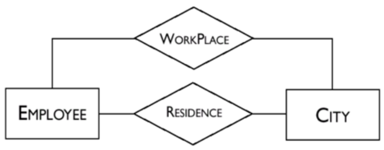
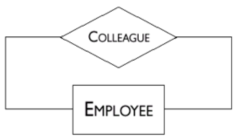
ENTITY RELATIONSHIP MODEL

- **Modeling** = mapping entities and relationships of world into concepts of a DB
  - Relational Model represent entities, relationships w/ relations
  - Mapping is not deterministics
- Entity Relationship Model Elements
  - **Entity Set** = a class of objects that have properties in common, and autonomous existence
    - ex. City, Employee
    - An **Entity** is an instance of an entity set
      - ex. Stockhom is a City, Johanseen is an Employee

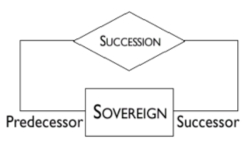
  - **Relationship Sets** = association btwn 2+ entity sets
    - ex. Residence (btwn entity sets City and Employee)
    - A Relationship is an instance of n-ary relationship set
      - ex. <Johanseen, Stockholm> is a Residence relationship



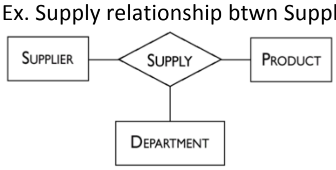
    - **Recursive Relationship** – relationships relate an entity self to itself
      - Ex. Colleague relation btwn Employees



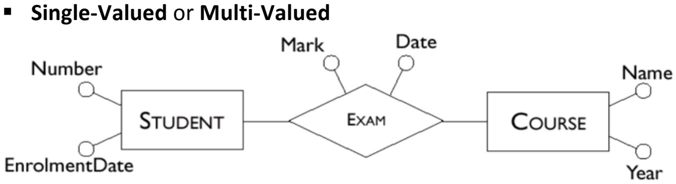
      - Recursive Relationship doesn't hav eto be symmetric, need to indicate the two roles that an Entity can be

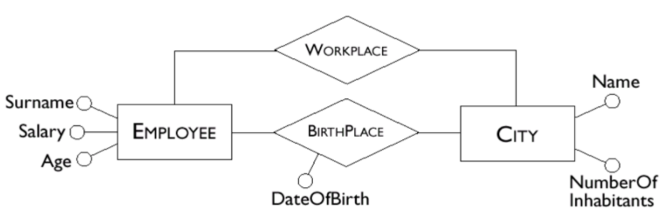


    - Ternary Relationships – relationships between 3 entity sets
      - Ex. Supply relationship btwn Supplier, Product, Department

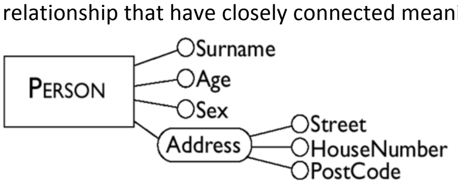


  - **Attributes** – describe elementary properties of entites and relationships
    - **Single-Valued** or **Multi-Valued**

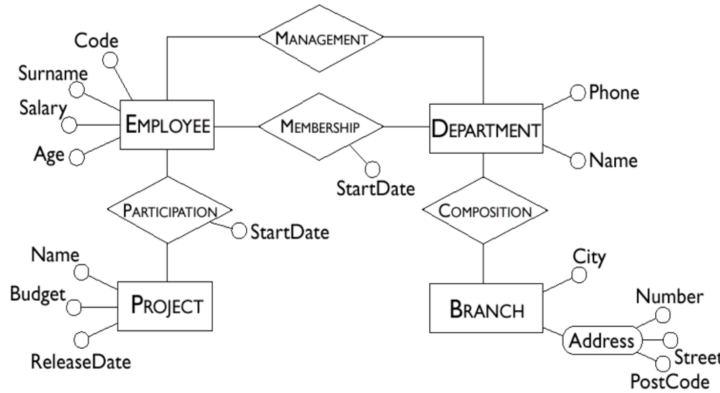




    - **Composite attributes** – grouped attributes of the same entity or relationship that have closely connected meaning or uses



  - Example ER Diagram

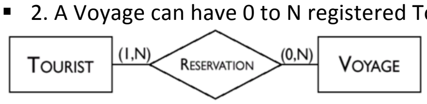


- **Cardinalities** – constrain how an Entity Instance participate in a Relationship Instance = (min, max)
  - Ex. Assignment relationship btwn Employee and Task
    - Employee → Assignment → Task: An Employee can be assigned to btwn 1 – 5 Tasks
    - Task → Assignment → Employee: A Task can be assigned to 0 – 50 Employees

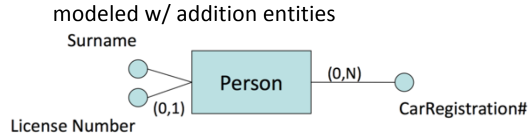
  - Cardinality = pairs of non-negative integers (Min, Max)
    - Min =
      - 0 → entity participation in a relationship is **optional**
      - 1 → entity participation in a relationship is **mandatory**
    - Max =
      - 1 → each instance of entity is associated w/ at most **single** intance of the relationship
      - N → each instance of entity is associated w/ **many** instances of the relationships
    - Multiplicity: $(1-1) \subset (N-1), (1-N) \subset (N-N)$
  - Ex1. Sale relationship btws Order and Invoice
    - 1. An Order of a Sale can have 0 to 1 Invoices
    - 2. An Invoice for a Sale is solely for an Order

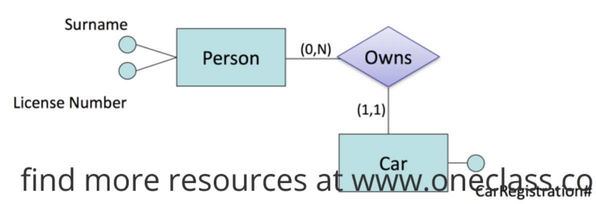
  - Ex2. Residence relationship btwn Person and City
    - 1. Person Resides in soley one City
    - 2. A City can have 0 to N Residences

  - Ex3. Reservation relationship btwn Tourist and Voyage
    - 1. A Tourist can Reserve 1 to N Voyages
    - 2. A Voyage can have 0 to N registered Tourists

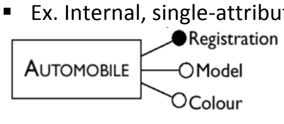


  - Cardinality of Attributes
    - **Single-valued attributes** (1, 1) can be omitted
    - **Optional attributes** (0, 1) or **Multi-valued attributes** (1, N), (0, N) have to be specified
      - Multi-valued attributes may represent situations th at can be modeled w/ addition entities

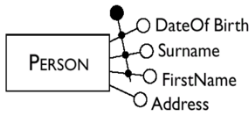




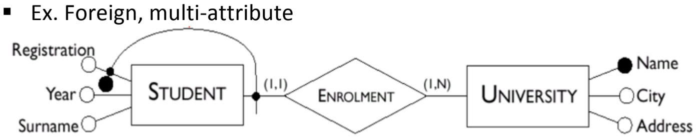
- **Keys** = minimal sets of attributes which identify unique instances of entity sets
  - Ex. SIN# attribute = key for entity set Person
  - **Internal Keys** = keys forms by one or more attributes from entity itself
    - Ex. Internal, single-attribute



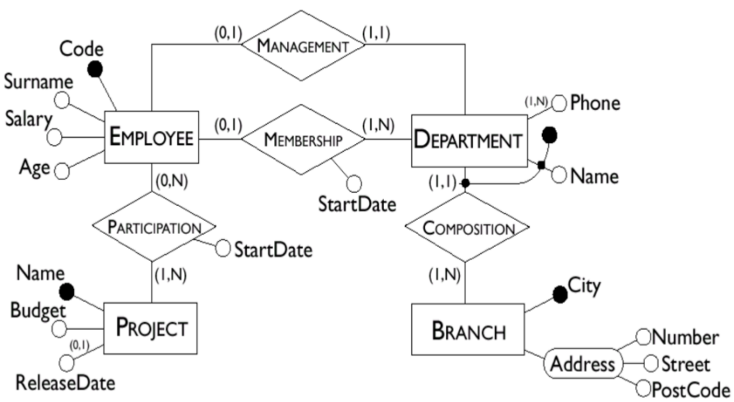
    - Ex. Internal, multi-attribute



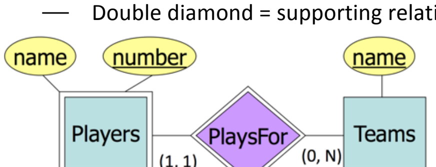
  - **Foreign Keys** = keys forms by attributes of the entity itself and attributes of other entities it has relations with
    - Ex. Foreign, multi-attribute



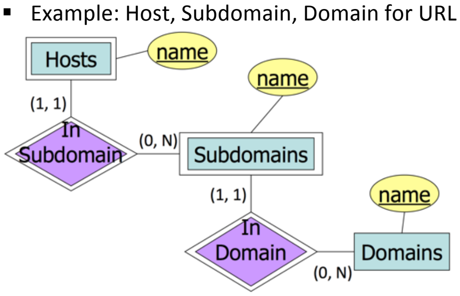
  - Example ER Diagram with Keys



- **Weak entities** = entities that requires other entities to identify itself uniquely
  - Need to follow one or more many-one relationships from the entity itself, and include the key of the related entities from the connected entity sets (foreign keys)
  - Weak entities never exist alone
    - Always require 1 **supporting** relationship for unique identification
  - Example: uniquely identify a player in the Players entity set
    - Player.name cannot be a key b/c players may share same name
    - Player.number cannot be a key b/c players on multiple teams may share numbers
    - Key: Player.number with Team.name through relationship PlaysFor
      - Double rectangle = weak entity set
      - Double diamond = supporting relationship

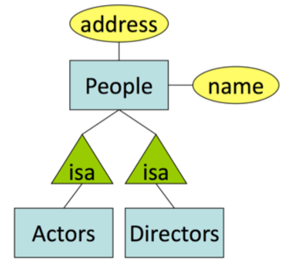


  - Chained weak entity sets
    - Example: Host, Subdomain, Domain for URL

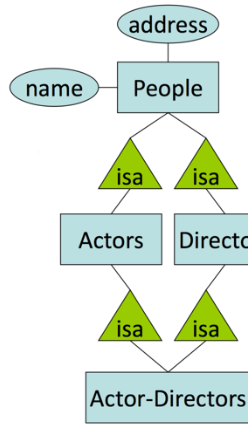


  - Weak entities usually not used in practice

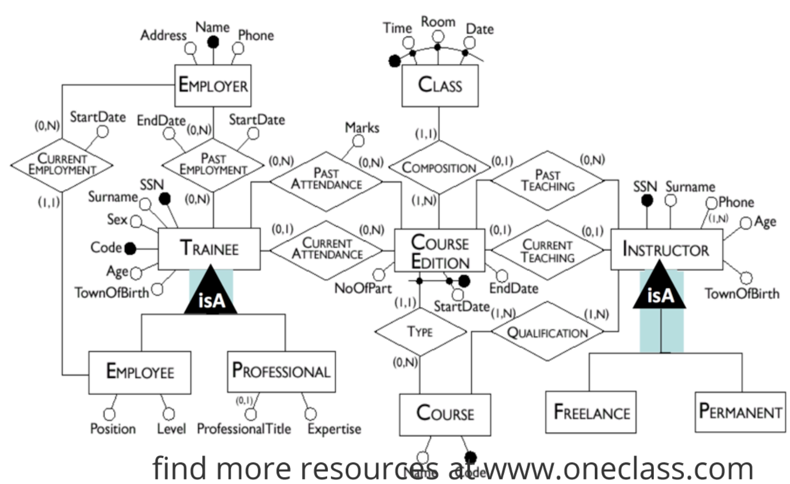
- Subclass in ER Diagrams
  - Subclass = inheritance (usually special case)
    - Fewer instances, more attributres
    - One-one relationship btwn classes



  - Multiple Inheritance not usually necessary
    - i.e. an entity can be many classes
    - Usually not good idea → naming collision
    - Usable classes usually form a tree

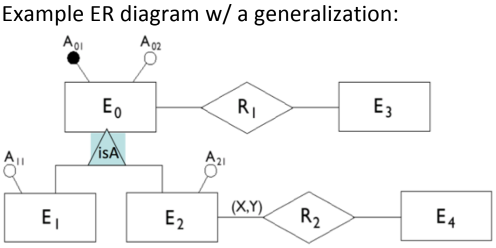


- Example: Real World problem to ER Model
  - We wish to create a database for a company that runs training courses. For this, we must store data about **trainees** and **instructors**. For each **course participant** (about 5,000), identified by a code, we want to store her social security number, surname, 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**.

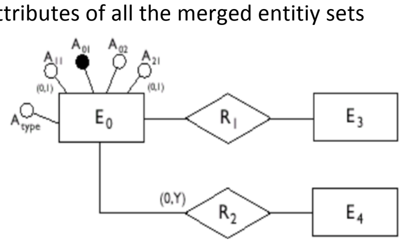


DESIGNING A DATABASE

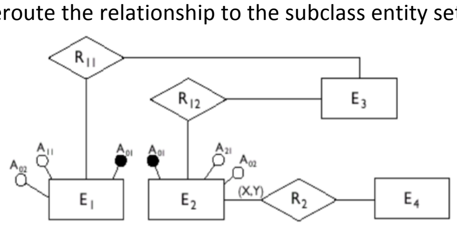
- Given conceptual schema (ER Diagram) generate a logical (relational) schema
  - Not a simple translation b/c:
    - 1. Not all constructs of ER model can be translated naturally into the relational mdoel
    - 2. Schema must be restructured in such a way to make the execution of projected operations as efficient as possible
- 1. **Restructure ER schema** based on criteria for optimization of the schema
  - Input: ER Schema → Output: Restructured E/R Schema
  - 1. **Analysis of Redundancies**
    - **Redundancy** = saying the same thing in two or more differen ways
      - — Wastes space, encourage inconsistency, indicates design flaw
    - Type 1: Repeated Information
    - Type 2: Repeated Design (same or similar attributes)
    - Entity Set vs. Attributes → an entity set satisfy one of the following conditions:
      - — 1. It is more than the name of something, it has atleast one nonkey attribute
      - — 2. It is "many" in a many-one or many-many relationship
    - **Rules**:
      - — A "thing" in its own right → Entity Set
      - — A "detail" about some other "thing → attribute
      - — A "detail" correlated among many"things" → Entity Set
    - Presence of Redundancy in a DB can be
      - — **Advantage** → a reduction in the number of accesses necessary to obtain derived information
      - — **Disadvantage** → need larger storage requirements, and need additional operations to keep data consistent
  - 2. **Removing Generalizations**
    - Relationship Model does not allow direct representations of generalizations in ER diagram
    - Example ER diagram w/ a generalization:



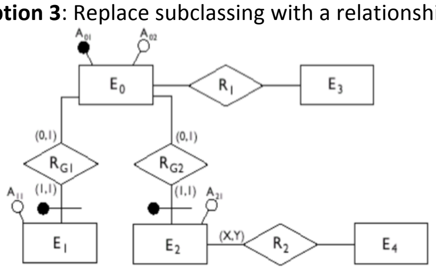
    - **Option 1**: Merge all generalizing entity sets in a single entity set with attributes of all the merged entitiy sets



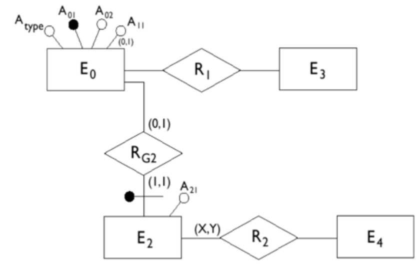
    - **Option 2**: Remove the entity set that has subclass entity sets, reroute the relationship to the subclass entity sets



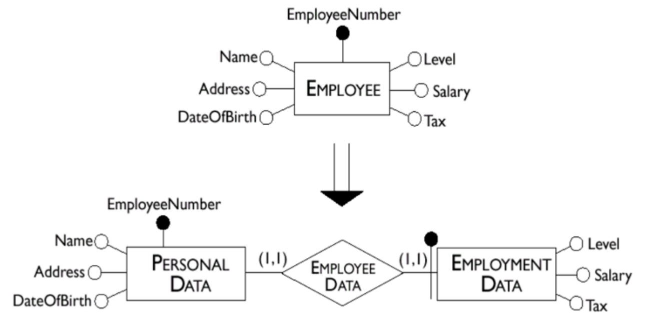
    - **Option 3**: Replace subclassing with a relationship

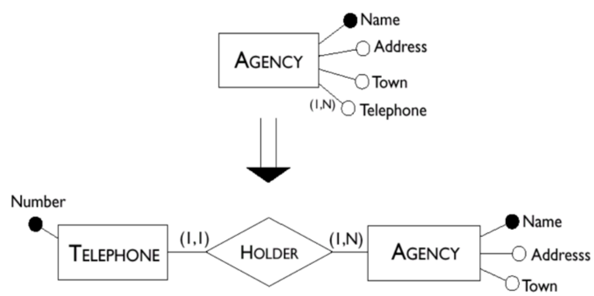


    - **Option 4**: Remove the entity set that has subclass entity sets, relate the subclass entity sets with a relationship

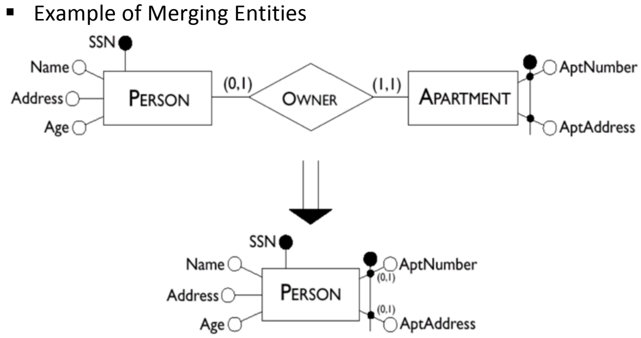


  - 3. **Partitioning/Merging of Entities and Relations**
    - Entities and Relationships can be partitioned/merged to improve efficiency, DB accesses can be reduced by
      - — **Separating Attributes** of the same concept that are accessed by **different operations**
      - — **Merging Attributes** of different concept that are accessed by the **same operation**
    - Example of Partitioning Entities

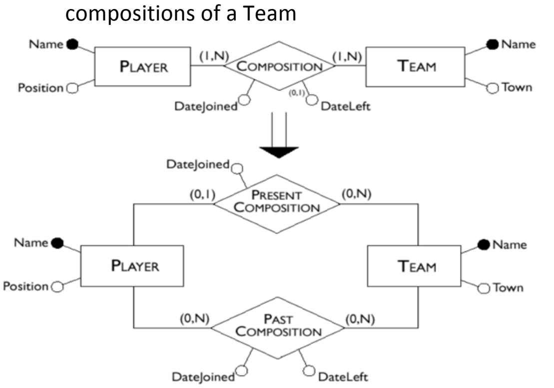




    - Example of Merging Entities

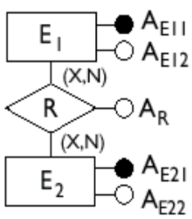


    - Example Partitoning a Relationship
      - — Suppose that Composition represents current and past compositions of a Team



- 4. **Limit the Use of Weak Entity Sets**
  - When to use Weak Entity Sets → when there's no global authority capable of creating unique IDs
    - Ex. unlikely to assign unique player numbers across all football teams in the world
  - In reality there's a way to identify each entity uniquely anyways
- 5. **Selection of Keys**
  - Every relation must have a unique primary key
  - Criteria for selecting primary keys
    - Attributes w/ null values cannot form primary keys
    - One/few attributes is preferable to many attributes
    - Internal keys preferable to external ones (weak entities)
    - A key that is used by many operations to access instnaces of an entity is preferable to others
  - Use IDs (integers) b/c easier than multi-attribute and/or string keys:

- 2. **Translate into the relational model**
  - Input: ER Schema → Output: Relational Schema
  - **Binary Many-to-Many Relationship**
    - Both Entity Sets and Relationship are all relations
      - Key of the Entity Set relation is the key of the Entity Set
      - Keys of the Relationship relation is the two keys from the Entity Sets as foreign keys



$E_1(\underline{A_{E11}}, A_{E12})$

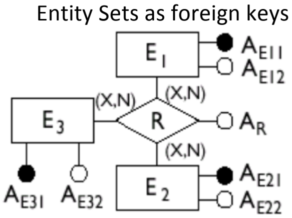
$E_2(\underline{A_{E21}}, A_{E22})$

$R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$

    - Example:

      - Relations:
        `Employee(`<u>`Number`</u>`, Surname, Salary)`
        `Project(`<u>`Code`</u>`, Name, Budget)`
        `Participation(`**<u>`Number`</u>**`, `**<u>`Code`</u>**`, StartDate)`

  - **Ternary Relationship**
    - Entity Sets and Relationship are all relations
      - Key of the Entity Set relation is the key of the Entity Set
      - Keys of the Relationship relation is the two keys from the Entity Sets as foreign keys



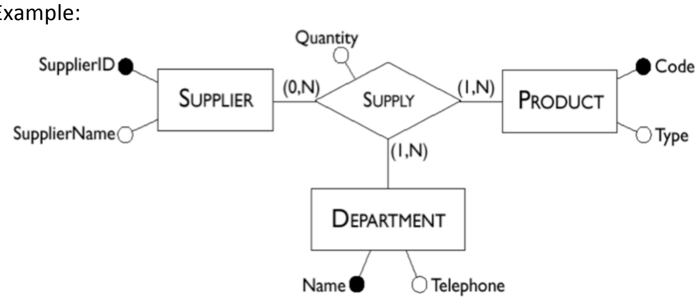
$E_1(\underline{A_{E11}}, A_{E12})$

$E_2(\underline{A_{E21}}, A_{E22})$

$E_3(\underline{A_{E31}}, A_{E32})$

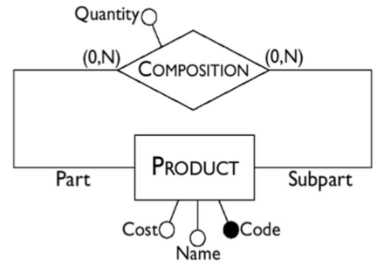
$R(\underline{A_{E11}}, \underline{A_{E21}}, \underline{A_{E31}}, A_R)$

  - Example:



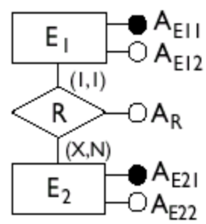
    - Relations:
      `Supplier(`<u>`SupplierID`</u>`, SupplierName)`
      `Product(`<u>`Code`</u>`, Type)`
      `Department(`<u>`Name`</u>`, Telephone)`
      `Supply(`**<u>`SupplierID`</u>**`, `**<u>`Code`</u>**`, `**<u>`Name`</u>**`, Quantity)`

  - **Many-to-Many Recursive Relationship**
    - Entity Set and Relationship are all relations
      - Key of the Entity Set relation is the key of the Entity Set
      - Key of the Relationship relation is the keys from the two instances of the Entity Set as foreign keys
    - Example:



      - Relations:
        `Product(`<u>`Code`</u>`, Name, Cost)`
        `Composition(`**<u>`PartCode`</u>**`, `**<u>`SubpartCode`</u>**`, Quantity)`

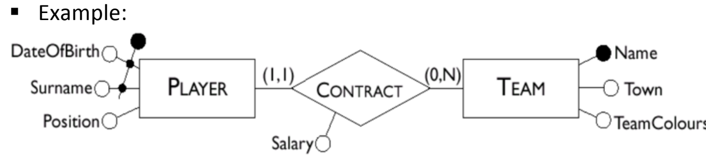
  - **One-to-Many Relationship (Mandatory)**
    - Entity Sets are relations, merge the Relationship with the mandatory "one" Entity Set
      - Key of the Entity Set relation is the key of the Entity Set
      - Mandotry "one" Entity Set given a foreign key of the "many" entity, and Relationship attributes



$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$

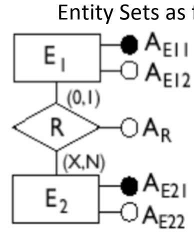
$E_2(\underline{A_{E21}}, A_{E22})$

    - Example:



      - Relations:
        `Player(`<u>`SurName`</u>`, `<u>`DateOfBirth`</u>`, Position, Salary, `**`TeamName`**`)`
        `Team(`<u>`Name`</u>`, Town, TeamColors)`

  - **One-to-Many Relationship (Optional)**
    - Option 1: Entity Sets and Relationships are all relations
      - Key of the Entity Set relation is the key of the Entity Set
      - Keys of the Relationship relation is the two keys from the Entity Sets as foreign keys

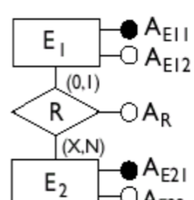


$E_1(\underline{A_{E11}}, A_{E12})$

$E_2(\underline{A_{E21}}, A_{E22})$

$R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$

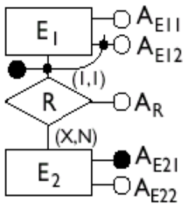
    - Option 2: Entity Sets are relations, merge the Relationship with the optional "one" Entity Set
      - Key of the Entity Set relation is the key of the Entity Set
      - Optional "one" Entity Set given a foreign key of the "many" entity (or NULL), and Relationship attributes (or NULL)



$E_1(\underline{A_{E11}}, A_{E21}, A^*_{E21}, A^*_R)$

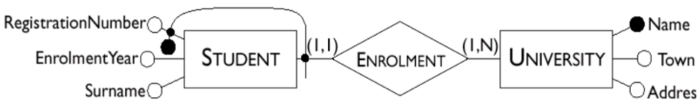
$E_2(\underline{A_{E21}}, A_{E22})$

- **Weak Entities**
  - Entity Sets are relations, merge Supporting Relationship into the "one" Entity Set
    - Key of the Entity Set relation is the key of the Entity Set
    - A key of the Weak Entity is a foreign key of "many" Entity Set
    - Relationship attributes given to Weak Entity



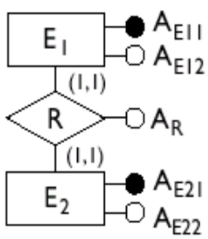
$$E_1(\underline{A_{E12}}, \underline{A_{E21}}, A_{E11}, A_R)$$
$$E_2(\underline{A_{E21}}, A_{E22})$$

  - Example:



    - Relations:
      University(Name, Town, Address)
      Student(RegistrationNumber, UniversityName, Surname, EnrolmentYear)

- **One-to-One Relationship (Mandatory)**
  - Both Entity Sets are relations
    - Key of the Entity Set relation is the key of the Entity Set
    - Merge the Relationship attributes into either one of the Entity Set and give it a foreign key of the other Entity Set



$$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$$
$$E_2(\underline{A_{E21}}, A_{E22})$$

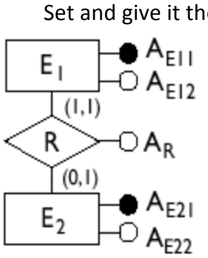
Alternatively:

$$E_2(\underline{A_{E21}}, A_{E22}, A_{E11}, A_R)$$
$$E_1(\underline{A_{E11}}, A_{E12})$$

  - Example:

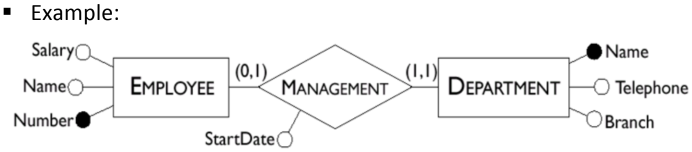
    - Relations:
      Head(Number, Name, Salary, StartDate, **DepartmentName**)
      Department(Name, Telephone, Branch)

- **One-to-One Relationship** (One Optional)
  - Both Entity Sets are relations
    - Key of the Entity Set relation is the key of the Entity Set
    - Merge the Relationship attributes into the mandatory Entity Set and give it the foreign key of the optional Entity Set



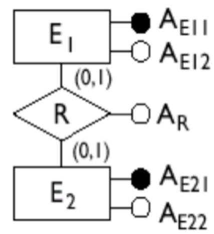
$$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$$
$$E_2(\underline{A_{E21}}, A_{E22})$$

  - Example:



    - Relations:
      Employee(Number, Name, Salary)
      Department(Name, Telephone, Branch, StartDate, **HeadEmployeeNumber**)

- **One-to-One Relationship** (Both Optional)
  - Both Entity Sets and Relationship are all relations
    - Key of the Entity Set relation is the key of the Entity Set
    - Keys of the Relationship relation is the two keys from the Entity Sets as foreign keys



$$E_1(\underline{A_{E11}}, A_{E12})$$
$$E_2(\underline{A_{E21}}, A_{E22})$$
$$R(\underline{A_{E11}}, A_{E21}, A_R)$$

  - Example:

    - Relations:
      Employee(Number, Name, Salary)
      Department(Name, Telephone, Branch)
      Management(**Head**, **Department**, StartDate)

Design Theory for Relational DBs: Functional Dependencies, Decompositions, Normal Forms

Introduction to Databases

Manos Papagelis

Thanks to Ryan Johnson, John Mylopoulos, Arnold Rosenbloom and Renee Miller for material in these slides

Database Design Theory

- Guides systematic improvements to database schemas
- General idea:
  - Express constraints on the data
  - Use these to decompose the relations
- Ultimately, get a schema that is in a “normal form” that guarantees certain desirable properties
- “Normal” in the sense of conforming to a standard
- The process of converting a schema to a normal form is called *normalization*

Goal #1: remove redundancy

- Consider this schema

Student Name	Student Email	Course	Instructor
Xiao	xiao@gmail	CSC333	Smith
Xiao	xiao@gmail	CSC444	Brown
Jaspreet	jaspreet@gmail	CSC333	Smith

- What if…
  - Xiao changes email addresses? (update anomaly)
  - Xiao drops CSC444? (deletion anomaly)
  - Need to create a new course, CSC222 (insertion anomaly)

Multiple relations => exponentially worse

Goal #2: expressing constraints

- Consider the following sets of schemas:

  Students(utorid, name, email)
  vs.
  Students(utorid, name)
  Emails(utorid, address)
- Consider also:

  House(street, city, value, owner, propertyTax)
  vs.
  House(street, city, value, owner)
  TaxRates(city, value, propertyTax)

Dependencies, constraints are domain-dependent

PART 1: FUNCTIONAL DEPENDENCIES

Functional dependencies

- Let X, Y be sets of attributes from relation R
- X -> Y is an assertion about tuples in R
  - Any tuples which agree in all attributes of X must also agree in all attributes of Y
- “X functionally determines Y”
  - Or, “The values of attributes Y are a function of those in X”
  - Not necessarily an easy function to compute, mind you

  => Consider X -> h, where h is the hash of attributes in X
- Notational conventions
  - “a”, “b”, “c” – specific attributes
  - “A”, “B”, “C” – sets of (unnamed) attributes
  - abc -> def – same as {a,b,c} -> {d,e,f}

Most common to see singletons (X -> y or abc -> d)

Rules and principles about FDs

- Rules
  - The splitting/combining rule
  - Trivial FDs
  - The transitive rule
- Algorithms related to FDs
  - the closure of a set of attributes of a relation
  - a minimal basis of a relation

The Splitting/Combining rule of FDs

- Attributes on right independent of each other
  - Consider *a,b,c -> d,e,f*
  - "Attributes a, b, and c functionally determine d, e, and f"

  => No mention of d relating to e or f directly
- Splitting rule (Useful to split up right side of FD)
  - *abc -> def* becomes *abc -> d*, *abc -> e* and *abc -> f*
- No safe way to split left side
  - *abc -> def* is NOT the same as *ab -> def* and *c -> def*!
- Combining rule (Useful to combine right sides):
  - *if abc -> d*, *abc -> e*, *abc -> f* holds, then *abc -> def* holds

Splitting FDs – example

- Consider the relation and FD
  - EmailAddress(user, domain, firstName, lastName)
  - user, domain -> firstName, lastName
- The following hold
  - user, domain -> firstName
  - user, domain -> lastName
- The following do NOT hold!
  - user -> firstName, lastName
  - domain -> firstName, lastName

Gotcha: "doesn't hold" = "not all tuples" != "all tuples not"

Trivial FDs

- Not all functional dependencies are useful
  - A -> A always holds
  - abc -> a also always holds (right side is subset of left side)
- FD with an attribute on both sides is "trivial"
  - Simplify by removing L ∩ R from R

    abc -> ad becomes abc -> d
  - Or, in singleton form, delete trivial FDs

    abc -> a and abc -> d becomes just abc -> d

Transitive rule

- The transitive rule holds for FDs
  - Consider the FDs: *a -> b* and *b -> c*; then *a->c* holds
  - Consider the FDs: *ad -> b* and *b -> cd*; then *ad->cd* holds or just *ad->c* (because of the trivial dependency rule)

Identifying functional dependencies

- FDs are domain knowledge
  - Intrinsic features of the data you're dealing with
  - Something you know (or assume) about the data
- Database engine cannot identify FDs for you
  - Designer must specify them as part of schema
  - DBMS can only enforce FDs when told to
- DBMS cannot safely "optimize" FDs either
  - It has only a finite sample of the data
  - An FD constrains the entire domain

14

Coincidence or FD?

ID	Email	City	Country	Surname
1983	tom@gmail.com	Toronto	Canada	Fairgrieve
8624	mar@bell.com	London	Canada	Samways
9141	scotty@gmail.com	Winnipeg	Canada	Samways
1204	birds@gmail.com	Aachen	Germany	Lakemeyer

- What if we try to infer FDs from the data?
  - ID -> email, city, country, surname
  - email -> city, country, surname
  - city -> country
  - surname -> country

Domain knowledge required to validate FDs

15

Keys and FDs

- Consider relation R with attributes A
- Superkey
  - Any S ⊆ A s.t. S -> A

  => Any subset of A which determines all remaining attributes in A
- Candidate key (or key)
  - C ⊆ A s.t. C -> A and X -> A does not hold for any X ⊂ C

  => A superkey which contains no other superkeys

  => Remove any attribute and you no longer have a key
- Primary key
  - The candidate key we use to identify the relation

  => Always exists, only one allowed, doesn't matter which C we use
- Prime attribute
  - ∃ candidate key C s.t. x ∈ C (attribute that participates in at least one key)

17

FD: relaxes the concept of a "key"

- Functional dependency: X -> Y
- Superkey: X -> R
- A superkey must include all remaining attributes of the relation on the RHS
- An FD can involve just a subset of them
- Example:

  Houses(street, city, value, owner, tax)
  - street,city -> value,owner,tax *(both FD and key)*
  - city,value -> tax *(FD only)*

18

Cyclic functional dependencies?

- Attributes on right side of one FD may appear on left side of another!
  - Simple example: assume relation (A, B) & FDs: A -> B, B -> A
  - What does this say about A and B?
- Example
  - studentID -> email   email -> studentID

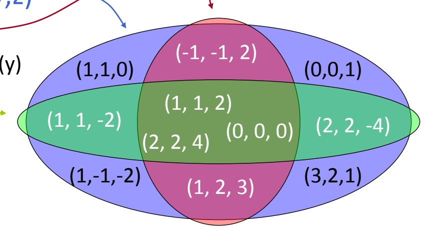
19

Geometric view of FDs

- Let D be the domain of tuples in R
  - Every possible tuple is a point in D
- FD X on R restricts tuples in R to a subset of D
  - Points in D which violate X cannot be in R
- Example: D(x,y,z)
  - xy -> z

  => z = abs(x) + abs(y)
  - z -> x,y

  => x=y=abs(z)/2



20

Inferring functional dependencies

- Problem
  - Given FDs X_1 -> a_1, X_2 -> a_2, etc.
  - Does some FD Y -> B (not given) also hold?
- Consider the dependencies

  A -> B   B -> C

  Intuitively, A -> C also holds

  The given FDs entail (imply) it (transitivity rule)

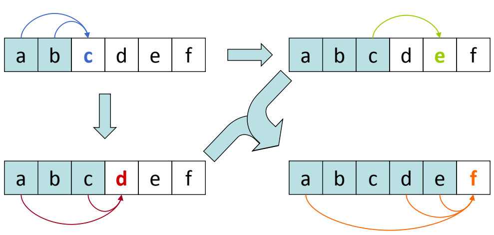
How to prove it in the general case?

Closure test for FDs

- Given attribute set A and FD set F
  - Denote A_F^+ as the closure of A relative to F
  - => A_F^+ = set of all FDs given or implied by A
- Computing the [transitive] closure of A
  - Start: $A_F^+ = A$, F' = F
  - While $\exists X \in F'$ s.t. $LHS(X) \subseteq A_F^+$:
    - $A_F^+ = A_F^+ \cup RHS(X)$
    - F' = F' - X
  - At end: A -> B $\forall B \in A_F^+$

Closure test – example

- Consider R(a,b,c,d,e,f) with FDs ab -> c, ac -> d, c -> e, ade -> f
- Find A^+ if A = ab or find $\{a,b\}^+$



$\{a,b\}^+$={a,b,c,d,e,f} or *ab -> cdef -- ab is a candidate key!*

Example : Closure Test

F: *AB -> C*
A -> D
D -> E
AC -> B

X	***X_F^+***
A	*{A, D, E}*
AB	*{A, B, C, D, E}*
AC	*{A, C, B, D, E}*
B	*{B}*
D	*{D, E}*

Is *AB -> E* entailed by ***F***? *Yes*
Is *D -> C* entailed by ***F***? *No*

Result: X_F^+ allows us to determine all FDs of the form *X -> Y* entailed by ***F***

Discarding redundant FDs

- Minimal basis: opposite extreme from closure
- Given a set of FDs F, want to minimize F' s.t.
  - $F' \subseteq F$
  - F' entails X $\forall X \in F$
- Properties of a minimal basis F'
  - RHS is always singleton
  - If any FD is removed from F', F' is no longer a minimal basis
  - If for any FD in F' we remove one or more attributes from the LHS of $X \in F'$, the result is no longer a minimal basis

Constructing a minimal basis

- Straightforward but time-consuming
  1. Split all RHS into singletons
  2. $\forall X \in F'$, test whether $J = (F'-X)^+$ is still equivalent to F^+

  => Might make F' too small
  3. $\forall i \in LHS(X)$ $\forall X \in F'$, let LHS(X')=LHS(X)-i
     Test whether $(F'-X+X')^+$ is still equivalent to F^+

  => Might make F' too big
  4. Repeat (2) and (3) until neither makes progress

Minimal Basis: Example

- Relation R: R(A, B, C, D)
- Defined FDs:
  - F = {A->AC, B->ABC, D->ABC}

Find the minimal Basis M of F

27

Minimal Basis: Example (cont.)

1st Step

- H = {A->A, A->C, B->A, B->B, B->C, D->A, D->B, D->C}

2nd Step

- A->A, B->B: can be removed as trivial
- A->C: can't be removed, as there is no other LHS with A
- B->A: can't be removed, because for J=H-{B->A} is B^+=BC
- B->C: can be removed, because for J=H-{B->C} is B^+=ABC
- D->A: can be removed, because for J=H-{D->A} is D^+=DBA
- D->B: can't be removed, because for J=H-{D->B} is D^+=DC
- D->C: can be removed, because for J=H-{D->C} is D^+=DBAC

Step outcome => H = {A->C, B->A, D->B}

28

Minimal Basis: Example (cont.)

3rd Step

- H doesn't change as all LHS in H are single attributes

4th Step

- H doesn't change

Minimal Basis: M = H = {A->C, B->A, D->B}

29

Minimal Basis: Example 2

- Relation R: R(A, B, C)
- Defined FDs:
  - A->B, A->C, B->C, B->A, C->A, C->B
  - AB->, AC-B, BC->A
  - A->BC
  - A->A
- Possible Minimal Bases:
  - {A->B, B->A, B->C, C->B} or
  - {A->B, B->C, C->A}
  - …