Database Systems
236363

Functional Dependencies

Database Design Process

- How do we design the relations schemas?
  - Option 1:
    - Directly from the ER diagram
  - Option 2:
    - Identify all attributes coming out of the system's requirement analysis. That is, all minimal units of data items, such as student name, course number, course name, etc.
    - We create a single "super schema" including all attributes, or a few large schemas
    - We then iteratively combine and split the schemas to obtain a "good representation"

Database Design Process - continued

- During the design process, it is important to verify that the following holds:
  - The entire data is stored and can be represented
  - All constraints on the data are preserved, e.g., keys
  - No unnecessary redundancy
  - Queries over the database can be implemented efficiently

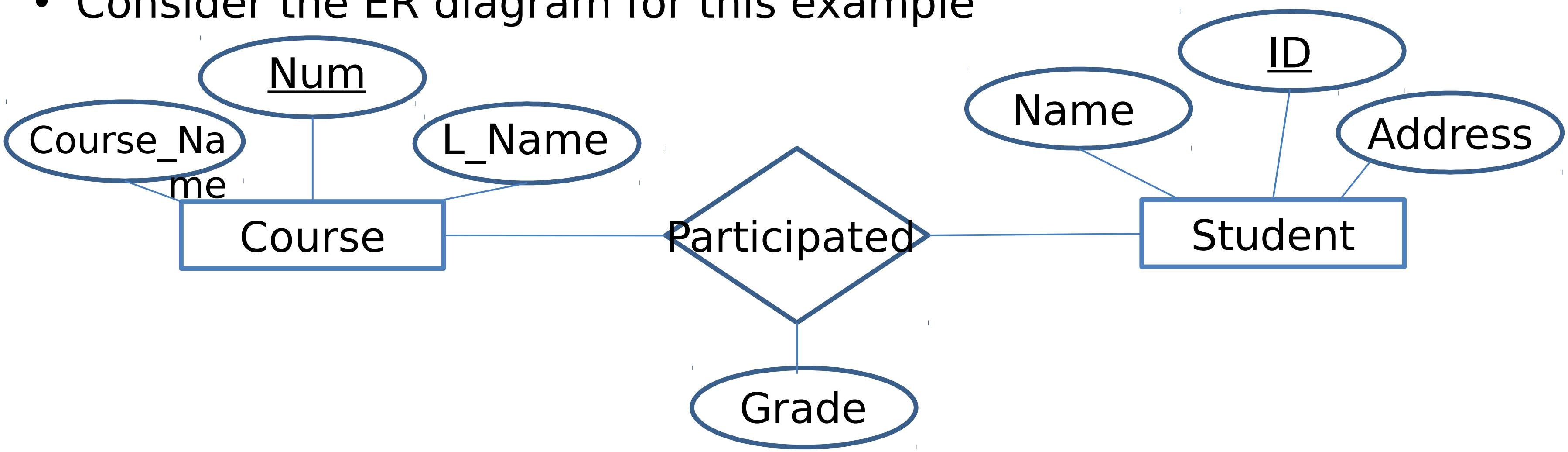
Schema Decomposition-reminder

- Recall the courses and students example from the introduction lecture (first week)
  - For each student we wish to save the student name, student id, and address
  - For each course we wish to store the course name, course number, and lecturer
  - For each occurrence of a student taking a course, we would like to record the identifying information about this event plus the final grade

  - As mention in the first lecture, saving a single student name, student id, address, course number, course name, lecturer, final grade schema is inefficient and leads to consistency problems
  - Hence, we prefer breaking this into several smaller schemas

Continued

- Consider the ER diagram for this example



- This diagram implies 3 schemas:
  - Course(Course_Name,Num,L_Name) Student(Name,ID,Address) Participated(Num,ID,Grade)

- As discussed in the first lecture, this decomposition is based on some implicit simplifying assumptions, but for the purpose of this discussion it will do

Functional Dependencies - Informally

- The ER diagram for the students and courses implies some conditions that must hold in the database
  - The ID is a primary key for the student entity
    - Hence, the name and address attributes are uniquely defined for each value of ID
  - The course name and lecturer are uniquely defined by the course number
  - For each combination of course number and student ID there is a single value of final grade (if exists)

- In general, a functional dependency exists whenever a subset of the attributes uniquely define the values of another subset of attributes

ERD Represents Functional Dependencies

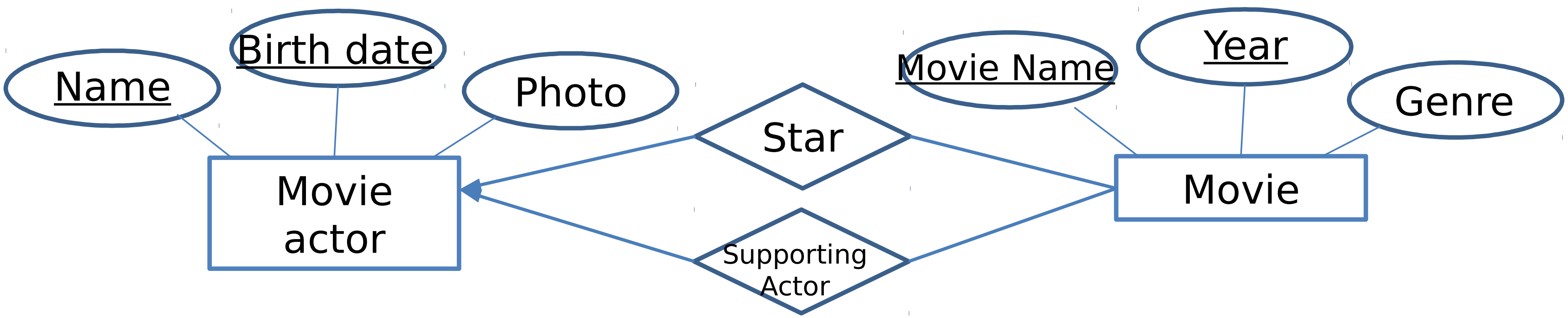
- The ER diagram defines functional dependencies
- When deciding on the entity sets and relationship sets, we also define the database attributes
- The primary keys and the relations between entity sets and relationship sets define functional dependencies that the database fulfills
- Yet, there may be additional dependencies that the ER diagram cannot express, e.g., that there are additional (non-primary) keys for a given entity set

Are the Schema Obtained by ERD Always the Best?

- In the next slides, we will learn formal measures that evaluate how good a given decomposition into schemas is
- We will see that sometimes ERD based schemas are good and other times they are not
- We will also learn how to design schemas in order to optimize these measures

Another Comment

- Treating all attributes as one large schema that is being decomposed during the design phase is sometimes also problematic
  - For example, consider the following diagram



  - If we derive the schema only by focusing on attributes, we will not be able to distinguish between the relationship sets “Star” and “Supporting Actor”
  - Further, the analysis of the functional dependency between them will also be incorrect

Anomalies in Inadequate Schema Design

- Recall the problems arising from treating the entire database as a single large schema
  - Redundancy
    - Useless repetitions, e.g., no point in writing the student’s address for each course s/he is registered to
  - Complicates updates
    - Each change, e.g., in a student’s address, would require updating multiple rows in the database
  - Representation difficulties
    - There are certain situations that are hard to represent this way, e.g., a student that did not take any course
    - Sometimes null values can help with this, but they add their own complexities and so better be avoided

To Decompose or not to Decompose

- Schemas decomposition
  - Enables solving the anomalies mentioned before
  - Reduces the chances of inconsistencies, by eliminating redundancies in the database

- However…
  - By decomposing the schemas, many queries would require performing joins
    - Why is this considered harmful?
  - Having many schemas may complicate the writing of queries and applications

Dependencies and Decomposition

- First, we will study how functional dependencies are identified and derived

- Next, we will learn how to perform decompositions based on functional dependencies
  - The basic idea is that certain dependencies in a given schema may indicate inefficiencies in it and therefore that it is better to further decompose it into smaller schemas

- After that, we will explore normal forms – what they are and how to obtain them
  - For a given decomposition into schemas, the higher the order of the normal form is, the better the design is

Functional Dependencies

The theory behind designing schemas for databases

Functional Dependencies

- Functional dependencies for a given relation are discovered from analyzing the system's requirements

- For a relation R with attributes A and B, we say that B is functionally dependent on A, denoted $A \rightarrow B$ if for every two records in R in which the value of A is the same, the value of B is also the same
  - $\forall x_1, x_2 \in R$, if $x_1[A]=x_2[A]$ then $x_1[B]=x_2[B]$

- Note that we are interested in functional dependencies that arise from the system's analysis rather than from the specific content of R

Functional Dependencies - continued

- For given properties A_1,A_2,B, denote $A_1A_2 \rightarrow B$ the following dependency
  - In every two records of R such that the values of A_1 and A_2 are the same, the values of B are the also same
    - Notice that $A_1A_2 \rightarrow B$ does not impose any restriction on records in which only the values of A_1 or only the values of A_2 are the same

- We denote $A_1A_2 \rightarrow B_1B_2$ the fact that both $A_1A_2 \rightarrow B_1$ and $A_1A_2 \rightarrow B_2$ hold for a given set of properties A_1A_2 and B_1B_2

- Further, we can generalize $A_1 \ldots A_k \rightarrow B_1 \ldots B_l$ to denote that

, $\forall x_1,x_2 \in R$

if for all $1 \leq i \leq k$ $x_1[A_i]=x_2[A_i]$ then for all $1 \leq j \leq l$ $x_1[B_j]=x_2[B_j]$

Example

S	C	T
Bart	Math	Mrs. Krabappel
Lisa	Math	Mrs. Krabappel
Lisa	Logic	Ms. Hoover

- This relation satisfies C→T but does not satisfy S→T

- In general, a schema in which for a given relation R there are two properties A,B such that A→B but A is not a key is problematic
  - Is the schema in the example problematic?

A Dependency Following From a Set of Dependencies

- Given a set of functional dependencies $F=\{X_1 \rightarrow Y_1, \ldots, X_n \rightarrow Y_n\}$ where each X_i and Y_i is a set of properties of the relation R, a dependency $X \rightarrow Y$ follows from F (denoted $F \vDash X \rightarrow Y$) if every relation with the corresponding properties that satisfies all dependencies in F also satisfies $X \rightarrow Y$

- Example
  - Given a schema $R=(A,B,C)$ and a set of dependencies $F=\{A \rightarrow B, B \rightarrow C\}$, the dependency $A \rightarrow C$ follows from F
    - This is because for any two records in which the properties in A are the same, the properties of B are the same (due to $A \rightarrow B$) and since the properties in B are the same, so are the properties in C (due to $B \rightarrow C$)

Keys

- Given a schema R and a set of functional dependencies F:
  - A superkey of R is a set of attributes $X \subseteq R$ such that $F \models X \rightarrow R$
  - A key of R is a set of attributes $X \subseteq R$ such:
    1. X is a superkey of R
    2. No proper subset of X is a superkey of R

- A key is also called a minimal key or an admissible key
  - As hinted before, keys are useful in identifying problematic schemas, e.g., the existence of a dependency $X \rightarrow Y$ in which X is not a key

- Are there relations that have no key?

Inferring Functional Dependencies

- When designing a database, it is important to know all functional dependencies in the system, both the ones that can be discovered directly from the system analysis F and the ones that follow from F

- In order to know whether $F \vDash X \rightarrow Y$ holds, we could look for a counter example of a relation that satisfies F but not $X \rightarrow Y$
  - If after scanning the entire database we are unable to find such an example, we conclude that $F \vDash X \rightarrow Y$ holds

- However, scanning the infinite set of all possible values of the relations is not feasible
  - Hence, we will focus on learning how to infer all functional dependencies that follow from F without scanning infinite sets

Trivial Functional Dependencies

- If A is an attribute of a relation R, then regardless of the current values in the database, $A \rightarrow A$ always hold
  - This is because there can be no two records $r, s \in R$ for which the values of A are both identical and different
- Generally, if $X = \{A_1, \ldots, A_k\}$ is a set of attributes of R, then $X \rightarrow A_i$ will always hold for every $1 \leq i \leq k$
  - Hence, for every subset $Y \subseteq X$ the functional dependency $X \rightarrow Y$ also holds
- This is called ***reflexivity***

An Example – the Insertion Rule

- **<u>Claim:</u>** If R is a relation whose content satisfies $X \rightarrow Y$ and let Z be any set of attributes, then R also satisfies the functional dependency $XZ \rightarrow YZ$
  - In other words, the dependency $XZ \rightarrow YZ$ follows from the dependency $X \rightarrow Y$ ($X \rightarrow Y \models XZ \rightarrow YZ$)

- Notice that we use the notation XZ to denote the unification of the sets of attributes in X and Z

Proof of the Insertion Rule

- Assume b.w.o.c. that there exist a relation R and attribute sets X,Y,Z such that R satisfies $X \rightarrow Y$ but does not satisfy $XZ \rightarrow YZ$
  - Hence, there are records $r,s \in R$ whose values are the same in all attributes of XZ but differ in at least one attribute A of YZ
  - If $A \in Y$ then r and s agree on all attributes of X but not on all attributes in Y, meaning that $X \rightarrow Y$ does not hold – a contradiction
  - If $A \in Z$ then by assumption they cannot obtain a different value on A since they have the same values on all attributes in XZ – a contradiction
  - This covers all options for A. Hence, there can be no relation that satisfies the assumption and the claim holds

Another Example - Transitivity

- If X,Y,Z are sets of attributes of a relation R, then $X \rightarrow Z$ follows from the combination of $X \rightarrow Y$ and $Y \rightarrow Z$
  - Proof concept:
    - As mentioned before, we show that if the contents of R satisfies both $X \rightarrow Y$ and $Y \rightarrow Z$, then for every two records $r,s \in R$, if the values of their X attributes are the same then the values of their Z attributes are also the same

Armstrong's Axioms

- The three inference rules we have seen are called Armstrong's axioms:
  - **Reflexivity:** if X is a set of attributes of R and $Y \subseteq X$, then $X \rightarrow Y$
  - **Insertion:** if R satisfies $X \rightarrow Y$ for two sets of attributes X,Y of R, then for every set of attributes Z in R it holds that $XZ \rightarrow YZ$
  - **Transitivity:** if $X \rightarrow Y$ and $Y \rightarrow Z$ both hold for a relation R, then $X \rightarrow Z$ also holds
- For a set of dependencies F and another dependency $X \rightarrow Y$, we say that $X \rightarrow Y$ can be deduced from F, denoted $F \vdash X \rightarrow Y$, if $X \rightarrow Y$ can be inferred from F using only Armstrong's axioms

Soundness and Completeness

- **Soundness**
  - Every functional dependency $X \rightarrow Y$ that can be inferred from a set of functional dependencies F using a finite number of usages of Armstrong's axioms indeed follows from F
    - The soundness proof follows from our proof that when using each of the axioms one can infer only a dependency that follows from F
    - We then apply induction on the number of axiom usages

- **Completeness**
  - Every functional dependency $X \rightarrow Y$ that follows from F can be inferred from F using a finite number of usages of Armstrong's axioms
    - The proof is given later

$$F \vdash X \rightarrow Y \;\Leftrightarrow\; F \models X \rightarrow Y$$

Additional Inference Rules

- The following inference rules follow from Armstrong's axioms (and can be inferred from them):
  - **Unification:**
    - If $X \rightarrow Y$ and $X \rightarrow Z$ both hold then $X \rightarrow YZ$ holds
  - **Split:**
    - If $X \rightarrow YZ$ holds then $X \rightarrow Y$ and $X \rightarrow Z$ both hold
  - **Pseudo-transitivity:**
    - If $X \rightarrow Y$ and $YW \rightarrow Z$ both hold then $XW \rightarrow Z$ holds

Proof of the Unification Rule

- We will prove using Armstrong's axioms the following claim:
  - If $X \rightarrow Y$ and $X \rightarrow Z$ both hold then $X \rightarrow YZ$ holds

- Proof
  - From $X \rightarrow Y$ and the insertion rule we have that $XZ \rightarrow YZ$ holds
  - From $X \rightarrow Z$ and the insertion rule we have that $X \rightarrow XZ$ holds (since $XX=X$)
  - From $X \rightarrow XZ$ and $XZ \rightarrow YZ$ and the transitivity rule we have $X \rightarrow YZ$

.Q.E.D

Closure of an Attributes Set

- For a relation R, a functional dependency set F, and a set of attributes X, denote by $X_F{}^+$ the set of attributes A of R for which $F\vdash(X\rightarrow A)$ holds
  - When F is obvious from the context, we simply write X^+

- Claim
  - For each set Y of attributes of R, $F\vdash(X\rightarrow Y)$ holds iff $Y\subseteq X_F{}^+$ holds

- Proof
  - If $Y\subseteq X_F{}^+$, then for each $A\in Y$, $F\vdash(X\rightarrow A)$ holds and by unification we have that $F\vdash(X\rightarrow Y)$
  - If $F\vdash(X\rightarrow Y)$, then by the split rule for each $A\in Y$, $F\vdash(X\rightarrow A)$ holds, meaning that $Y\subseteq X_F{}^+$

- Corollary
  - For each set of attributes X, $(X_F{}^+)_F{}^+ = X_F{}^+$

Completeness of Armstrong's Axioms

- For the completeness proof, assume that for a set of dependencies F, $F \nvdash (X \rightarrow Y)$ holds
- We will show that $F \not\models (X \rightarrow Y)$ holds by constructing possible content for R that satisfies all the dependencies in F but does not satisfy $X \rightarrow Y$
  - From the previous claim, if $F \nvdash (X \rightarrow Y)$ then $Y \subseteq X_F^+$ does not hold, meaning that there is an attribute $A \in Y$ such that does not belong to X_F^+
  - We will now create two records: the first will have "0" in all the attributes of R while the other will have "0" in all attributes of X_F^+ and "1" on all other attributes of R
  - Both records agree on all attributes of X_F^+ and therefore also on all attributes of X
  - Yet, they do not agree on A and thus R does not satisfy $X \rightarrow Y$
  - On the other hand, R satisfies F; otherwise, it would contradict the claim that $(X_F^+)_F^+ = X_F^+$

Q.E.D.

Completeness Proof – Example

$R \setminus X^+$	X^+
0…0	0…0
1…1	0…0

In this relation, if there is a dependency that is violated, then there is one of the form $W \rightarrow A$ where $W \subseteq X^+$ and $A \in R \setminus X^+$

If such a dependency $W \rightarrow A$ exists, then $A \in (X^+)^+$

$A \in R \setminus X^+$ is a contradiction to $X^+ = (X^+)^+$

A Simple Algorithm for Computing the Closure

- The closure of an attributes set has an important role in the design of relational schemas
  - For example, X is a superkey of R iff $X^+=R$
  - Thus, we need an algorithm to compute it
- The following simple algorithm computes the closure X_F^+ for a given dependency set F and attributes set X
- Claim:
  - The algorithm always terminates and returns the correct answer

```
A_List := X
Repeat
   For every Y→Z∈F do
      If Y⊆A_List then
         A_List := A_List∪Z
Until no change to A_List
Return A_List
```

Execution Example

- Compute the closure of $X=\{A,B\}$ for the dependency set $F=\{A \rightarrow C,\ BC \rightarrow A,\ AC \rightarrow D,\ CE \rightarrow F\ \}$
  - Initialization: A_List={A,B}
  - From $A \rightarrow C$, we get A_List={A,B,C}
  - From $AC \rightarrow D$, we get A_List={A,B,C,D}
  - The other dependencies do not add anything
  - The final result is $X_F^+ = \{\ A,B,C,D\ \}$

An Improved Closure Algorithm

- The following algorithm ensures that the running time will be linear with the length of the input (according to Beeri-Bernstein theorem)

```
A_List := X
F_List := F
Repeat
   For every Y→Z∈F_List do
      Y := Y \ A_List
      If Y=∅ then
         A_List := A_List∪Z
Until no change to A_List
Return A_List
```

Closure of Dependency Sets

- For a dependency set F over a set of attributes U or a relation R[U], denote F^+ the set of all dependencies implied by F
  - This set is called the closure of F

- The closures of dependency set and attributes sets are used to define criteria for the goodness of a relation split

- Yet, the size of F^+ could be exponential in the size of F
  - We need to worry about this when we design protocols for finding splits

Closure Comparisons

- Given a dependency set F and a new dependency $X \rightarrow Y$, we can compute whether $X \rightarrow Y \in F^+$ even without explicitly computing F^+
  - For this, we can compute X_F^+, which runs in linear time, and then check if $Y \subseteq X_F^+$

- Consequently, given two sets of dependencies F and G, we can verify in polynomial time whether $F^+ = G^+$
  - First, we check for each dependency in G whether it follows from F
    - If so, we deduce that $G^+ \subseteq (F^+)^+ = F^+$
  - Similarly, we can verify if $F^+ \subseteq G^+$

Covers and Minimal Covers

- A dependency set G is called a ***cover*** of F if $G^+ = F^+$
  - As we saw, this can be done in polynomial time

- A cover G of F is said to be ***minimal*** if the following additional requirements are met
  - All dependencies in G are of the form $X \rightarrow A$ (where A is a single attribute)
  - No dependency in G can be inferred from another dependency in G
  - There does not exists in G a dependency $X \rightarrow A$ such that there exists a proper subset Y of X for which $Y \rightarrow A \in G^+ = F^+$

- For each dependency set F there is at least one minimal cover whose size is polynomial in F
- It is possible that there will be multiple minimal covers for the same dependency set F
  - Note that each minimal cover can potentially be of different size, i.e., “minimal” does not imply anything about the size of the cover

The Basic Idea for Finding Minimal Covers

- Given F:
  1. Split all dependencies such that the right side of each dependency will include a single attribute
  2. Eliminate from all added dependencies all attributes that are redundant (from the left side)
  3. Eliminate all dependencies that can be deduced from others

- Notice:
  - Eliminating an attribute from the right side of a functional dependency may reduce F
    - We should verify that after the change, the dependency remains
  - Eliminating an attribute from the left side of a functional dependency may increase F
    - We should verify that the resulting dependency already existed before the change
- An exact algorithm will be given in the recitation

Additional Dependency Types

- Relational schemas may include additional dependency types
  - *Multivalued dependencies*
    - Here, there may be multiple different values of Y for the same values of X, yet the values of X fix the set of values for Y
  - *Inclusion dependencies*
    - This types of dependency relates between the values of attributes in two relations in the schema
    - E.g., in the train operation π_{S_Name} (Serves) $\subseteq$ π_{S_Name} (Station)

- In this course, we focus on design considerations that follow from functional dependencies only