

Database Constraints and Design

- We know that databases are often required to satisfy some *integrity constraints*.
- The most common ones are functional and inclusion dependencies.
- We'll study properties of integrity constraints.
- We show how they influence database design, in particular, how they tell us what type of information should be recorded in each particular relation

3/3

Introduction to Databases

1

Review: functional dependencies and keys

- A functional dependency is $X \rightarrow Y$ where X, Y are sequences of attributes. It holds in a relation R if for every two tuples t_1, t_2 in R :

$$\pi_X(t_1) = \pi_X(t_2) \text{ implies } \pi_Y(t_1) = \pi_Y(t_2)$$
- A very important special case: *keys*
- Let K be a set of attributes of R , and U the set of *all* attributes of R . Then K is a key if R satisfies functional dependency $K \rightarrow U$.
- In other words, a set of attributes K is a key in R if for any two tuples t_1, t_2 in R ,

$$\pi_K(t_1) = \pi_K(t_2) \text{ implies } t_1 = t_2$$
- That is, a key is a set of attributes that uniquely identify a tuple in a relation.

3/3

Introduction to Databases

2

Problems

- Good constraints vs Bad constraints: some constraints may be undesirable as they lead to problems.
- Implication problem: Suppose we are given some constraints. Do they imply others?

This is important, as we never get the list of *all* constraints that hold in a database (such a list could be very large, or just unknown to the designer).

It is possible that all constraints given to us look OK, but they imply some bad constraints.
- Axiomatization of constraints: a simple way to state when some constraints imply others.

3/3

Introduction to Databases

3

Bad constraints: example

DME	Dept	Manager	Employee	ES	Employee	Salary
D1	Smith	Jones			Jones	10
D1	Smith	Brown			Brown	20
D2	Turner	White			White	20
D3	Smith	Taylor		
D4	Smith	Clarke		
...

- Functional dependencies
- in DME: $\text{Dept} \rightarrow \text{Manager}$, but none of the following:
 - $\text{Dept} \rightarrow \text{Employee}$
 - $\text{Manager} \rightarrow \text{Dept}$
 - $\text{Manager} \rightarrow \text{Employee}$
- in ES: $\text{Employee} \rightarrow \text{Salary}$ (Employee is a key for ES) but not $\text{Salary} \rightarrow \text{Employee}$

3/3

Introduction to Databases

4

Update anomalies

- **Insertion anomaly:** A company hires a new employee, but doesn't immediately assign him/her to a department. We cannot record this fact in relation DME.
- **Deletion anomaly:** Employee White leaves the company. We have to delete a tuple from DME. But this results in deleting manager Turner as well, even though he hasn't left!
- Reason for anomalies: the association between managers and employees is represented in the same relation as the association between managers and departments. Furthermore, the same fact (a department D is managed by M) could be represented more than once.
- Putting this into the language of functional dependencies: we have a dependency $\text{Dept} \rightarrow \text{Manager}$, but Dept is not a key.
- In general, one tries to avoid situations like this.

3/3

Implication for functional dependencies

- Normally we try to find nontrivial dependencies
A trivial dependency is $X \rightarrow Y$ with $Y \subseteq X$
- Suppose we are given a set U of attributes of a relation, a set F of functional dependencies (FDs), and a FD f over U .
 F implies f , written $F \vdash f$
- if for every relation over attributes U , if R satisfies all FDs in F , then it is the case that it also satisfies f .
- Problem: Given U and F , find all nontrivial FDs f such that $F \vdash f$.

3/3

dependencies and implication

- Suppose we have relation R with 3 attributes A, B, C
- Someone tells us that R satisfies $A \rightarrow B$ and $B \rightarrow \{A, C\}$.
- This looks like the bad case before: we have $A \rightarrow B$ that does not mention C .
- But A is a key: if $\pi_A(t_1) = \pi_A(t_2)$, then $\pi_B(t_1) = \pi_B(t_2)$, and since B is a key, $t_1 = t_2$.
- Thus, even though we did not have information that A is a key, we were able to derive it, as it is implied by other dependencies.
- Using implication, we can find all dependencies and figure out if there is a problem with the design.

3/3

Implication for functional dependencies cont'd

- Closed set with respect to F : a subset V of U such that, for every $X \rightarrow Y$ in F ,
if $X \subseteq V$, then $Y \subseteq V$
- Property of FDs: For every set V , there exists a unique set $C_F(V)$, called the *closure* of V with respect to F , such that $C_F(V)$ is closed
 $V \subseteq C_F(V)$
For every closed set W , $V \subseteq W$ implies $C_F(V) \subseteq W$.
- Solution to the implication problem:
A FD $X \rightarrow Y$ is implied by F
if and only if
 $Y \subseteq C_F(X)$

3/3

Implication and closure

- To solve the implication problem, it suffices to find closure of each set of attributes.
- A naive approach to finding $C_F(X)$: check all subsets $V \subseteq U$, verify if they are closed, and select the smallest closed set containing X .
- Problem: this is too expensive.
- If U has n elements and X has $m < n$ elements, then there are 2^{n-m} subsets of U that contain V .
- But there is a very fast, $O(n)$, algorithm to compute $C_F(X)$.
- We will see a very simple $O(n^2)$ algorithm instead.

Implication and closure cont'd

ALGORITHM CLOSURE

Input: a set F of FDs, and a set X

Output: $C_F(X)$

1. $unused := F$
2. $closure := X$
3. repeat until no change:
 - if $Y \rightarrow Z \in unused$ and $Y \subseteq closure$ then
 - (i) $unused := unused - \{Y \rightarrow Z\}$
 - (ii) $closure := closure \cup Z$
4. Output $closure$.

Homework: Prove that CLOSURE returns $C_F(X)$ and that its running time is $O(n^2)$.

Properties of the closure

- $X \subseteq C_F(X)$
- $X \subseteq Y$ implies $C_F(X) \subseteq C_F(Y)$
- $C_F(C_F(X)) = C_F(X)$
- It turns out that for any operator C satisfying
 - $X \subseteq C(X)$
 - $X \subseteq Y$ implies $C(X) \subseteq C(Y)$
 - $C(C(X)) = C(X)$there exists a family of FDs F such that $C = C_F$.

Closure: Example

- Common practice: write sets as AB for $\{A, B\}$, BC for $\{B, C\}$ etc
- $U = \{A, B, C\}$, $F = \{A \rightarrow B, B \rightarrow AC\}$
- Closure:
 - $C_F(\emptyset) = \emptyset$
 - $C_F(C) = C$
 - $C_F(B) = ABC$
 - hence $C_F(X) = ABC$ for any X that contains B
 - $C_F(A) = ABC$
 - hence $C_F(X) = ABC$ for any X that contains A

Keys, candidate keys, and prime attributes

- A set X of attributes is a key with respect to F if $X \rightarrow U$ is implied by F
 - That is, X is a key if $C_F(X) = U$
 - In the previous example: any set containing A or B is a key.
 - *Candidate keys*: smallest keys. That is, keys X such that for each $Y \subset X$, Y is not a key.
 - In the previous example: A and B .
 - Suppose U has n attributes. What is the maximum number of candidate keys?
- answer: $\binom{n}{\lfloor n/2 \rfloor}$
and that's very large: 252 for $n = 10$, $> 180,000$ for $n = 20$
- *Prime attribute*: an attribute of a candidate key.

Implication of inclusion dependencies

- There are simple rules:
- $R[A] \subseteq R[A]$
- if $R[A_1, \dots, A_m] \subseteq S[B_1, \dots, B_m]$
then $R[A_{i_1}, \dots, A_{i_m}] \subseteq S[B_{i_1}, \dots, B_{i_m}]$
where $\{i_1, \dots, i_m\}$ is a permutation of $\{1, \dots, m\}$.
- if $R[A_1, \dots, A_m, A_{m+1}, \dots, A_n] \subseteq S[B_1, \dots, B_m, B_{m+1}, \dots, B_n]$
then $R[A_1, \dots, A_m] \subseteq S[B_1, \dots, B_m]$
- if $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$ then $R[X] \subseteq T[Z]$

Inclusion dependencies

- Functional and inclusion dependencies are the most common ones encountered in applications
- Reminder: $R[A_1, \dots, A_m] \subseteq S[B_1, \dots, B_m]$ says that for any tuple t in R , there is a tuple t' in S such that

$$\pi_{A_1, \dots, A_m}(t) = \pi_{B_1, \dots, B_m}(t')$$
- Suppose we have a set of relation names and inclusion dependencies (IDs) between them. Can we derive *all* IDs that are valid?
- Formally, let G be a set of IDs and g and ID. We say that G implies g , written

$$G \vdash g$$
 if any set of relations that satisfies all IDs in G , also satisfies g .
- The answer is positive, just as for FDs.

Implication of inclusion dependencies

- An ID g is implied by G iff it can be derived by repeated application of the four rules shown above
- This immediately gives an expensive algorithm: apply all the rules until none are applicable.
- It turns out that the problem is inherently hard; no reasonable algorithm for solving it will ever be found
- But an important special case admits efficient solution
- Unary IDs: those of the form $R[X] \subseteq S[Y]$
- Implication $G \vdash g$ can be tested in polynomial time for unary IDs

Functional and inclusion dependencies

- In majority of applications, one has to deal with only two kinds of dependencies: FDs and IDs
- Implication problem can be solved for FDs and IDs.
- Can it be solved for FDs and IDs together?
- That is, given a set of FDs F and a set of IDs G , a FD f and an ID g . Can we determine if

$$\begin{aligned} F \cup G \vdash f \\ F \cup G \vdash g \end{aligned}$$

- That is, any database that satisfies F and G , must satisfy f (or g).
- It turns out that no algorithm can possibly solve this problem.

When is implication solvable for FDs and IDs?

- Recall: an ID is unary if it is of the form $R[X] \subseteq S[Y]$.
- Implication can be tested for unary IDs and arbitrary FDs
- Moreover, it can be done in polynomial time.
- However, the algorithm for doing this is quite complex.

Even more restrictions

- Relations are typically declared with just primary keys
- Inclusion dependencies typically occur in foreign key declarations
- Can implication be solved algorithmically for just primary keys and foreign keys?
- The answer is still NO.
- Thus, it is hard to reason about some of the most common classes of constraints found in relational databases.

Dependencies: summary

- FDs: $X \rightarrow Y$
- Keys: $X \rightarrow U$, where U is the set of all attributes of a relation.
- Candidate key: a minimal key X ; no subset of X is a key.
- Inclusion dependency: $R[A_1, \dots, A_n] \subseteq S[B_1, \dots, B_n]$ (unary if $n = 1$)
- Foreign key: $R[A_1, \dots, A_n] \subseteq S[B_1, \dots, B_n]$ and $\{B_1, \dots, B_n\}$ is a key
- Implication problem:
 - easy for FDs alone
 - hard but solvable for IDs alone
 - solvable by a complex algorithm for FDs and unary IDs
 - unsolvable for FDs and IDs, and even
 - unsolvable for primary keys and foreign keys

Database Design

- Finding database schemas with good properties
- Good properties:
 - no update anomalies
 - no redundancies
 - no information loss
- Input: list of all attributes and constraints (usually functional dependencies)
- Output: list of relations and constraints they satisfy

3/3

21

Introduction to Databases

Why is BAD bad?

- **Redundancy:** many facts are repeated
- Director is determined by Title
 - For every showing, we list both director and title
- Address is determined by Theater
 - For every movie playing, we repeat the address
- **Update anomalies:**
 - If Address changes in one tuple, we have inconsistency, as it must be changed in all tuples corresponding to all movies and showtimes.
 - If a movie stops playing, we lose association between Title and Director
 - Cannot add a movie before it starts playing

3/3

23

Introduction to Databases

Example: bad design

- Attributes: Title, Director, Theater, Address, Phone, Time, Price
- Constraints:

FD1 Theater \rightarrow Address, Phone
FD2 Theater, Time, Title \rightarrow Price
FD3 Title \rightarrow Director

- Bad design: put everything in one relation

BAD[Title, Director, Theater, Address, Phone, Time, Price]

3/3

22

Introduction to Databases

Good design

- Split BAD into 3 relations:

Relational Schema GOOD:

Table	attributes	constraints
T1	Theater, Address, Phone	FD1: Theater \rightarrow Address, Phone
T2	Theater, Title, Time, Price	FD2: Theater, Time, Title \rightarrow Price
T3	Title, Director	FD3: Title \rightarrow Director

3/3

24

Introduction to Databases

Why is GOOD good?

- No update anomalies:
Every FD defines a key
- No information loss:

$$T1 = \pi_{\text{Theater,Address,Phone}}(\text{BAD})$$

$$T2 = \pi_{\text{Theater,Title,Time,Price}}(\text{BAD})$$

$$T3 = \pi_{\text{Title,Director}}(\text{BAD})$$

$$\text{BAD} = T1 \bowtie T2 \bowtie T3$$

- No constraints are lost

FD1, FD2, FD3 all appear as constraints for T1, T2, T3

Decompositions: Criteria for good design

Given a set of attributes U and a set F of functional dependencies, a *decomposition* of (U, F) is a set

$$(U_1, F_1), \dots, (U_n, F_n)$$

where $U_i \subseteq U$ and F_i is a set of FDs over attributes U_i .

A decomposition is called:

- **BCNF** decomposition if each (U_i, F_i) is in BCNF.

Boyce-Codd Normal Form (BCNF)

- What causes updates anomalies?
- Functional dependencies $X \rightarrow Y$ where X is not a key.
- A relation is in Boyce-Codd Normal Form (BCNF) if for every nontrivial FD $X \rightarrow Y$, X is a key.
- A database is in BCNF if every relation in it is in BCNF.

Decompositions: Criteria for good design cont d

- **lossless** if for every relation R over U that satisfies all FDs in F , each $\pi_{U_i}(R)$ satisfies F_i and

$$R = \pi_{U_1}(R) \bowtie \pi_{U_2}(R) \bowtie \dots \bowtie \pi_{U_n}(R)$$

- **Dependency preserving** if

$$F \text{ and } F^* = \bigcup_i F_i$$

are equivalent.

That is, for every FD f ,

$$F \vdash f \Leftrightarrow F^* \vdash f$$

or

$$\forall X \subseteq U \quad C_{F^*}(X) = C_F(X)$$

Projecting FDs

- Let U be a set of attributes and F a set of FDs
- Let $V \subseteq U$
- Then
 - $\pi_V(F) = \{X \rightarrow Y \mid X, Y \subseteq V, Y \subseteq C_F(X)\}$
 - In other words, these are all FDs on V that are implied by F :
 - $\pi_V(F) = \{X \rightarrow Y \mid X, Y \subseteq V, F \vdash X \rightarrow Y\}$
- Even though as defined $\pi_V(F)$ could be very large, it can often be compactly represented by a set F' of FDs on V such that:
 - $\forall X, Y \subseteq V: F' \vdash X \rightarrow Y \Leftrightarrow F \vdash X \rightarrow Y$

Decomposition algorithm: example

Consider BAD[Title, Director, Theater, Address, Phone, Time, Price] and constraints FD1, FD2, FD3.

Initialization: $S = (\text{BAD}, \text{FD1}, \text{FD2}, \text{FD3})$

While loop

- Step 1: Select BAD; it is not in BCNF because Theater \rightarrow Address, Phone but Theater $\not\rightarrow$ Title, Director, Time, Price
- Our sets are:
 - $X = \{\text{Theater}\}, Y = \{\text{Address, Phone}\},$
 - $Z = \{\text{Title, Director, Time, Price}\}$

Decomposition algorithm

Input: A set U of attributes and F of FDs
 Output: A database schema $S = \{(U_1, F_1), \dots, (U_n, F_n)\}$

1. Set $S := \{(U, F)\}$
2. While S is not in BCNF do:
 - (a) Choose $(V, F') \in S$ not in BCNF
 - (b) Choose nonempty disjoint X, Y, Z such that:
 - (i) $X \cup Y \cup Z = V$
 - (ii) $Y = C_{F'}(X) - X$
 (that is, $F' \vdash X \rightarrow Y$ and $F' \not\vdash X \rightarrow A$ for all $A \in Z$)
 - (c) Replace (V, F') by
 - $(X \cup Y, \pi_{X \cup Y}(F'))$ and $(X \cup Z, \pi_{X \cup Z}(F'))$
 - (d) If there are (V', F') and (V'', F'') in S with $V' \subseteq V''$ then remove (V', F')

Decomposition algorithm: example cont'd

- Step 1, cont'd
 - $\pi_{X \cup Y}(\{\text{FD1, FD2, FD3}\}) = \text{FD1}$
 - $\pi_{X \cup Z}(\{\text{FD1, FD2, FD3}\}) = \{\text{FD2, FD3}\}$
- Thus, after the first step we have two schemas:
 - $S_1 = (\{\text{Theater, Address, Phone}\}, \text{FD1})$
 - $S'_1 = (\{\text{Theater, Title, Director, Time, Price}\}, \text{FD2, FD3})$
- Step 2: S_1 is in BCNF, there is nothing to do.
 - S'_1 is not in BCNF: Title is not a key.
 - Let $X = \{\text{Title}\}, Y = \{\text{Director}\}, Z = \{\text{Theater, Time, Price}\}$
 - $\pi_{X \cup Y}(\{\text{FD1, FD2, FD3}\}) = \text{FD3}$
 - $\pi_{X \cup Z}(\{\text{FD1, FD2, FD3}\}) = \text{FD2}$

Decomposition algorithm: example cont'd

- After two steps, we have:
 $S_1 = (\{\text{Theater, Address, Phone}\}, \text{FD1})$
 $S_2 = (\{\text{Theater, Title, Time, Price}\}, \text{FD2})$
 $S_3 = (\{\text{Title, Director}\}, \text{FD3})$
- S_1, S_2, S_3 are all in BCNF, this completes the algorithm.

Surprises with BCNF

- Example: attributes Theater (th), Screen (sc), Title (tl), Snack (sn), Price (pr)
- Two FDs
Theater, Screen \rightarrow Title
Theater, Snack \rightarrow Price
- Decomposition into BCNF: after one step
(th, sc, tl; th, sc \rightarrow tl), (th, sc, sn, pr; th, sn \rightarrow pr)
- After two steps:
(th, sc, tl; th, sc \rightarrow tl), (th, sn, pr; th, sn \rightarrow pr), (th, sc, sn; \emptyset)
- The last relation looks very unnatural!

Properties of the decomposition algorithm

- For any relational schema, the decomposition algorithm yields a new relational schema that is:
in BCNF, and
lossless
- However, the output is not guaranteed to be dependency preserving.

Surprises with BCNF cont'd

- The extra relation (th, sc, sn; \emptyset) is not needed
- Why? Because it does not add any information
- All FDs are accounted for, so are all attributes, as well as all tuples
- BCNF tells us that for any R satisfying the FDs:

$$R = \pi_{\text{th,sc,tl}}(R) \bowtie \pi_{\text{th,sn,pr}}(R) \bowtie \pi_{\text{th,sc,sn}}(R)$$

- However, in our case we also have
 $R = \pi_{\text{th,sc,tl}}(R) \bowtie \pi_{\text{th,sn,pr}}(R)$
- This follows from the intuitive semantics of the data.
- But what is there in the schema to tell us that such an equation holds?

Multivalued dependencies

- Tell us when a relation is a join of two projections
- A multivalued dependency (MVD) is an expression of the form

$$X \twoheadrightarrow Y$$

where X and Y are sets of attributes

- Given a relation R on the set of attributes U , MVD $X \twoheadrightarrow Y$ holds in it if

$$R = \pi_{XY}(R) \bowtie \pi_{X(U-Y)}(R)$$

- Simple property: $X \rightarrow Y$ implies $X \twoheadrightarrow Y$

- Another definition: R satisfies $X \twoheadrightarrow Y$ if for every two tuples t_1, t_2 in R with $\pi_X(t_1) = \pi_X(t_2)$, there exists a tuple t with

$$\begin{aligned} \pi_{XY}(t) &= \pi_{XY}(t_1) \\ \pi_{X(U-Y)}(t) &= \pi_{X(U-Y)}(t_2) \end{aligned}$$

MVDs and decomposition

- If a relation satisfies a nontrivial MVD $X \twoheadrightarrow Y$ and X is not a key, it should be decomposed into relations with attribute sets XY and $X(U - Y)$.
- Returning to the example, assume that we have a MVD
Theater \twoheadrightarrow Screen
- Apply this to the "bad" schema (th, sc, sn; \emptyset) and obtain
(th, sc; \emptyset) (th, sn; \emptyset)
- Both are subsets of already produced schemas and can be eliminated.
- Thus, the FDs plus Theater \twoheadrightarrow Screen give rise to decomposition
(th, sc, tl; th, sc \twoheadrightarrow tl) (th, sn, pr; th, sn \twoheadrightarrow pr)

4th Normal Form (4NF)

- Decompositions like the one just produced are called *4th Normal Form* decompositions
- They can only be produced in the presence of MVDs
- Formally, a relation is in 4NF if for any nontrivial MVD $X \twoheadrightarrow Y$, either $X \cup Y = U$, or X is a key.
- Since $X \rightarrow Y$ implies $X \twoheadrightarrow Y$, 4NF implies BCNF
- General rule: if BCNF decomposition has "unnatural relations", try to use MVDs to decompose further into 4NF
- A useful rule: any BCNF schema that has one key that consists of a single attribute, is in 4NF
- Warning: the outcome of a decomposition algorithm may depend on the order in which constraints are considered.

BCNF and dependency preservation

- Schema Lecture[C(lass), P(rofessor), T(ime)]
- Set of FDs $F = \{C \rightarrow P, PT \rightarrow C\}$
- (Lectures, F) not in BCNF: $C \rightarrow P \in F$, but C is not a key.
- Apply BCNF decomposition algorithm: $X = \{C\}, Y = \{P\}, Z = \{T\}$
- Output: $\{(C, P), C \rightarrow P\}, \{(C, T), \emptyset\}$
- We lose $PT \rightarrow C!$
- In fact, there is no relational schema in BCNF that is equivalent to Lectures and is lossless and dependency preserving.
- Proof: there are just a few schemas on 3 attributes ... check them all ... exercise!

Third Normal Form (3NF)

- If we want a decomposition that guarantees losslessness and dependency preservation, we cannot always have BCNF
- Thus we need a slightly weaker condition
- Recall: a candidate key is a key that is not a subset of another key
- An attribute is prime if it belongs to a candidate key
- Recall (U, F) is in BCNF if for any FD $X \rightarrow A$, where $A \notin X$ is an attribute, $F \vdash X \rightarrow A$ implies that X is a key
- (U, F) is in the third normal form (3NF) if for any FD $X \rightarrow A$, where $A \notin X$ is an attribute, $F \vdash X \rightarrow A$ implies that one of the following is true:
 - either X is a key, or
 - A is prime

Decomposition into third normal form: covers

- Decomposition algorithm needs a small set that represents all FDs in a set F
- Such a set is called *minimal cover*. Formally:
 - F' is a *cover* of F iff $C_F = C_{F'}$. That is, for every f ,
$$F \vdash f \Leftrightarrow F' \vdash f$$
 - F' is a *minimal cover* if F' is a cover,
no subset $F'' \subset F'$ is a cover of F ,
each FD in F' is of the form $X \rightarrow A$, where A is an attribute,
For $X \rightarrow A \in F'$ and $X' \subset X$, $A \notin C_{F'}(X')$.

Third Normal Form (3NF) cont'd

- Main difference between BCNF and 3NF: in 3NF non-key FDs are OK, as long as they imply only prime attributes
- Lectures[C, P, T], $F = \{C \rightarrow P, PT \rightarrow C\}$ is in 3NF:
- PT is a candidate key, so P is a prime attribute.
- More redundancy than in BCNF: each time a class appears in a tuple, professor's name is repeated.
- We tolerate this redundancy because there is no BCNF decomposition.

Decomposition into third normal form: covers cont'd

- A minimal cover is a small set of FDs that give us all the same information as F .
- Example: let $\{A \rightarrow AB, A \rightarrow AC, A \rightarrow B, A \rightarrow C, B \rightarrow BC\}$
- Closure:
 - $C_F(A) = ABC$,
 - $C_F(B) = BC$,
 - $C_F(C) = C$, and hence:
- Minimal cover: $A \rightarrow B, B \rightarrow C$

Decomposition into third normal form: algorithm

Input: A set U of attributes and F of FDs

Output: A database schema $S = \{(U_1, F_1), \dots, (U_n, F_n)\}$

Step 1. Find a minimal cover F' of F .

Step 2. If there is $X \rightarrow A$ in F' with $XA = U$, then output (U, F') .

Otherwise, select a key K , and output:

2a) $(XA, X \rightarrow A)$ for each $X \rightarrow A$ in F' , and

2b) (K, \emptyset)

3NF cont'd

- Properties of the decomposition algorithm: it produces a schema which is in 3NF
- lossless, and dependency preserving
- Complexity of the algorithm?
- Given the a minimal cover F' , it is linear time. (Why?)
- But how hard is it to find a minimal cover?
- Naive approach: try all sets F'' , check if they are minimal covers.
- This is too expensive (exponential); can one do better?
- There is a polynomial-time algorithm. How?

Decomposition into third normal form: example

• $F = \{A \rightarrow AB, A \rightarrow AC, A \rightarrow B, A \rightarrow C, B \rightarrow BC\}$

• $F' = \{A \rightarrow B, B \rightarrow C\}$

• A is a key, so output:

$(A, \emptyset), (AB, A \rightarrow B), (BC, B \rightarrow C)$

• Simplification: if one of attribute-sets produced in 2a) is a key, then step 2b) is not needed

• Hence the result is:

$(AB, A \rightarrow B), (BC, B \rightarrow C)$

• Other potential simplifications: if we have $(XA_1, X \rightarrow A_1), \dots, (XA_k, X \rightarrow A_k)$ in the output, they can be replaced by $(XA_1 \dots A_k, X \rightarrow A_1 \dots A_k)$

Overview of schema design

- Choose the set of attributes U
- Choose the set of FDs F
- Find a lossless dependency preserving decomposition into: BCNF, if it exists
3NF, if BCNF decomposition cannot be found

Other constraints

- SQL lets you specify a variety of other constraints:
 - Local (refer to a tuple)
 - global (refer to tables)
- These constraints are enforced by a DBMS, that is, they are checked when a database is modify.
- Local constraints occur in the CREATE TABLE statement and use the keyword CHECK after attribute declaration:

```
CREATE TABLE T ( ...
    ....
    A <type> CHECK <condition>,
    B <type> CHECK <condition>,
    ....
)
```

343

Assertions

- These assert some conditions that must be satisfied by the whole database.
- Typically assertions say that all elements in a database satisfy certain condition.
- All salaries are at least 10K:
CREATE ASSERTION A1 CHECK
 ((SELECT MIN (Emp1.Salary) FROM Emp1) >= 10000))
- Most assertions use NOT EXISTS in them:
SQL way of saying

“every x satisfies F ”

is to say

“does not exist x that satisfies $\neg F$ ”

343

Local constraints

- Example: the value of attribute Rank must be between 1 and 5:
Rank INT CHECK (1 <= Rank AND Rank <= 5)
- Example: the value of attribute A must be less than 10, and occur as a value of attribute C of relation R:
A INT CHECK (A IN
 SELECT R.C FROM R WHERE R.C < 10)
- Example: each value of attribute Name occurs precisely once as a value of attribute LastName in relation S:

```
Name VARCHAR(20) CHECK (1 = SELECT COUNT(*)
FROM S
WHERE Name=S.LastName)
```

343

Assertions cont'd

- Example: all employees of department 'sales' have salary at least 20K:
CREATE ASSERTION A2 CHECK
 (NOT EXISTS (SELECT * FROM Emp1
 WHERE Dept='sales' AND Salary < 20000))
- All theaters play movies that are at most 3hrs long:
CREATE ASSERTION A2 CHECK
 (NOT EXISTS (SELECT * FROM Schedule S, Movies M
 WHERE S.Title=M.Title AND M.Length > 180))

343

Assertions cont'd

- Some assertions use counting. For example, to ensure that table T is never empty, use:

```
CREATE ASSERTION T_notempty CHECK
  ( 0 <> (SELECT COUNT(*) FROM T) )
```
- Assertions are not forever: they can be created, and dropped later:

```
DROP ASSERTION T_notempty.
```

Analysis of the trigger

- AFTER UPDATE OF Length ON Movies specifies the **event**: Relation Movies was modified, and attribute Length changed its value.
- REFERENCING

```
  OLD ROW AS OldTuple
  NEW ROW AS NewTuple
```

says how we refer to tuples before and after the update.
- FOR EACH ROW – the trigger is executed once for each update row.
- WHEN (OldTuple.Length <> NewTuple.Length) – **condition** for the trigger to be executed (Length changed its value).
- SET Length = OldTuple.Length
WHERE title = NewTuple.title
is the **action**: Length must be restored to its previous value.

Triggers

- They specify a set of actions to be taken if certain event(s) took place.
- Follow the *event-condition-action* scheme.
- Less declarative and more procedural than assertions.
- Example: If an attempt is made to change the length of a movie, it should not go through.

```
CREATE TRIGGER NoLengthUpdate
AFTER UPDATE OF Length ON Movies
REFERENCING
  OLD ROW AS OldTuple
  NEW ROW AS NewTuple
FOR EACH ROW
WHEN (OldTuple.Length <> NewTuple.Length)
SET Length = OldTuple.Length
WHERE title = NewTuple.title
```

Another trigger example

- Table Emp(emp_id,rank,salary)
- Requirement: the average salary of managers should never go below \$100,000.
- Problem: suppose there is a complicated update operation that affects many tuples. It may initially decrease the average, and then increase it again. Thus, we want the trigger to run after *all* the statements of the update operation have been executed.
- Hence, FOR EACH ROW trigger cannot work here.

Another trigger example cont'd

Trigger for the previous example:

```
CREATE TRIGGER MaintainAvgSal
AFTER UPDATE OF Salary ON Emp1
REFERENCING
    OLD TABLE AS OldTuples
    NEW TABLE AS NewTuples
FOR EACH STATEMENT
WHEN ( 100000 > (SELECT AVG(Salary)
                FROM Emp1 WHERE Rank='manager') )
BEGIN
    DELETE FROM Emp1
    WHERE (emp_id, rank, salary) in NewTuples;
    INSERT INTO Emp1
    (SELECT * FROM OldTuples)
END;
```

343

Introduction to Databases

57

Analysis of the trigger

- AFTER UPDATE OF Salary ON Emp1 specifies the **event**: Relation Emp1 was modified, and attribute Salary changed its value.

- REFERENCING

```
    OLD TABLE AS OldTuples
    NEW TABLE AS NewTuples
```

says that we refer to the set of tuples that were inserted into Emp1 as NewTuples and to the set of tuples that were deleted from Emp1 as OldTuples.

- FOR EACH STATEMENT – the trigger is executed once for the entire update operation, not once for each updated row.

- WHEN (100000 > (SELECT AVG(Salary)
 FROM Emp1 WHERE Rank='manager'))

is the **condition** for the trigger to be executed: after the entire update, the average Salary of managers is less than \$100K.

343

Introduction to Databases

58

Analysis of the trigger cont'd

- BEGIN
 DELETE FROM Emp1
 WHERE (emp_id, rank, salary) in NewTuples;
 INSERT INTO Emp1
 (SELECT * FROM OldTuples)
END;
is the **action**, that consists of two updates between BEGIN and END.
- DELETE FROM Emp1
 WHERE (emp_id, rank, salary) in NewTuples;
deletes all the tuples that were inserted in the illegal update, and
- INSERT INTO Emp1
 (SELECT * FROM OldTuples)
re-inserts all the tuples that were deleted in that update.

343

Introduction to Databases

59

Conceptual modeling

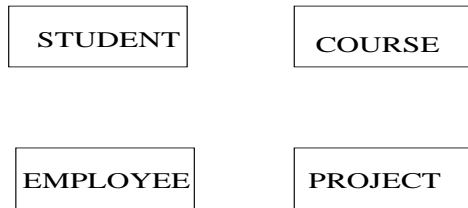
- How does one come up with a set of attributes and FDs?
- Often by using conceptual design techniques, typically represented by entity-relationship models
- Database describes objects (entities), relationships between them, and their attributes.
- Goal: describe entities, relationships, and attributes based on a real-world situation that needs to be modeled by a database, find relational database design, and then apply normalization techniques.
- Steps:
 - describe entities
 - describe relationships between them
 - describe attributes
 - translate into relational model

343

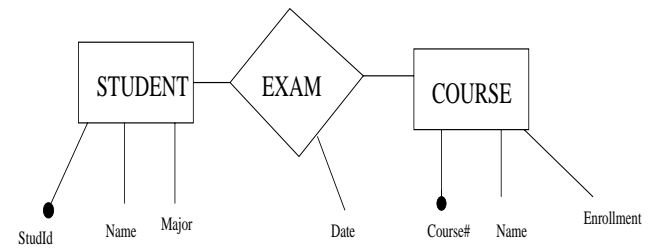
Introduction to Databases

60

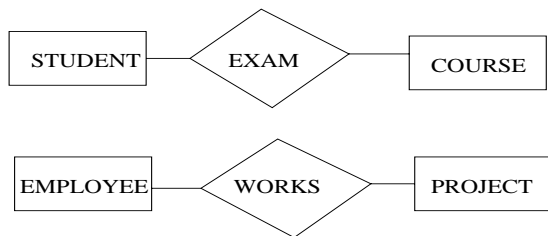
ENTITIES



ATTRIBUTES



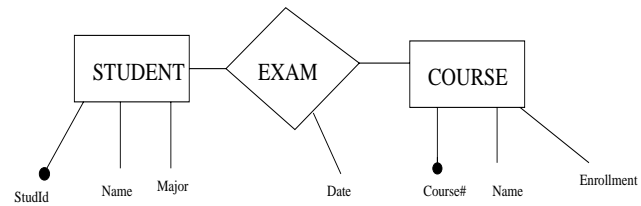
RELATIONSHIPS



Translating into relational model: rules

- Each entity E_i is described by a relation T_i
- Key attributes of E_i form a key of T_i
- Each relationship R_j is described by a relation S_j
- For a relationship R_j between E_1, \dots, E_k , the key of S_j is the set of all attributes which are keys for T_1, \dots, T_k
- Sometimes attributes have to be renamed

TRANSLATION INTO RELATIONS



Student(StudId, Name, Major)

Course(Course#, Name, Enrollment)

Exam(StudId, Course#, Date)

Database design in SQL

```
CREATE TABLE Student
  (StudId char(10) not null primary key,
   Name char(20), Major char(5))
```

```
CREATE TABLE Course
  (Course# int not null primary key,
   Name char(30), Enrollment int)
```

```
CREATE TABLE Exam
  (Student char(10) not null,
   Course int not null,
   Date int,
   unique (Student, Course))
```