

# Relational Database Design

CPS 216  
Advanced Database Systems

## Announcements

- ❖ DB2 accounts have been set up
  - Let me know if you have not received an email from me regarding your account
- ❖ Recitation session this Friday (January 17) on E/R database design and review of relational algebra and design theory
- ❖ Office hours
  - Me (D327): Wed. 3:35-4:35pm & Fri. 2:00-3:00pm
  - TA (D328): Tue. & Thu. 12:00-1:00pm

## Database (schema) design

- ❖ Understand the real-world domain being modeled
- ❖ Specify it using a database design model
  - Design models are especially convenient for schema design, but are not necessarily implemented by DBMS
  - Popular ones include
    - Entity/Relationship (E/R) model
    - Object Definition Language (ODL)
- ❖ Translate the design to the data model of DBMS
  - Relational, XML, object-oriented, etc.
- ❖ Apply database design theory to check the design
- ❖ Create DBMS schema

## Entity-relationship (E/R) model

- ❖ Historically very popular
  - Primarily a design model; not implemented by any major DBMS nowadays
- ❖ Can think of as a “watered-down” object-oriented design model
- ❖ E/R diagrams represent designs

## E/R example



- ❖ Entity: a “thing,” like a record or an object
- ❖ Entity set (rectangle): a collection of things of the same type, like a relation of tuples or a class of objects
- ❖ Relationship: an association among two or more entities
- ❖ Relationship set (diamond): a set of relationships of the same type; an association among two or more entity sets
- ❖ Attributes (ovals): properties of entities or relationships, like attributes of tuples or objects

## ODL (Object Definition Language)

- ❖ Standardized by ODMG (Object Data Management Group)
  - Comes with a declarative query language OQL (Object Query Language)
  - Implemented by OODBMS (Object-Oriented DataBase Management Systems)
- ❖ Object oriented
- ❖ Based on C++ syntax
- ❖ Class declarations represent designs

## ODL example

7

```
class Student {
  attribute integer SID;
  attribute string name;
  relationship Set<Course> enrolledIn inverse Course::students;
};
class Course {
  attribute string CID;
  attribute string title;
  relationship Set<Student> students inverse Student::enrolledIn;
};
```

- ❖ Easy to map them to C++ classes
  - ODL attributes correspond to attributes of objects; complex types are allowed
  - ODL relationships can be mapped to pointers to other objects (e.g., `Set<Course>` → set of pointers to objects of `Course` class)

## Not covered in this lecture

8

- ❖ E/R and ODL design
- ❖ Translating E/R and ODL designs into relational designs
  - ☞ Reference book (GMUW) has all the details
  - ☞ E/R design and E/R-relational translation will be covered in recitation session this Friday
- ❖ Next: relational design theory

## Relational model: review

9

- ❖ A database is a collection of relations (or tables)
- ❖ Each relation has a list of attributes (or columns)
- ❖ Each attribute has a domain (or type)
- ❖ Each relation contains a set of tuples (or rows)

## Keys

10

- ❖ A set of attributes  $K$  is a key for a relation  $R$  if
  - In no instance of  $R$  will two different tuples agree on all attributes of  $K$ 
    - That is,  $K$  is a “tuple identifier”
  - No proper subset of  $K$  satisfies the above condition
    - That is,  $K$  is minimal
- ❖ Example: *Student* ( $SID$ ,  $name$ ,  $age$ ,  $GPA$ )
  - $SID$  is a key of *Student*
  - $\{SID, name\}$  is not a key (not minimal)

## Schema vs. data

11

*Student*

| <i>SID</i> | <i>name</i> | <i>age</i> | <i>GPA</i> |
|------------|-------------|------------|------------|
| 142        | Bart        | 10         | 2.3        |
| 123        | Milhouse    | 10         | 3.1        |
| 857        | Lisa        | 8          | 4.3        |
| 456        | Ralph       | 8          | 2.3        |
| ...        | ...         | ...        | ...        |

- ❖ Is *name* a key of *Student*?
  - Yes? Seems reasonable for this instance
  - No! Student names are not unique in general
- ❖ Key declarations are part of the schema

## More examples of keys

12

- ❖ *Enroll* ( $SID$ ,  $CID$ )
  - $\{SID, CID\}$
- ❖ *Address* ( $street\_address$ ,  $city$ ,  $state$ ,  $zip$ )
  - $\{street\_address, city, state\}$
  - $\{street\_address, zip\}$
- ❖ *Course* ( $CID$ ,  $title$ ,  $room$ ,  $day\_of\_week$ ,  $begin\_time$ ,  $end\_time$ )
  - $\{CID, day\_of\_week, begin\_time\}$
  - $\{CID, day\_of\_week, end\_time\}$
  - $\{room, day\_of\_week, begin\_time\}$
  - $\{room, day\_of\_week, end\_time\}$
  - ☞ Not a good design, and we will see why later

## Usage of keys

13

- ❖ More constraints on data, fewer mistakes
- ❖ Look up a row by its key value
  - Many selection conditions are “key = value”
- ❖ “Pointers”
  - Example: *Enroll* (*SID*, *CID*)
    - *SID* is a key of *Student*
    - *CID* is a key of *Course*
    - An *Enroll* tuple “links” a *Student* tuple with a *Course* tuple
  - Many join conditions are “key = key value stored in another table”

## Motivation for a design theory

14

| <i>SID</i> | <i>name</i> | <i>CID</i> |
|------------|-------------|------------|
| 142        | Bart        | CPS216     |
| 142        | Bart        | CPS214     |
| 857        | Lisa        | CPS216     |
| 857        | Lisa        | CPS230     |
| ...        | ...         | ...        |

- ❖ Why is this design is bad?
  - This design has redundancy, because the name of a student is recorded multiple times, once for each course the student is taking
- ❖ Why is redundancy bad?
  - Wastes space, complicates updates, and promotes inconsistency
- ❖ How about a systematic approach to detecting and removing redundancy in designs?
  - Dependencies, decompositions, and normal forms

## Functional dependencies

15

- ❖ A functional dependency (FD) has the form  $X \rightarrow Y$ , where  $X$  and  $Y$  are sets of attributes in a relation  $R$
- ❖  $X \rightarrow Y$  means that whenever two tuples in  $R$  agree on all the attributes in  $X$ , they must also agree on all attributes of  $Y$

| $X$ | $Y$ | $Z$ |
|-----|-----|-----|
| $a$ | $b$ | $c$ |
| $a$ | $b$ | ?   |
| ... | ... | ... |

Must be  $b$  ←      ← Could be anything

## FD examples

16

*Address* (*street\_address*, *city*, *state*, *zip*)

- ❖  $street\_address, city, state \rightarrow zip$
- ❖  $zip \rightarrow city, state$
- ❖  $zip, state \rightarrow zip$ ?
  - This is a trivial FD
  - Trivial FD:  $LHS \supseteq RHS$
- ❖  $zip \rightarrow state, zip$ ?
  - This is non-trivial, but not completely non-trivial
  - Completely non-trivial FD:  $LHS \cap RHS = \emptyset$

## Keys redefined using FD's

17

A set of attributes  $K$  is a key for a relation  $R$  if

- ❖  $K \rightarrow$  all (other) attributes of  $R$ 
  - That is,  $K$  is a “super key”
- ❖ No proper subset of  $K$  satisfies the above condition
  - That is,  $K$  is minimal

## Reasoning with FD's

18

Given a relation  $R$  and a set of FD's  $\mathcal{F}$

- ❖ Does another FD follow from  $\mathcal{F}$ ?
  - Are some of the FD's in  $\mathcal{F}$  redundant (i.e., they follow from the others)?
- ❖ Is  $K$  a key of  $R$ ?
  - What are all the keys of  $R$ ?

## Attribute closure

19

- ❖ Given  $R$ , a set of FD's  $\mathcal{F}$  that hold in  $R$ , and a set of attributes  $Z$  in  $R$ :  
The closure of  $Z$  (denoted  $Z^+$ ) with respect to  $\mathcal{F}$  is the set of all attributes functionally determined by  $Z$
- ❖ Algorithm for computing the closure
  - Start with closure =  $Z$
  - If  $X \rightarrow Y$  is in  $\mathcal{F}$  and  $X$  is already in the closure, then also add  $Y$  to the closure
  - Repeat until no more attributes can be added

## A more complex example

20

*StudentGrade* ( $SID, name, email, CID, grade$ )

- ❖  $SID \rightarrow name, email$
  - ❖  $email \rightarrow SID$
  - ❖  $SID, CID \rightarrow grade$
- ☞ Not a good design, and we will see why later

## Example of computing closure

21

- ❖  $\mathcal{F}$  includes:
  - $SID \rightarrow name, email$
  - $email \rightarrow SID$
  - $SID, CID \rightarrow grade$
- ❖  $\{CID, email\}^+ = ?$
- ❖  $email \rightarrow SID$ 
  - Add  $SID$ ; closure is now  $\{CID, email, SID\}$
- ❖  $SID \rightarrow name, email$ 
  - Add  $name, email$ ; closure is now  $\{CID, email, SID, name\}$
- ❖  $SID, CID \rightarrow grade$ 
  - Add  $grade$ ; closure is now all the attributes in *StudentGrade*

## Using attribute closure

22

Given a relation  $R$  and set of FD's  $\mathcal{F}$

- ❖ Does another FD  $X \rightarrow Y$  follow from  $\mathcal{F}$ ?
  - Compute  $X^+$  with respect to  $\mathcal{F}$
  - If  $Y \subseteq X^+$ , then  $X \rightarrow Y$  follow from  $\mathcal{F}$
- ❖ Is  $K$  a key of  $R$ ?
  - Compute  $K^+$  with respect to  $\mathcal{F}$
  - If  $K^+$  contains all the attributes of  $R$ ,  $K$  is a super key
  - Still need to verify that  $K$  is *minimal* (how?)

## Useful rules of FD's

23

- ❖ Armstrong's axioms
  - Reflexivity: If  $Y \subseteq X$ , then  $X \rightarrow Y$
  - Augmentation: If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$  for any  $Z$
  - Transitivity: If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$
- ❖ Rules derived from axioms
  - Splitting: If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$
  - Combining: If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$

## Non-key FD's

24

- ❖ Consider a non-trivial FD  $X \rightarrow Y$  where  $X$  is not a super key
  - Since  $X$  is not a super key, there are some attributes (say  $Z$ ) that are not functionally determined by  $X$

| $X$ | $Y$ | $Z$  |
|-----|-----|------|
| $a$ | $b$ | $c1$ |
| $a$ | $b$ | $c2$ |
| ... | ... | ...  |

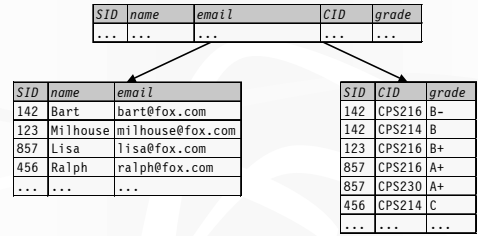
The fact that  $a$  is always associated with  $b$  is recorded in multiple rows: redundancy!

## Example of redundancy

- ❖ *StudentGrade* (*SID*, *name*, *email*, *CID*, *grade*)
- ❖ *SID* → *name*, *email*

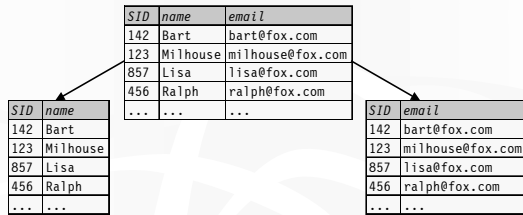
| <i>SID</i> | <i>name</i> | <i>email</i>     | <i>CID</i> | <i>grade</i> |
|------------|-------------|------------------|------------|--------------|
| 142        | Bart        | bart@fox.com     | CPS216     | B-           |
| 142        | Bart        | bart@fox.com     | CPS214     | B            |
| 123        | Milhouse    | milhouse@fox.com | CPS216     | B+           |
| 857        | Lisa        | lisa@fox.com     | CPS216     | A+           |
| 857        | Lisa        | lisa@fox.com     | CPS230     | A+           |
| 456        | Ralph       | ralph@fox.com    | CPS214     | C            |
| ...        | ...         | ...              | ...        | ...          |

## Decomposition



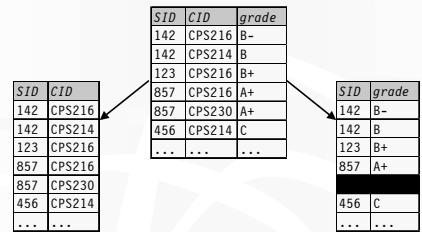
- ❖ Eliminates redundancy
- ❖ To get back to the original relation: ⋈

## Unnecessary decomposition



- ❖ Fine: join returns the original relation
- ❖ Unnecessary: no redundancy is removed, and now *SID* is stored twice!

## Bad decomposition



- ❖ Association between *CID* and *grade* is lost
- ❖ Join returns more rows than the original relation

## Questions about decomposition

- ❖ When to decompose
- ❖ How to come up with a correct decomposition

## An answer: BCNF

- ❖ A relation *R* is in Boyce-Codd Normal Form if
  - For every non-trivial FD  $X \rightarrow Y$  in *R*, *X* is a super key
  - That is, all FDs follow from “key → other attributes”
- ❖ When to decompose
  - As long as some relation is not in BCNF
- ❖ How to come up with a correct decomposition
  - Always decompose on a BCNF violation
  - ☞ Then it is guaranteed to be a correct decomposition!

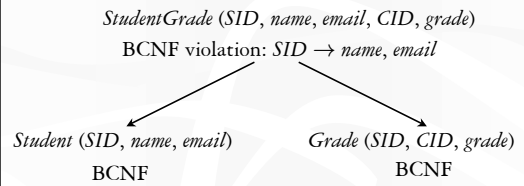
## BCNF decomposition algorithm

31

- ❖ Find a BCNF violation
  - That is, a non-trivial FD  $X \rightarrow Y$  in  $R$  where  $X$  is not a super key of  $R$
- ❖ Decompose  $R$  into  $R_1$  and  $R_2$ , where
  - $R_1$  has attributes  $X \cup Y$
  - $R_2$  has attributes  $X \cup Z$ , where  $Z$  contains all attributes of  $R$  that are in neither  $X$  nor  $Y$
- ❖ Repeat until all relations are in BCNF

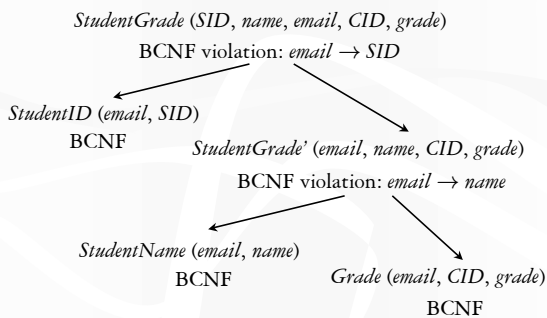
## BCNF decomposition example

32



## Another example

33



## Recap

34

- ❖ Functional dependencies: generalization of keys
- ❖ Non-key functional dependencies: a source of redundancy
- ❖ BCNF decomposition: a method of removing redundancies due to FD's
- ❖ BCNF: schema in this normal form has no redundancy due to FD's
- ☞ Not covered in this lecture: many other types of dependencies (e.g., MVD) and normal forms (e.g., 4NF)
  - GMUW has all the details
  - Relational design theory was a big research area in the 1970's, but there is not much now