Module 9: Data Dependencies and Normal Forms

Winter 2026

Cheriton School of Computer Science

CS 348: Intro to Database Management

Reading Assignments and References

To be read during the Week of March 9–13:

- Chapter 7 of course textbook.[1]

[1] Silberschatz, Korth and Sudarshan, *Database Systems Concepts*, 7^{th} edition

Outline

Relational Schema Diagnosis

There are many possibilities for the choice of a relational schema design.

1. Can one design be preferable to another?
2. If so, what are criteria to apply in arbitrating among the possibilities?

- (*usability*) How does a choice of design impact productivity?

  $\Rightarrow$ *ease of expressing queries*

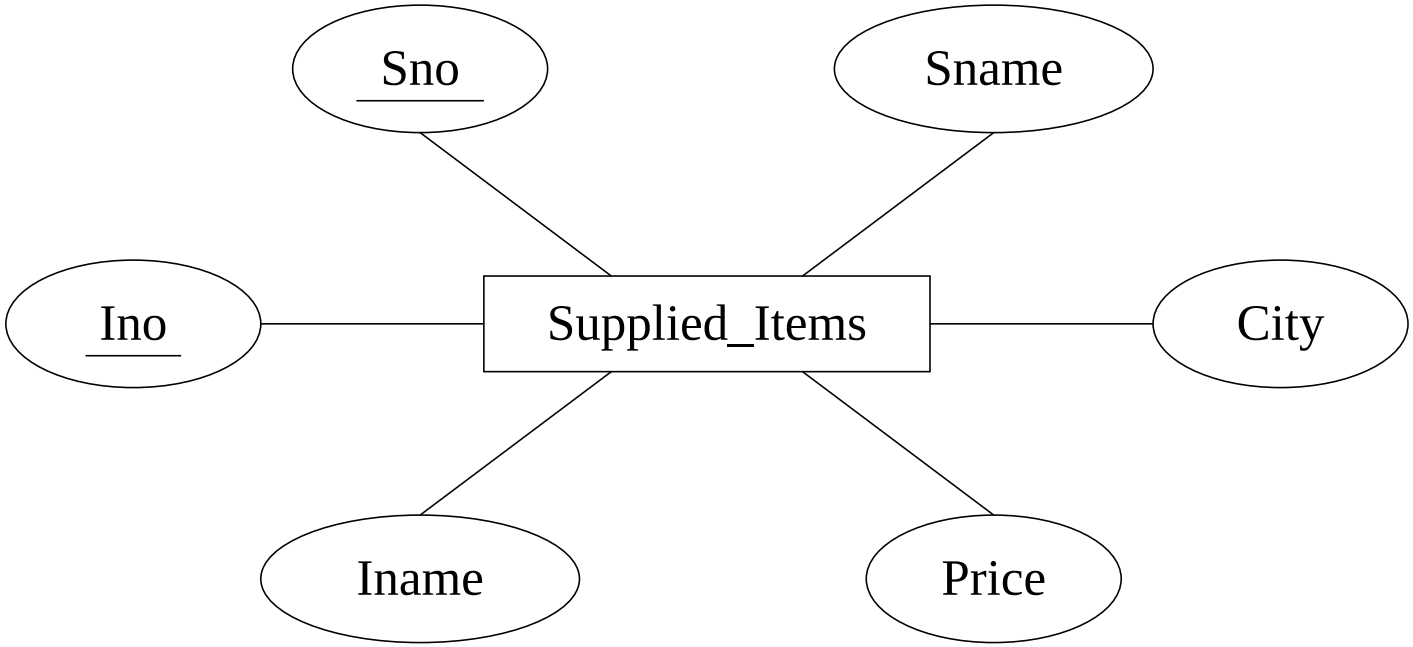
  $\Rightarrow$ *ease of expressing data revision*

  $\Rightarrow$ *metadata transparency*

- (*resource requirements*) What are the allowed instances of the schema?

Change Anomalies

Assume conceptual design has yielded the following ER diagram for a hypothetical inventory application.



Informally: *A supplied item entity has facts about the supplier, item and price.*

Change Anomalies (cont'd)

Consider our prescribed relational schema design and a selection of sample data:

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

Change Anomalies (cont'd)

Consider our prescribed relational schema design and a selection of sample data:

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

- No clear problems with expressing queries, e.g., *all cities with suppliers that supply screws*.

Change Anomalies (cont'd)

Consider our prescribed relational schema design and a selection of sample data:

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

- No clear problems with expressing queries, e.g., *all cities with suppliers that supply screws*.
- Clearly problems with expressing data revision however:
  1. updating existing data, e.g. *change the name of supplier* S1 *to* "ACME",
  2. adding new data, e.g. *add item* I4 *with the name* "Washer", and
  3. deleting existing data, e.g. *supplier named* "Budd" *no longer supplies screws*.

Change Anomalies (cont'd)

Consider our prescribed relational schema design and a selection of sample data:

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

- No clear problems with expressing queries, e.g., *all cities with suppliers that supply screws*.
- Clearly problems with expressing data revision however:
  1. updating existing data, e.g. *change the name of supplier* S1 *to* "ACME",
  2. adding new data, e.g. *add item* I4 *with the name* "Washer", and
  3. deleting existing data, e.g. *supplier named* "Budd" *no longer supplies screws*.
- No clear problems with metadata transparency, but very likely problems in resource requirements with a choice of RDBMS vendor.

Change Anomalies (cont'd)

Compare to an alternative relational schema design that decomposes table Supplied_Items into three separate tables:

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

The decomposition makes data revision much easier.

Exercise: *Does the decomposition make query writing easier? If not, how can this also be repaired?* (Hint: Consider defining views.)

Change Anomalies (cont'd)

Replacing a table with two or more tables is not always a good idea:

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

Change Anomalies (cont'd)

Replacing a table with two or more tables is not always a good idea:

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

Question: How can one *diagnose* the possibility of anomalous data revision, and of a repair by decomposition?

Diagnosis and Repair of Change Anomalies

Explore diagnosis and repair *by appeal to the integrity constraints of a relational database schema*.

- Integrity constraints can imply regularities in database instances that lead to change anomalies.

A possible repair is to replace a table by two or more other tables.

Decomposition

Assume a relation schema is given by set of attributes $\{A_1, \ldots, A_k\}$, and let R and $\{R_1, \ldots, R_n\}$ be a relational schema and a set of n relational schemata, respectively. Then $\{R_1, \ldots, R_n\}$ is a *decomposition* of R if

$$R = \bigcup_{1 \leq i \leq n} R_i.$$

- Integrity constraints can determine that a decomposition does not lose information.

Outline

Functional Dependencies

A small fragment of the variety of *equality generating dependencies* in RC over a single table have proven to be ubiquitous in conceptual design.

Functional Dependency (FD)

Let R be a k-ary relation $R/(A_1, \ldots, A_k)$, where $\ell(i) = A_i$, $1 \leq i \leq k$, and let X, Y be (not necessarily proper) subsets of $\{A_1, \ldots, A_k\}$. A *functional dependency* over $\{A_1, \ldots, A_k\}$ (the attributes of R) is written

$$X \to Y$$

and is shorthand for the RC integrity constraint

$$\forall v_1, \ldots, v_k, v'_1, \ldots, v'_k . R(v_1, \ldots, v_k) \wedge R(v'_1, \ldots, v'_k) \wedge \left(\bigwedge_{A \in X} v_{\ell^{-1}(A)} \approx v'_{\ell^{-1}(A)} \right) \to \left(\bigwedge_{A \in Y} v_{\ell^{-1}(A)} \approx v'_{\ell^{-1}(A)} \right).$$

NOTE: FD syntax assumes attributes are given for columns.

We say that (the set of attributes) X *functionally determines* (the set of attributes) Y in R, and usually conflate a relation schema with its set of attributes.

Functional Dependencies (cont'd)

Consider the following relation schema:

EmpProj

SIN	PNum	Hours	EName	PName	PLoc	Allowance

Functional Dependencies (cont'd)

Consider the following relation schema:

EmpProj

SIN	PNum	Hours	EName	PName	PLoc	Allowance

Examples of functional dependencies over EmpProj:[†]

- *Employee SIN numbers functionally determine employee names.*

  $\text{SIN} \rightarrow \text{EName}$

[†] We elide any use of curly braces in defining the left or right-hand-sides of FDs.

Functional Dependencies (cont'd)

Consider the following relation schema:

EmpProj

SIN	PNum	Hours	EName	PName	PLoc	Allowance

Examples of functional dependencies over EmpProj:[†]

- *Employee SIN numbers functionally determine employee names.*

  $\text{SIN} \rightarrow \text{EName}$

- *Project numbers functionally determine project names and locations.*

  $\text{PNum} \rightarrow \text{PName, PLoc}$

[†] We elide any use of curly braces in defining the left or right-hand-sides of FDs.

Functional Dependencies (cont'd)

Consider the following relation schema:

EmpProj

SIN	PNum	Hours	EName	PName	PLoc	Allowance

Examples of functional dependencies over EmpProj:[†]

- *Employee SIN numbers functionally determine employee names.*

  $\text{SIN} \rightarrow \text{EName}$

- *Project numbers functionally determine project names and locations.*

  $\text{PNum} \rightarrow \text{PName, PLoc}$

- *Project locations and the number of hours functionally determine allowances.*

  $\text{PLoc, Hours} \rightarrow \text{Allowance}$

[†] We elide any use of curly braces in defining the left or right-hand-sides of FDs.

Functional Dependencies (cont'd)

Consider the following relation schema:

EmpProj

SIN	PNum	Hours	EName	PName	PLoc	Allowance

Examples of functional dependencies over EmpProj:[†]

- *Employee SIN numbers functionally determine employee names.*

  $\text{SIN} \rightarrow \text{EName}$

- *Project numbers functionally determine project names and locations.*

  $\text{PNum} \rightarrow \text{PName, PLoc}$

- *Project locations and the number of hours functionally determine allowances.*

  $\text{PLoc, Hours} \rightarrow \text{Allowance}$

Exercise: *What functional dependencies should hold as a consequence of the primary key of* EmpProj*?*

[†] We elide any use of curly braces in defining the left or right-hand-sides of FDs.

Logical Consequence with Functional Dependencies

Functional Dependency Closure

Given a relation R with attributes $\{A_1, \ldots, A_k\}$ and a set $F \cup \{X \to Y\}$ of functional dependencies over R, we say

$$F \textit{ logically implies } X \to Y$$

whenever $X \to Y$ holds in all instances of R that satisfies each FD in F.

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

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

Logical Consequence with Functional Dependencies

Functional Dependency Closure

Given a relation R with attributes $\{A_1, \ldots, A_k\}$ and a set $F \cup \{X \to Y\}$ of functional dependencies over R, we say

$$F \textit{ logically implies } X \to Y$$

whenever $X \to Y$ holds in all instances of R that satisfies each FD in F.

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

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

Example: *If* $F = \{A \to B, B \to C\}$, *then* $\{A \to B, B \to C, A \to C\} \subseteq F^+$.

Reasoning about Functional Dependencies

Armstrong's Axioms

Armstrong's axioms for inferring FDs from FDs are as follows:

1. (*reflexivity*) If $Y \subseteq X$, then infer $X \rightarrow Y$ holds.
2. (*augmentation*) If $X \rightarrow Y$ holds, then infer $\Rightarrow XZ \rightarrow YZ$ holds.
3. (*transitivity*) If $X \rightarrow Y$ and $Y \rightarrow Z$ hold, then infer $X \rightarrow Z$ holds.

where XZ and YZ are shorthands for $X \cup Z$ and $Y \cup Z$.

The axioms are *sound*: anything derived from F is in F^+.
The axioms are also *complete*: anything in F^+ can be derived.

Reasoning about Functional Dependencies

Armstrong's Axioms

Armstrong's axioms for inferring FDs from FDs are as follows:

1. (*reflexivity*) If $Y \subseteq X$, then infer $X \rightarrow Y$ holds.
2. (*augmentation*) If $X \rightarrow Y$ holds, then infer $\Rightarrow XZ \rightarrow YZ$ holds.
3. (*transitivity*) If $X \rightarrow Y$ and $Y \rightarrow Z$ hold, then infer $X \rightarrow Z$ holds.

where XZ and YZ are shorthands for $X \cup Z$ and $Y \cup Z$.

The axioms are *sound*: anything derived from F is in F^+.
The axioms are also *complete*: anything in F^+ can be derived.

Additional rules are useful:

4. (*union*) If $X \rightarrow Y$ and $X \rightarrow Z$ hold, then infer $X \rightarrow YZ$ holds.
5. (*decomposition*) If $X \rightarrow YZ$ holds, then infer $X \rightarrow Y$ holds.

Reasoning about Functional Dependencies

Armstrong's Axioms

Armstrong's axioms for inferring FDs from FDs are as follows:

1. (*reflexivity*) If $Y \subseteq X$, then infer $X \rightarrow Y$ holds.
2. (*augmentation*) If $X \rightarrow Y$ holds, then infer $\Rightarrow XZ \rightarrow YZ$ holds.
3. (*transitivity*) If $X \rightarrow Y$ and $Y \rightarrow Z$ hold, then infer $X \rightarrow Z$ holds.

where XZ and YZ are shorthands for $X \cup Z$ and $Y \cup Z$.

The axioms are *sound*: anything derived from F is in F^+.
The axioms are also *complete*: anything in F^+ can be derived.

Additional rules are useful:

4. (*union*) If $X \rightarrow Y$ and $X \rightarrow Z$ hold, then infer $X \rightarrow YZ$ holds.
5. (*decomposition*) If $X \rightarrow YZ$ holds, then infer $X \rightarrow Y$ holds.

Exercise: *Show how the union and decomposition axioms can themselves be derived from Armstrong's axioms.*

Reasoning about Functional Dependencies (cont'd)

Example: *Let F be the following FDs over EmpProj*:

$$F = \{ \quad \begin{array}{l} \text{SIN} \rightarrow \text{EName} \\ \text{PNum} \rightarrow \text{PName, PLoc} \\ \text{PLoc, Hours} \rightarrow \text{Allowance} \\ \text{SIN, PNum} \rightarrow \text{Hours}^{\dagger} \ \} \end{array}$$

The FD SIN, PNum $\rightarrow$ Allowance can be derived from F as follows:

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

---

†Exercise: *Why should any* EmpProj *instance satisfy this FD?*

Functional Dependencies and Keys

Superkey

Given a relation R with attributes $\{A_1, \ldots, A_k\}$ and subset K of R, K is a *superkey* of R if the FD $K \rightarrow R$ holds on any instance of R.

Candidate Key

Given a relation R with attributes $\{A_1, \ldots, A_k\}$ and subset K of R, K is a *candidate key* of R if the FD $K \rightarrow R$ holds on any instance of R, and no strict subset of K is a superkey.

Primary Key

Given a relation R with attributes $\{A_1, \ldots, A_k\}$, the *primary key* of R is a distinguished *candidate key* of R chosen by the DBA group.

Exercise: *Determine criteria for a DBA group to apply in selecting a primary key.*

An Optimal Algorithm for Deciding Logical Consequence of FDs

Computing an Attribute Closure

Assume R is a set of attributes, X a subset of R, and F a set of FDs over R. *Compute*X^+ returns a subset of R computed as follows:

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

ComputeX^+ can be implemented to run in linear time.

Correctness Theorem for *Compute*X^+

$(X \rightarrow Y) \in F^+$ if and only if $Y \subseteq$ *Compute*$X^+(X, F)$.

An Optimal Algorithm (cont'd)

Assume R is a set of attributes, X and Y are subsets of R, and F a set of FDs over R in the following.

Superkey Theorem for Functional Dependencies

X is a superkey of a relation with schema R if and only if $R \subseteq \mathit{ComputeX}^{+}(X, F)$.

An Optimal Algorithm (cont'd)

Assume R is a set of attributes, X and Y are subsets of R, and F a set of FDs over R in the following.

Superkey Theorem for Functional Dependencies

X is a superkey of a relation with schema R if and only if $R \subseteq \mathit{ComputeX}^+(X, F)$.

Exercises:

- *Assuming a set semantics, derive an efficient algorithm for computing a candidate key of R.*[†]
- *Can an efficient algorithm exist that computes all candidate keys of R?*
- *Does one lose expressiveness by disallowing any FD $X \rightarrow Y$ for which Y has more than one attribute?*
- *Does one lose expressiveness by disallowng any FD $X \rightarrow Y$ for which X has more than two attibutes?*
- *Prove the correctness theorem for ComputeX$^+$.*

[†] ... where efficient means polynomial time.

Outline

Repair by Decomposition

Assume a given relational database schema $\langle \rho, \Sigma \rangle$, has already undergone a repair by decomposition.

Also assume for this unit that Σ consists of a set of functional dependencies F.

Repair by Decomposition

Assume a given relational database schema $\langle \rho, \Sigma \rangle$, has already undergone a repair by decomposition.

Also assume for this unit that Σ consists of a set of functional dependencies F.

A good decomposition obtaining the schema should satisfy the following conditions:

- (lossless-join) no loss of information in comparison to the original schema before decomposition

Repair by Decomposition

Assume a given relational database schema $\langle \rho, \Sigma \rangle$, has already undergone a repair by decomposition.

Also assume for this unit that Σ consists of a set of functional dependencies F.

A good decomposition obtaining the schema should satisfy the following conditions:

- (lossless-join) no loss of information in comparison to the original schema before decomposition,
- (dependency preserving) no added complications in checking for violation of FDs in F

Repair by Decomposition

Assume a given relational database schema $\langle \rho, \Sigma \rangle$, has already undergone a repair by decomposition.

Also assume for this unit that Σ consists of a set of functional dependencies F.

A good decomposition obtaining the schema should satisfy the following conditions:

- (lossless-join) no loss of information in comparison to the original schema before decomposition,
- (dependency preserving) no added complications in checking for violation of FDs in F, and
- (normal form) no change anomalies (or at least fewer anomalies) in expressing data revision.

Lossless-Join Decompositions

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

Example: *Consider where the table*

Marks

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

has been replaced by the following two tables in the decomposition.

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.)

A query in SQL that expresses the *natural join* of tables SGM and AM is given as follows:

```
select distinct Student, Assignment, Group, x.Mark as Mark
form SGM x, AM y
where x.Mark = y.Mark
```

Computing the natural join of SGM and AM produces the following table.

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

Lossless-Join Decompositions (cont.)

A query in SQL that expresses the *natural join* of tables SGM and AM is given as follows:

```
select distinct Student, Assignment, Group, x.Mark as Mark
form SGM x, AM y
where x.Mark = y.Mark
```

Computing the natural join of SGM and AM produces the following table.

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

This table *looses* information in comparison to the original table by virtue of computing extra tuples, called spurious tuples.

Lossless-Join Decompositions (cont.)

Lossless-Join Decomposition

A decomposition $\{R_1, R_2\}$ of R is a *lossless-join* decomposition if, for any instance of R, the natural join of R_1 and R_2 has no spurious tuples.

Theorem

A decomposition $\{R_1, R_2\}$ of R is a lossless-join decomposition if and only if the common attributes of R_1 and R_2 form a superkey for either schema, that is, either $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$ is a logical consequence of the integrity contraints.

Lossless-Join Decompositions (cont.)

Lossless-Join Decomposition

A decomposition $\{R_1, R_2\}$ of R is a *lossless-join* decomposition if, for any instance of R, the natural join of R_1 and R_2 has no spurious tuples.

Theorem

A decomposition $\{R_1, R_2\}$ of R is a lossless-join decomposition if and only if the common attributes of R_1 and R_2 form a superkey for either schema, that is, either $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$ is a logical consequence of the integrity contraints.

Example: *With table* Marks *we have*

R = {Student, Assignment, Group, Mark}
F = {Student, Assignment $\rightarrow$ Group, Mark}

R_1 = SGM = {Student, Group, Mark}
R_2 = AM = {Assignment, Mark}

Decomposition $\{R_1, R_2\}$ *is lossy because* $\{\text{Mark}\}$ $(= R_1 \cap R_2)$ *is not a superkey of either* SGM *or* AM *according to integrity constraints* F.

Dependency Preservation

How do we check that integrity constraints are not violated after data revision on a decomposition?

Dependency Preservation

How do we check that integrity constraints are not violated after data revision on a decomposition?

Example: *Assume an original table and integrity constraints over the table are given as follows:*

R / (Proj, Dept, Div)

F = { FD1: Proj $\rightarrow$ Dept, FD2: Dept $\rightarrow$ Div, and FD3: Proj $\rightarrow$ Div }

and consider two possible decompositions:

D_1 = { R1 / (Proj, Dept), R2 / (Dept, Div)}

D_2 = { R1 / (Proj, Dept), R3 / (Proj, Div)}

Observe that both are lossless. (Why?)

Dependency Preservation (cont.)

Which decomposition makes it easier to check that all constraints in F continue to hold after a revision to one or more of its tables?

- Decomposition D_1 lets us test FD1 on table R1 and FD2 on table R2; if they are both satisfied, FD3 is automatically satisfied.
- Decomposition D_2 lets us test FD1 on table R1 and FD3 on table R3. Dependency FD2 is an *inter-relational constraint* that needs to be checked on a natural join of tables R1 and R3.

D_1 is therefore preferable to D_2 since it avoids the need to check inter-relational constraints.

Dependency Preservation (cont.)

Which decomposition makes it easier to check that all constraints in F continue to hold after a revision to one or more of its tables?

- Decomposition D_1 lets us test FD1 on table R1 and FD2 on table R2; if they are both satisfied, FD3 is automatically satisfied.
- Decomposition D_2 lets us test FD1 on table R1 and FD3 on table R3. Dependency FD2 is an *inter-relational constraint* that needs to be checked on a natural join of tables R1 and R3.

D_1 is therefore preferable to D_2 since it avoids the need to check inter-relational constraints.

Dependency Preserving Decomposition

Let R be a relational schema and F a set of functional dependencies over R. A decomposition $D = \{R_1, \ldots, R_n\}$ of R is *dependency preserving* if there is an equivalent set G of functional dependencies, none of which is inter-relational in D.

Normal Forms that Avoid Change Anomalies

Rule of Thumb: Facts about *different* entites should be recorded in separate tables, that is:

> *each relation schema should consist of two separate sets of attributes: the first comprising a primary key to record facts that identify an entity, and the second to record independent facts about the entity.*

Normal Forms that Avoid Change Anomalies

Rule of Thumb: Facts about *different* entites should be recorded in separate tables, that is:

> *each relation schema should consist of two separate sets of attributes: the first comprising a primary key to record facts that identify an entity, and the second to record independent facts about the entity.*

We attempt to achieve this by decomposition, if necessary, of a schema to a normal form.

Goals:

- intuitive and straightforward data revision that avoids anomalies, and
- nonredundant representation of data.

Normal Forms that Avoid Change Anomalies

Rule of Thumb: Facts about *different* entites should be recorded in separate tables, that is:

> *each relation schema should consist of two separate sets of attributes: the first comprising a primary key to record facts that identify an entity, and the second to record independent facts about the entity.*

We attempt to achieve this by decomposition, if necessary, of a schema to a normal form.

Goals:

- intuitive and straightforward data revision that avoids anomalies, and
- nonredundant representation of data.

We discuss:

- Boyce-Codd Normal Form (BCNF), and
- Third Normal Form (3NF).

Both normal forms are based on the integrity constraints that are FDs.

Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF)

Schema R is in *BCNF* with respect to a set of functional dependencies F if and only if, whenever $(X \to Y) \in F^+$ where $XY \subseteq R$, then either

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

A database schema $\langle \rho = \{R_1, \ldots, R_n\}, F \rangle$ is in BCNF if each relation schema R_i is in BCNF with respect to F.

BCNF is our first attempt to formalize our rule of thumb.

Boyce-Codd Normal Form (cont'd)

Why does BCNF avoid redundancy?

Example: *For the schema* Supplied_Items *we had the FD:*

$$\text{Sno} \rightarrow \text{Sname, City}$$

Consequently, the supplier name Magna *and the 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:

1. Sno *is a superkey for R (by definition of BCNF).*
2. *Therefore* S1 *appears on one row only.*
3. *And therefore* Magna *and the city* Ajax *are not repeatedly recorded in R for supplier* S1.

An Algorithm for Computing a BCNF Decomposition

Computing a BCNF Decomposition

Assume R is a set of attributes and F a set of FDs. *ComputeBCNF*(R, F) returns a decomposition of R as follows:

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 $X \cup Y$ to Result;
    end;
    return Result;
end

Correctness Theorem for *ComputeBCNF*

Function *ComputeBCNF*(R, F) returns a lossless-join decomposition of R for which each table in the decomposition is in BCNF with respect to F.

On Computing a Lossless Join BCNF Decomposition

- No *efficient* procedure to do this exists.

On Computing a Lossless Join BCNF Decomposition

- No *efficient* procedure to do this exists.
- Results depend on sequence of FDs used to decompose the relations.

On Computing a 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.

  Example: *Consider* $R = \{A, B, C\}$ *and* $F = \{AB \rightarrow C, C \rightarrow B\}$.

  $\Rightarrow$ R is not in BCNF with respect to F

  $\Rightarrow$ $AB \rightarrow C$ will be an inter-relational FD in any decomposition of R

Third Normal Form

Third Normal Form (3NF)

Schema R is in *3NF* with respect to a set of functional dependencies F if and only if, whenever $(X \to Y) \in F^+$ and $XY \subseteq R$, then either

- $(X \to Y)$ is trivial,
- X is a superkey of R, or
- each attribute of $Y - X$ is contained in a candidate key of R.

A database schema $\langle \rho = \{R_1, \ldots, R_n\}, F \rangle$ is in 3NF if each relation schema R_i is in 3NF with respect to F.

Third Normal Form

Third Normal Form (3NF)

Schema R is in *3NF* with respect to a set of functional dependencies F if and only if, whenever $(X \rightarrow Y) \in F^+$ and $XY \subseteq R$, then either

- $(X \rightarrow Y)$ is trivial,
- X is a superkey of R, or
- each attribute of $Y - X$ is contained in a candidate key of R.

A database schema $\langle \rho = \{R_1, \ldots, R_n\}, F \rangle$ is in 3NF if each relation schema R_i is in 3NF with respect to F.

- BCNF implies 3NF, but not the reverse.

  $\Rightarrow$ allows more redundancy

  Example: $R =$ {A, B, C} is in 3NF with respect to $F =$ {AB $\rightarrow$ C, C $\rightarrow$ B}

Third Normal Form

Third Normal Form (3NF)

Schema R is in *3NF* with respect to a set of functional dependencies F if and only if, whenever $(X \rightarrow Y) \in F^+$ and $XY \subseteq R$, then either

- $(X \rightarrow Y)$ is trivial,
- X is a superkey of R, or
- each attribute of $Y - X$ is contained in a candidate key of R.

A database schema $\langle \rho = \{R_1, \ldots, R_n\}, F \rangle$ is in 3NF if each relation schema R_i is in 3NF with respect to F.

- BCNF implies 3NF, but not the reverse.

  $\Rightarrow$ allows more redundancy

  Example: $R =$ {A, B, C} is in 3NF with respect to $F =$ {AB $\rightarrow$ C, C $\rightarrow$ B}

- A lossless-join and dependency-preserving decomposition into 3NF relation schemata always exists.

Minimal Cover

Dependency Set Equivalence for Functional Dependencies

Two sets of functional dependencies F and G are *equivalent* if and only if $F^+ = G^+$.

Minimal Cover

Dependency Set Equivalence for Functional Dependencies

Two sets of functional dependencies F and G are *equivalent* if and only if $F^+ = G^+$.

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

Minimal FD Sets

A set of dependencies F is *minimal* if it satisfies the following conditions:

1. Every right-hand-side of a 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.

Minimal Cover

Dependency Set Equivalence for Functional Dependencies

Two sets of functional dependencies F and G are *equivalent* if and only if $F^+ = G^+$.

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

Minimal FD Sets

A set of dependencies F is *minimal* if it satisfies the following conditions:

1. Every right-hand-side of a 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, called a *minimal cover*.

Minimal Covers (cont'd)

Computing a Minimal Cover

Given a set of functional dependencies F, the following steps modify F to a minimal cover, where each step is repeatedly applied 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 $A \in \mathit{ComputeX}^{+}(X, F - \{X \rightarrow A\})$.

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

Minimal Covers (cont'd)

Computing a Minimal Cover

Given a set of functional dependencies F, the following steps modify F to a minimal cover, where each step is repeatedly applied 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 $A \in \mathit{ComputeX}^{+}(X, F - \{X \rightarrow A\})$.

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

A further step can be taken to reduce the number of FDs:

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

Computing a 3NF Decomposition

Computing a 3NF Decomposition

Assume R is a set of attributes and F a set of FDs over R. *Compute*3NF(R, F) returns a decomposition of R as follows:

function *Compute*3NF(R, F)
begin
    Result := $\emptyset$;
    G := a minimal cover for F;
    for each $(X \rightarrow Y) \in G$ **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

Computing a 3NF Decomposition (cont'd)

Correctness Theorem for *Compute*3NF

Function *Compute*3NF(R, F) returns a lossless-join decomposition of R that is also dependency preserving and for which each table in the decomposition is in 3NF with respect to F.

Summary

- Functional dependencies provide clues towards elimination of (some) *redundancies* in a relational schema.
- Goals: to repair relational schemata by decomposition which is
  1. not lossy,
  2. dependency preserving, and
  3. for which each resulting table is in BCNF, or at least in 3NF.

Outline

Beyond Functional Dependencies

There exist update anomalies and data redundancies in relational schemata that cannot be captured by FDs.

Example: *Consider the following relation schema and sample valid data:*

Course	**T**eacher	**B**ook
Math	Smith	Algebra
Math	Smith	Calculus
Math	Jones	Algebra
Math	Jones	Calculus
Advanced Math	Smith	Calculus
Physics	Black	Mechanics
Physics	Black	Optics

There are no non-trivial FDs that hold on this instance.

The scheme (**C**ourse, **T**eacher, **B**ook) *or* CTB *for short, is therefore in BCNF.*

Multivalued Dependencies

- The CTB table contains redundant information for the following reason.

  Whenever a pair of tuples (c, t_1, b_1) *and* (c, t_2, b_2) *occurs in* CTB *then the tuple* (c, t_1, b_2) *also occurs in* CTB.

  By symmetry, the tuple (c, t_2, b_1) *also occurs in* CTB.

Multivalued Dependencies

- The CTB table contains redundant information for the following reason.

  Whenever a pair of tuples (c, t_1, b_1) *and* (c, t_2, b_2) *occurs in* CTB *then the tuple* (c, t_1, b_2) *also occurs in* CTB.

  By symmetry, the tuple (c, t_2, b_1) *also occurs in* CTB.

- We say that a multivalued dependency (MVD) written

$$C \twoheadrightarrow T$$

  holds on CTB (and, by symmetry, the MVD $C \twoheadrightarrow B$ as well).

Multivalued Dependencies

- The CTB table contains redundant information for the following reason.

  Whenever a pair of tuples (c, t_1, b_1) *and* (c, t_2, b_2) *occurs in* CTB *then the tuple* (c, t_1, b_2) *also occurs in* CTB.

  By symmetry, the tuple (c, t_2, b_1) *also occurs in* CTB.

- We say that a multivalued dependency (MVD) written

$$C \twoheadrightarrow T$$

holds on CTB (and, by symmetry, the MVD C $\twoheadrightarrow$ B as well).

Given a course, the set of teachers are uniquely determined and independent of the remaining attributes. (The course also uniquely determines a set of books.)

Another Example

Course	**T**eacher	**H**our	**R**oom	**S**tudent	**G**rade
CS101	Jones	M-9	2222	Smith	A
CS101	Jones	W-9	3333	Smith	A
CS101	Jones	F-9	2222	Smith	A
CS101	Jones	M-9	2222	Black	B
CS101	Jones	W-9	3333	Black	B
CS101	Jones	F-9	2222	Black	B

- *FDs that hold on any instance:*

$$C \rightarrow T,\ CS \rightarrow G,\ HR \rightarrow C,\ HT \rightarrow R,\ HS \rightarrow R$$

- *An MVD that also holds on any instance:*

$$C \twoheadrightarrow HR$$

Inferring FDs and MVDs

Inference Axioms

The following axioms infer both FDs and MVDs from FDs and MVDs:

1. (*reflexivity*) $Y \subset X \Rightarrow X \twoheadrightarrow Y$
2. (*complementation*) $X \twoheadrightarrow Y \Rightarrow X \twoheadrightarrow (R - Y)$
3. (*augmentation*) $X \twoheadrightarrow Y \Rightarrow XZ \twoheadrightarrow YZ$
4. (*transitivity*) $X \twoheadrightarrow Y, Y \twoheadrightarrow Z \Rightarrow X \twoheadrightarrow (Z - Y)$
5. (*conversion*) $X \rightarrow Y \Rightarrow X \twoheadrightarrow Y$
6. (*interaction*) $X \twoheadrightarrow Y, XY \rightarrow Z \Rightarrow X \rightarrow (Z - Y)$

Theorem

The inference axioms are sound and complete for logical implication of FDs and MVDs.

Inferring FDs and MVDs (cont'd)

Example: *In the* CTHRSG *relation schema* C $\twoheadrightarrow$ SG *can be derived as follows:*

1.	C $\twoheadrightarrow$ HR	(given)
2.	C $\twoheadrightarrow$ T	(from C $\rightarrow$ T)
3.	C $\twoheadrightarrow$ CTSG	(complementation of 1)
4.	C $\twoheadrightarrow$ CT	(augmentation of 2 by C)
5.	CT $\twoheadrightarrow$ CTSG	(augmentation of 3 by T)
6.	C $\twoheadrightarrow$ SG	(transitivity on 4 and 5)

Reasoning about FDs and MVDs

Dependency Basis

Given a relation schema R and $X \subseteq R$, a *dependency basis* for X with respect to a set of FDs and MVDs F is a partition of $R - X$ to subsets $\{Y_1, \ldots, Y_k\}$ such that F logically implies $X \twoheadrightarrow Z$ if and only if $Z - X$ is a union over some subset of $\{Y_1, \ldots, Y_k\}$.

Reasoning about FDs and MVDs

Dependency Basis

Given a relation schema R and $X \subseteq R$, a *dependency basis* for X with respect to a set of FDs and MVDs F is a partition of $R - X$ to subsets $\{Y_1, \ldots, Y_k\}$ such that F logically implies $X \twoheadrightarrow Z$ if and only if $Z - X$ is a union over some subset of $\{Y_1, \ldots, Y_k\}$.

- Unlike the case with FDs, it is not possible split right-hand-sides of MVDs to single attributes (cf. minimal cover).

Reasoning about FDs and MVDs

Dependency Basis

Given a relation schema R and $X \subseteq R$, a *dependency basis* for X with respect to a set of FDs and MVDs F is a partition of $R - X$ to subsets $\{Y_1, \ldots, Y_k\}$ such that F logically implies $X \twoheadrightarrow Z$ if and only if $Z - X$ is a union over some subset of $\{Y_1, \ldots, Y_k\}$.

- Unlike the case with FDs, it is not possible split right-hand-sides of MVDs to single attributes (cf. minimal cover).
- The dependency basis of X w.r.t. F can be computed in PTIME.

  Example: *For the relation schema* CTHRSG*, the dependency basis of* C *with respect to the schema's FDs and MVDs is* {T, HR, SG}.

Lossless-Join Decomposition

As with FDs, lossless join decompositions can be defined in terms of MVD inference.

Theorem

Given a relational schema R and a set F of FDs and MVDs over R, a decomposition $\{R_1, R_2\}$ of R is a lossless-join decomposition if and only if either

$$F \models (R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)$$

or, by symmetry

$$F \models (R_1 \cap R_2) \twoheadrightarrow (R_2 - R_1)$$

(where we use "$\models$" as shorthand for "logically implies").

The theorem is a generalization of the case for FDs alone.

Fourth Normal Form

Fourth Normal Form (4NF)

Schema R is in *fourth normal form* (4NF) with respect to a set of FDs and MVDs F if and only if, whenever $(X \twoheadrightarrow Y) \in F^+$ and $XY \subseteq R$, then either

- $(X \twoheadrightarrow Y)$ is trivial ($Y \subseteq X$ or $XY = R$), or
- X is a superkey of R.

A database schema $\langle \rho = \{R_1, \ldots, R_n\}, F \rangle$ is in 4NF if each relation schema R_i is in 4NF with respect to F.

- One can use a BCNF-like decomposition procedure to obtain a lossless-join decomposition into 4NF.

Fourth Normal Form (cont'd)

Example: *The* CTB *relation schema can be decomposed to 4NF by appeal to the MVD* C $\twoheadrightarrow$ T *to the following pair of tables:*

Course	Teacher
Math	Smith
Math	Jones
Physics	Black
Advanced Math	Smith

Course	Book
Math	Algebra
Math	Calculus
Physics	Mechanics
Physics	Optics
Advanced Math	Calculus

Note again that no FDs hold over CTB.

Join and Inclusion Dependencies

A join dependency on a relation schema R is expressed with the syntax

$$\bowtie [R_1, \ldots, R_k]$$

where $\{R_1, \ldots, R_k\}$ is a decomposition of R, and holds over an instance of R if the instance satisfies the following integrity constraints:

1. $\forall \mathbf{x_i}.R_i(\mathbf{x_i}) \leftrightarrow \exists \mathbf{y}.R(\mathbf{x_i}, \mathbf{y})$, for $1 \leq i \leq k$ (each R_i is a *projection* of R), and
2. $\forall \mathbf{x}.R(\mathbf{x}) \leftrightarrow R_1(\mathbf{x_1}) \wedge \ldots \wedge R_k(\mathbf{x_k})$ (the natural join of the R_i is R).

Join and Inclusion Dependencies

A join dependency on a relation schema R is expressed with the syntax

$$\bowtie [R_1, \ldots, R_k]$$

where $\{R_1, \ldots, R_k\}$ is a decomposition of R, and holds over an instance of R if the instance satisfies the following integrity constraints:

1. $\forall \mathbf{x_i}.R_i(\mathbf{x_i}) \leftrightarrow \exists \mathbf{y}.R(\mathbf{x_i}, \mathbf{y})$, for $1 \leq i \leq k$ (each R_i is a *projection* of R), and
2. $\forall \mathbf{x}.R(\mathbf{x}) \leftrightarrow R_1(\mathbf{x_1}) \wedge \ldots \wedge R_k(\mathbf{x_k})$ (the natural join of the R_i is R).

An inclusion dependency on relation schemata R and S is expressed with the syntax

$$R[X] \subseteq S[Y]$$

where X and Y are respective subsets of R and S, and holds over an instance of R and an instance of S if the *projection* of R on X is a subset of the *projection* of S on Y.

$\Rightarrow$ *generalizes foreign-key constraints*

Join and Inclusion Dependencies (cont'd)

- Join dependencies are a generalization of MVDs: $X \twoheadrightarrow Y$ is the same as $\bowtie [XY, X(R - Y)]$
- Join dependencies *cannot* be simulated by MVDs.
- No axiomatization of join dependencies exists.
- There is a *project-join normal form* (5NF): $\bowtie [R_1, \ldots, R_k]$ implies each R_i is a superkey.
- Logical consequence for FDs and binary inclusion dependencies is undecidable.