The Entity/Relationship (ER) Model & DB Design

Introduction to databases

CSC343 Winter 2012

Ryan Johnson

Thanks to Manos Papagelis, John Mylopoulos, Arnold Rosenbloom and Renee Miller for material in these slides

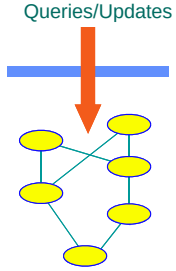
Overview

- The Entity/Relationship (ER) Model
- Designing a database schema
  - Restructuring of an E/R model
  - Translation of an E/R model into the logical model (DB Schema)

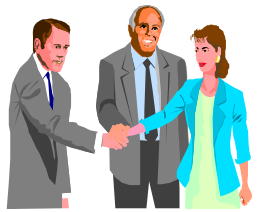
THE ENTITY/RELATIONSHIP (ER) MODEL

Conceptualizing the real-world

Modeling is about mapping entities and relationships of the world into the concepts of a database



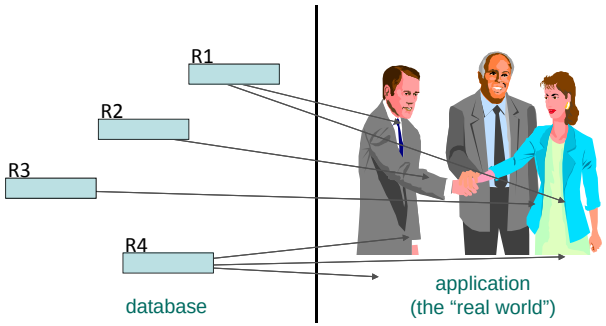
database



application
(the "real world")

Mapping is Not Deterministic

The Relational Model uses relations to represent entities, relationships, or combinations thereof



The mapping process is not always clear

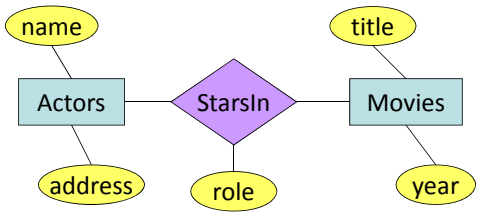
Framework for E/R

- Design is serious business
  - The boss (or client) wants a database

  => but has no clue what to put in it
- Sketching the key components is an efficient way to develop a working database
  - Sketch out (and debug) schema designs
  - Express as many constraints as possible
  - Convert to relational DB once the client is happy

Entity/Relationship Model

- Visual data model (diagram-based)
  - Quickly "chart out" a database design
  - Easier to "see" big picture
  - Comparable to class diagrams in UML
- Basic concept: *entities* and their *relationships*, along with the *attributes* describing them

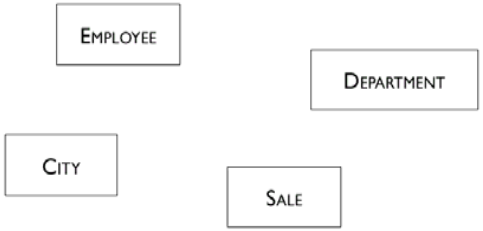


Entity/Relationship vs other models

	E/R	OO	RA
"thing" to be modeled	*Entity*	Object	Tuple
set of similar "things"	*Entity set*	Class	Relation
relationship	*Relationship*	Object?	Tuple?
set of similar relationships	*Relationship Set*	Class?	Relation?
property of a "thing" or of a relationship	*Attribute*	Field	Attribute

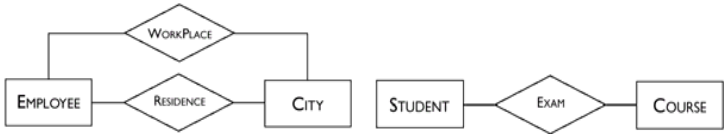
Entity Sets

- An entity set represents a class of objects that have properties in common and an autonomous existence (e.g., City, Department, Employee, Sale)
- An entity is an instance of an entity set (e.g., Stockholm is a City; Peterson is an Employee)

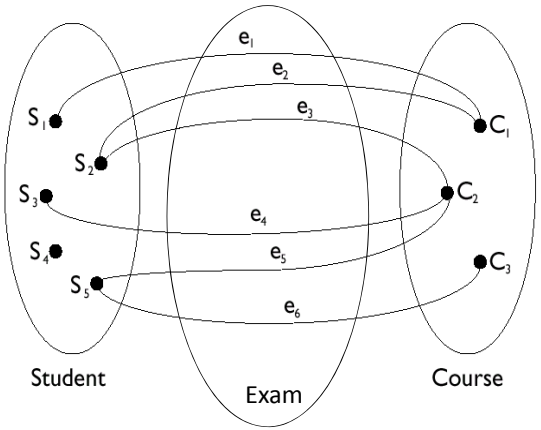


Relationship Sets

- A relationship set is an association between 2+ entity sets (e.g., Residence is a relationship set between entity sets City and Employee)
- A relationship is an instance of a n-ary relationship set (e.g., the pair <Johanssen, Stockholm> is a relationship instance of Residence)



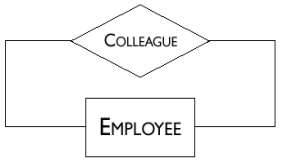
Example of Instances for Exam

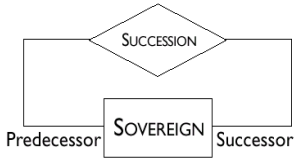


A student can't take more than one exam for a particular course

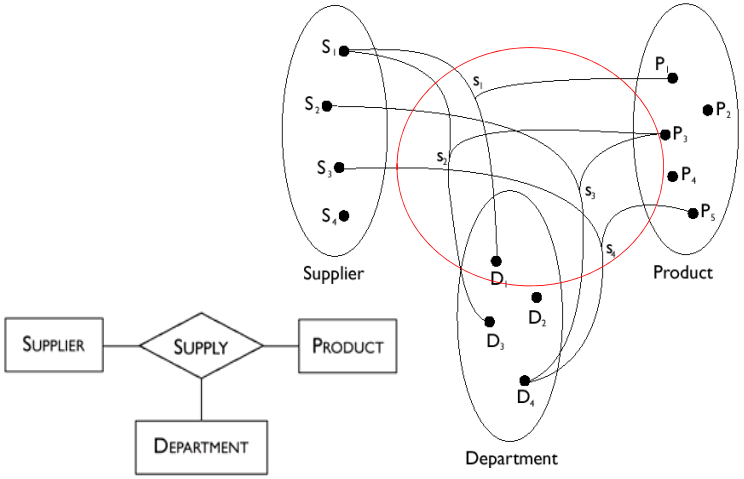
Recursive Relationships

- Recursive relationships relate an entity to itself
- Note in the second example that the relationship is not symmetric
  - In this case, it is necessary to indicate the two roles that the entity plays in the relationship



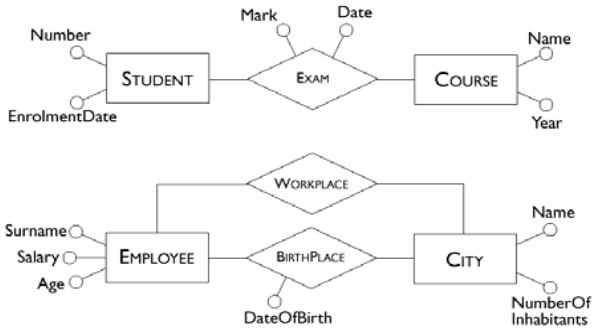


Ternary Relationships



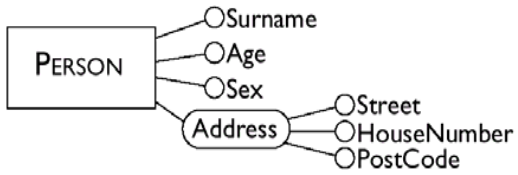
Attributes

- Describe elementary properties of entities or relationships (e.g., Surname, Salary and Age are attributes of Employee)
- May be single-valued, or multi-valued

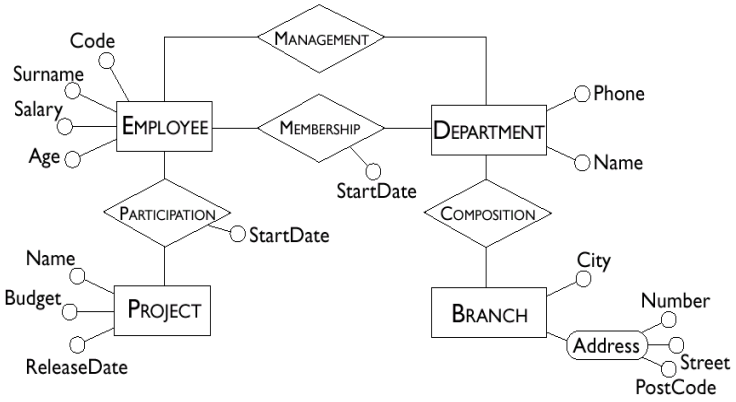


Composite Attributes

- composite attributes are grouped attributes of the same entity or relationship that have closely connected meaning or uses



Example Schema with Attributes



Cardinalities

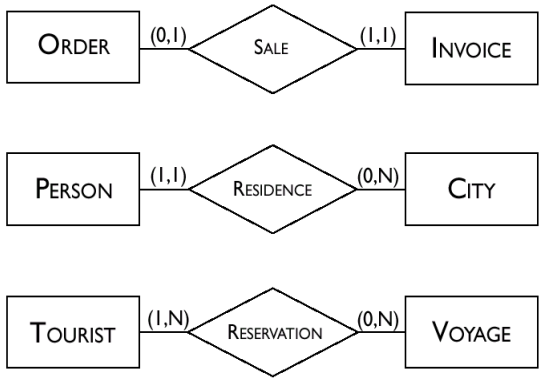
- Each entity set participates in a relationship set with a minimum (min) and a maximum (max) cardinality
- Cardinalities constrain how entity instances participate in relationship instances
- Graphical representation in E/R Diagrams: pairs of (min, max) values for each entity set

An entity might not participate in any relationship

Cardinalities (cont.)

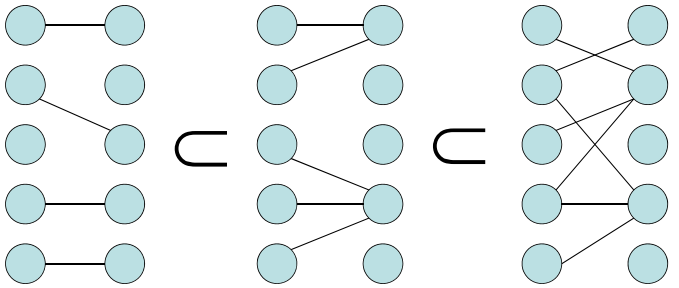
- In principle, cardinalities are pairs of non-negative integers (n, N) such that $n \leq N$, where N means "any number"
- minimum cardinality n:
  - If 0, entity participation in a relationship is optional
  - If 1, entity participation in a relationship is mandatory
- maximum cardinality N:
  - If 1, each instance of the entity is associated at most with a single instance of the relationship
  - If N, then each instance of the entity is associated with many instances of the relationship

Cardinality Examples



Multiplicity of relationships

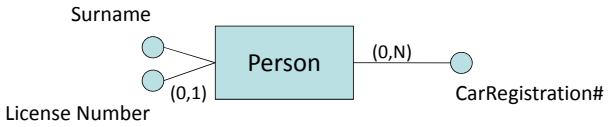
If entities E1 and E2 participate in relationship R with cardinalities (n1, N1) and (n2, N2) then the multiplicity of R is N1-N2 (= N2-N1)



1-1 N-1 OR 1-N N-N

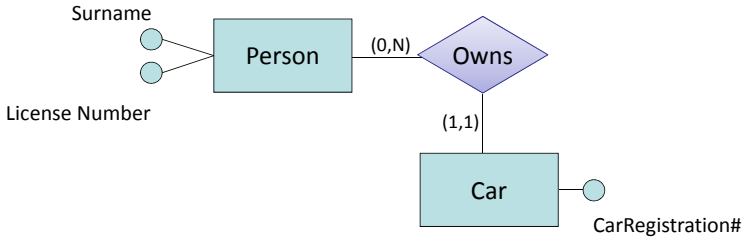
Cardinalities of Attributes

- Describe min/max number of values an attribute can have
- When the cardinality of an attribute is (1, 1) it can be omitted (single-valued attributes)
- The value of an attribute, may also be null, or have several values (multi-valued attributes)



Cardinalities of Attributes (cont.)

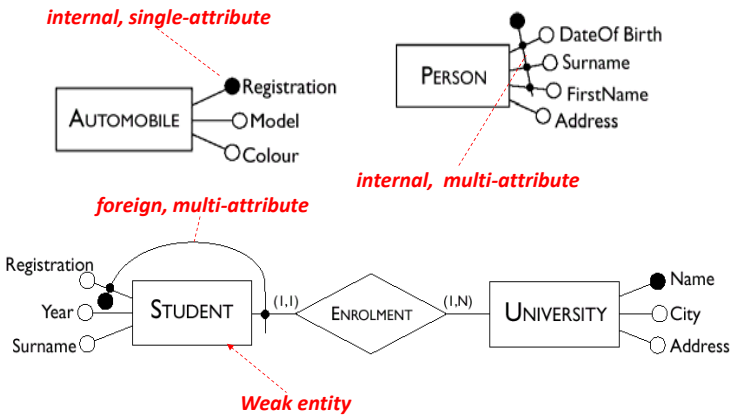
- Multi-valued attributes often represent situations that can be modeled with additional entities. E.g., the ER schema of the previous slide can be revised into:



Keys in E/R

- Keys consist of minimal sets of attributes which identify uniquely instances of an entity set
  - socialInsurance# may be a key for Person
  - firstName, middleName, lastName, address may be a key for Person
- In most cases, a key is formed by one or more attributes of the entity itself (*internal* keys)
- Sometimes, other entities are involved in the identification (*foreign keys*, *weak entities*)
- A key for a relationship consists of keys of entities it relates

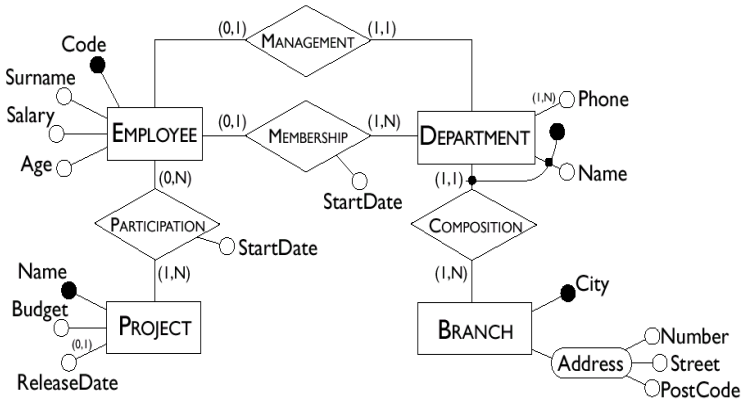
Examples of Keys in E/R



General Observations on Keys

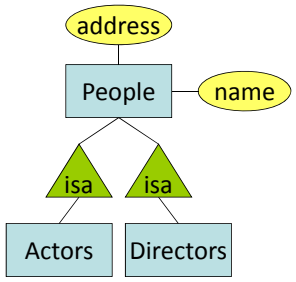
- A key may consist of one or more attributes, provided that each of them has (1,1) cardinality
- A foreign key can involve one or more entities, provided that each of them is member of a relationship to which the entity to be identified participates with cardinality equal to (1,1)
- A foreign key may involve an entity that has itself a foreign key, as long as cycles are not generated
- Each entity must have at least one (internal or foreign) key

Schema with Keys



Subclasses in E/R

- Subclass = special case → Inheritance
  - Fewer instances, more attributes (usually)
  - One-one relationship between classes
  - Attributes: union of classes involved

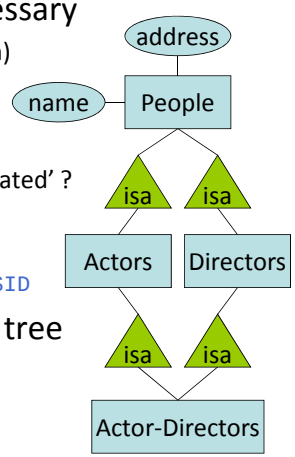


Multiple inheritance in E/R

- Allowed, but not usually necessary
  - Entity can "be" many classes (union)
- Usually not a good idea
  - Naming collisions, semantic clashes

  => What if both have attribute 'nominated' ?
  - Queries often work just as well

  => `SELECT A.* FROM Actors A, Directors D WHERE A.SID = D.SID`
- Usable classes usually form a tree

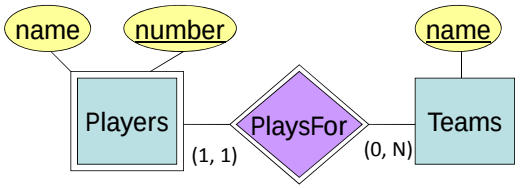


Weak entity sets

- Occasionally, entities of an entity set need "help" to identify them uniquely
- Entity set *E* is said to be *weak* if in order to identify entities of *E* uniquely, we need to follow one or more many-one relationships from *E* and include the key of the related entities from the connected entity sets
- Weak entities never exist alone
  - Always at least one *supporting* relationship to identify them
  - Other relationships allowed as well

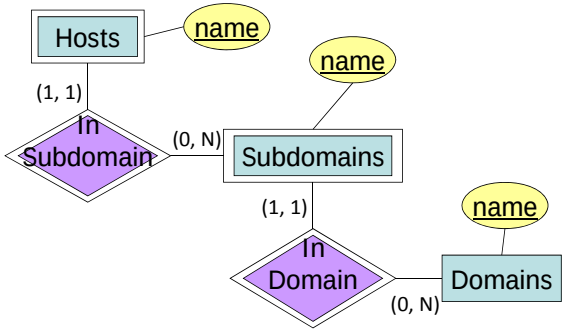
Weak entity sets – example

- name is almost a key for football players, but there might be two with the same name
- number is certainly not a key, since players on two teams could have the same number
- But number, together with the team name related to the player by PlaysFor should be unique



Double rectangle for the weak entity set
Double diamond for *supporting* many-one relationship

"Chained" weak entity sets



Key: host + subdomain + domain

Weak entity sets in practice

- Question: how does a supporting relationship identify an entity having an incomplete key?
  - Example: print servers

  => CS: inkblot, treekiller

  => Math: papershredder, treekiller
- Answer: it doesn't.
  - Option 1: replicate Printers (CSPrinters and MathPrinters)
  - Option 2: create/use some artificial key (serial number, etc.)
  - Option 3: store full key in weak entity (most common)
- Weak entities: a (sometimes useful) myth

Recommendation: need a good reason to use

Challenge: modeling the "real world"

- Life is arbitrarily complex
  - Directors who are also actors? Actors who play multiple roles in one movie? Animal actors?
- Design choices: Should a concept be modeled as an entity, an attribute, or a relationship?
- Constraints on the ER Model: A lot of data semantics can be captured but some cannot
- Key to successful model: parsimony
  - As complex as necessary, but no more
  - Choose to represent only "relevant" things

EXAMPLE

From real world to E/R Model

We wish to create a database for a company that runs training courses. For this, we must store data about trainees and instructors. For each course participant (about 5,000 in all), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent the seminars that each participant is attending at present and, for each day, the places and times the classes are held.

Each course has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each instructor (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the tutor is qualified to teach. All the instructors' telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.

From real world to E/R Model

We wish to create a database for a company that runs training courses. For this, we must store data about the *trainees* and the *instructors*. For each *course participant* (about 5,000), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent the *seminars* that each participant is attending at present and, for each day, the places and times the classes are held.

Each *course* has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each *instructor* (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the *tutor* is qualified to teach. All the instructors' telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.

Glossary

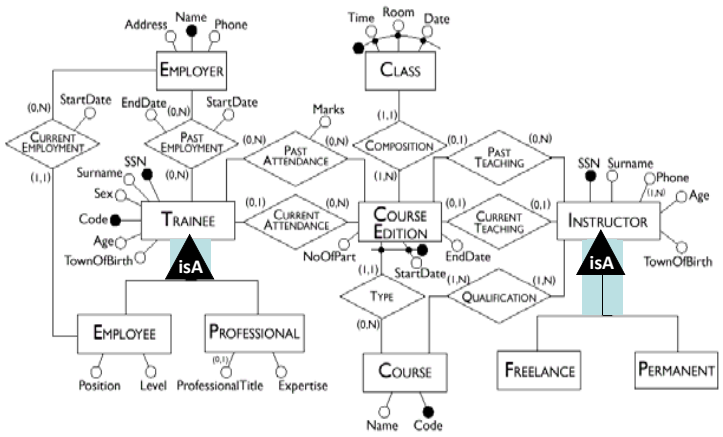
Term	Description	Synonym	Links
Trainee	Participant in a course. Can be an employee or self-employed.	Participant	Course, Company
Instructor	Course tutor. Can be freelance.	Tutor	Course
Course	Course offered. Can have various editions.	Seminar	Instructor, Trainee
Company	Company by which a trainee is employed or has been employed.		Trainee

More Annotations

We wish to create a database for a company that runs training courses. For this, we must store data about *trainees* and *instructors*. For each *course participant* (about 5,000), identified by a code, we want to store her *social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed)*, courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent *seminars* that each participant is attending at present and, *for each day, the places and times the classes are held.*

Each course has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. For *each edition, we represent the start date, the end date, and the number of participants.* If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each *instructor* (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the *tutor* is qualified to teach. All the instructors' telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.

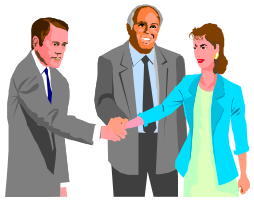
… the E/R model result



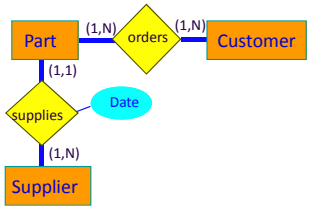
DESIGNING A DATABASE SCHEMA

Designing a Database Schema

The "real world"



The E/R Model (Conceptual Model)



The Relational Schema

Part (Name,Description,Part#)
Supplier (Name, Addr)
Customer (Name, Addr)
Supplies (Name,Part#, Date)
Orders (Name,Part#)

(Relational) Database Design

- Given a conceptual schema (ER, but could also be UML), generate a logical (relational) schema
- This is *not* just a simple translation from one model to another for two main reasons:
  - not all the constructs of the ER model can be translated naturally into the relational model
  - the schema must be restructured in such a way as to make the execution of the projected operations as efficient as possible

Logical Design Steps

It is helpful to divide the design into two steps:

- *Restructuring of the Entity-Relationship schema*, based on criteria for the optimization of the schema
- *Translation into the logical model*, based on the features of the logical model (in our case, the relational model)

RESTRUCTURING OF AN E/R MODEL

Restructuring Overview

Input: E/R Schema

Output: Restructured E/R Schema

Restructuring parts:

- Analysis of Redundancies
- Removing Generalizations (Subclasses)
- Partitioning/Merging of Entities and Relations
- Limit the Use of Weak Entity Sets
- Selection of Primary Identifiers (Keys)

Analysis of Redundancies

- *Redundancy* = saying the same thing in two (or more) different ways
- Wastes space and (more importantly) encourages inconsistency
  - Two representations of the same fact become inconsistent if we change one and forget to change the other
- Usually indicates a design flaw as well
  - Example: storing actor's address with movies

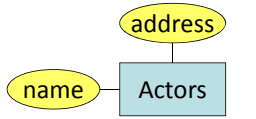
  => Address at time of filming? Now? Hotel near studio?

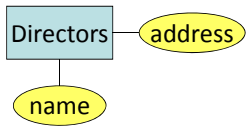
Two types of redundancy

- Repeated information

name	address	role
James Jones	Villa, CA	Vader
James Jones	Villa, CA	Vader
James Jones	Villa, CA	Vader
James Jones	Villa, CA	Greer
James Jones	Villa, CA	Mustafa

- Repeated designs (same or similar attributes)





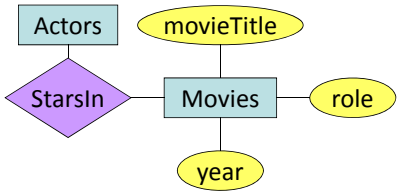
Entity Sets Versus Attributes

- An entity set should satisfy at least one of the following conditions:
  - It is more than the name of something; it has at least one nonkey attribute.

    or
  - It is the "many" in a many-one or many-many relationship.
- Rules of thumb
  - A "thing" in its own right => Entity Set
  - A "detail" about some other "thing" => Attribute
  - A "detail" correlated among many "things" => Entity Set

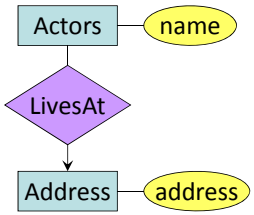
Really this is just about avoiding redundancy

E.S. vs. attributes: bad examples



Many movies, one role?

Many roles, one movie?



Redundant Entity Set and Relationship Set

Deciding about Redundancy

The presence of a redundancy in a database may be

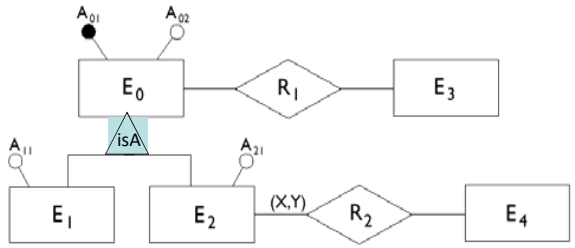
- an advantage: a reduction in the number of accesses necessary to obtain derived information
- a disadvantage: because of larger storage requirements, (but, usually at negligible cost) and the necessity to carry out additional operations in order to keep the derived data consistent

The decision to maintain or eliminate a redundancy is made by comparing the cost of operations that involve the redundant information and the storage needed, in the case of presence or absence of redundancy.

Performance analysis is required to decide about redundancy

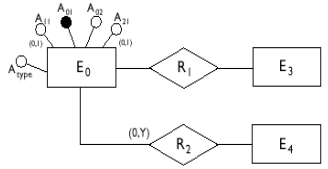
Removing Generalizations

- The relational model does not allow direct representation of generalizations that may be present in an ER diagram. For example, here is an ER schema with generalizations:

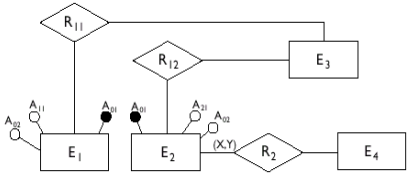


Possible Restructurings

Option 1

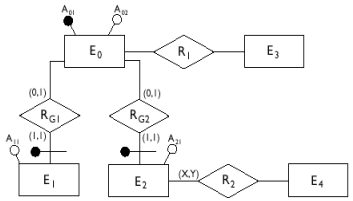


Option 2

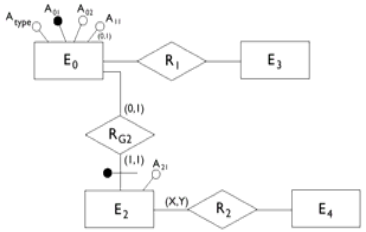


Possible Restructurings

Option 3



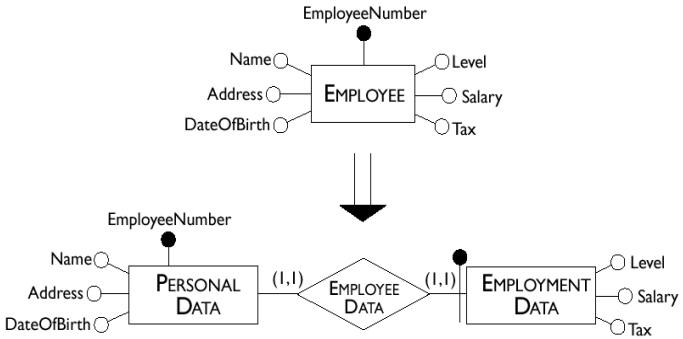
Option 4



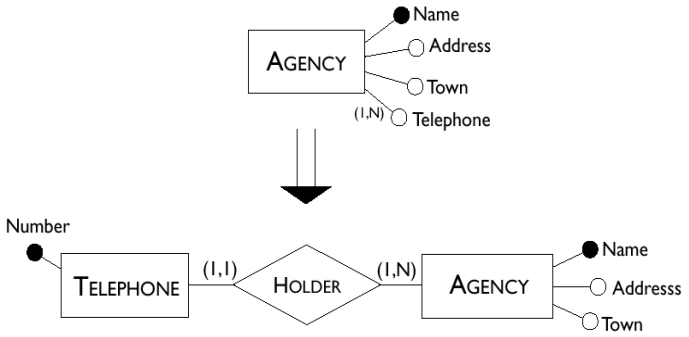
Partitioning and Merging of E/R

- Entities and relationships of an E-R schema can be partitioned or merged to improve the efficiency of operations
- Accesses are reduced by:
  - separating attributes of the same concept that are accessed by different operations and
  - merging attributes of different concepts that are accessed by the same operations

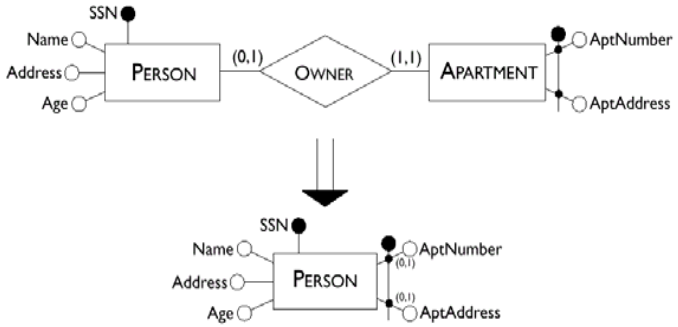
Example of Partitioning



Elimination of Multi-valued Attrib.

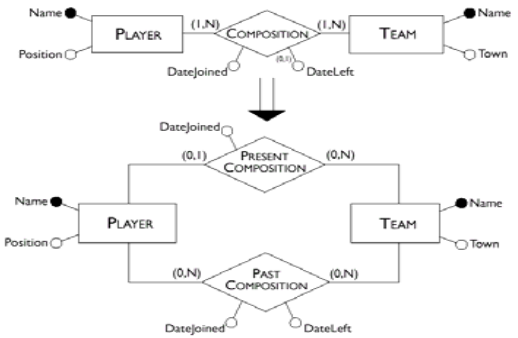


Merging Entities



Partitioning of a Relationship

Suppose that composition represents current and past compositions of a team



When to use weak entity sets?

- The usual reason is that there is no global authority capable of creating unique ID's
- Example: it is unlikely that there could be an agreement to assign unique player numbers across all football teams in the world

Don't Overuse Weak Entity Sets

- Beginning database designers often doubt that anything could be a key by itself
  - They make all entity sets weak, supported by all other entity sets to which they are linked
- In reality, each entity gets a unique ID anyway
  - Social insurance number, automobile VIN, etc.
  - Useful for many reasons (next slide)

Selecting a Primary Key

- Every relation must have a unique primary key
- The criteria for this decision are as follows:
  - Attributes with null values cannot form primary keys
  - One/few attributes is preferable to many attributes
  - Internal keys preferable to external ones (weak entities depend for their existence on other entities)
  - A key that is used by many operations to access instances of an entity is preferable to others

Keeping keys simple

Multi-attribute and/or string keys…

- … are redundant
  - e.g. Movies(title, year, …): 2 attributes, ~16 bytes
  - Number of movies ever made << 2^{32} (4 bytes)

  => Integer movieID key saves 75% space and a lot of typing
- … break encapsulation
  - e.g. Patient(firstName, lastName, phone, …)
  - Security/privacy hole

  => Integer patientID prevents information leaks
- … are brittle (nasty interaction of above two points)
  - Name or phone number change? Parent and child with same name?
  - Patient with no phone? Two movies with same title and year?

  => Internal ID always exists, immutable, unique

Also: computers are really good at integers…

TRANSLATION OF AN E/R MODEL INTO THE LOGICAL MODEL (DB SCHEMA)

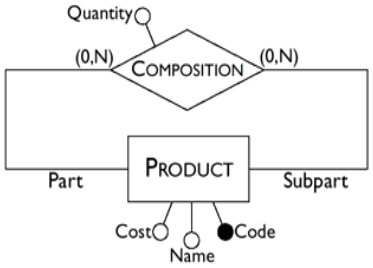
Translation into a Logical Schema

- The second step of logical design consists of a translation between different data models
- Starting from an E-R schema, an equivalent relational schema is constructed
  - "equivalent": a schema capable of representing the same information
- We will deal with the translation problem systematically, beginning with the fundamental case, that of entities linked by many-to-many relationships

Many-to-Many Relationships

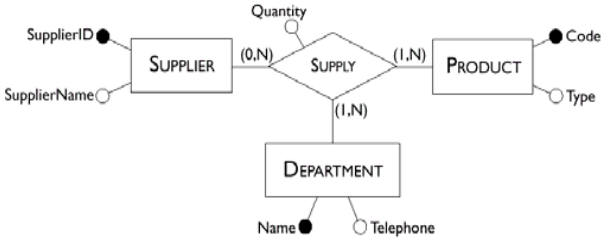
Employee(Number, Surname, Salary)
Project(Code, Name, Budget)
Participation(Number, Code, StartDate)

Many-to-Many Recursive Relationsh.



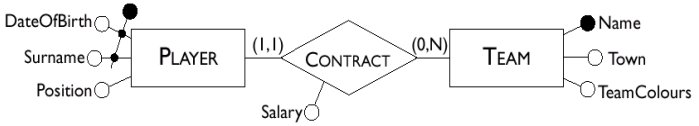
Product(Code, Name, Cost)
Composition(Part, SubPart, Quantity)

Ternary Relationships



Supplier(SupplierID, SupplierName)
Product(Code, Type)
Department(Name, Telephone)
Supply(Supplier, Product, Department, Quantity)

One-to-Many Relationships

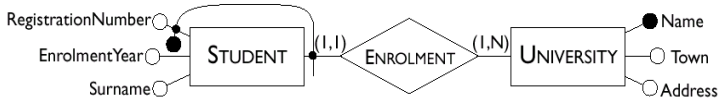


Player(Surname,DateOfBirth, Position)
Team(Name, Town, TeamColours)
Contract(PlayerSurname, PlayerDateOfBirth, Team, Salary)

OR

Player(Surname,DateOfBirth, Position, TeamName, Salary)
Team(Name, Town, TeamColours)

Weak Entities



Student(<u>RegistrationNumber</u>, <u>University</u>, Surname, EnrolmentYear)
University(<u>Name</u>, Town, Address)

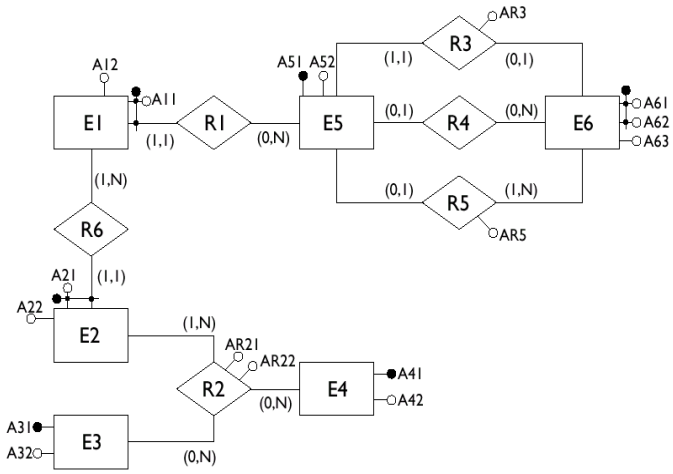
One-to-One Relationships

Head(<u>Number</u>, Name, Salary, Department, StartDate)
Department(<u>Name</u>, Telephone, Branch)
Or
Head(<u>Number</u>, Name, Salary, StartDate)
Department(<u>Name</u>, Telephone, HeadNumber, Branch)

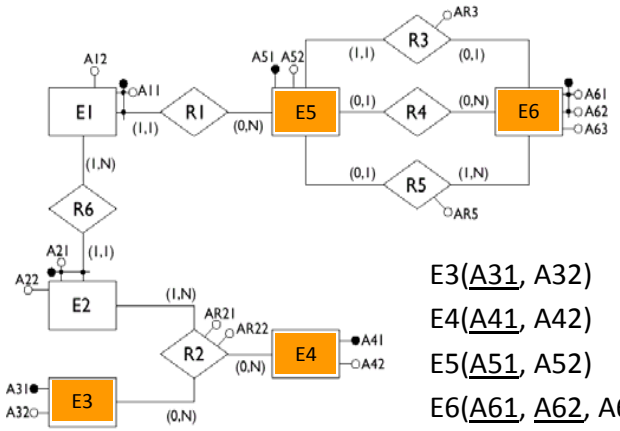
Optional One-to-One Relationships

Employee(<u>Number</u>, Name, Salary)
Department(<u>Name</u>, Telephone, Branch, Head, StartDate)
Or, if both entities are optional
Employee(<u>Number</u>, Name, Salary)
Department(<u>Name</u>, Telephone, Branch)
Management(<u>Head</u>, Department, StartDate)

A Sample ER Schema

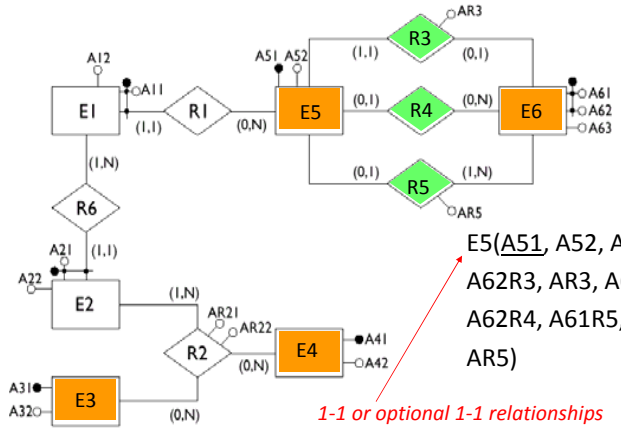


Entities with Internal Identifiers



E3(<u>A31</u>, A32)
E4(<u>A41</u>, A42)
E5(<u>A51</u>, A52)
E6(<u>A61</u>, <u>A62</u>, A63)

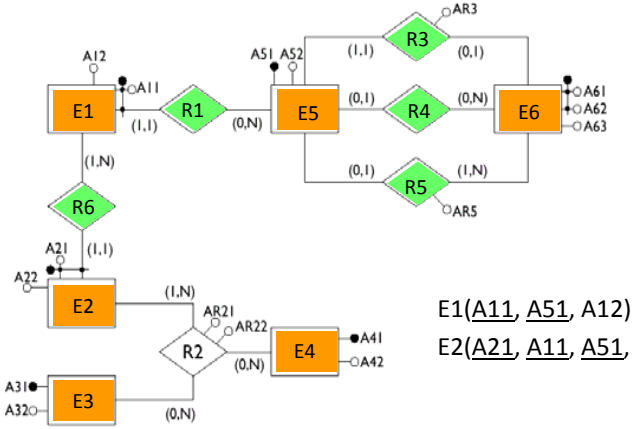
1-1 and Optional 1-1 Relationships



E5(<u>A51</u>, A52, A61R3, A62R3, AR3, A61R4, A62R4, A61R5, A62R5, AR5)

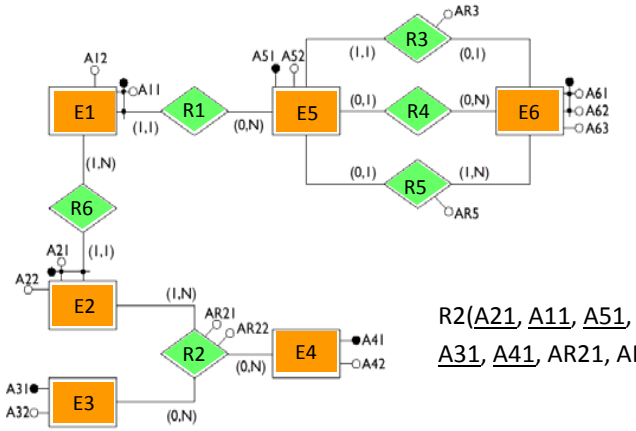
1-1 or optional 1-1 relationships
Can lead to messy transformations

Weak Entities



E1(<u>A11</u>, <u>A51</u>, A12)
E2(<u>A21</u>, <u>A11</u>, <u>A51</u>, A22)

Many-to-Many Relationships



R2(<u>A21</u>, <u>A11</u>, <u>A51</u>, <u>A31</u>, <u>A41</u>, AR21, AR22)

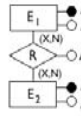
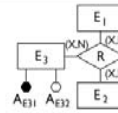
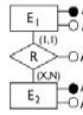
Result of the Translation

E1(A11, A51, A12)
E2(A21, A11, A51, A22)
E3(A31, A32)
E4(A41,A42)
E5(A51, A52, A61R3, A62R3, AR3, A61R4, A62R4, A61R5, A62R5, AR5)
E6(A61, A62, A63)
R2(A21, A11, A51, A31, A41, AR21, AR22)

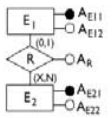
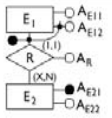
We have a Database Schema!

Ready to create our tables in the DBMS!

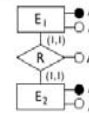
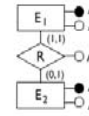
Summary of Transformation Rules

Type	Initial schema	Possible translation
Binary many-to-many relationship		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$
Ternary many-to-many relationship		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $E_3(\underline{A_{E31}}, A_{E32})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, \underline{A_{E31}}, A_R)$
One-to-many relationship with mandatory participation		$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

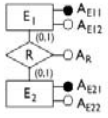
...More Rules...

Type	Initial schema	Possible translation
One-to-many relationship with optional participation		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$ Alternatively: $E_1(\underline{A_{E11}}, A_{E12}, \underline{A^*_{E21}}, A^*_R)$ $E_2(\underline{A_{E21}}, A_{E22})$
Relationship with external identifiers		$E_1(\underline{A_{E12}}, \underline{A_{E21}}, A_{E11}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

…Even More Rules...

Type	Initial schema	Possible translation
One-to-one relationship with mandatory participation for both entities		$E_1(\underline{A_{E11}}, A_{E12}, \underline{A_{E21}}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$ Alternatively: $E_2(\underline{A_{E21}}, A_{E22}, \underline{A_{E11}}, A_R)$ $E_1(\underline{A_{E11}}, A_{E12})$
One-to-one relationship with optional participation for one entity		$E_1(\underline{A_{E11}}, A_{E12}, \underline{A_{E21}}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

…and the Last One...

Type	Initial schema	Possible translation
One-to-one relationship with optional participation for both entities		$E_1(\underline{A_{E11}}, A_{E21})$ $E_2(\underline{A_{E21}}, A_{E22}, A^{*}_{E11}, A^{*}_{R})$ Alternatively: $E_1(\underline{A_{E11}}, A_{E12}, A^{*}_{E21}, A^{*}_{R})$ $E_2(\underline{A_{E21}}, A_{E22})$ Alternatively: $E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, A_{E21}, A_R)$

What is Next?

- Systematic improvements to database schemas
  - functional dependencies
  - decompositions
  - normal forms