

Principles of Database Systems With Internet and Java Applications

Today's Topic

Chapter 5: Improving the Quality of Database Designs

Instructor's name and information goes here
Please see the notes pages for more information.

1

Chapter 4 The Relational Data Model

- A *Relation* is a two-dimensional table
 - Fixed list of columns
 - One object per row
- An *attribute* represents a single column of a table and has a name and a type
- A *relation schema* is the name and the list of attributes of a relation
 - Grade (studentId, assignmentId, points, dateSubmitted)
- A *tuple* is a row of a table, one value for each attribute
 - (123, 14, 27, 5/28/98)

2



Model Concepts

- **Domain D** is a set of atomic values
 - Atomic means that each value is indivisible.
 - Specify domain by specifying data type and name
- **Attribute A** is the name of a role played by some domain
- **Relation schema R** is the specification of a relation (**Intension**)
 - Given by a name and list of attributes
 - Degree of a relation is the number of attributes
- **Relation (instance) $r(R)$** of a relation schema (**Extension**)
 - A set of n -tuples, n is degree of relation schema

3



Characteristics of Relational Model

- Relation is a set of tuples
 - No ordering of tuples
 - No duplicate tuples
 - no two rows have all the same values
- Each attribute value is atomic
 - hence no multiple-valued or composite attributes
 - called **first normal form**
- Each relation is a set of assertions
 - Each represents a fact
 - Some facts are about relationships
- That's it!
 - no other data structures
 - no explicit representation of relationships

4

Representing E-R Model as Relations

- Entity class → Relation schema
- Entity → row of table
 - set of all entities of class → table
- Attribute → column definition (attribute)
 - attribute value → table element
- Relationship type →
 - relation schema
 - attribute(s) of relation schema

5

Example of Relation Schema and Table

- Figure 4.2
 - Customer
 - (accountId, lastName, firstName, street, city, state, zipcode)

<u>accountId</u>	lastName	firstName	street	city	state	zipcode	balance
101	Block	Jane	1010 Main St.	Apopka	FL	30458	0.00
102	Hamilton	Cherry	3230 Dade St.	Dade City	FL	30555	4.47
103	Harrison	Kate	103 Dodd Hall	Apopka	FL	30457	30.57
104	Breaux	Carroll	76 Main St.	Apopka	FL	30458	34.58

6

Attributes are atomic

- Composite attributes
 - simple attribute for each field
 - Customer (accountId, lastName, firstName, street, city, state, zipcode)
- Multi-valued Attributes
 - Represent as single-valued attribute? What's the key? See Fig. 4.3

<i>accountId</i>	<i>lastName</i>	<i>firstName</i>	<i>street</i>	<i>city</i>	<i>state</i>	<i>zipcode</i>	<i>otherUser</i>
104	Breaux	Carroll	76 Main St.	Apopka	FL	30458	Judy Breaux
104	Breaux	Carroll	76 Main St.	Apopka	FL	30458	Cyrus Lambeaux
104	Breaux	Carroll	76 Main St.	Apopka	FL	30458	Jean Deaux

- Represent as table!

7

Represent Relationship Type

- Represent as *foreign key* attribute(s), Fig. 4.5

<i>accountId</i>	<i>otherUser</i>
104	Judy Breaux
104	Cyrus Lambeaux
104	Jean Deaux

<i>videoid</i>	<i>title</i>	<i>genre</i>	<i>storeId (foreign key)</i>
101	Men in Black	action comedy	5
145	The Prince of Egypt	animated drama	5
90987	Elizabeth	costume drama	3
99787	The Waterboy	comedy	3
123	The Truth about Cats and Dogs	romantic comedy	5

8

Representing relationships as attributes

■ One-to-many

- For each one-to-many relationship type R
 - subject class S (one side)
 - target class T (many side),
- add the key attributes of S to the schema of T
 - as foreign keys.
- Name the foreign key attributes
 - uses the role that S plays in relationship type R.
- Add the attributes of the relationship type R to schema for T.

■ One-to-one

- choose one side and use above rule

9

Representing relationships as tables

- Create a relation schema for the relationship type
 - foreign key attributes for the key of the related schema
 - add attributes of the relationship type
- Schema: IsChildOf (child, parent)

<i>child</i>	<i>parent</i>
358-44-7865	269-02-8765
579-98-8778	479-98-0098
358-44-7865	579-98-8778

10

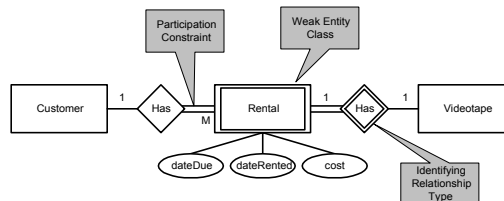
Many-to-many relationship types

- Create a relation schema for the relationship type
 - foreign key attributes for the key of the related schema
 - add attributes of the relationship type
- Examples in class!

11

Representing Weak Entity Classes

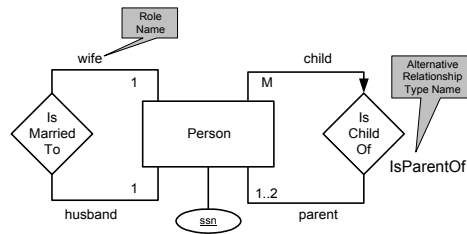
- Create a relation schema
 - Add foreign key for each defining relationship type
 - Key is partial key plus defining foreign keys
- Consider Fig. 2.5, weak class Rental
- Schema: Rental (videoid, dateDue, dateRented, cost)
 - key
 - videoid (foreign key)



12

Interpreting ER Diagrams

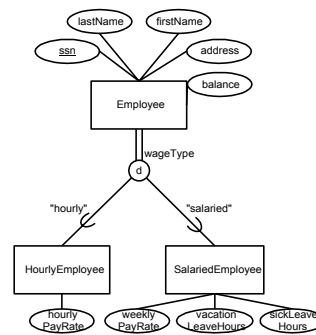
- Model of a partial representation of real objects
- Cardinalities should reflect meaning of representation, not meaning of reality!
- Consider Fig. 2.6, class Person and the IsChildOf relationship type



13

Representing specialization hierarchies

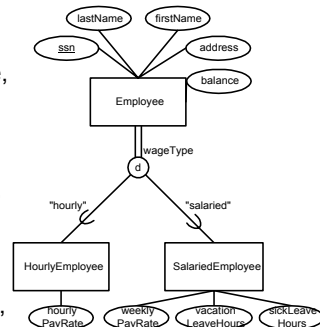
- Three possibilities
 - 1. Create a table for the superclass with its attributes and a table for each subclass with its attributes
 - 2. Create a table for the superclass with all of the subclass attributes
 - 3. Create a table for each subclass that includes both subclass and superclass attributes



14

Examples of specialization

- Applying rules to Fig. 4.12 of page 82
 - Fig. 4.13 of page 83
 - Employee:(ssn, lastName, firstName, address, balance, wageType)
 - HourlyEmployee:(ssn, hourlyRate)
 - SalariedEmployee:(ssn, weeklyPayRate, vacationLeaveHours, sickLeaveHours)
 - Fig. 4.14 of page 83
 - Employee:(ssn, lastName, firstName, wageType, hourlyRate, weeklyPayRate, vacationLeaveHours, sickLeaveHours)
 - Fig. 4.15 of page 84
 - HourlyEmployee:(ssn, lastName, firstName, address, balance, wageType, hourlyRate)
 - SalariedEmployee:(ssn, lastName, firstName, address, balance, wageType, weeklyPayRate, vacationLeaveHours, sickLeaveHours)



15

Modeling of Constraints

- A *constraint* is a declaration of a restriction on the data that can be part of a database
- ER or OR model specify *type* and *structural* constraints
 - E.g. how many and what type for attributes and relationships
- Some commonly used constraint types
 - A *key* is an attribute or set of attributes that uniquely identifies the objects in a class
 - A *single-valued constraint* requires that values of a certain attribute are unique
 - A *referential integrity constraint* requires that a value referred to by some object actually exist
 - A *domain constraint* requires that the value of an attribute be drawn from a specific set of values
 - A *general constraint* is an arbitrary assertion that must hold

16



Referential Integrity

- A *referential integrity constraint* asserts that exactly one value exists in a given role.
 - Also called a *foreign key constraint*
- In particular, a related object must exist;
 - e.g. Grade.assignmentId (attribute)
 - R.E. can be used to ensure that
 - for each Grade, there must be an Assignment with the specified assignmentId
 - for each Grade, there must be a Student with the specified studentId