Logical Database Design: Outline

- Motivation
  - Update anomalies
  - Properties of a good design
- Logical Design
  - Lossless-Join Decomposition
  - Functional Dependency
  - Normalization
- Normal Forms
  - Boyce-Codd Normal Form (BCNF)
  - 3th Normal Form (3NF)

Logical Database Design

- Motivation
  - Update anomalies
  - Properties of a good design
- Logical Design
- Normal Forms

Redundant Information

- Suppose a "big" schema *Rents(r)* is designed for both films and reservations of our Video Store
  - *Title*, *Price*, and *Kind* is repeated for each film.

CustomerID	*Title*	*Price*	*Kind*	*ResDate*
0001	True Lies	3.25	D	2006-04-19
0002	True Lies	3.25	D	2006-04-21
0001	The Lion King	3.25	C	2006-04-19
0003	The Lion King	3.25	C	2006-04-19
0001	Henry V	1.75	D	2006-04-18
NULL	The Matrix 4	3.25	D	NULL

- Wastes space
- Potential for inconsistent data is increased
  - Murphy's Law
- New movies?

Anomalies Resulting From A Bad Design

CustomerID	*Title*	*Price*	*Kind*	*ResDate*
0001	True Lies	3.25	D	2006-04-19
0002	True Lies	3.25	D	2006-04-21
0001	The Lion King	3.25	C	2006-04-19
0003	The Lion King	3.25	C	2006-04-19
0001	Henry V	1.75	D	2006-04-18

- Update anomaly
  - Update rental price to $4 for 'True Lies', have to change several tuples => anomaly if changes in some but not in all.
- Insertion anomaly
  - Cannot insert information about a film if it has no rentals.
- Deletion anomaly
  - If no rentals, information about the film disappears!

Goals of Logical Database Design

- Logical database design requires that we find a “good” collection of relation schemes.
  - Avoid redundant data.
  - Avoid modification anomalies.
  - Ensure that relationships among attributes are represented.
  - Facilitate the checking of updates for violation of database integrity constraints.
- Logical design methods apply even if initial relational schema is obtained without first designing an ER model.

Fix the Bad Design Example

- Solution: *decompose* the relation schema *Rents (r)* into:
  - R_1 = (CustomerID, Title, ResDate)
  - R_2 = (Title, Price, Kind)
- Requirements
  - All attributes of the original schema (R) must appear in the decomposition

    $$R = R_1 \cup R_2$$

  - *Lossless-join decomposition*: For all possible relations r on schema R,

    $$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$$

- How do we ensure that the decomposition is a lossless-join decomposition?

A Good Decomposition

CustomerID	*Title*	*ResDate*
0001	True Lies	2006-04-19
0002	True Lies	2006-04-21
0001	The Lion King	2006-04-19
0003	The Lion King	2006-04-19
0001	Henry V	2006-04-18

$$r_1 = \pi_{CustomerID,\ Title,\ ResDate}(r)$$

Title	*Price*	*Kind*
True Lies	3.25	D
The Lion King	3.25	C
Henry V	1.75	D

$$r_2 = \pi_{Title,\ Price,\ Kind}(r)$$

CustomerID	*Title*	*Price*	*Kind*	*Date*
0001	True Lies	3.25	D	2006-04-19
0002	True Lies	3.25	D	2006-04-21
0001	The Lion King	3.25	C	2006-04-19
0003	The Lion King	3.25	C	2006-04-19
0001	Henry V	1.75	D	2006-04-18

$$r_1 \bowtie r_2$$

A Bad Decomposition

CustomerID	*Price*	*ResDate*
0001	3.25	2006-04-19
0002	3.25	2006-04-21
0003	3.25	2006-04-19
0001	1.75	2006-04-18

Title	*Price*	*Kind*
True Lies	3.25	D
The Lion King	3.25	C
Henry V	1.75	D

$r_1 = \pi_{CustomerID,\ Price,\ ResDate}(r)$

$r_2 = \pi_{Title,\ Price,\ Kind}(r)$

CustomerID	*Title*	*Price*	*Kind*	*ResDate*
0001	True Lies	3.25	D	2006-04-19
0002	True Lies	3.25	D	2006-04-21
0001	The Lion King	3.25	C	2006-04-19
0003	The Lion King	3.25	C	2006-04-19
0001	Henry V	1.75	D	2006-04-18
0002	The Lion King	3.25	C	2006-04-21
0003	True Lies	3.25	D	2006-04-19

$r_1 \bowtie r_2$

Logical Database Design

- Motivation
- Logical Design
  - Lossless-Join Decomposition
  - Functional Dependency
  - Normalization
- Normal Forms

Process of Logical Design

- Normalization
  - "Norm" means *ideal* (as in a *normative* process)
- What is norm?
  - Set of conditions: *normal form*
  - Many different normal forms
- How to achieve norm?
  - Identify violating condition
  - Decompose relation(s) to avoid violation
    - Making more relations, but fewer columns
    - More joins during queries
    - *Denormalizing*: process of undoing normalization to improve query performance

Decompositions

- Let U be a relation schema.
- A set of relation schemas $\{R_1, R_2, \ldots, R_n\}$ is a *decomposition* of U if and only if

$$U = R_1 \cup R_2 \cup \ldots \cup R_n$$

- Extra information is needed to guarantee lossless join decomposition.
  - An *integrity constraint* is a condition which must be satisfied by all instances of a set of relational schemas.
  - Domain and key constraints are examples of integrity constraints.

Lossless-Join Decomposition

- A decomposition $\{R, T\}$ of U is a *lossless-join decomposition* (*with respect to a set of constraints*) if the constraints imply that

$$u = r \bowtie t$$

  for all ***possible*** instances of R, T, and U.
- The decomposition is said to be *lossy* otherwise.

- It is always the case for any *decomposition* $\{R, T\}$ of U that

$$u \subseteq r \bowtie t\,.$$

Functional Dependencies

- Functional dependencies generalize the previously introduced notions of keys.
- They also allow us to identify possible information loss from a given decomposition.

- Functional dependencies
  - Definition
  - Example
  - Closure of a set of dependencies
  - Closure of a set of attributes
  - Canonical representation: minimal cover

Functional Dependency Definition

- Definition: Let α and β be subsets of schema *R*. A *functional dependency* $\alpha \rightarrow \beta$ holds on *R* if for all legal instances *r* of *R*,

$$\forall\, t_1, t_2 \in r\ (t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta])$$

- We use a functional dependency *F* for two different purposes.
  - Intensionally: *F* holds for schema *R*
  - Extensionally: relation instance *r* satisfies *F*
- A functional dependency $\alpha \rightarrow \beta$ is *trivial* if $\beta \subseteq \alpha$

Functional Dependency Examples

- "A film has a unique title, rental price, and distributor."
  - Title → RentalPrice, Distributor
- "The CustomerID uniquely identifies the customer and his/her address."
  - CustomerID → Name, Street, City, State
- "Each video tape has a unique status."
  - FilmID, TapeNum → Status

More Functional Dependency Examples

- "On any particular day, a video tape can be checked out to at most one customer."
  - CheckDate, FilmID, TapeNum $\rightarrow$ CustomerID

- "A performer can have only one role in a particular film."
  - PerformerID, FilmID $\rightarrow$ Role

Closure of Functional Dependency

- Let F be a set of functional dependencies. The *closure* of F, denoted by F^+, consists of all dependencies implied by the dependencies in F. ($F \subseteq F^+$)
- Example
  - $R = (A, B, C, D)$
  - $F = \{ A \to B, A \to C, CD \to A\}$
  - Some members of F^+
    - $A \to BC$
    - $CD \to B$
    - $AD \to B$
    - $AD \to ABCD$
- If $\alpha \to R \in F^+$ then α is a superkey of R.
  - candidate key, primary key

Armstrong's Axioms

- Armstrong's Axioms help us compute closures.
  - *Reflexivity rule*: If $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ (for any two sets of attributes α and β).
  - *Augmentation rule*: If $\alpha \rightarrow \beta$ then $\gamma\alpha \rightarrow \gamma\beta$.
  - *Transitivity rule*: If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$.

- Armstrong's axioms are sound and complete.
  - They are *sound* in that they generate only correct functional dependencies.
  - They are *complete* in that they generate all possible FDs (F^+) from a given set F.

Armstrong's Axioms, cont.

- Additional rules:
  - *Union rule*: If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ then $\alpha \rightarrow \beta\gamma$.
  - *Decomposition rule*: If $\alpha \rightarrow \beta\gamma$ then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$.
  - *Pseudotransitivity rule*: If $\alpha \rightarrow \beta$ and $\gamma\beta \rightarrow \delta$ then $\alpha\gamma \rightarrow \delta$.

- These rules are sound.

Example

- Given the following functional dependencies F, compute F^+
  - $A \to BC$
  - $CD \to E$
  - $B \to D$
  - $E \to A$.

- Computing F^+
  - $E \to A$ and $A \to BC$, so $E \to BC$ (transitivity)
  - $B \to D$, so $CB \to CD$ (augmentation)
  - $CB \to CD$ and $CD \to E$, so $CB \to E$ (transitivity)
  - And many more…

Algorithm for Computing F^+

- $F^+ = F$

  repeat

  for each FD f in F^+

  apply *reflexivity* and *augmentation* rules on f

  add the resulting FDs to F^+

  for each pair of FDs f_1 and f_2 in F^+

  if f_1 and f_2 can be combined using *transitivity*

  then add the resulting FD to F^+

  until F^+ does not change any further

Closure of a Set of Attributes

- The *closure* of a set of attributes α *with respect to* a set of dependencies F is all attributes determined by α.

$$\alpha^+ = \{A \mid \alpha \rightarrow A \in F^+\}$$

- Observation:
  - If $\alpha \rightarrow \beta$ is in the closure of F, then β is in the closure of α.

- This yields the following algorithm for computing the closure of α.

Closure Algorithm

- Algorithm for computing the closure of α.

$result$ **=** α
while $result$ changes **do**
  for each $\gamma \rightarrow \delta$ in F **do**
    if $\gamma \subseteq result$ **then** $result := result \cup \delta$

- The algorithm computes the closure of α correctly.
  - All parts of *result* upon termination are in the closure of α.
  - All parts of the closure of α are in *result* upon termination.
  - The algorithm terminates.

Example of Attribute Closure

- $R = (A, B, C, D)$
- $F = \{ A \rightarrow B, A \rightarrow C, CD \rightarrow A\}$
- AD^+
  - $result = AD$
  - $result = ADB$ $(A \rightarrow B$ and $A \subseteq AD)$
  - $result = ADBC$ $(A \rightarrow C$ and $A \subseteq ABD)$

- Is AD a candidate key?
  - Does $AD \rightarrow R$?
  - Does $A \rightarrow R$?
  - Does $D \rightarrow R$?

Minimal Covers

- A *minimal cover* of a set F of functional dependencies is a canonical representation, itself a set of functional dependencies.
  - In each functional dependency, the right-hand side is a single attribute
  - No redundant attributes on the left-hand side. That is, there is no $X \rightarrow A \in F$ such that:
    - $Z \subseteq X$
    - $Z \rightarrow A$
  - There are no redundant functional dependencies. That is, there is no $X \rightarrow A$ in F such that $(F - (X \rightarrow A))^+ = F^+$

Minimal Covers, cont.

- Example: A minimal cover for

  $F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, A \rightarrow E\}$ is

  $F_c = \{A \rightarrow B,\ A \rightarrow C,\ CD \rightarrow E,\ B \rightarrow D\}$.

- Every set of dependencies *F* has a minimal cover.
- The minimal cover is not necessarily unique.

- How to compute minimal covers?
  - Three properties in the previous slide

Minimal Cover Algorithm

$G := F$

replace each functional dependency $X \to A_1 A_2 \ldots A_n \in G$ with functional dependencies $X \to A_1, X \to A_2, \ldots, X \to A_n$

for each FD $X \to A \in G$
  for each attribute $B \in X$
    compute $(X - B)^+$ with respect to the functional dependencies G
    if $(X - B)^+$ contains A
    then replace $X \to A$ with $(X - B) \to A$ in G

for each remaining FD $X \to A \in G$
    compute X^+ with respect to the set of dependencies $G - (\{X \to A\})$
  if X^+ contains A
  then remove $X \to A$ from G

Minimal Cover Example

- $F = \{S \rightarrow T, ST \rightarrow U, S \rightarrow TU\}$

Step 1: Break up right-hand sides.

- $G = \{S \rightarrow T, ST \rightarrow U, S \rightarrow U\}$

Step 2: Shrink left-hand sides.

- $S\,T \rightarrow U$
  - S^+ with respect to $G' = \{S \rightarrow T, S \rightarrow U\}$ is $\{S, T, U\}$: T is not needed.
  - T^+ with respect to $G' = \{S \rightarrow T, S \rightarrow U\}$ is $\{\, T \,\}$: S **is** needed.
- Result: $G = \{S \rightarrow T, S \rightarrow U, S \rightarrow U\} = \{S \rightarrow T, S \rightarrow U\}$

Minimal Cover Example, cont.

- $G = \{S \rightarrow T, S \rightarrow U\}$

- Step 3: eliminate superfluous functional dependencies.
  - $S \rightarrow T$: S^+ with respect to $G' = \{ S \rightarrow U \}$ is $\{ S, U \}$, so dependency $S \rightarrow T$ is *not* superfluous.
  - $S \rightarrow U$: S^+ with respect to $G' = \{ S \rightarrow T \}$ is $\{ S, T \}$, so dependency $S \rightarrow U$ is not superfluous.

- Result: $G = \{S \rightarrow T, S \rightarrow U\}$

Important Points So Far

- Properties of a good relational design
  - No redundancy
  - No update anomalies
  - Ability to represent all the information
- Converting a bad design to a good design
  - Decompose large relation schemas into smaller ones (Then we will need to do more joins to answer queries.)
- Ensuring lossless join decomposition
  - Specify semantic integrity constraints to be satisfied by all instances of the schemas.
  - Use constraints to argue that decompositions are lossless.
- Functional dependencies
- Armstrong's axioms to determine closure

Properties of a Good Decomposition

- A relational schema *R* with a set of functional dependencies *F* is decomposed into R_1 and R_2.

1. Lossless-join decomposition
   - Test to see if *at least one* of the following dependencies are in F^+
     - $R_1 \cap R_2 \rightarrow R_1$
     - $R_1 \cap R_2 \rightarrow R_2$
   - If not, decomposition may be lossy
2. Dependency preservation
   - Let F_i be the set of dependencies in F^+ that include only attributes in R_i (Notation: $F_i = \pi_{R_i}(F^+)$)
   - Test to see if $(F_1 \cup F_2)^+ = F^+$
   - When a relation is modified, no other relations need to be checked to preserve dependencies.
3. No redundancy

Example

- $R = (A, B, C)$
- $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B), R_2 = (B, C)$
  - Lossless-join decomposition?
    - $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC \in F^+$
  - Dependency preserving?
    - $F^+ = \{A \rightarrow B, A \rightarrow C, A \rightarrow AB, A \rightarrow BC, A \rightarrow AC, A \rightarrow ABC,$
      $AB \rightarrow C, AB \rightarrow AC, AB \rightarrow BC, AB \rightarrow ABC, AC \rightarrow BC,$
      $AC \rightarrow B, B \rightarrow C, B \rightarrow BC\} \cup \{many\ trivial\ dependencies\}$
    - $F_1 = \{A \rightarrow B, A \rightarrow AB\} \cup \{many\ trivial\ dependencies\}$
    - $F_2 = \{B \rightarrow C, B \rightarrow BC\} \cup \{many\ trivial\ dependencies\}$
    - $(F_1 \cup F_2)^+ = F^+$

Another Example

- $R = (A, B, C)$
- $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B), R_2 = (A, C)$
  - Lossless-join decomposition: yes
    - $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - Dependency preserving: no
    - $F^+ = \{A \rightarrow B, A \rightarrow C, A \rightarrow AB, A \rightarrow BC, A \rightarrow AC, A \rightarrow ABC,$
      $AB \rightarrow C, AB \rightarrow AC, AB \rightarrow BC, AB \rightarrow ABC, AC \rightarrow BC,$
      $AC \rightarrow B, B \rightarrow C, B \rightarrow BC\} \cup \{\textit{many trivial dependencies}\}$
    - $F_1 = \{A \rightarrow B, A \rightarrow AB\} \cup \{\textit{many trivial dependencies}\}$
    - $F_2 = \{A \rightarrow C, A \rightarrow AC\} \cup \{\textit{many trivial dependencies}\}$
    - $(F_1 \cup F_2)^+ \neq F^+$
    - Cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$

Impact of Non-Dependency Preservation

- r_1:

<u>*A*</u>	*B*
1	17
2	18

- r_2:

<u>*A*</u>	*C*
1	109
2	231

Solution:
using Norm Forms to guide decomposition!

- r_1 satisfies $\{A \rightarrow B\}$.
- r_2 satisfies $\{A \rightarrow C\}$.
- The programmer must maintain $\{B \rightarrow C\}$ manually, which is difficult.

Logical Database Design

- Motivation
- Logical Design
- Normal Forms
  - Boyce-Codd Normal Form (BCNF)
  - 3th Normal Form (3NF)

Normal Forms

- First Normal Form (1NF)
  - Domains of all attributes of relation schema are atomic.
- Second Normal Form (2NF)
  - Of historical interest only.
- Boyce-Codd Normal Form (BCNF)
- Third Normal Form (3NF)
- Fourth Normal Form (4NF)
  - Multivalued Dependencies (MVDs)
- Fifth Normal Form (5NF)
  - Join Dependencies (JDs)

Boyce-Codd Normal Form

- A relation schema R is in *Boyce-Codd Normal Form* (BCNF) if for all $X \rightarrow A \in F^+$, at least one of the following holds:
  - $X \rightarrow A$ is *trivial* (i.e., $A \subseteq X$), or
  - X is a superkey for R.

- Note that A is a single attribute.

Example

- $R = (A, B, C)$
  - $F = \{A \rightarrow B, B \rightarrow C\}$
  - Key = $\{A\}$

- Is R is in BCNF?
  - NO. Why?

- Decomposition: $R_1 = (\underline{A}, B)$, $R_2 = (\underline{B}, C)$
  - R_1 and R_2 are in BCNF.
  - Decomposition is a lossless-join decomposition.
  - Resulting schema is dependency preserving.

BCNF Decomposition Algorithm

- Algorithm to decompose a relation schema R into a set of relation schemas $\{R_1, R_2, \ldots, R_n\}$ such that:
  - each relation schema R_i is in BCNF, and
  - get a lossless-join decomposition.

result := $\{R\}$

done := false

while (**not** *done*) **do**

if (there is a schema R_i in result not in BCNF) **then**

let $X \rightarrow A$ be a BCNF violating FD, that is, a nontrivial functional dependency that holds on R_i such that $X \rightarrow R_i \notin F^+$ and $X \cap A = \emptyset$

$result := (result - R_i) \cup \{(R_i - A)\} \cup \{(X, A)\}$

else *done* := true

Combine R_i and R_j if:

R_i was obtained by using $X_i \rightarrow A_i$

R_j was obtained by using $X_i \rightarrow A_j$

Decompose Bad Schema *Rents(r)*

- R={*CustomerID*, *Title*, *Price*, *Kind*, *ResDate*}
  - Identify FDs.
  - Identify candidate key(s).
  - Which FD(s) violates the BCNF?

CustomerID	*Title*	*Price*	*Kind*	*ResDate*
0001	True Lies	3.25	D	2006-04-19
0002	True Lies	3.25	D	2006-04-21
0001	The Lion King	3.25	C	2006-04-19
0003	The Lion King	3.25	C	2006-04-19
0001	Henry V	1.75	D	2006-04-18

Dependency Preservation

- It is not always possible to get a BCNF decomposition that is dependency preserving.

- $R = (C, S, Z)$ (*City*, *State*, and *Zip code*)
- $F = \{CS \rightarrow Z, Z \rightarrow S\}$
- There are two candidate keys: CS and CZ.

- R is not in BCNF.
- Any decomposition of R will fail to preserve $CS \rightarrow Z$.

- It is not always possible to achieve both BCNF and dependency preservation.
  - Consider a weaker normal form – 3NF.

Third Normal Form

- A relation schema R is in *third normal form* (3NF) if for all $X \to A \in F^+$ at least one of the following holds:
  - $X \to A$ is trivial (i.e., $A \subseteq X$),
  - X is a superkey for R, or
  - A is contained in a candidate key.

- Note that A is a single attribute.
- If a relation is in BCNF it is in 3NF.

Example, Cont.

- $R = (C, S, Z)$ (*City*, *State*, and *Zip code*)
- $F = \{CS \rightarrow Z, Z \rightarrow S\}$
- There are two candidates keys: *CS* and *CZ*.
- *R* is in 3NF.
  - $CS \rightarrow Z$ (*CS* is a superkey.)
  - $Z \rightarrow S$ (*S* is contained in a key.)
- 3NF admits some redundancy

C	*S*	*Z*
Tucson	Arizona	85718
Marana	?	85718

- ? = Arizona by $Z \rightarrow S$.
- Thus, all redundancy is not eliminated.

3NF Decomposition Algorithm

- Algorithm to decompose a relation schema R into a set of relation schemas $\{R_1, R_2, \ldots, R_n\}$ such that:
  - each relation schema R_i is in 3NF,
  - lossless-join decomposition, and
  - decomposition is dependency preserving.

Let F_C be the minimal cover of F
$m := 0$
`for each` functional dependency $X \rightarrow A \in F_C$ **`do`**
    $m := m + 1$
    $R_m := XA$
`if` none of the schemas R_j $1 \leq j \leq m$ contains a candidate key for R **`then`**
    $m := m + 1$
    $R_m :=$ any candidate key for R
Combine R_i and R_j if:
    R_i was obtained by using $X_i \rightarrow A_i$
    R_j was obtained by using $X_i \rightarrow A_j$

An Exercise on 3NF

- R={A, B, C, D, E}
- F={A $\rightarrow$BC, CD $\rightarrow$E, B $\rightarrow$D, E $\rightarrow$A}

- Give a lossless, dependency preserving decomposition into 3NF of schema R.

What Can Be Achieved?

- It is always possible to decompose a relation into relations in 3NF such that
  - the decomposition is lossless, and
  - dependencies are preserved.
- It is always possible to decompose a relation into relations in BCNF such that
  - the decomposition is lossless.
- It may not be possible to preserve dependencies and BCNF.

Summary

- Properties of a “good” relational design
  - No redundancy
  - Ability to represent all the information
- Functional dependencies (FDs)
  - The single most important concept in relational database design.
  - Armstrong's axioms to determine closure
  - Minimal covers
  - FDs are a tool to ensure a “good” relational design.
- Normal Forms
  - BCNF: Lossless but not always dependency preserving
  - 3NF: Lossless and dependency preserving