Database Technologies

Session -2

Content

- Database Design, Entity-Relationship Diagram (ERD)
- Relation model
  - Database Constraints (Primary Key, Foreign Key, Candidate key)
- ER to Relation mapping
- Codd’s 12 rules for RDBMS

Database Design

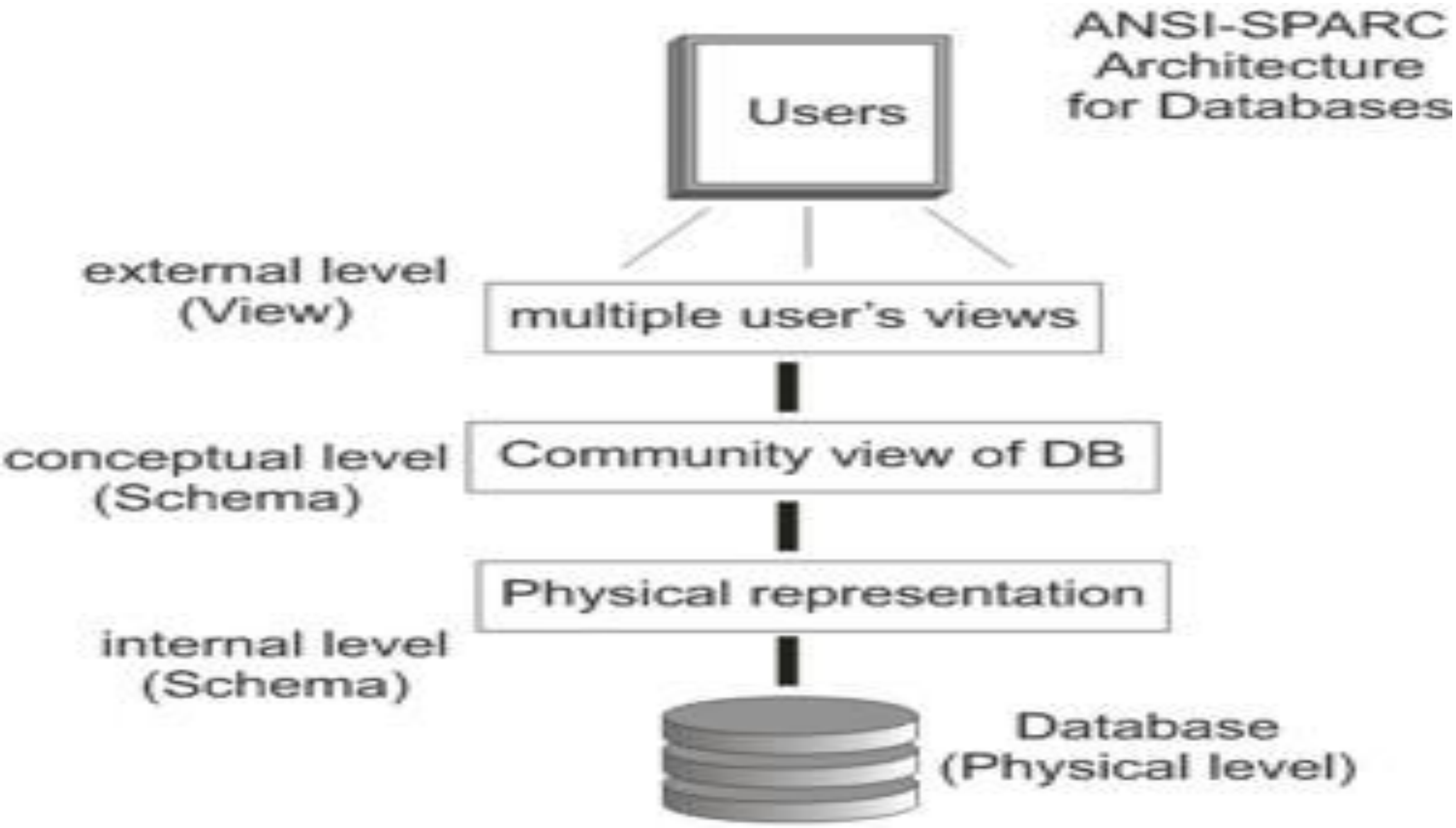
Database Design -Requirement

- Database should be easy to maintain
  - Storing only a limited amount (if any) of repetitive data.
  - If you have a lot of repetitive data and one instance of that data undergoes a change (such as a name change), that change has to be made for all occurrences of the data.
    - Create a master table

The Conceptual Model

- ANSI/SPARC model advocates the 3-tier architecture - external model, conceptual model and the internal model.
- "conceptual model" captures the global/ institutional view of the data semantics.
  - investigates and enumerates the various entities that participate in the business environment being modelled.

Three Level Architecture



ANSI-SPARC Architecture for Databases

Users

external level (View)

multiple user's views

conceptual level (Schema)

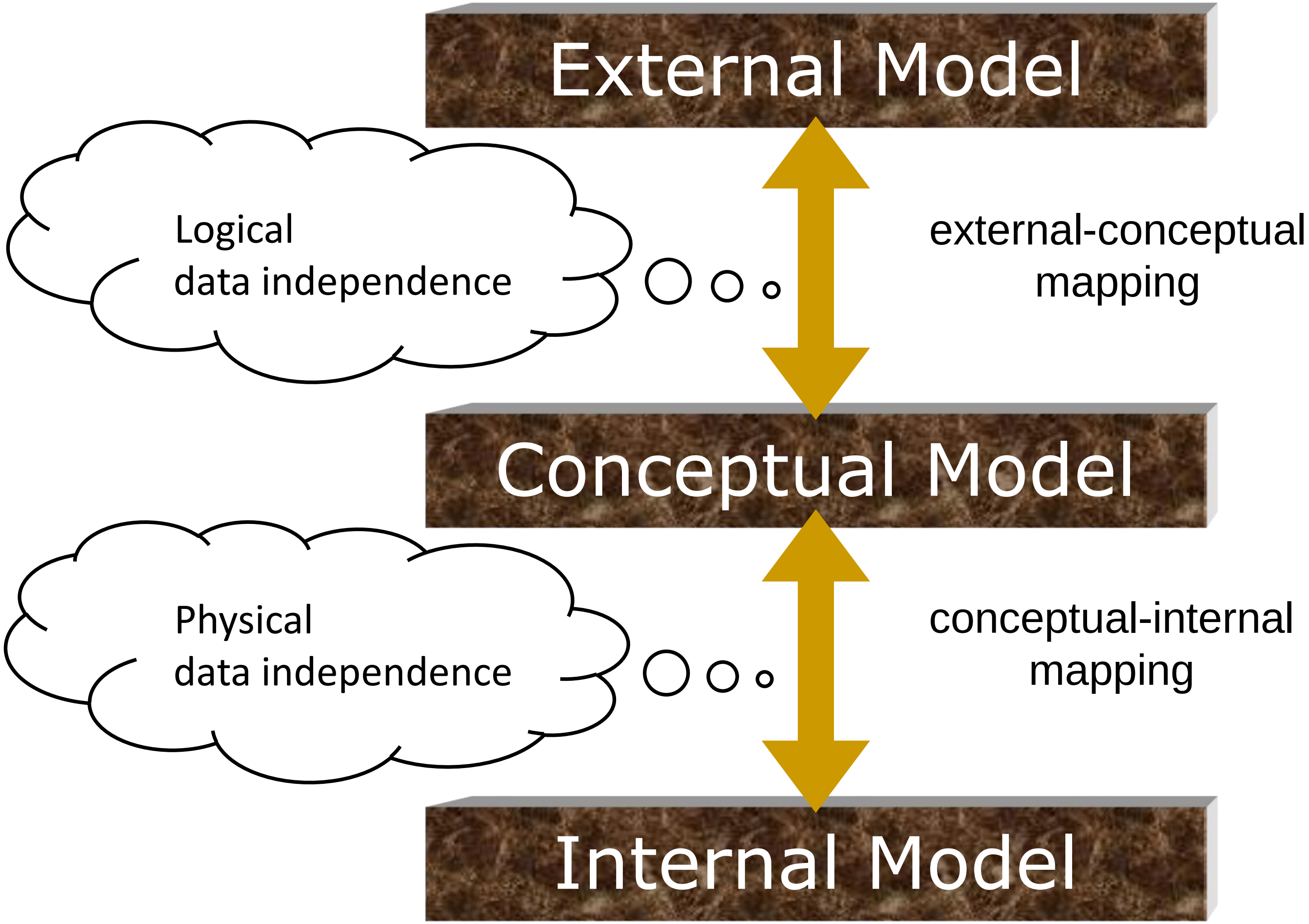
Community view of DB

Physical representation

internal level (Schema)

Database (Physical level)

ANSI/SPARC 3-Level Architecture


External Model

Conceptual Model

Internal Model

external-conceptual mapping

conceptual-internal mapping

Logical data independence

Physical data independence

Levels of Abstraction

- **Physical level:** describes how a record (e.g., customer) is stored.
- **Logical level:** describes data stored in database, and the relationships among the data.

```
type customer = record
        customer_id : string;
        customer_name : string;
        customer_street : string;
        customer_city : string;
    end;
```

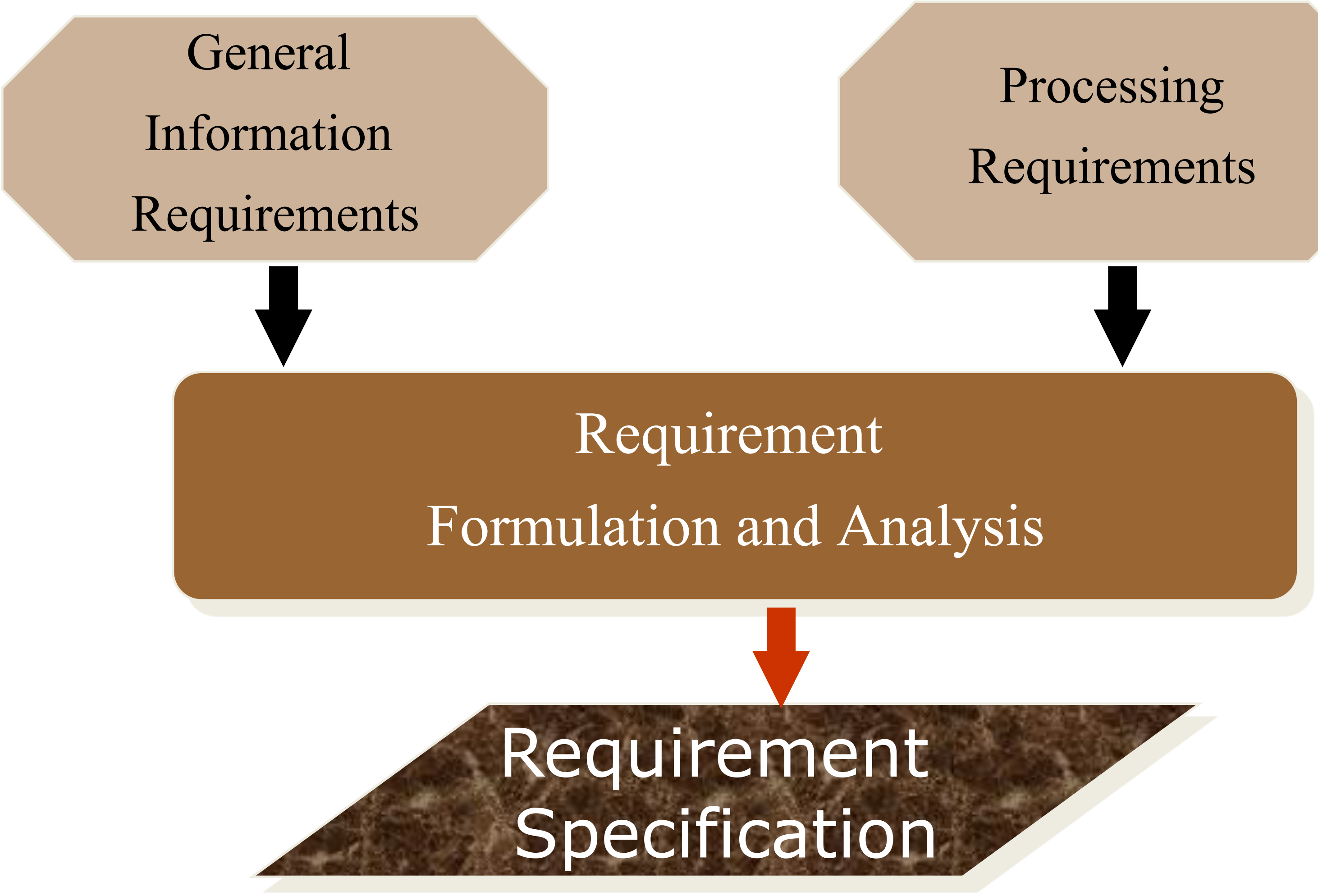
- **View level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

Database Design

The process of designing the general structure of the database:

- Logical Design –
  - Deciding on the database schema.
  - Database design requires that we find a “good” collection of relation schemas.
  - Business decision –
    - What attributes should we record in the database?
  - Computer Science decision –
    - What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database

Database Design Process
Step 1



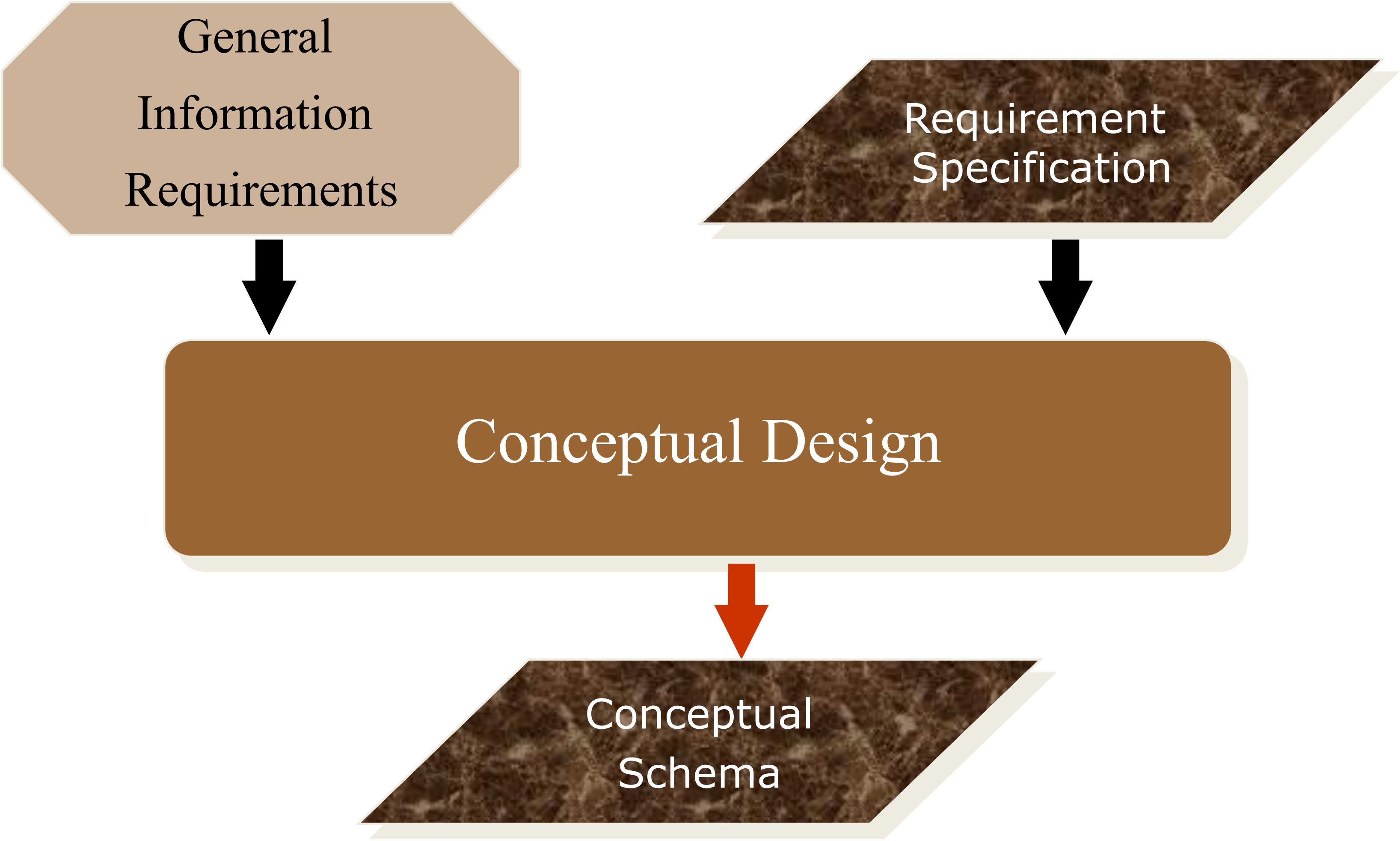
General Information Requirements

Processing Requirements

Requirement Formulation and Analysis

Requirement Specification

Database Design Process
Step 2

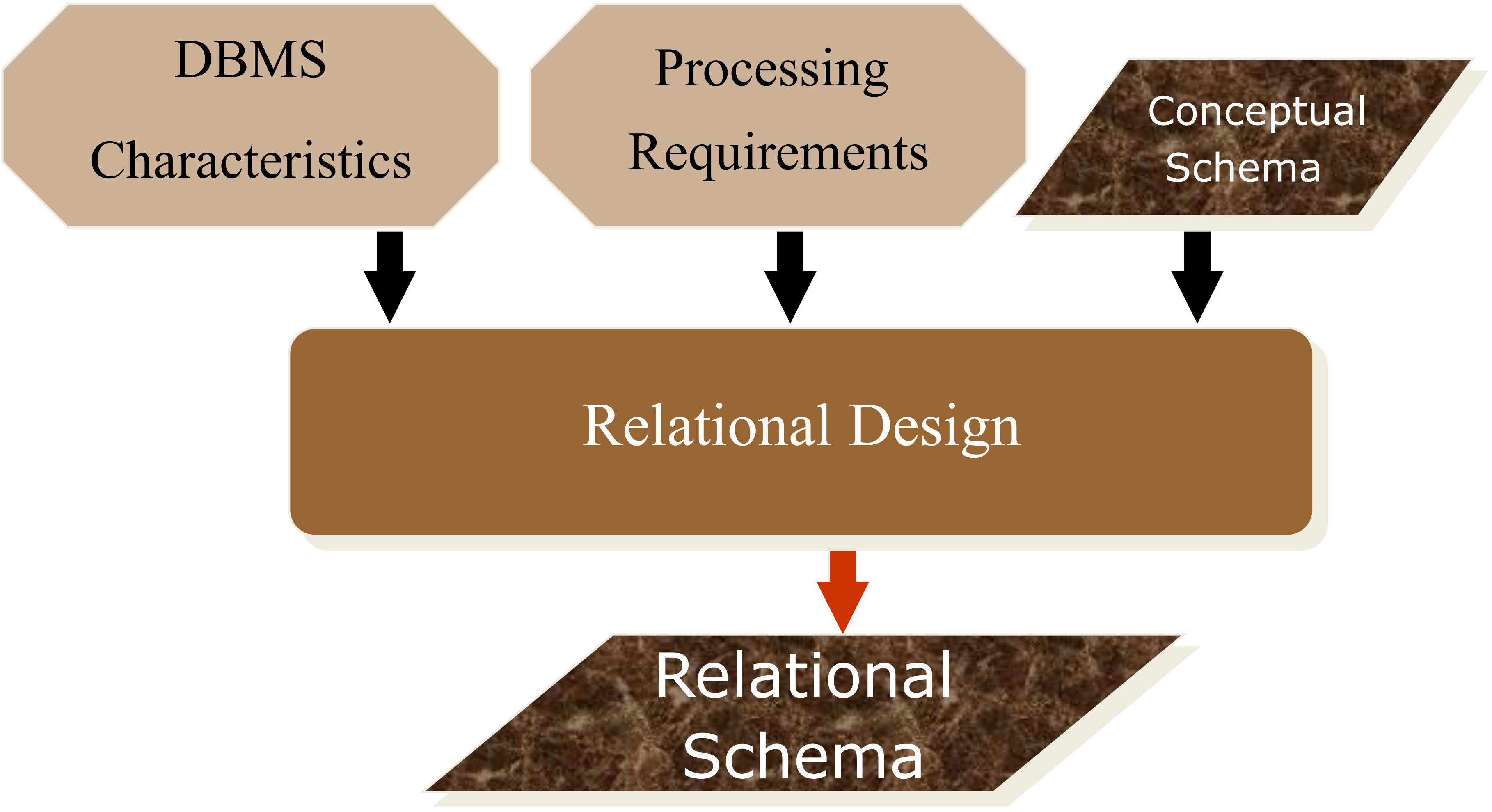

General Information Requirements

Requirement Specification

Conceptual Design

Conceptual Schema

Database Design Process
Step 3

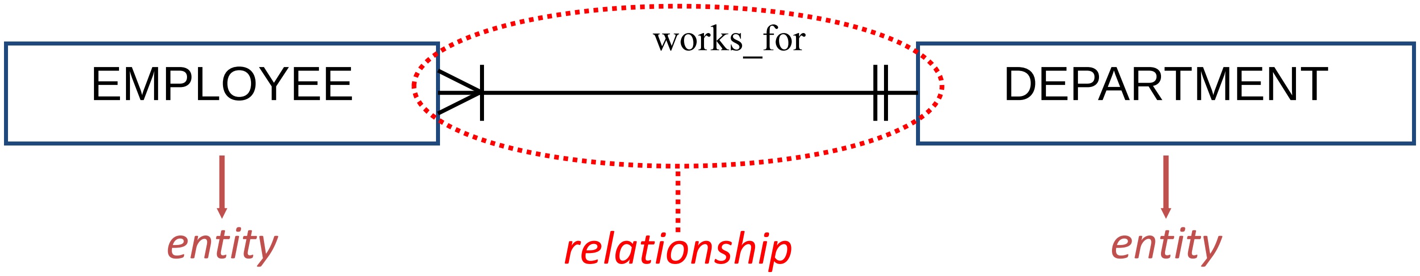


ER Modeling

E-R Modelling

- Entity-Relationship (E-R) Modelling is a conceptual modelling tool.
- perceives the business environment in terms of participating "entities" and the "relationship" between them.

e.g. many employees work for a department.



The E-R Model

- “entity” represents some object of the real-world; “relationship” captures the association between entities (real-world objects).
- E-R model must capture all the business requirements, as well as the natural associations between business objects.
- must be complete and sound, but must be easily readable - to the designer as well as to the naive user.

Building the ER Model

- ♦ the requirements specification is the first step to any design; it captures the ‘*what*’ of the business environment.
- ♦ also documents the “business rules” - i.e., the constraints that will apply to your database.

  e.g. *every department must have a manager;*

  and only one manager.

- ♦ the ER model must capture the participating entities as well as these business rules.

Building the ER Model : Entity

- an “entity” (set) is a data object.
- depicts a *set* of related/similar objects in the real world (not necessarily tangible).
- usually identified by the nouns in the requirements specification.
  e.g. ‘department’, ‘invoice’, ‘vehicle’, etc..
  - not all nouns are entities, nor are all entities identified by nouns.

Building the ER Model : Attributes

- “attributes” are properties of an entity.
- each instance (member) of an entity (set) will have the same attributes (properties), but different attribute *values*.

e.g. “department” entity may have attributes
“dept_id” and “dept_name”.

Every department will have these attributes; one department is differentiated from another by its id (value of “dept_id”) and its name (value of “dept_name”).

Building the ER Model : Relationship

- ♦ represents the real-world association between two or more entities; or even of an entity with itself.
- ♦ represents the association between entity *sets*; not between entity *instances*.
- ♦ are usually omni-directional; “roles” may be used where required, to add meaning.
- ♦ a “role” is a name (usually a noun) of one end of a relationship.

Building the ER Model : Relationship

- a role indicates the function of an individual entity in the relationship.
- relationships are usually identified by the verbs in the requirements specification.

  e.g. “employee works for a department”.

  entities: *“employee”, “department”*
  relationship: *“works_for”*

  - not all verbs are relationships; nor are all relationships identified by verbs.

Building the ER Model : a recap

- the requirements specification is the stepping stone to an ERD.
- typically, the nouns identify the entities, and the verbs identify the relationships.
- entities are defined in terms of its attributes; which serve to capture its real-world properties.
- relationships denote associations, and therefore capture business rules (constraints).

Building the ER Model : an example

ABC Traders is a trading concern that is a distributor for several categories of products, and sells numerous products of each category.

Apart from its fixed retail customers (retail outlets), ABC Traders also have counter sales at their sales depot.

Fixed retail customers are extended an agreed period of credit and to an extent specified. Counter sales are strictly cash sales.

ABC employs several salesmen who are responsible for collecting orders from the customers and to follow-up on that order - delivery as well as payment.

Building the ER Model : example

A given order can be for several products, but will be for only one category of products.

In case of non-availability of an ordered product, no back-order is generated.

Alternatively, a customer can cancel an order he has placed, notwithstanding availability of goods.

No advance is collected along with the order; full payment is expected after invoicing and within the specified credit period.

Part payments are not allowed, but a customer can pay several invoices in one payment. Payment may either be in cash or through cheque or draft.

Building the ER Model : example

Potential Entities:

~~ABC Traders~~
~~trading concern~~
~~distributor~~
categories
products
retail customers
~~sales depot~~
? credit period
salesmen

order
back-order
invoice
availability
~~delivery~~
? payment
advance
cash
cheque
draft

Building the ER Model : example

Attributes: (some sample cases)

- CUSTOMER

  cust_id
  cust_name
  cust_class
  cust_contactperson
  cust_address
  cust_city
  cust_tel1
  cust_tel2
  cust_fax
  cust_email

- SALESMAN

  slman_id
  slman_name
  slman_birthdt
  slman_joindt
  slman_address
  slman_city
  slman_tel
  slman_fuelallowance

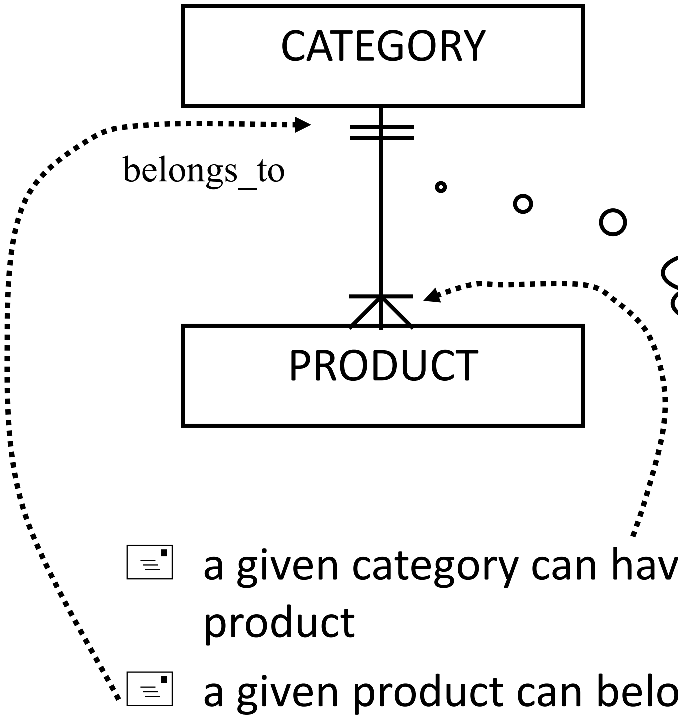
Building the ER Model : example

Relationships: (*a sample case*)

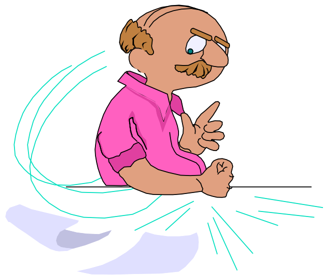
ABC sells many categories of products;
each category has several products

entities: category, product

relationship: product belongs to category



one:many



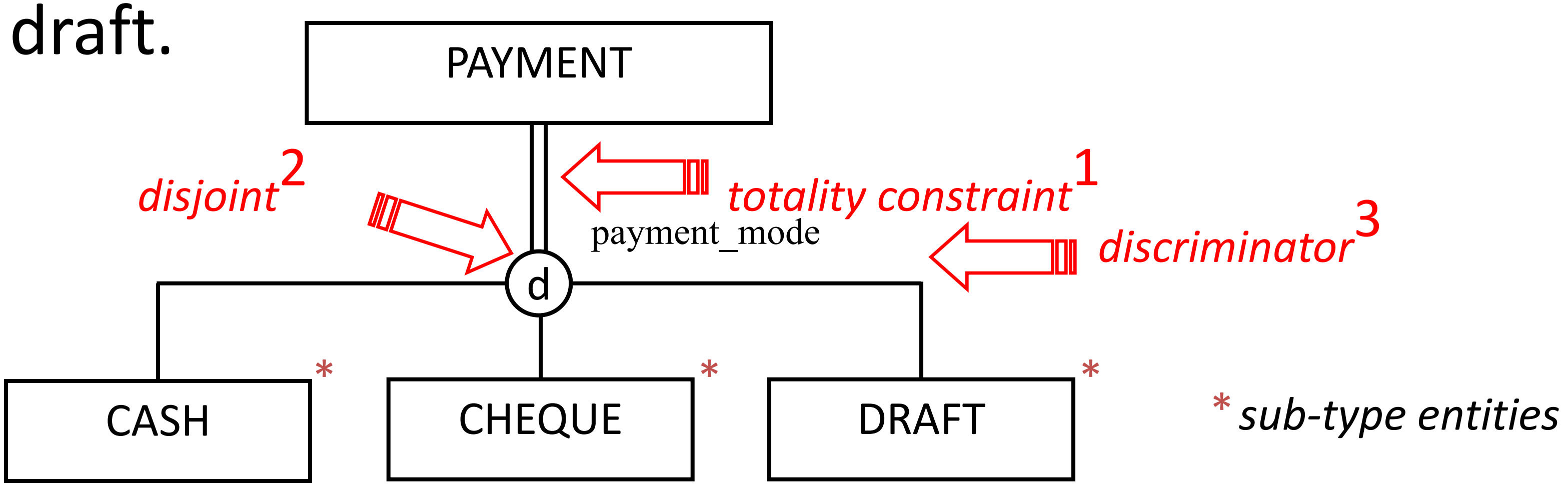
Business Rules !!

- a given category can have many products, but must have at least one product
- a given product can belong to one and only one category.

Building the ER Model : example

Advanced Constructs (*Specialisation*)

Customers pay their invoices in cash, cheque or draft.



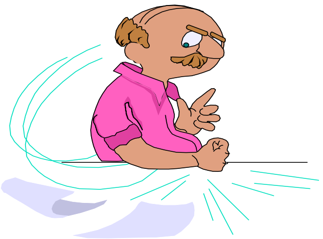
1 every instance of *payment* has to be of *at least* one sub-type

2 an instance of *payment* can be of *exactly* one sub-type

3 the criteria on which an instance of *payment* is classified into one of the sub-types.

Building the ER Model : example

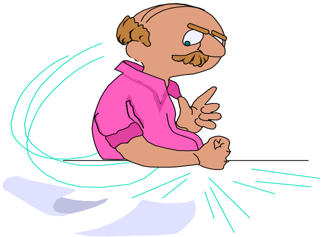
Invoices are delivered to customers: ⇐ “delivery” : attribute or entity ??



Entity !!

- acknowledge delivery
- record delivery challan number
- record delivered items
- record mode of delivery

- customer signs duplicate invoice; only store whether invoice is delivered or not
- a given invoice is delivered at most, once.
- delivered items are same as invoice items

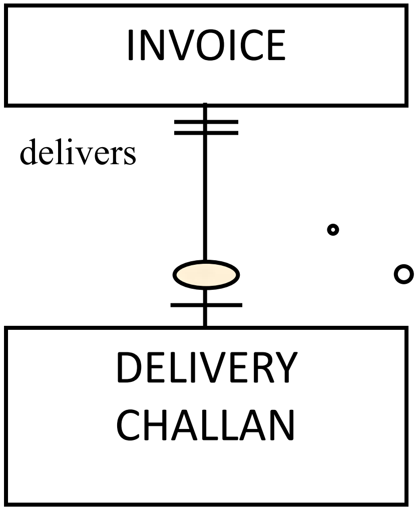


Attribute !!

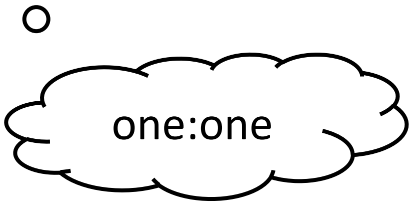
Building the ER Model : example

ER modelling : attribute or entity ??

Invoices are delivered to customers.



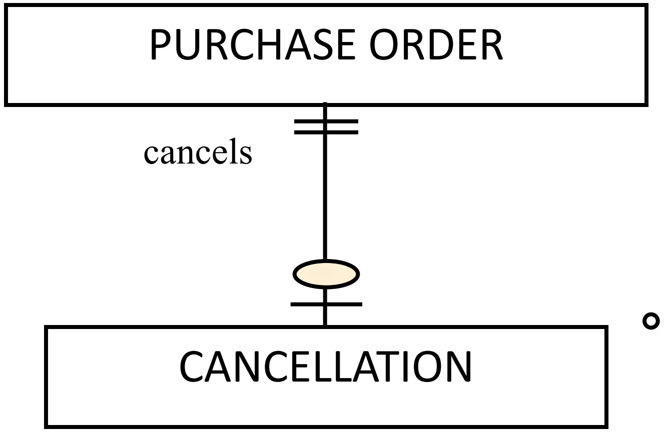
- a delivery challan delivers one and only one invoice
- a given invoice is delivered entirely or not delivered at all.



Generally, in the case one:one relationships, one entity can be “collapsed” into the other entity and modelled as attributes of the other entity.

Building the ER Model : example

Alternatively, a customer can cancel an order......



PURCHASE ORDER

cancels

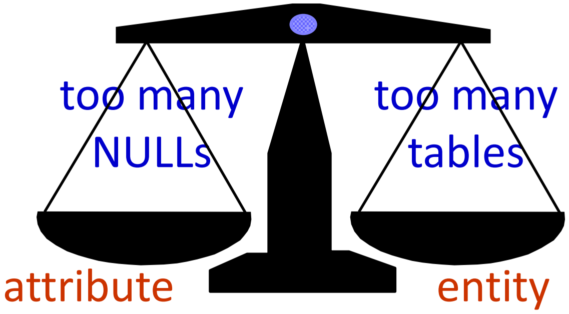
CANCELLATION

......nor are all entities identified by nouns.

“*cancellation*”, “delivery” : Attribute or entity ?

- Business Rules
  - ⇒ what is the usual practice ?
  - ⇒ what is the data ?
  - ⇒ how much data ?
  - ⇒ retrieve how often ?

- Performance tuning issues



Entity : Properties

An entity must exhibit the following properties:

- lies within the scope of the business world being modelled
- represents a set of similar objects *about which the enterprise needs to store information*.
- provides the ability to distinguish between various instances of the entity
- satisfies the rules of “normalization”.

Entity : Types

Entities may be categorised on the basis of (common) characteristics into:

- ♦ fundamental/strong entity

  an entity that is capable of its “own existence” - i.e. an entity whose instances exist notwithstanding the existence of other entities.

- ♦ weak entities
- ♦ associative entities
- ♦ sub-type entities

Attributes

♦ characteristics/properties of an entity, that provide descriptive details of it.

♦ an entity can thus be thought of, as an ordered set of attributes.

♦ every *instance* of an entity is then, merely an ordered set of attribute *values*.

♦ each attribute is associated with a set of possible values; this set is called a "*domain*".

Attributes : Identifier

- an attribute that takes unique values, such that it can uniquely *identify* an entity instance. *e.g. “dept_id” in “department” entity*

- an identifier is also known as a “key”.

- need not comprise only one attribute, can have two or more attributes (*composite key*)

- no attribute can identify more than one entity except in a “*specialisation hierarchy*”.

Attributes : values

- the rules of “normalisation” dictate that every attribute shall take atomic values; multi-valued attributes are generally not allowed.
- multi-valued attributes suggest an entity by itself.
- further constraints can be imposed on attribute values (apart from the domain)

  e.g. “not null” *prevents an attribute from taking nulls.*

Relationship

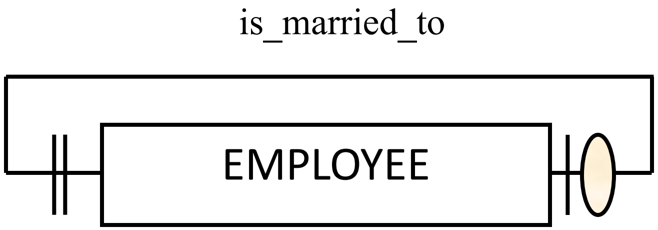
- ♦ models the real-world association between two or more entities (*binary, n-ary relationship*).
- ♦ may also model the association of an entity with itself (*recursive* relationship).
- ♦ must have a name that is unique across the entire model.
- ♦ must wherever possible, be supported by a well-documented description.

Relationship : Properties

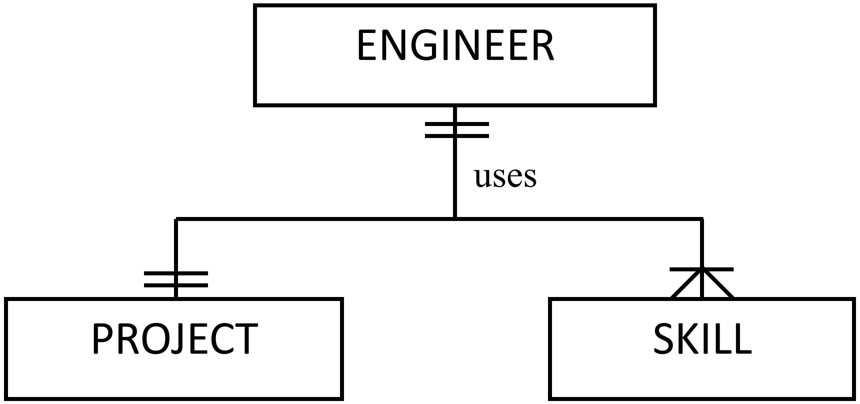
- ♦ described in terms of "*degree*", "*connectivity*", and "*existence*".
- ♦ translates into "*migration of the key*" as follows:
  - ⇒ in the case of one:one relationships, from either entity to the other.
  - ⇒ in the case of one:many relationships, from the entity at the "one" end to that at the "many" end.
  - ⇒ in the case of many:many relationships, from both entities "into the relationship"

Relationship : Degree

- "degree" describes the number of entities involved in the relationship.
- typically 2 (binary); other common degrees are 1 (recursive) and 3 (ternary).



Recursive



Ternary

Relationship : Connectivity

- “connectivity” indicates the entity *occurrences* (instances) participating in a relationship.
- takes values “one” or “many”.

  e.g. a one:many *relationship indicates that for every occurrence of one entity, there are many related instances of the other entity.*
- an actual count of this connectivity (instead of “one” & “many”) if specified, is called the *cardinality* of the relationship.

Relationship : Existence

- “existence” defines whether the relationship is optional or mandatory.
- determined by business rules.
- existence implicitly defines the minimum connectivity of a relationship (‘0’ if optional, ‘1’ if mandatory).

  e.g. a project has to be managed by one employee, but not every employee manages a project.

Relationship : other issues

- a relationship name is usually read from left (entity) to right (entity) in an ER diagram; or from top to bottom*.

  *e.g. entities are “*customer*” and “*order*”.*
  *relationship is “*places*”.*

  *In the ERD, place “*customer*” to the left of “*order*”.*

  The construct will then be read as
  *“*customer *places* order*”.*

** applicable only to crow’s-foot and IDEF1X notation*

Entity types : Weak

- an entity that is *not* capable of “its own existence”.
- characterised by the need to have at least one external identifier (of another entity) as part of its own identifier.

*e.g. consider “*payment*” and “*pmt_items*”*

*“*pmt_items*” cannot exist without a corresponding*

*“*payment*” instance. “pmt_id” of “*payment*” will be part of the identifier of “*pmt_items*”*

Entity types : Associative

- a relationship translates into migration of a key - many:many relationship implies the keys migrating many times, *both ways*.
- such migration leads to redundancy and many:many relationships must therefore be resolved.
- “Associative entity” is an entity that is used to resolve a many:many relationship.

Entity types : Associative

- also called “*intersecting entity*”.
- identifier of an associative entity is a composite of the identifiers of the entities involved in the many:many relationship.
- associative entities by their very definition, are also weak entities.
- must be appropriately named

Entity types : Sub-type

- part of a specialisation hierarchy.
- categorise a subset of the occurrences of the parent entity.
- has (inherits) all the attributes of the parent entity, but also has additional attributes that are specific to that subset of occurrences.
- every entity must be associated with at least one super-entity.

Entity types : Sub-type

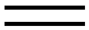
- specialisation can either be *total* (every instance of super-type must belong to at least one sub-type) or can be *partial*.
- specialisation can either be *disjoint* (every super-type instance can belong to at most, one sub-type) or can be *overlapping*.

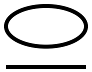
The Crow's-Foot Notation

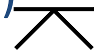
Entity

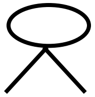
ENTITY_NAME

Connectivity & Existence

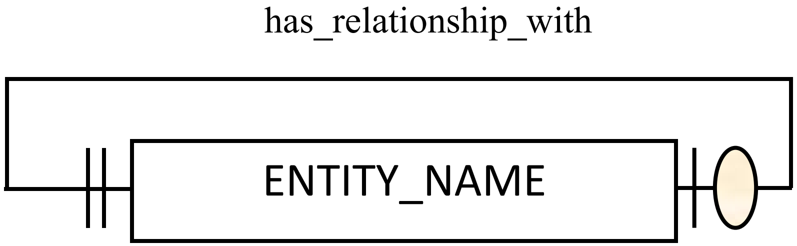
one, mandatory
(*one and only one*) 

one, optional
(*zero or one*) 

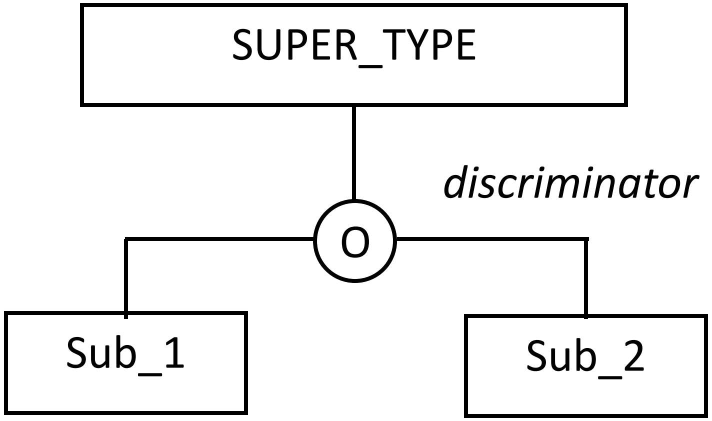
many, mandatory
(*many, but at least one*) 

many, optional
(*zero, one or more*) 

Recursive Relationship

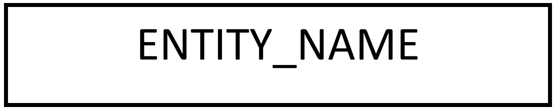


Specialisation Hierarchy



The Chen Notation

Entity

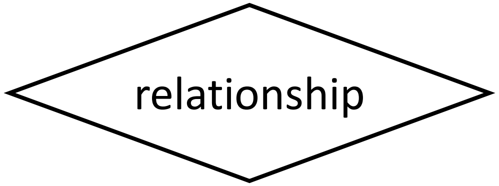


Cardinality

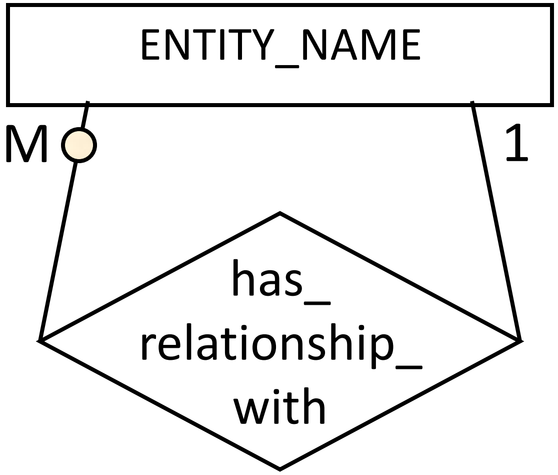
⇒ ‘M’ and/or ‘N’

⇒ ‘1’

Relationship



Recursive Relationship

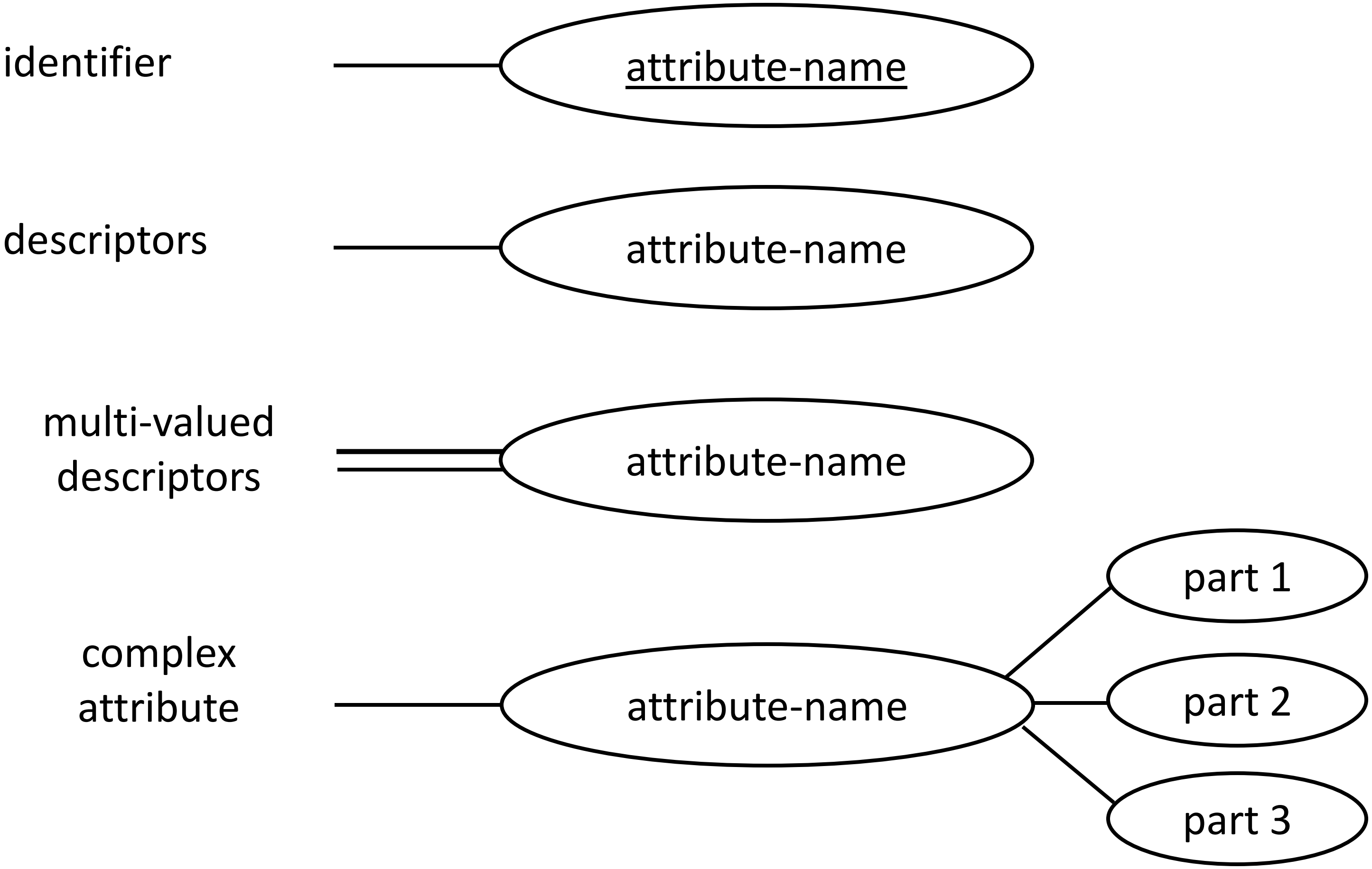


Existence

optionality ○

The Chen Notation : Attributes

Attributes



identifier — attribute-name

descriptors — attribute-name

multi-valued descriptors — attribute-name

complex attribute — attribute-name

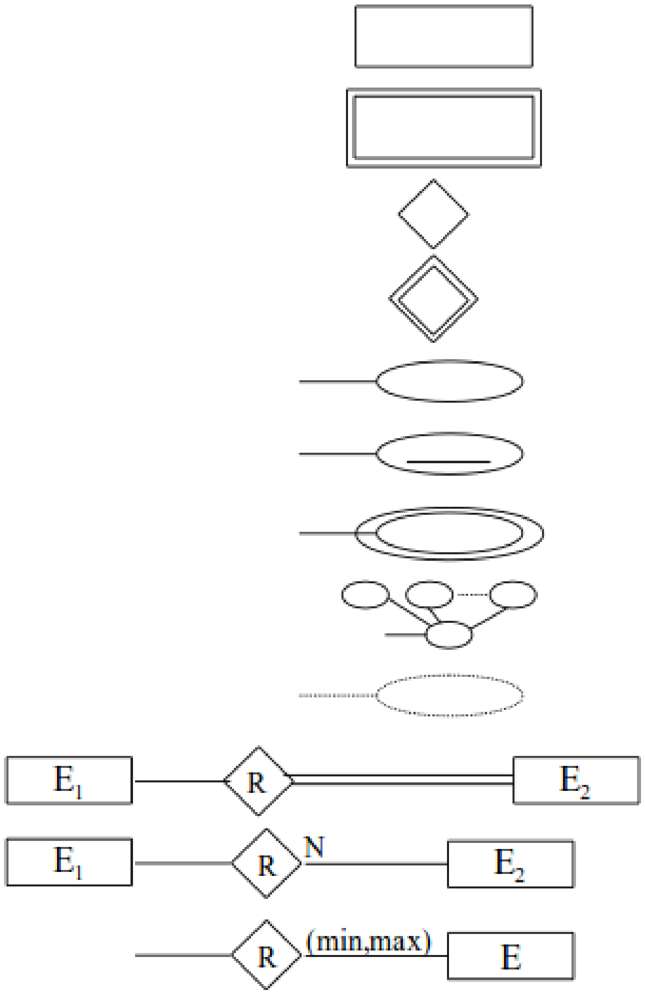
part 1

part 2

part 3

SUMMARY OF ER-DIAGRAM NOTATION FOR ER SCHEMAS

Symbol



Meaning

ENTITY TYPE

WEAK ENTITY TYPE

RELATIONSHIP TYPE

IDENTIFYING RELATIONSHIP TYPE

ATTRIBUTE

KEY ATTRIBUTE

MULTIVALUED ATTRIBUTE

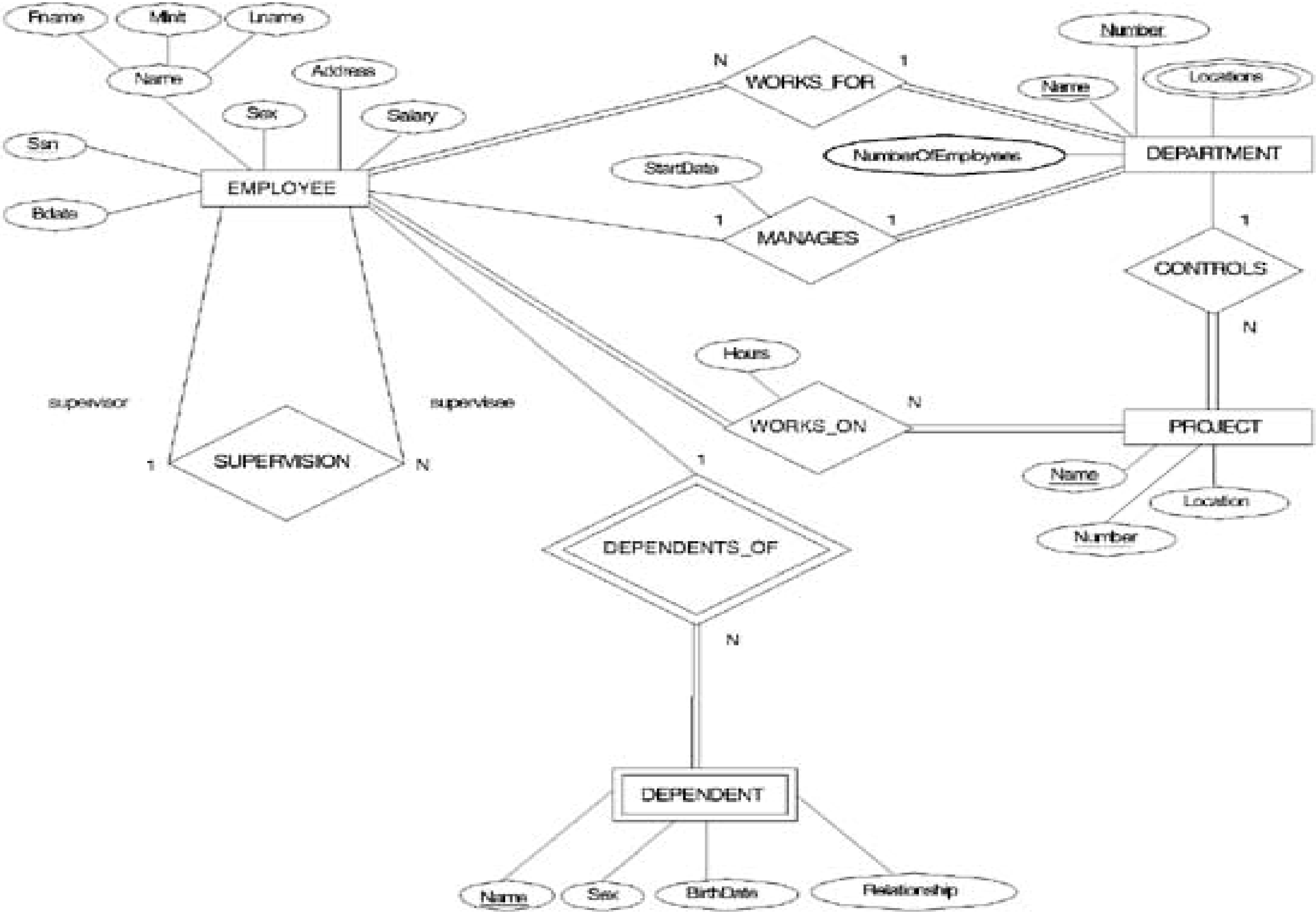
COMPOSITE ATTRIBUTE

DERIVED ATTRIBUTE

TOTAL PARTICIPATION OF E_2 IN R

CARDINALITY RATIO 1:N FOR E_1:E_2 IN R

STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R


Fname
MInit
Lname
Name
Address
Sex
Salary
Ssn
Bdate
EMPLOYEE
N
WORKS_FOR
1
Number
Name
Locations
DEPARTMENT
NumberOfEmployees
StartDate
1
MANAGES
1
1
CONTROLS
N
Hours
WORKS_ON
N
PROJECT
Name
Location
Number
supevisor
supervisee
1
SUPERVISION
N
1
DEPENDENTS_OF
N
DEPENDENT
Name
Sex
BirthDate
Relationship

ER to Relational Mapping

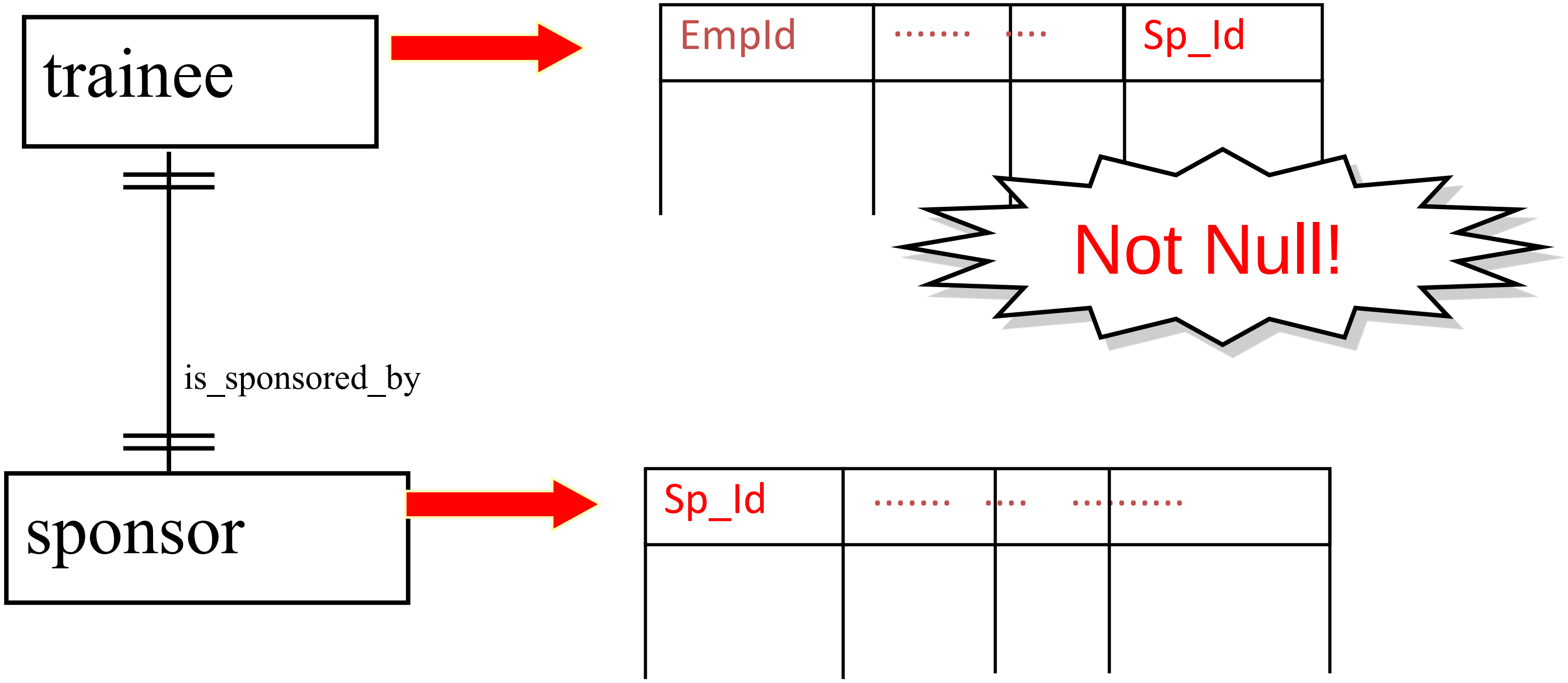
Translating the E-R Model

- E-R model captures the database at the conceptual level - translating the model into the relational model (or any other model) yields the actual database.

- a Data-Definition Language (DDL) is used to set up the actual database schema.

One-to-One Mandatory

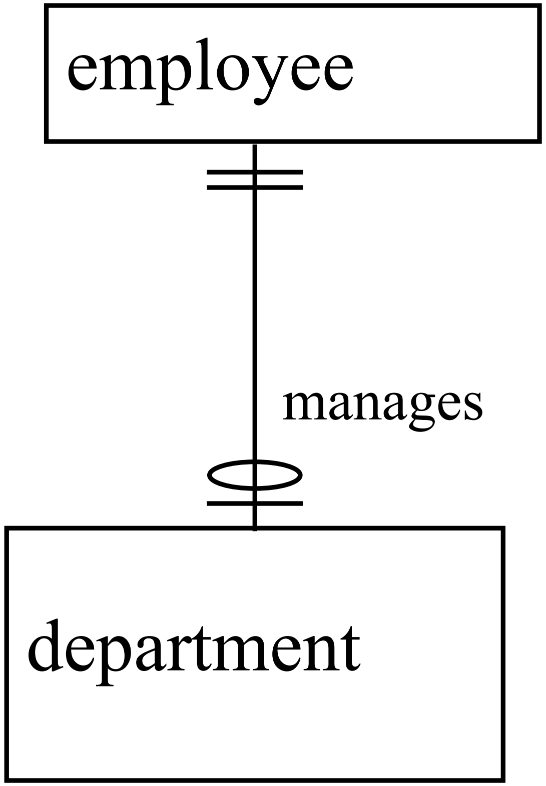
Every trainee has one and only one sponsor and every sponsor sponsors one and only one trainee.



- In a one:one relationship, one entity can usually be modelled as an attribute of the other entity.
  - For example, the entire 'sponsor' entity could be attributes of the 'trainee' entity (or vice-versa).
- When one of these two entities ALONE has an association with some other entity, then the two entities may be modelled separately.

One-to-One Optional

Every dept has one and only one manager; an employee can be the manager of at most one dept.



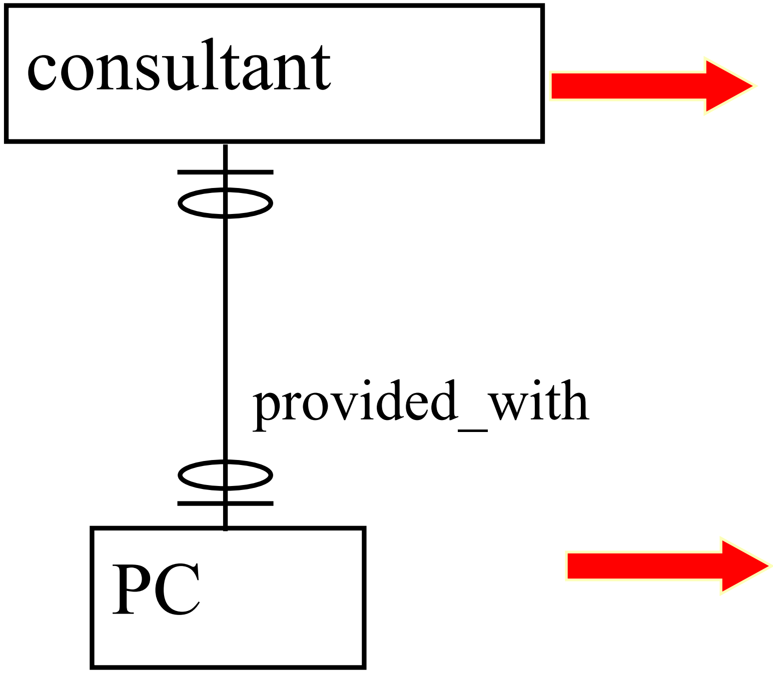
Emp_Id	....	.....	....

DeptId	....	....	Emp_Id

Not Null!

One-to-One Optional

Some of the consultants are provided PCs at their desks.



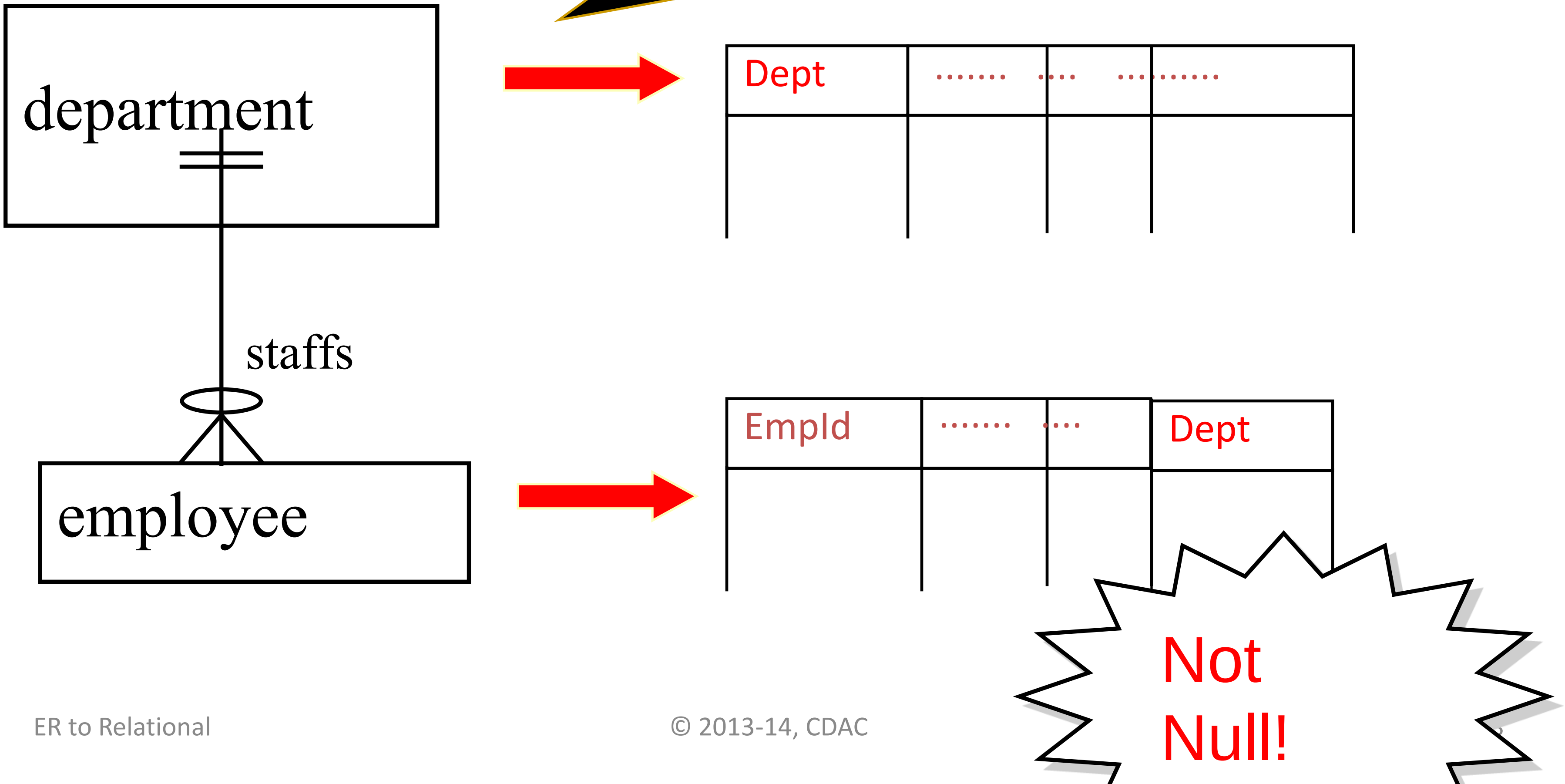
EmpId	…….	….	PC_Id

Null Allowed!

PC_Id	…….	….	……….

One-to-Many Optional

Every employee is alloted a department. A department however, staffs none, one or many employees.



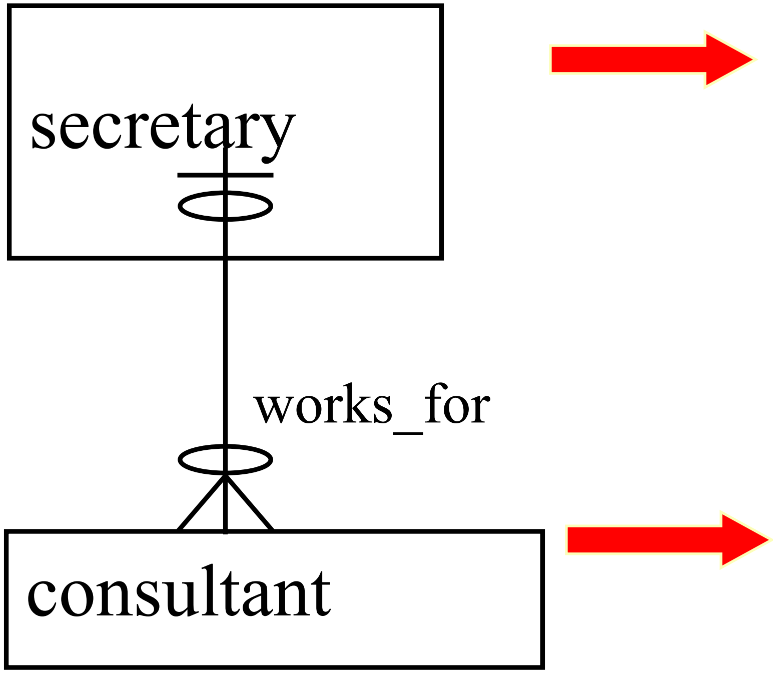
Dept	…….	….	……….

EmpId	…….	….	Dept

Not Null!

One-to-Many Optional

A consultant can have at most one secretary. A secretary may work for several consultants.



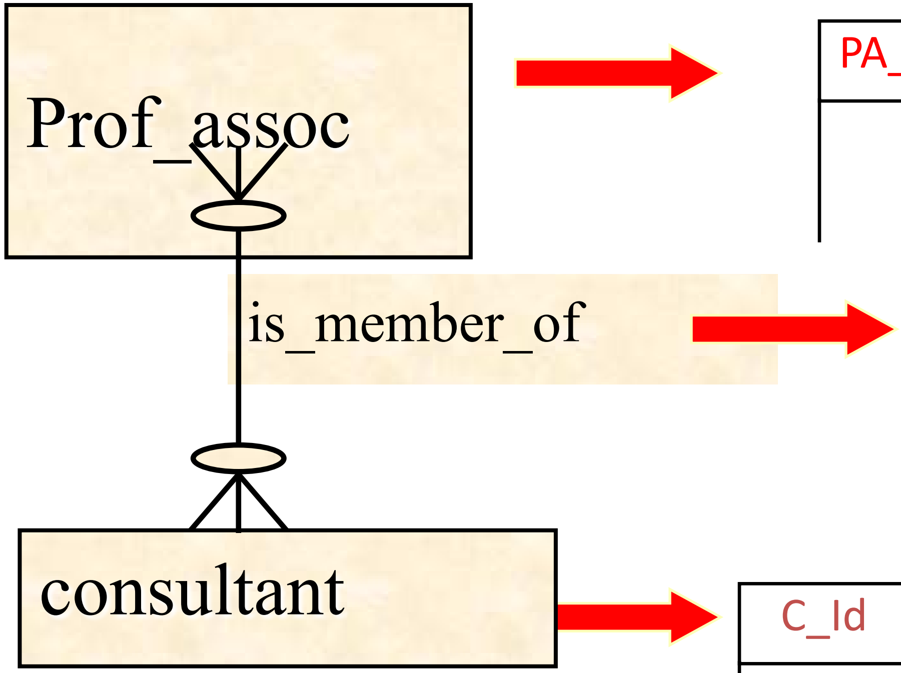
Sec_Id	…….	….	……….

C_Id	…….	….	Sec_Id

Null Allowed!

Many-to-Many Optional

A professional association may have none, one or more consultants as members. Similarly, a consultant may be a member of none, one or many professional associations.



PA_Id	....	....	....

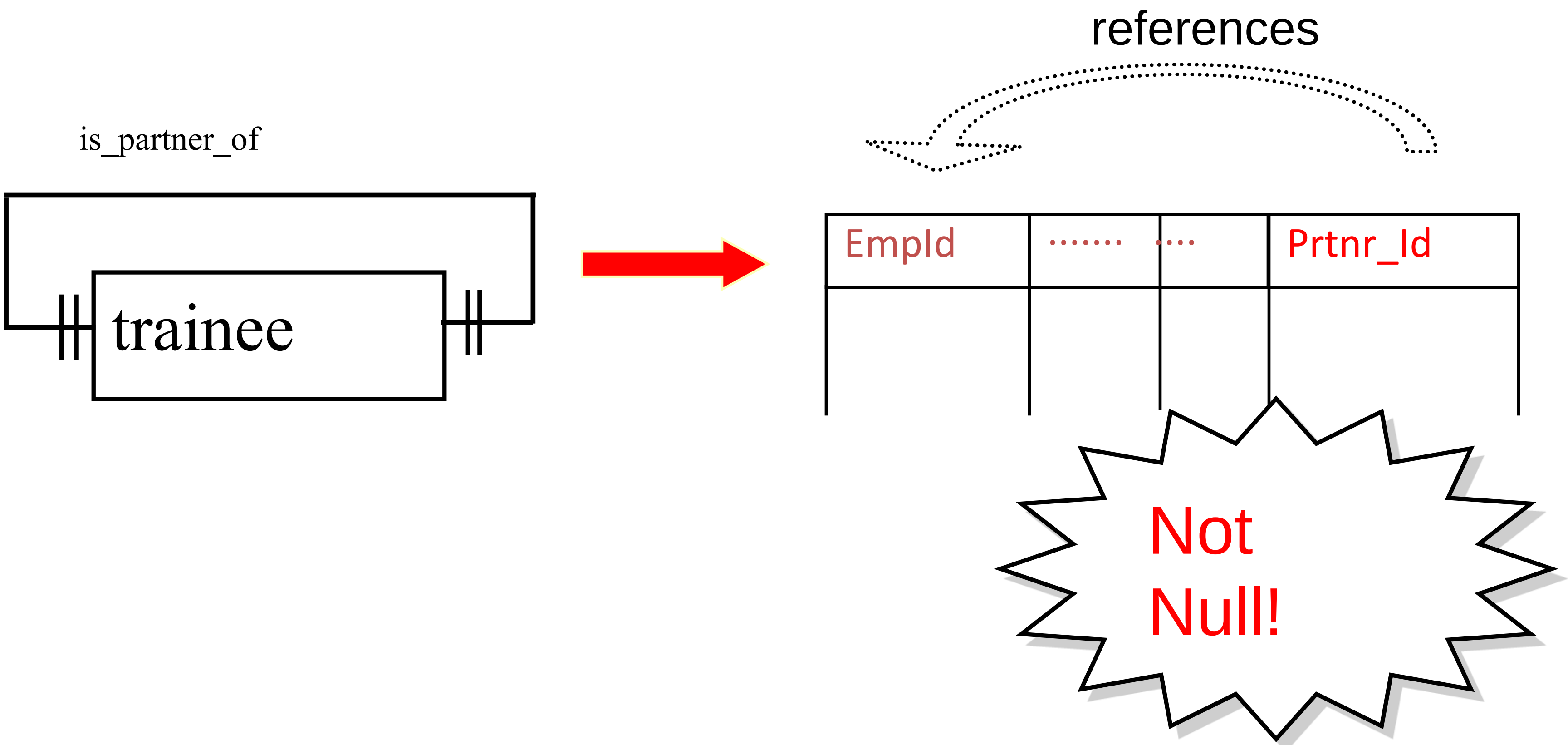
is_member_of

PA_Id	C_Id	....	....

C_Id	....	....

Recursive One-to-One Mandatory

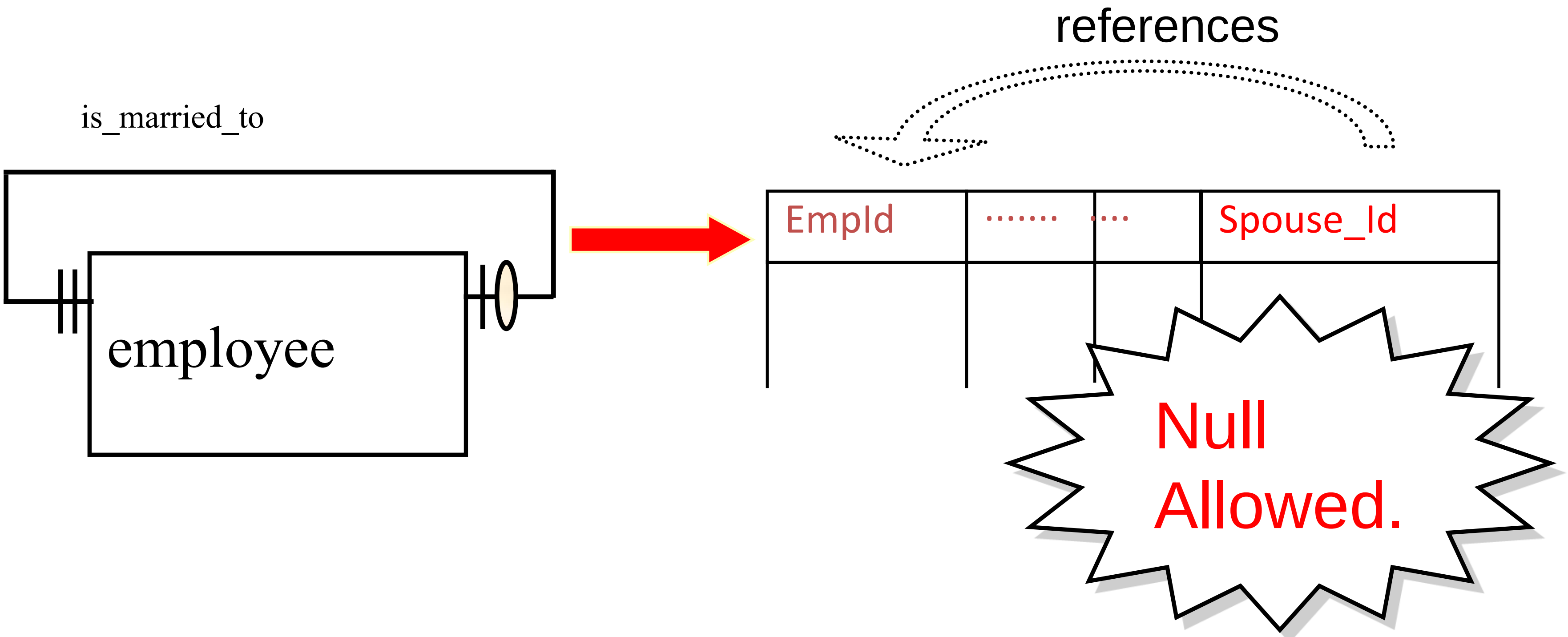
Every trainee has another trainee as a partner.



EmpId	....	....	Prtnr_Id

Recursive One-to-One Optional

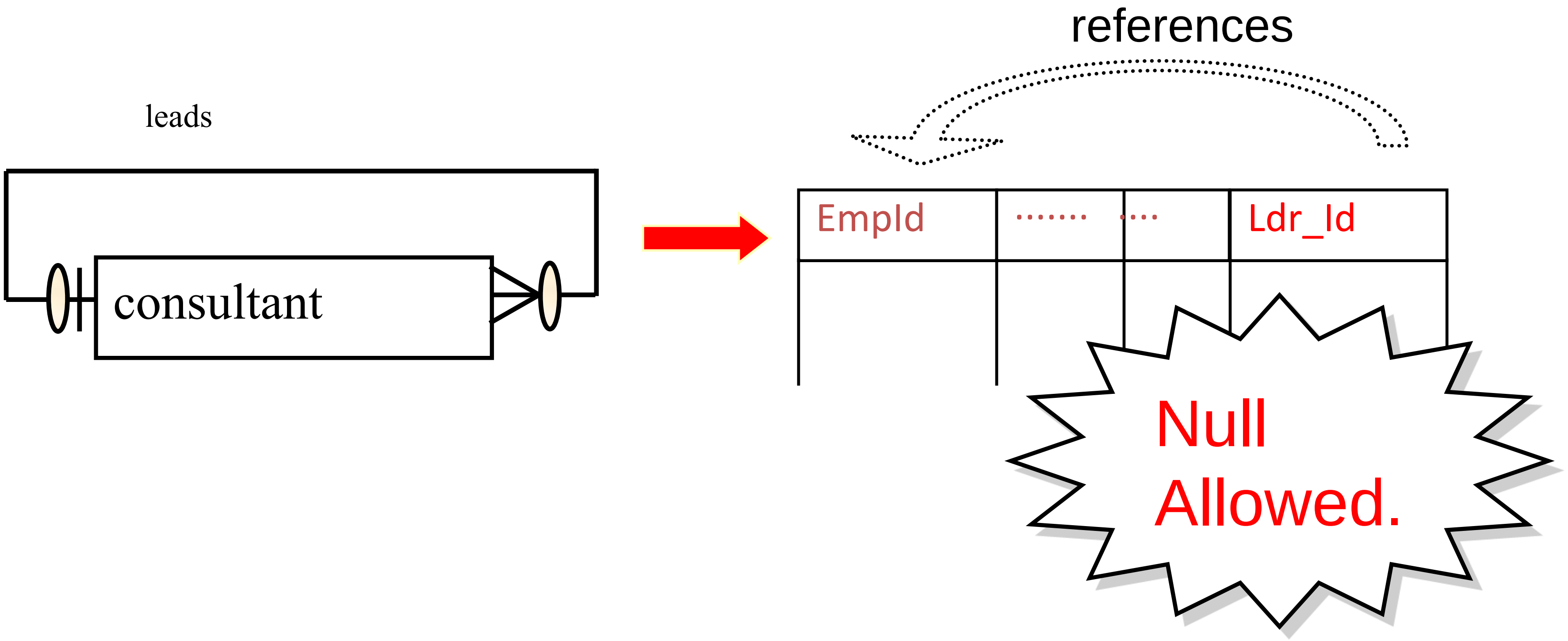
An employee may have another employee as a spouse.



EmpId	....	....	Spouse_Id

Recursive One-to-Many Optional

Consultants are divided into groups with each group having a leader.

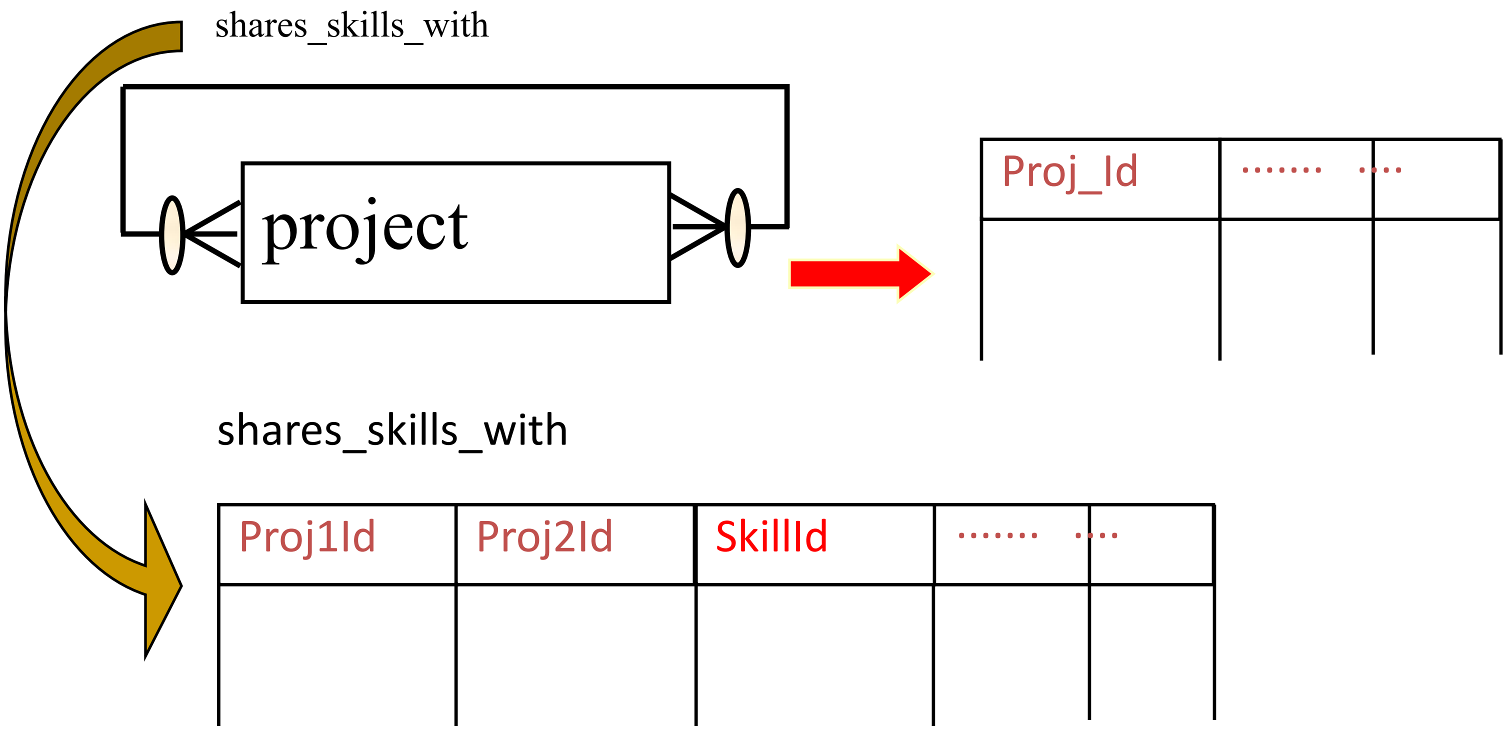


EmpId	....	...	Ldr_Id

Null Allowed.

Recursive Many-to-Many Optional

Several projects employ similar skill sets.
i.e., projects share skills.



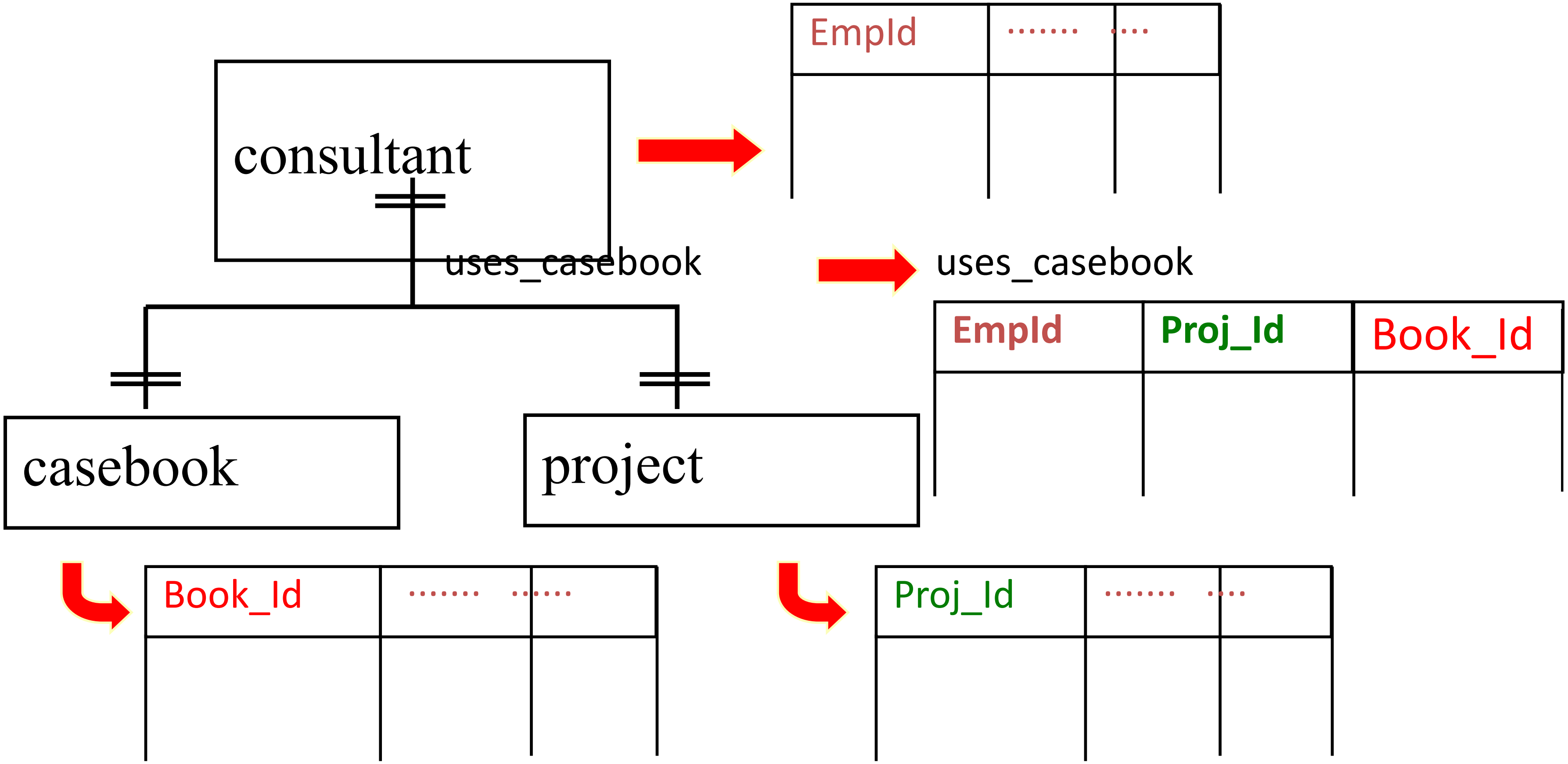
Proj_Id	....	....

shares_skills_with

Proj1Id	Proj2Id	SkillId	....	....

Ternary One-One-One Mandatory

Consultants use casebooks for projects. Each consultant uses a different casebook for different projects.



EmpId	....	....

uses_casebook

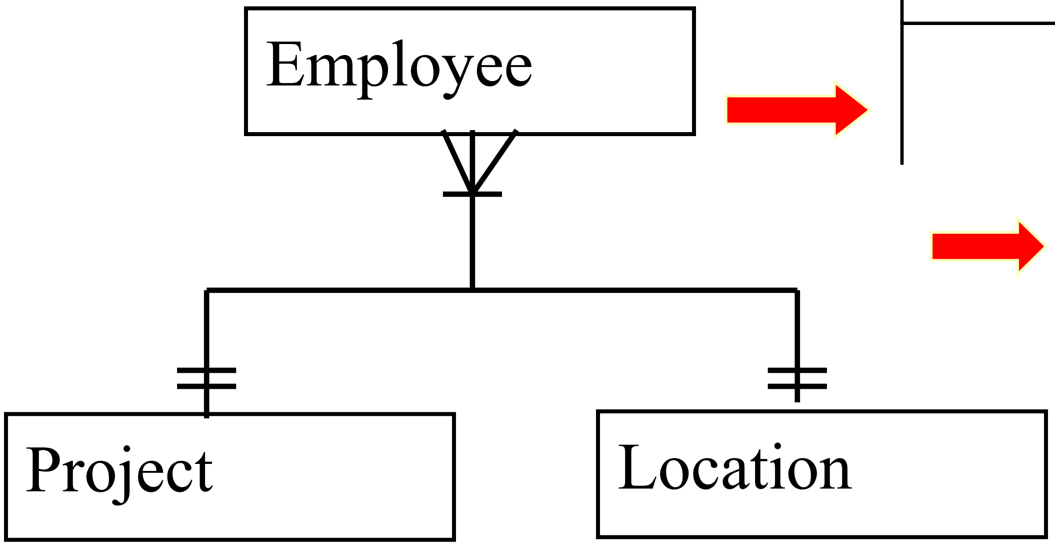
EmpId	**Proj_Id**	Book_Id

Book_Id	....	......

Proj_Id	....	....

Ternary Many-One-One Mandatory

Each employee assigned to a project works at only one location, but can be at a different location for a different project.



EmpId	....	....

assigned_to

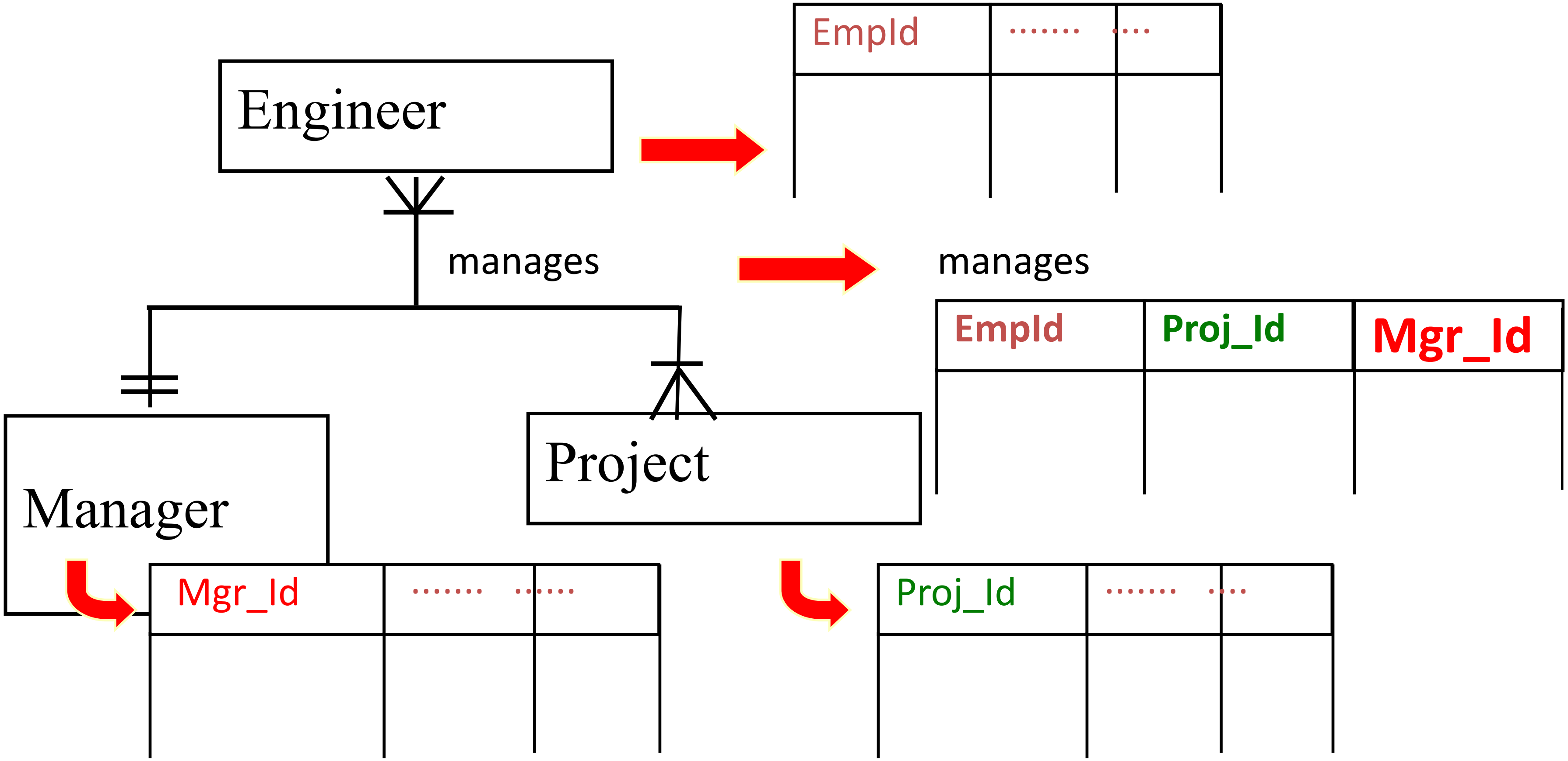
EmpId	Proj_Id	Loc_Id

Proj_Id	....	......

Loc_Id	....	....

Ternary Many-Many-One Mandatory

Each engineer working on a particular project has exactly one manager, but a project may have many managers and many projects.



EmpId	....	....

manages

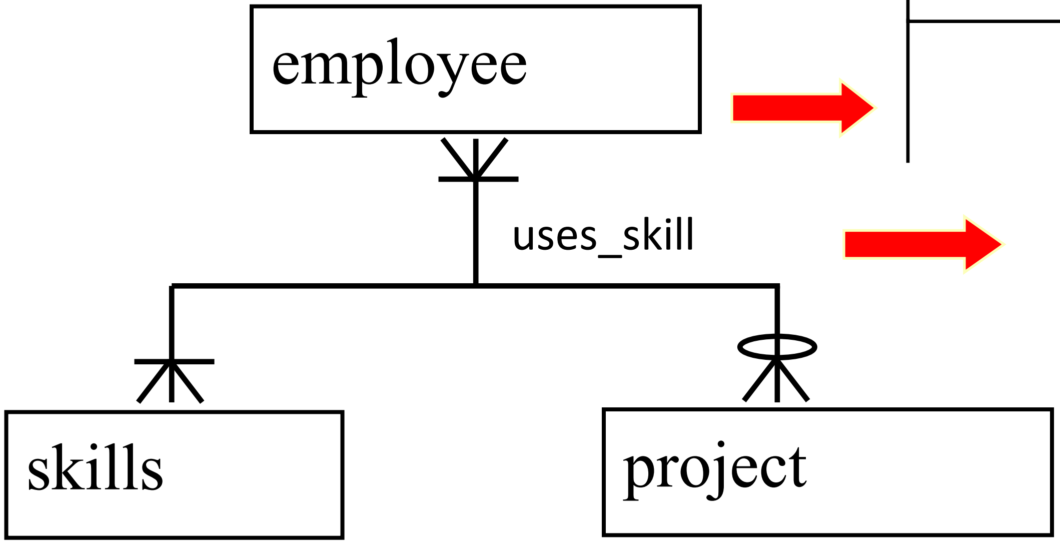
EmpId	Proj_Id	Mgr_Id

Mgr_Id	....	......

Proj_Id	....	....

Ternary Many-Many-Many Optional

Employees use a wide range of different skills for each project.



EmpId	....	....

uses_skill

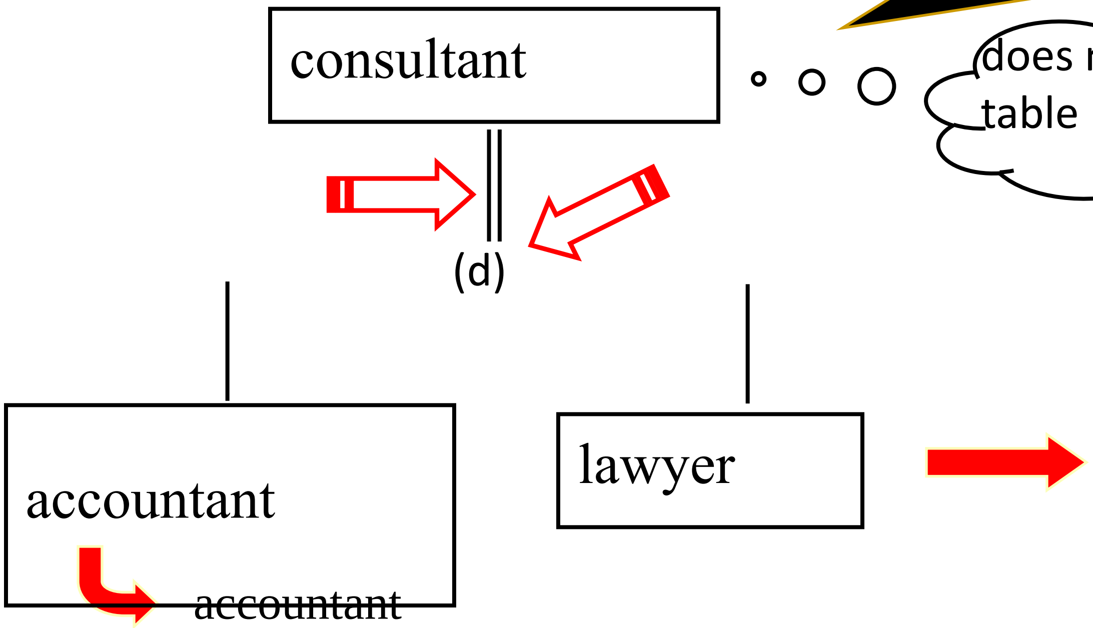
EmpId	Proj_Id	**Skill_Id**

Skill_Id	....	......

Proj_Id	....	....

Specialisation Hierarchies 1

A consultant is either an accountant or a lawyer but not none and not both.



does not map to a table

lawyer

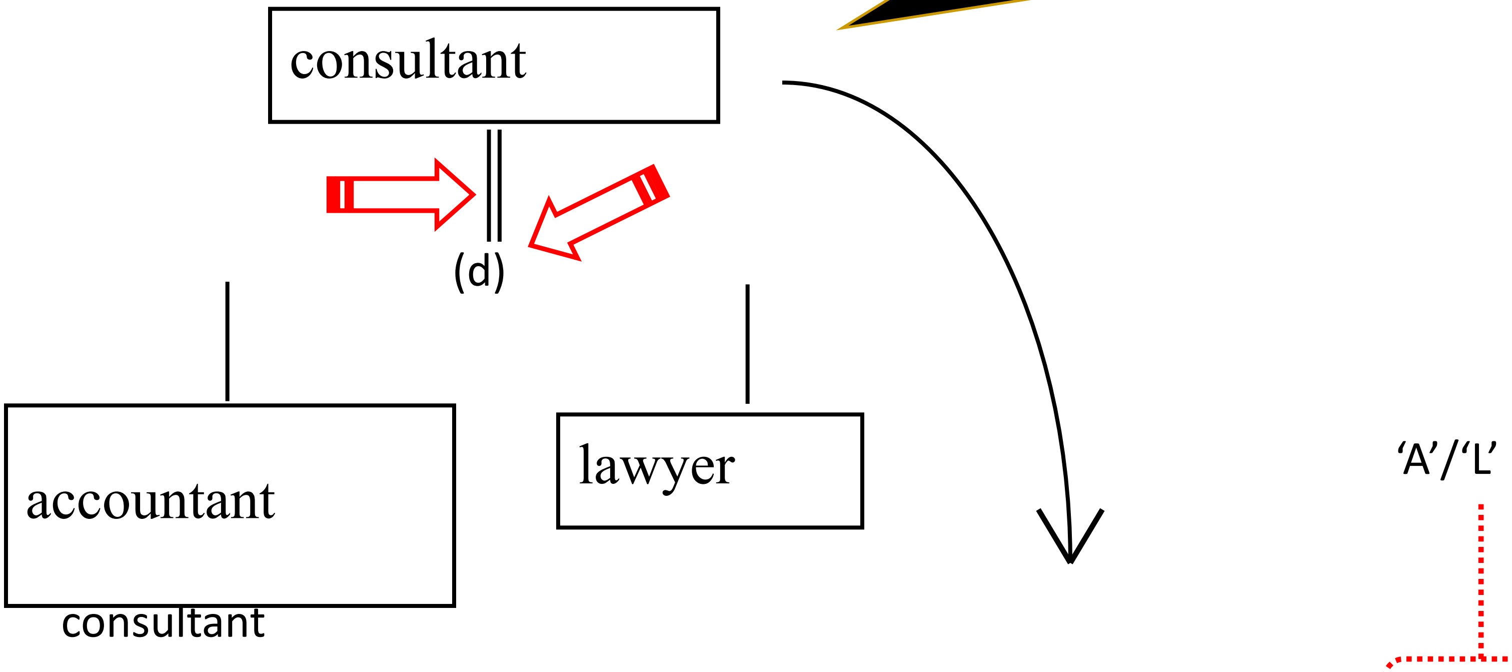
Emp_Id	....	.....

accountant

Emp_Id	....	......

Specialisation Hierarchies 2

A consultant is either an accountant or a lawyer but not none and not both.



consultant

Emp_Id	....	......	acct	....	...	lwyr	....	...	Type

Introduction to Relational Model

Session: 2

Example of a RELATION

Consider the relation EMPLOYEE represented by the following table:

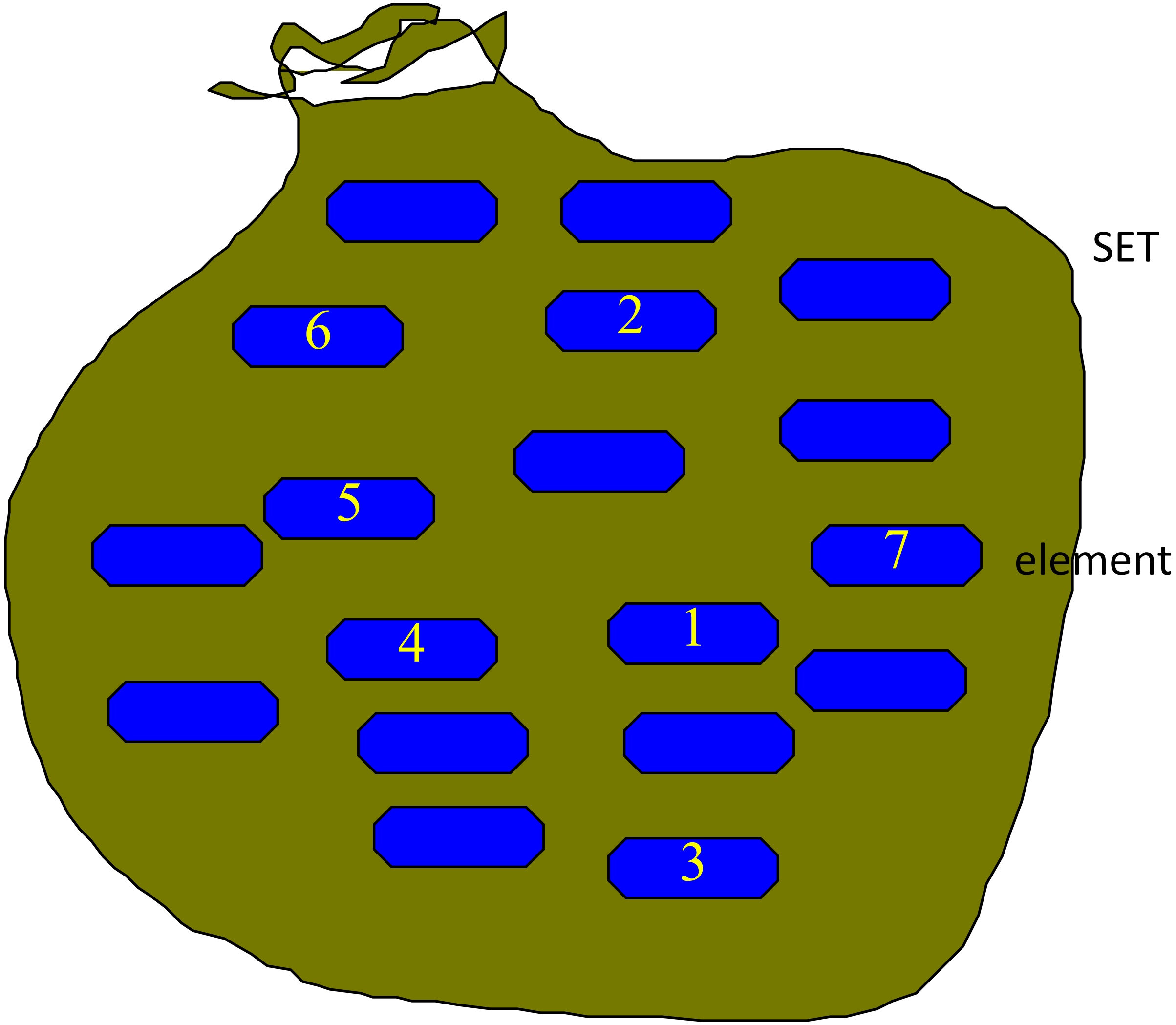
Emp Code	Name	Desig Code	Grade	Join Date	Basic Salary	Sex	Dept Code

Tuples of a RELATION

Each row here is a tuple.

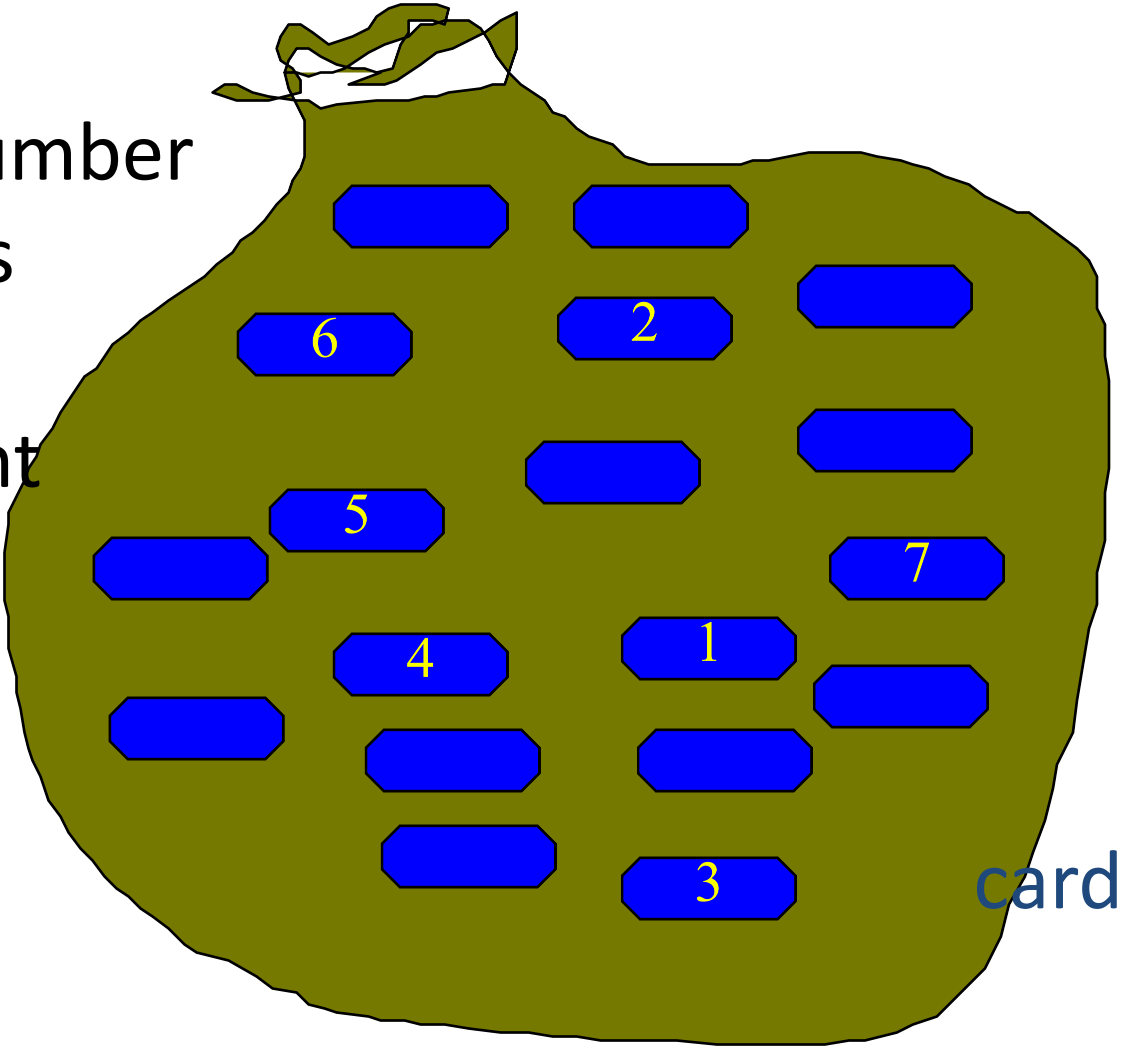
Emp Code	Name	Desig Code	Grade	Join Date	Basic Salary	Sex	Dept Code	
								1
								2
								3
								4
								5
								6
								7
								8

RELATION is a set of Tuples



Cardinality of a RELATION

is the number
of tuples
in it,
at a point
in time



cardinality = 18

Arity (Degree) of a Relation

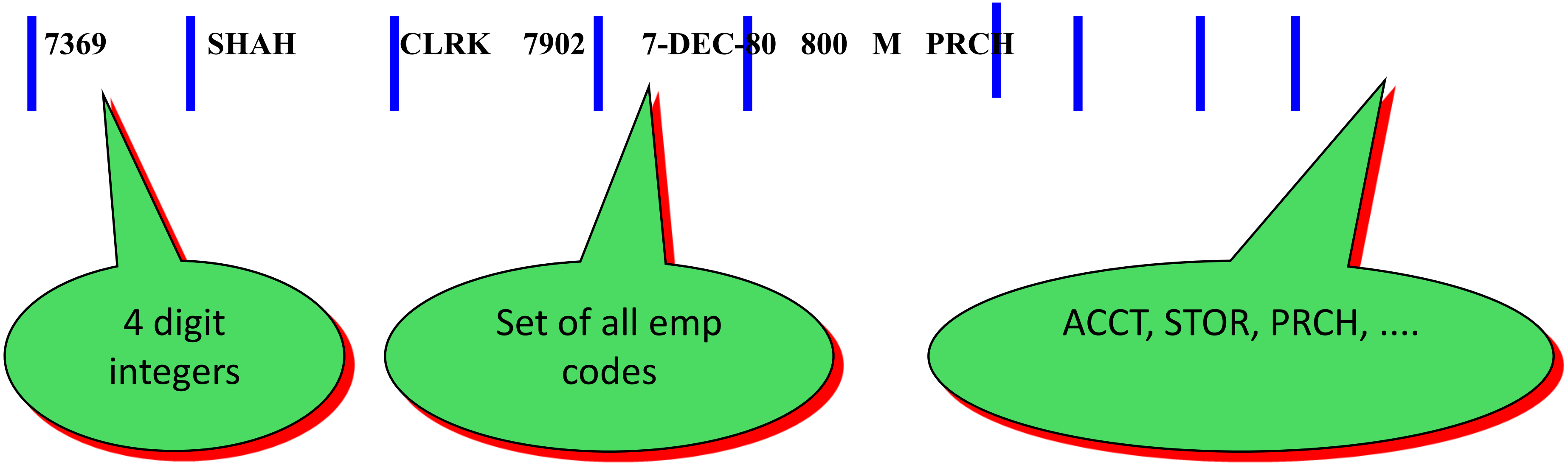
is the number of attributes in it.

7369	SHAH	CLRK	7902	17-DEC-80	800	M	PRCH

arity = 8

Domains

Each attribute has a domain associated with it. Attribute values in a relation are restricted to the values from its domain.



7369 SHAH CLRK 7902 7-DEC-80 800 M PRCH

4 digit integers

Set of all emp codes

ACCT, STOR, PRCH, ...

Consider the relation EMPLOYEE defined as:

```
create table EMPLOYEE{
        EmpCode     integer(4),
        Name                char(30),
        DesigCode   char(4),
        Grade       integer(4),
        JoinDate            date,
        Basic       money,
        Sex                 char(1),
        DeptCode            char(4) }
```

Domains of attributes of Employee

EmpCode	set of all 4-digit numbers
Name	set of all 30-alpha characters
DesigCode	set of all designation codes
Grade	set of all grade values
JoinDate	set of all dates (in a given range)
Basic	set of all possible values for basic
Sex	set {'M','F'}
DeptCode	set of all dept codes

A relation may be *represented* as a table where

Relation	Table
Tuple	Row
Attribute	Column
Arity	Number of columns
Primary key	Unique identifier
Domain	Pool of acceptable values
Cardinality	Number of rows

Super key

A set of attributes is said to be super key if and only if it satisfies:

- ***Uniqueness property*: No two distinct tuples have the same value for the key.**

- **If R is a relation, the set of attributes K is a Super Key, iff, for two distinct tuples t_1 and t_2 in R, $t_1[K] \neq t_2[K]$**

Candidate key

A set of attributes is said to be candidate key if and only if it satisfies the following time-independent properties:

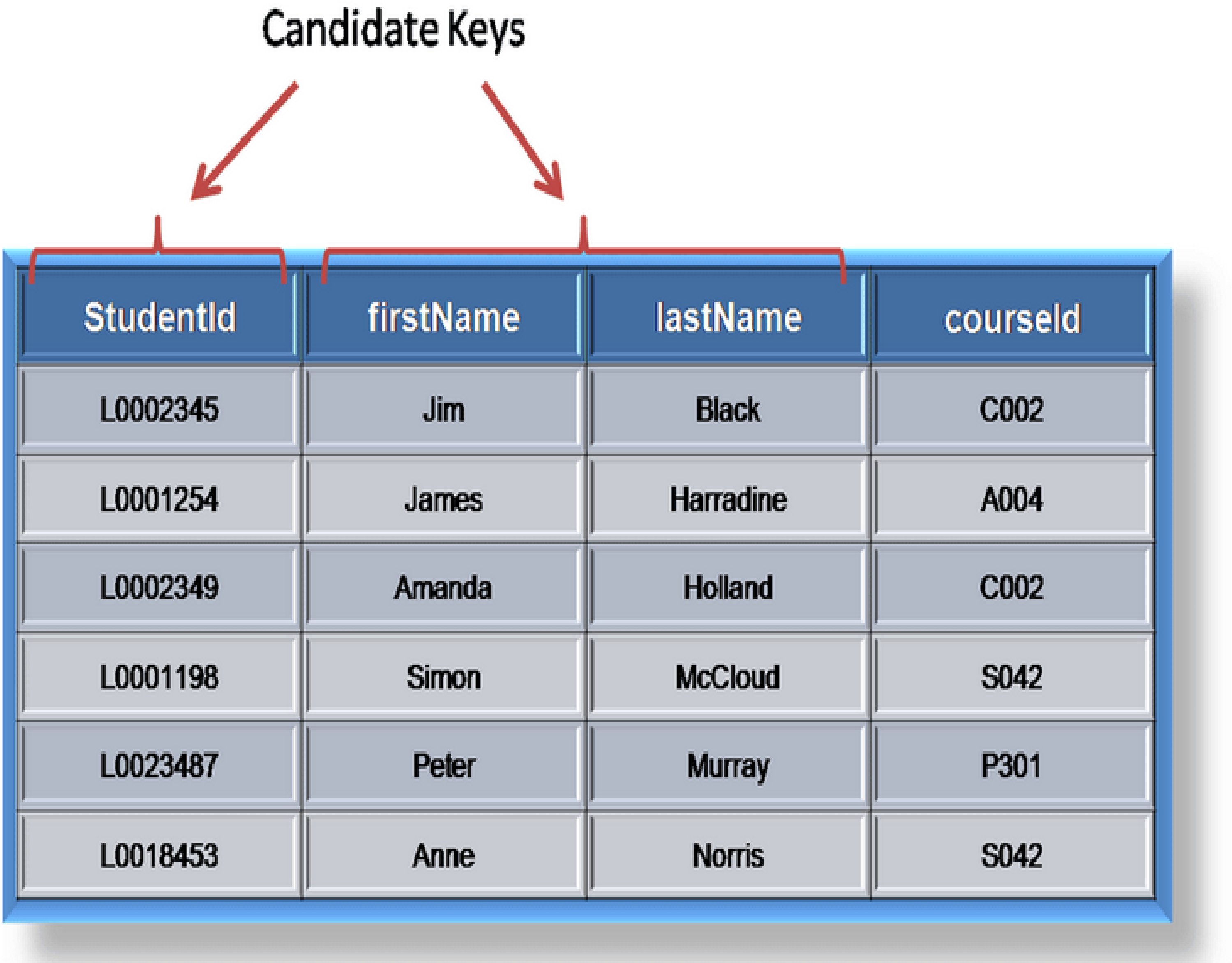
- ***Uniqueness property*: No two distinct tuples have the same value for the key.**
- ***Minimality property*: None of the attributes of the key can be discarded from the key without destroying the uniqueness property.**
- **A super key such that no proper subset is a super key within the relation**

Candidate key

- A candidate key is a single field or the least combination of fields that uniquely identifies each record in the table
  - The least combination of fields distinguishes a candidate key from a super key.
- A candidate is a subset of a super key
  - Every table must have at least one candidate key but at the same time can have several.

Candidate key

- It must contain unique values
- It must not contain null values
- It contains the minimum number of fields to ensure uniqueness
- It must uniquely identify each record in the table



Candidate Keys

StudentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301
L0018453	Anne	Norris	S042

- student_id uniquely identifies the students in a student table. This would be a candidate key.
- student’s first name and last name that also, when combined, uniquely identify the student in a student table.
- Both can be candidate keys.

Primary key

One of the candidate keys is chosen as the Primary Key.

There can be only one primary key per relation.

Primary key may be a compound key.

Primary Key

```
create table EMPLOYEE{
          EmpCode     char(6)                  primary key,
          Name                char(30),
          DesigCode   char(4),
          GradeCode   integer(4),
          JoinDate            date,
          Basic       money,
          Sex                 char(1),
          DeptCode            char(4) }
```

Secondary Key or Alternative Key

Any candidate key which is not a Primary Key is an Alternate Key.

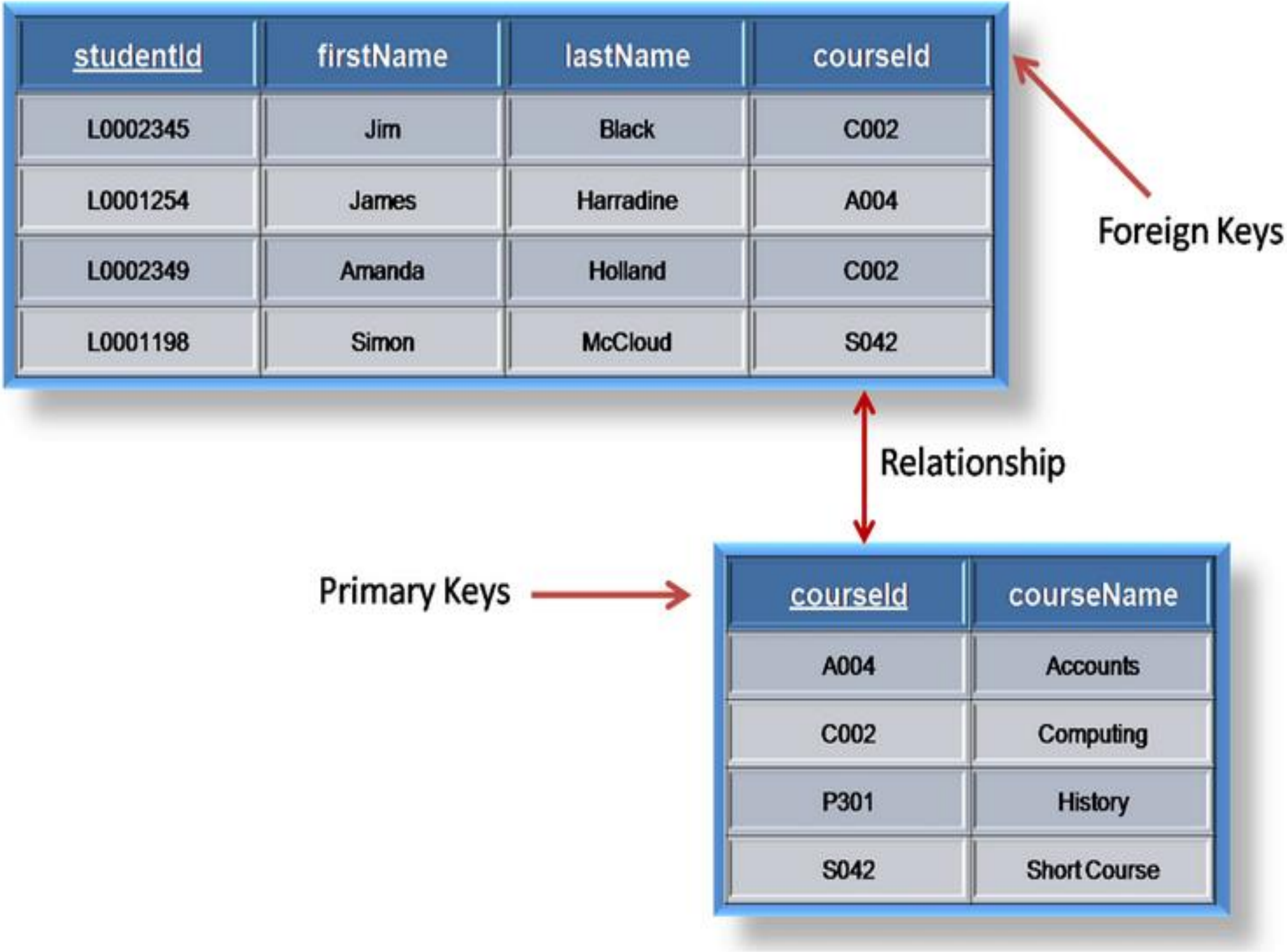
There can be more than one alternate key for any given relation.

Alternate Key

```
create table EMPLOYEE{
        EmpCode      char(6),
        Name              char(30)      alternate key,
        DesigCode    char(4),
        GradeCode    integer(4),
        JoinDate          date,
        Basic        money,
        Sex               char(1),
        DeptCode          char(4) }
```

Foreign Key

- A foreign key is generally a primary key from one table that appears as a field in another where the first table has a relationship to the second.
- If we had a table A with a primary key X that linked to a table B where X was a field in B, then X would be a foreign key in B.
- There can be more than one foreign key in a given relation.



studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

Foreign Keys

Relationship

Primary Keys

courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

Fundamental Integrity Rules

Entity Integrity - No attribute participating in the primary key of a base relation may accept null values.

Guarantees that each entity will have a unique identity.

Referential Integrity

Values of the foreign key (a) must be either null, or (b) if non-null, must match with the primary key value of some tuple of the `parent’ relation.

The reference can be to the same relation.

Codd’s 12 rules for RDBMS

Codd’s 12 Rules

Rule(0) –

The system must qualify as relational, as a database and a management system.

Rule(1) – Informational Rule

All information in a relational database (including table and column names) is represented in only one way, namely as a value in a table.

Codd’s 12 Rules

Rule(2) – Guaranteed Access Rule

All data must be accessible. *Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.*

Codd’s 12 Rules

- Rule(3) – *Systematic treatment of null values*
  - The DBMS must allow each field to remain null (or empty) to represent ‘missing’ or ‘not applicable’ information.
  - NULL can be interpreted as one the following: data is missing, data is not known, data is not applicable etc

Codd’s 12 Rules

- Rule(4) – *Active online catalog based on the relational model*
    - the structure description of whole database must be stored in an online catalog
    - data dictionary, which can be accessed by the authorized users.
    - Users can use the same query language to access the catalog which they use to access the database itself.

Codd’s 12 Rules

Rule(5) – *Comprehensive Data Sub-language*

The system must support at least one relational language that

- – Has a linear syntax
- – Can be used both interactively and within application programs,
- – Supports data definition operations (including view definitions), data manipulation operations (update as well as retrieval), security and integrity constraints, and transaction management operations (begin, commit, and rollback).

Codd’s 12 Rules

Rule(6) – The *view updating rule*

All views that are theoretically updatable must be updatable by the system.

Rule(7) – High-level insert, update, and delete

Codd’s 12 Rules

Rule(8) – Physical Data Independence

- Application should not have any concern about how the data is physically stored.
- Any change in its physical structure must not have any impact on application.

Rule(9) – Logical Data Independence

- Logical data must be independent of its user’s view (application).
- Any change in logical data must not imply any change in the application using it.
- if two tables are merged or one is split into two different tables, there should be no impact the change on user application.

Codd’s 12 Rules

Rule(10) – Integrity Independence

- Database must be independent of the application using it.
- All its integrity constraints can be independently modified without the need of any change in the application.
- This rule makes database independent of the front-end application and its interface.

Codd’s 12 Rules

Rule(11) – Distribution Independence

End user must not be able to see that the data is distributed over various locations.

User must also see that data is located at one site only.

This rule has been proven as a foundation of distributed database systems.

Codd’s 12 Rules

Rule(12) – Subversion Rule

If a system has an interface that provides access to low level records, this interface then must not be able to subvert the system and bypass security and integrity constraints.

Codd’s Objectives in Developing the Relational Model

- To accomplish a high degree of data independence
- To introduce a theoretical foundation for database management
- To eventually infuse inferential capabilities into a data management system

References

- Database Modeling & Design
  Toby J. Teorey
  Morgan Kaufmann Publishers Inc.

- Designing Quality Databases with IDEF1X Information Models
  Thomas A. Bruce
  Dorset Publishing House

- Database System Concepts (3/e)
  Silberschatz, Korth, Sudarshan
  McGraw-Hill International Edition (CS Series)