

CS5614
Database Management Systems
Department of Computer Science
Virginia Tech, VA 24061
Fall 1999

Who? Where? When?

- Instructor**
 - Naren Ramakrishnan, 629 McBryde, 231-8451, naren@cs.vt.edu
 - Office Hours: MW 2-4pm (or walk in any time)

 - Teaching Assistant**
 - Wei Yu (weyu@vt.edu)
 - Office Hours: TBA

 - Class Meeting Times**
 - MWF 12-12:50, RAND 206A

 - Keeping in Touch**
 - Web Page: <http://courses.cs.vt.edu/~cs5614>
 - Enter your name on sign-on sheet
 - Listserv: cs5614@listserv.vt.edu
-

Text etc.

- Recommended**
 - (The Heiro Book) Database System Implementation, Hector Garcia-Molina, Jeffrey D. Ullman and Jennifer Widom, Prentice Hall, June 1999.
 - (The Boat Book) Database System Concepts, Abraham Silberschatz, Henry F. Korth, and S. Sudarshan, McGraw-Hill, Third Edition, 1999.
 - (The Cow Book) Database Management Systems, Raghu Ramakrishnan, McGraw-Hill, Third Edition, 1999.
 - Prerequisites**
 - Graduate Student Standing
 - Work, Grading etc.**
 - 50% Homeworks
 - 20% Midterm
 - 30% Final
-

Other References

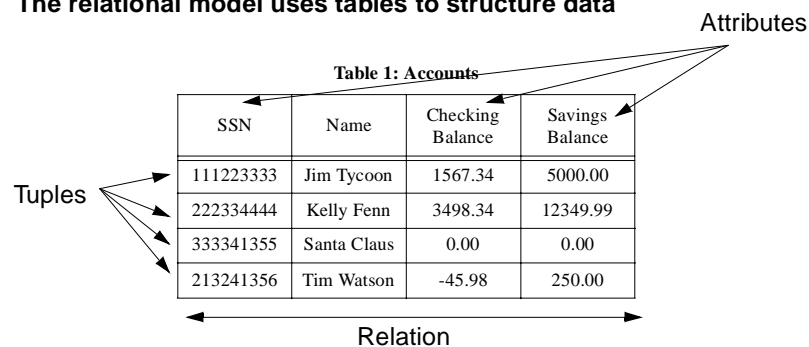
- For CS4604-like Background**
 - A First Course in Database Systems, Jeffrey D. Ullman and Jennifer Widom, Prentice Hall.
 - In-depth Coverage of Specific Topics**
 - Fundamentals of Database Systems, Elmasri and Navathe, Addison-Wesley.
 - Active Database Systems, Widom and Ceri, Morgan Kaufmann.
 - How is CS5614 Different from CS4604?**
 - Overlap for only about a month
 - More Emphasis on Implementation
 - An Insider's View
 - More Sleepless Nights
 - More Nightmare-ish than BLAIR Witch Project
-

The DB Industry

- **RDBMSs are a runaway success of simple theoretical ideas**
- **'Big 3' Database companies are among the largest in the s/w industry**
 - Oracle, Informix, Sybase
 - Other Market Forces: Microsoft, IBM
- **Exciting area for R&D too (3 Turing awards)**
 - Charles W. Bachman (1973)
 - Edgar F. Codd (1981)
 - James N. Gray (1998)
- **New applications and paradigms**
 - Biological databases
 - Semistructured data, Web-enabled databases
 - Multimedia and other forms of data
 - Data Warehousing, Information Integration
 - On-line Analytical Processing
 - Data Mining
 - Electronic Commerce

Example Scenario

- **The relational model uses tables to structure data**



- **Separates the logical view (externals) from the physical view (internals)**
- **Simple query languages (SQL) exist for accessing/modifying data**
 - Find all people who have negative balances in their checking acc.


```
SELECT Name FROM Accounts
WHERE CheckingBalance < 0
```
 - How is the answer determined?

An efficient way will be figured out by a query processor

Three Aspects of Database Systems

- Implementation**
 - How do you build a system such as ORACLE?
- Design**
 - How do you model your data and structure your info. in a database?
- Programming**
 - How do you use the capabilities of a DBMS?
- CS 5614 includes topics that address all aspects!**

Course Outline

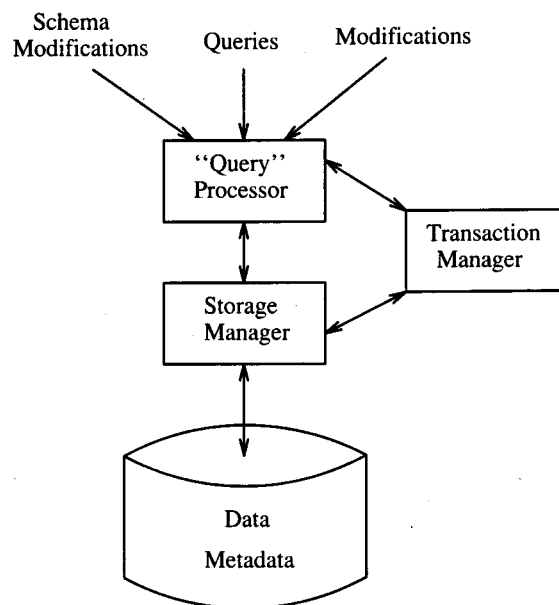
- Module 1: Data Modeling**
 - Entity-Relationship (ER) approaches
 - Specifying Constraints
 - The Relational Model
 - Converting ER to "R"
 - Perfecting Schemas
 - Normalization
 - Applications
 - Module 2: Query Processing etc.**
 - Relational Algebra
 - Datalog
 - SQL (Intergalactic dataspeak)
 - Recursion in Queries
 - Logic and Databases
 - Database Tuning
 - Plan Selection
 - Query Compilation
-

Course Outline (Contd.)

- **Module 3: Transaction Processing**
 - Concurrency Control
 - Serializable Schedules
 - Conflicts
 - Locking Mechanisms
 - Properties of Locking Mechanisms
 - Recovery Strategies
 - Logging, Resolving Deadlocks
 - OLTP
 - Active and Rule-Based Elements

- **Module 4: Information Integration**
 - Mediator-Based Approaches, Wrappers
 - Data Warehousing (CUBE operator)
 - OLAP
 - Data Mining etc.

DBMS Architecture (Simplified)



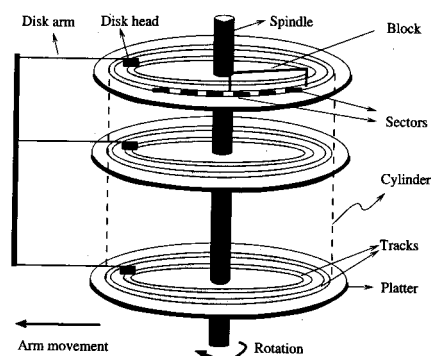
(courtesy UW)

Storage Manager

- **Memory Hierarchy**
 - Primary (Caches, Main Memory)
 - Secondary (Disks)
 - Tertiary (Tapes)
- **DBMS Storage = {Secondary + Tertiary}: Why?**
 - Nonvolatility
 - Addressability (the Y32 problem)
 - Cost
- **Disks: Operational Data, Tapes: Archival Data**
 - why?
- **Storage Manager = File Manager + Buffer Manager**
 - File Manager: Secondary Storage
 - Buffer Manager: Main Memory

Structure of a Disk

- **100 times cheaper, 2000 times slower**
- **Data must be in memory for DBMS to operate on it**
 - (not counting main memory database systems)
- **The unit of data transfer between disk and main memory = 1 block**
- **access time = seek time + rotational delay + transfer time**



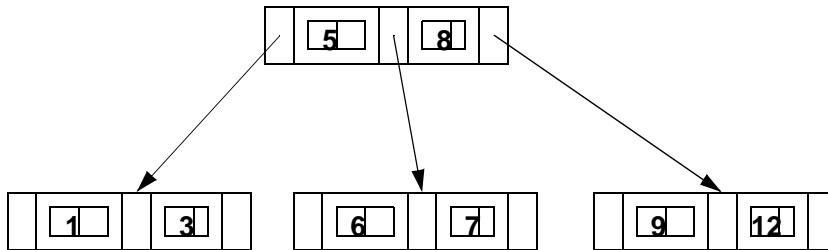
courtesy RR

More about Disks

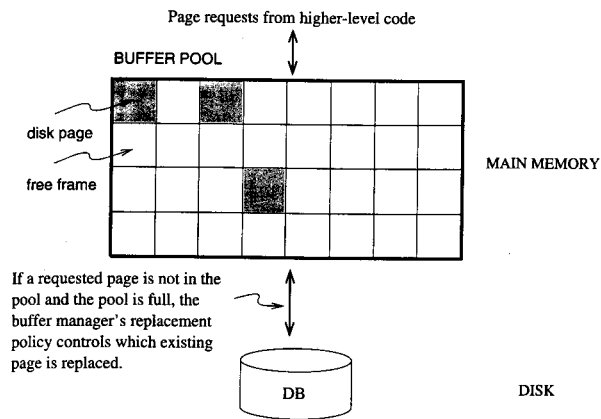
- Uses Indexes for Managing Data
 - Indexes implemented by B-trees

- Each Node of a B-Tree = 1 Disk Block ($2^{12} = 4096$ bytes)

- Why B-Trees? Why not Binary Search Trees?



Buffer Manager



courtesy RR

Query Processor

- Find the balances of all accounts of Santa Claus**

vs.

- Find the balances of all accounts of SSN: 333341355**
 - Which is more efficient?
- RDBMSs are declarative!**
 - One of the main reasons for their runaway success
 - specify what you want; not how to do it

Transaction Manager

- Enforces ACID Properties**
 - Atomicity
 - Consistency
 - Isolation
 - Durability
 - Locking (addresses I)**
 - Lock individual tuples
 - Lock whole relation
 - Logging (addresses D)**
 - performed on nonvolatile storage
 - Commitment (addresses A, D)**
-

Active Database Elements

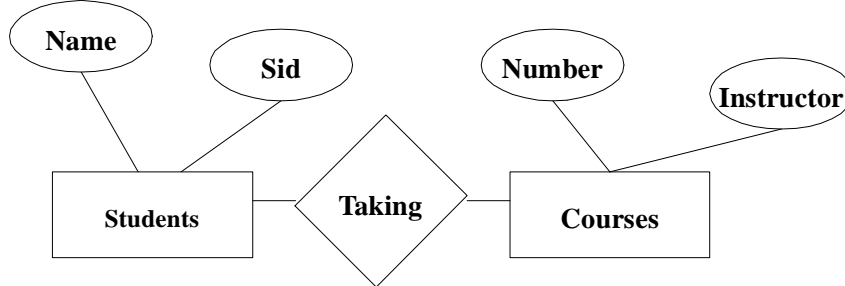
- Follow the ‘Event-Condition-Action’ Paradigm**
 - Helps a database to be reactive
 - Helps a database to incorporate domain specific knowledge
 - Available in most current commercial systems
- What Distinguishes an Active Database Element?**
 - Features that are “traditionally” implemented in application programs
- Constraints**
 - Integrity Constraints
- Triggers**
 - Alerters or Monitors
 - Authorization
 - Statistics Gathering
 - Views

Module 1: Database Modeling

- Why?**
 - To determine structure of the system before implementation
 - Start with {Ideas, Thoughts} in English**
 - Classes, Objects, Relationships, Constraints, Decisions etc.
 - Use a “design language”**
 - E/R Model: Entity-Relationship Modeling
 - Convert to Relations (for an RDBMS)**
 - in a fairly standardized and “automatic” manner
-

Entity-Relationship Modeling

- **Simple Diagrammatic approach to data modeling**
 - Very similar to Object Oriented Modeling
 - Classes = Entity Sets
 - Objects = 'Entitys'
 - Attributes = Properties

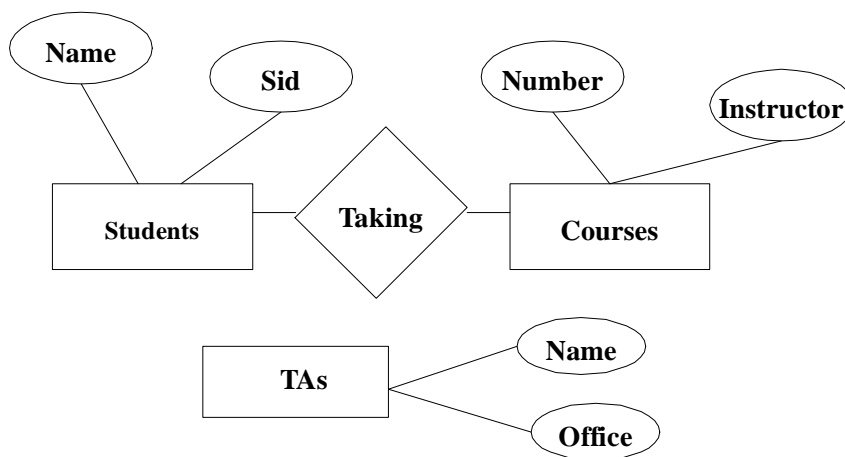


- **A Relationship = A table of associated connecting entities**

Table 2: Taking

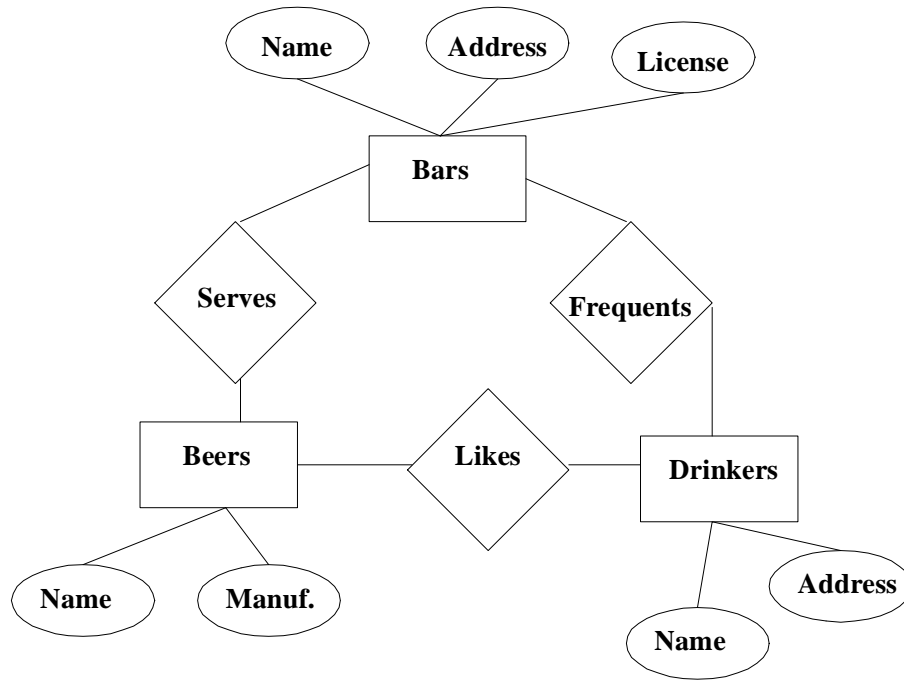
Name	SID	Number	Instructor
Ab Kader	231432345	9999	Mark
Ab Kader	231432345	9998	Dave

Three-Way Relationships



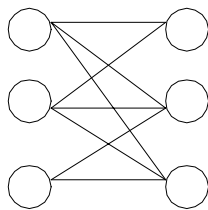
- **Exercise: How do you connect TAs to the rest of the diagram?**
 - What are your assumptions?

A 'Hello-World' for Database Systems (courtesy Widom)

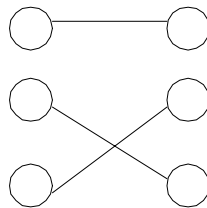
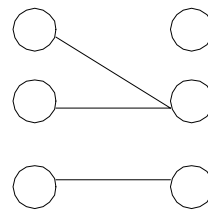


Multiplicity of Relationships

Many-Many



Many-One



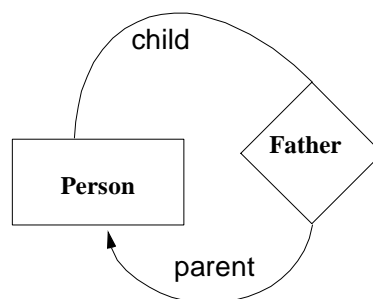
One-One

Representing a “-one” relationship

- In E-R diagrams
 - Use a pointed arrow towards the “one” set
- What to do with one-one relationships?
 - Use a pointed arrow towards both sets!
- Simple, right? :-)

Relationship Roles

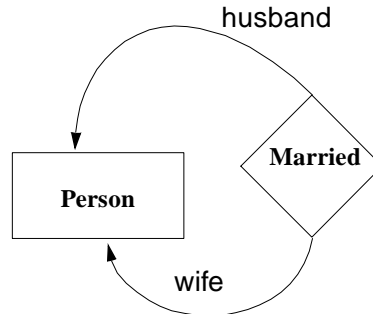
- when an entity set/class appears more than once in a relationship
 - Label the edges with roles to distinguish



- Why is there an arrow on only one side of the relationship?
- Arrow notation gets cumbersome when we have more than 2/3 sets

More Role Examples

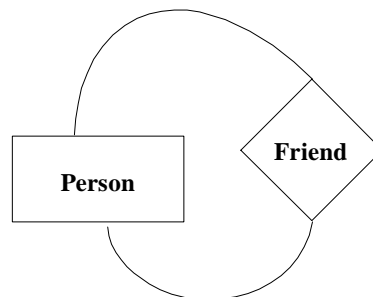
- An Arrow on Both Sides**
 - Is this relation symmetric?



- What if we replace husband and wife by “spouse”?**
-

A Symmetric Role Example

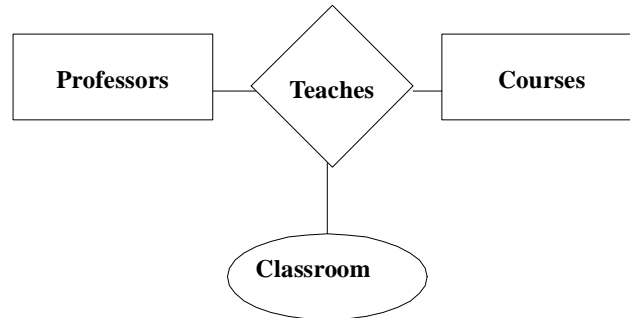
- 'Friend' is symmetric, 'Married' is not!**



- How do you encode symmetry?**
 - in E/R: No existing way, unless you invent your own shorthand
-

Attributes on Relationships

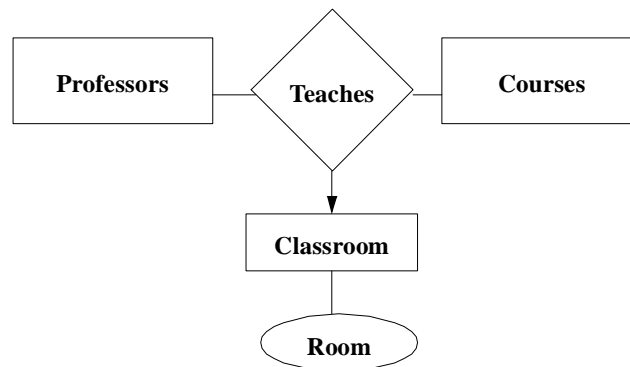
- Sometimes helpful to attach attributes to relationships



- Shorthand for three-way relationship
 - Can “push” classroom into another entity set

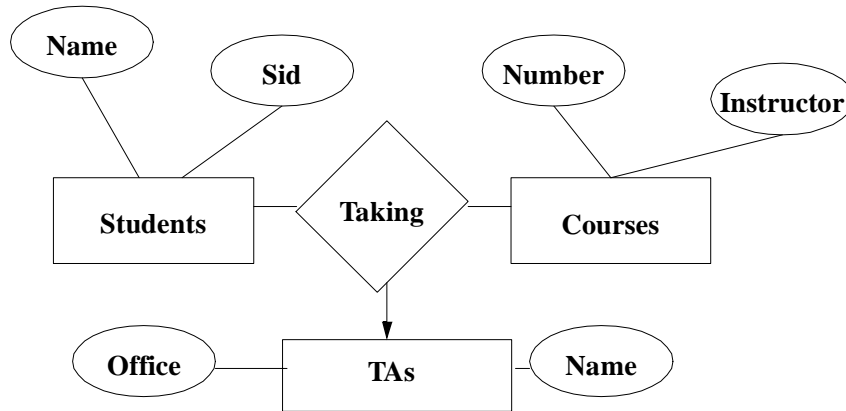
Like so...

- Notice the arrow!
 - Not sufficiently general to support all possibilities



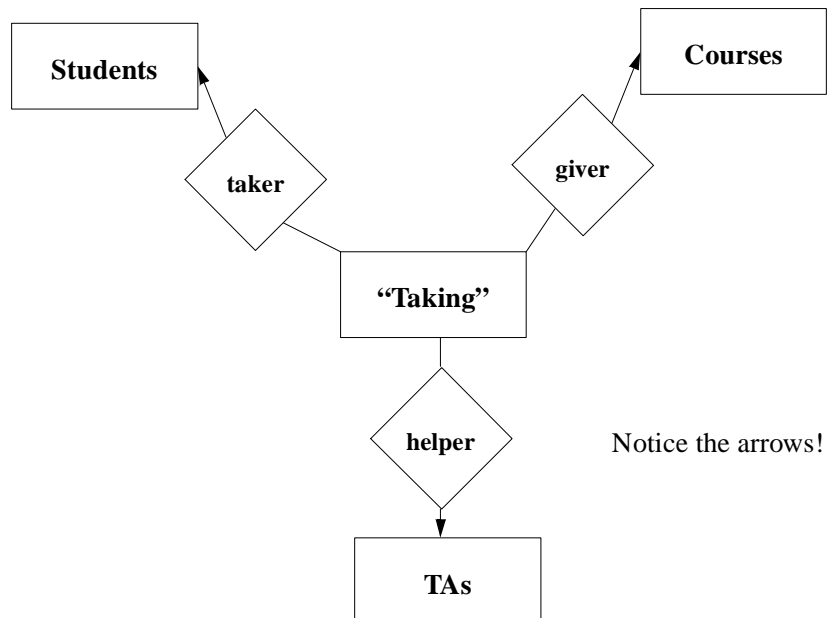
Converting Multiway Relationships to Binary

- Why would we want to do this?
- Create new entity sets for representing tuples of a relationship
 - One entity set tuple for every relationship tuple
- Recall the Student/Course/TA example



Turning a Relationship Inside Out

- Easy in E/R



Good Design Principles

- Be Faithful to requirements**
 - Talk to your client!
- Don't talk too much**
 - Avoid redundancy
 - Save space and minimize inconsistency
- Simplify life; do not complicate matters**
- Think about the end-user(s)**
- Find an appropriate way to say things**
 - Pick the right kind of element {entity set/relationship}
- Consider tradeoffs in design decisions**

Inheritance in Database Modeling

- Examples**
 - Ales are a kind of beer
 - CS courses are a kind of courses
 - Mammals are a kind of animals
 - Subclass = special case = more properties = fewer entities**
 - Inheritance in E/R**
 - Use a triangle ISA link between rectangles
 - Object belongs to both classes
-

Multiple Inheritance

- Examples**
 - “Who Killed Roger Rabbit” is a cartoon AND a murder mystery
 - Platypus has properties of both a mammal AND a non-mammal
 - CS/MATH 5485 is cross-listed as a CScourse AND a Mathcourse
 - Square is a rectangle AND a rhombus AND a quadrilateral
- Gets Messy**
 - Left to implementations to figure out disputes and conflicts

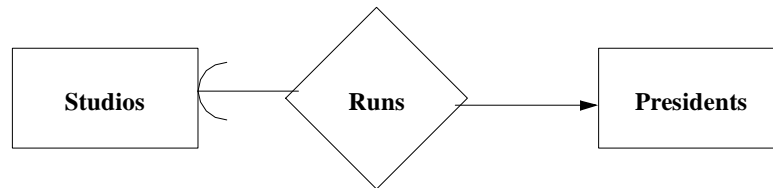
Keys

- Set of attributes whose values can belong to at most 1 entity/object**
 - SID is a key for Students
- In E/R Model**
 - Every Entity Set should have a key
 - Underline the respective attributes

Referential Integrity

- Stricter than Keys**
 - Requires that exactly one object exists in a certain role
 - Enforcing Mechanisms**
 - Forbid deletion of a referenced object
 - If a referenced object is deleted, delete all that reference it
 - When a new object is created, require that an existing object be related
 - Referential Integrity in E/R Diagrams**
 - Use rounded arrow entering the relevant entity set
-

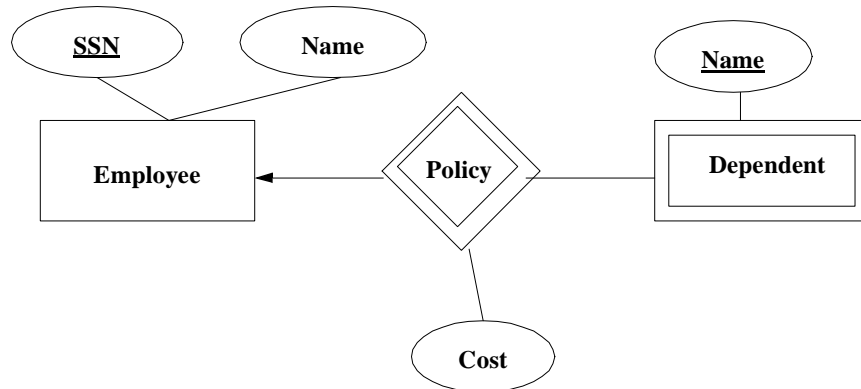
Referential Integrity Example



Weak Entity Sets

- Situation where a set's key does not come (completely) from itself**
 - comes from the key(s) of one or more sets to which it is connected to by a many-one relationship
- Primary Causes of Weak Entity Sets**
 - Elimination of multiway relationships
 - Hierarchy of Entity Sets (not inheritance)
- Representing Weak Sets in E/R Diagrams**
 - Use double border for Weak Entity Set
 - Use double border for Many-One Relationships that contribute some portion of the key

Weak Sets Example



- Key for Dependent = {Name,SSN}**
 - Note: keys can come only via many-one (or one-one) connections
- Can extend this to a “chain” of weak links**

The Relational Model

- Why?**
 - DB implementations are based on it (SQL, etc.)
 - Extremely simple: only one concept (the relation/table)
 - A good match for how we think about our data (mostly :-)
 - Has an elegant mathematical design theory
- Start with E/R**
 - Convert to Relations
 - “Normalize” Relations to improve choices for a particular design

Example

- Simple Correspondences**
 - Table = Relation
 - Column Headers = Attributes
 - Rows = Tuples

Table 3: Students

name	id	academic_level	address
Mark	1	Senior	123 Maple St, Blacksburg
Kathy	2	Senior	30 Nostreet, Christiansburg
David	3	Junior	49 Deadend, Blacksburg

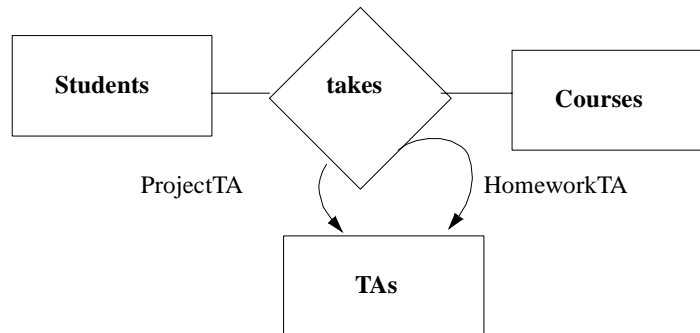
- Relation Schema**
 - Identifies name, attributes etc.
 - e.g., `student(name, id, academic_level, address)`
 - Does the order matter? also notice the underline above!
 - Relation Instance = Current Set of Tuples in the Relation Schema**
 - Database Schema = A Set of Relation Schemas**
-

DDL and DML

- DDL: Data Definition Language**
 - Use to specify relational schema
 - DML: Data Manipulation Language**
 - Use to specify relation instances
 - can also query/modify/insert/delete relation instances
 - Most modern versions of SQL serve these purposes**
-

Converting from E/R to Relations (1)

- Easier than other design languages! :-)**
 - Most of the hardwork done for us already
- Each entity set gets its own relation**
- Each relationship also gets its own relation**
 - Contains key attributes of connecting entity sets + relationship attributes



Converting from E/R to Relations (2)

- What about Weak Entity Sets?**
 - For each such set, include all the attributes that form its key
 - Easily recognizable by many-one relationships (via double diamonds)
- What about the Double Diamond Relationships?**
 - Can safely omit (!) Why?
- Handling Inheritance**
 - Every subclass gets its own relation
 - Every such relation will have both inherited and non-inherited attributes
 - No separate relation created for ISA connection
 - Information disbursed across various entity sets
- Can collapse a whole ISA hierarchy into one huge relation**
 - Saves space, preserves all info. in a single relation
 - Uses NULLs to represent "inappropriate" entries
 - Best of both worlds

Functional Dependencies

- **X -> A on R**
 - Assertion that when two tuples agree on X, they also agree on A
 - specifies a schema-level constraint on R
 - useful as an aid to “normalization”

- **Example**

```
Student(name, id, academic_level, advisor_id,  
        favorite_advisor)
```

```
id -> name  
id -> academic_level  
id -> favorite_advisor
```

but not

```
id -> advisor_id
```

even though id was a key for Student in E-R !

- **Moral of the Story: Keys for Relations are different from Keys for E-R**

Functional Dependencies (More General Stuff)

- **In general, key determines all other attributes**
 - Be careful about the way you converted from E-R to relations
- **Shorthand Notations**
 - Combine FDs with common LHSs by concatenating their RHSs
 - Omit the commas and curly braces around attributes
- **Intuitive Meaning**
 - If FD not of the form **key -> other attributes**, then the relation has too much stuff

Keys for Relations

- **K is a Key for Relation R iff**
 - K determines all other attributes of R
 - No proper subset of K determines all other attributes
 - called “Superkey” if only the first condition is satisfied

- **Example**

```
Student(name, id, academic_level, advisor_id,
        favorite_advisor)
```

```
id advisor_id -> name academic_level favorite_advisor
```

- `id` `advisor_id` is a key for `Student`
 - Neither `id` nor `advisor_id` by itself determines all other attributes
 - No other key for this example
 - `id` `advisor_id` `name` is an example of a superkey
 - Neither `id` nor `advisor_id` can be on the right of any FD, so, they must be part of any superkey
- **Doesn't sound good to have `advisor_id` be part of a key, right?**
 - Patience: we will show how to solve that problem soon! :-)

How does one identify FDs?

- **From**
 - “keyness”
 - many-one relationships
 - from the english description of the problem domain
- **Example**
 - “A professor cannot teach in two places at the same time”

```
professor_id time -> classroom
```

- **Reduce Trivial FDs**
 - ones which have the same attribute appearing on both LHS and RHS

Normalization

- **Final Goal = Boyce-Codd Normal Form (BCNF)**
 - R is in BCNF if for every nontrivial FD $X \rightarrow A$ in R, X is a superkey
- **Advantages**
 - Removes redundancy
 - Removes update anomalies
 - Removes deletion anomalies

Examples of Problems

- **Back to student**
 - “???” means redundant; can figure out from other tuples

name	id	academic_level	advisor_id	advisor_room	favorite_advisor
Mark	1	Senior	349	McBryde 630	350
???	1	???	350	McBryde 629	???
Kathy	2	Senior	146	McBryde 640	146
???	1	???	351	McBryde 641	???
???	1	???	352	McBryde 642	???
???	2	???	351	???	???
David	3	Junior	349	???	349

```
id -> name academic_level favorite_advisor
advisor_id -> advisor_room
```

- **Update Anomaly**
 - What happens if we update the `favorite_advisor` of Mark?
- **Deletion Anomaly**
 - If 350 is not the advisor for anybody, we lose his room info! :-)

The Root Cause of the Problem

- Each of the FDs

```
id -> name
id -> academic_level
id -> favorite_advisor
advisor_id -> advisor_room
```

has a LHS that is a proper subset of the key! (id advisor_id)

- A BCNF violation!
- Moral of the Story: Inferring FDs is very important in identifying pitfalls

Inferring FDs

- Given FDs

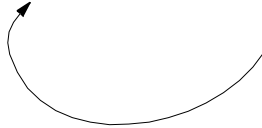
```
X1 -> A1
X2 -> A2
X3 -> A3
...
Xn -> An
```

- Does a new FD $Y \rightarrow B$ hold?
 - Solution: Compute the Closure of Y and see if it contains B !
-

Computing the Closure of Y

□ Algorithm

Set $Y' = Y$
 If an FD $X \rightarrow A$ holds and X belongs in Y'
 then add A to Y'



keep looping to Step 2 when Y' cannot be changed anymore

□ Example

$A \rightarrow B, BC \rightarrow D$

$A' = AB, B' = B, C' = C, D' = D, (AC)' = ABCD$

□ So, AC is a key! (why?)

More on Inferring FDs

□ Example:

$F = AB \rightarrow C, C \rightarrow D, D \rightarrow A$
 What FDs follow from this?

□ Do it Systematically (Brute Force; exponential complexity)

$A' = A, B' = B$ (nothing new)
 $C' = ACD$ (add $C \rightarrow A$)
 $D' = AD$ (nothing new)
 $(AB)' = ABCD$ (add $AB \rightarrow D$, skip all supersets of AB , why?)
 $(AC)' = ACD$ (nothing new)
 $(AD)' = AD$ (nothing new)
 $(BC)' = ABCD$ (nothing new, skip all supersets of BC)
 $(BD)' = ABCD$ (add $BD \rightarrow C$, skip all supersets of BD)
 $(CD)' = ACD$ (nothing new)
 skip ABC , because it is a superset of AB
 $(ACD)' = ACD$ (nothing new)
 skip BCD , because it is a superset of BC as well as BD

□ Answer: $C \rightarrow A, AB \rightarrow D, BD \rightarrow C$

Decomposing Relations

- Needed to remove anomalies**
 - Given relation R and FDs F
 - Decompose R so that BCNF violations $X \rightarrow A$ are removed
- Algorithm**
 - Expand $X \rightarrow A$ so that the right side includes X'
 - Decompose R into X' and $(R - X') \cup X$
 - Find FDs for the decomposed relations

Back to Student

- Consider the BCNF violation**

```
id -> name academic_level favorite_advisor
```
- Decompose student to**

```
Student1(id name academic_level favorite_advisor)
Student2(id advisor_id advisor_room)
```
- Project FDs down to these two relations**
 - For student1, $id \rightarrow name$, $id \rightarrow academic_level$ and $id \rightarrow favorite_advisor$
No problems here! Why? :-)
 - For student2, $id \rightarrow advisor_id$ is the key
but $advisor_id \rightarrow advisor_room$ still violates BCNF
- What is the solution?**

Yet Another Decomposition

- Decompose `student2` this time

- Consider the BCNF violation

```
adviser_id -> adviser_room
```

- Decompose `student2` to

```
Student21(adviser_id adviser_room)
Student22(id adviser_id)
```

- Rename `Student21` etc. to something more appropriate

- Final Relations

```
Student1(id name academic_level favorite_advisor)
Student21(adviser_id adviser_room)
Student22(id adviser_id)
```

- All are in BCNF! (see for yourself); no more redundancies or anomalies!

Normalization

- Simplest Case: BCNF

- Encountered previously

- Why?

- Guarantees that certain kinds of problems cannot arise
- BCNF guarantees removal of redundancy due to functional dependencies

- Other Normal Forms

- 3NF (weaker than BCNF)
- 4NF (stronger than BCNF)

Review of BCNF

- For every BCNF violating FD $A \rightarrow B$**
 - split relation into two parts
 - first part contains $\{A, B\}$
 - second part contains everything except B
- After Splitting**
 - check if the newer relations are in BCNF (how?)
 - else split them again!
- What happens if we split according to some other rule?**
 - we might not be able to recover the original relation
 - Lossy decomposition!
- Need not split a 2-attribute relation $R(a, b)$**
 - It's always in BCNF! why?

Example

- Consider**

```
Movie(title theater city)
```

```
theater -> city  
title city -> theater
```

- Keys are:**

```
{title, city}  
{theater, title}
```

- What happens if we split?**

```
Movie1(theater, city)  
Movie2(theater, title)
```

- This is a lossy decomposition!**
 - When we join them back, we might get corrupt data!
 - The original FDs may no longer be obeyed

Root Cause of the Problem

- There is a FD that has part of a key on the right hand side!**
 - Normally key attribute(s) should appear only on the left!
- Solution**
 - Relax our BCNF condition (called 3NF)
- A relation R is in 3NF if**
 - For every nontrivial FD $X \rightarrow A$, X is a superkey or A is part of a key
- For our example**

```

Movie(title theater city)

theater -> city
title city -> theater

```

 - The first FD has a part of a key (**city**) on the right
 - The second FD is not a BCNF violation, anyway!
 - so, no splitting!

Another interesting example

- Consider**

```

Student(name id address car)

```

 - where one Student can have many addresses and many cars
- Like so..**
 - Assume that Mark has two addresses and two cars

Table 4: Student

Name	id	Address	Car
Mark	1	Blacksburg	Pontiac
Mark	1	Christiansburg	Honda
Mark	1	Blacksburg	Honda
Mark	1	Christiansburg	Pontiac

- We are forced to repeat every combination
- Yet, there is no BCNF or 3NF violation!

Multivalued dependencies

- **Written as:**

```
id --> address
id --> car
```

- i.e., one id implies many addresses
- i.e., one id implies many cars
- but $id \rightarrow name$ is a normal FD

- **Generalization of regular FDs**

- **How to split?**

- Similar to regular splitting procedure

```
Student1(id address)
Student2(id car)
Student2(id name)
```

Multivalued dependencies (contd.)

- **Notice that in**

```
Student1(id address)
```

- $id \twoheadrightarrow address$ still holds
- we ignore it, since it involves all attributes of `Student1`
- this is called a trivial multivalued dependency!

- **4NF**

- removes redundancies due to multivalued dependencies

- **4NF implies BCNF implies 3NF**

- **Rules about multivalued dependencies**

- transitivity (If $A \twoheadrightarrow B$, $B \twoheadrightarrow C$, then $A \twoheadrightarrow C$)
- Every FD is a MD (why?)
- If $A \twoheadrightarrow B$ is a MD for R , then $A \twoheadrightarrow C$ is also an MD where C includes all attributes of R not among the A s or B s

Other Normal Forms

- 1NF**
 - requires that every column has an atomic value

- 2NF**
 - primarily of historical interest

- 5NF**
 - uses a further generalization of MDs called JDs! :-)
 - outside the scope of CS5614!

Summary of Normal Forms

- Desired Properties of “Breakups”**
 - Removal of Redundancy
 - Lossless Join Decomposition
 - Dependency Preservation

 - BCNF**
 - Guarantees the first two

 - 3NF**
 - Guarantees the second two

 - Always Aim for**
 - BCNF
 - Lossless Join
 - Dependency Preservation

 - If not possible, accept**
 - 3NF
 - Lossless Join
 - Dependency Preservation
-