# *XXI. Database Design*

**Databases and DBMS**
**Data Models, Hierarchical, Network, Relational**
**Database Design**
**Restructuring an ER schema**
**Performance analysis**
**Analysis of Redundancies, Removing generalizations**
**Translation into a Relational Schema**
**The Training Company Example**
**Normal Forms and Normalization of Relational Schemata**

---

# *Databases*

- A *database* is a collection of persistent data shared by a number of applications
- Databases have been founded on the concept of *data independence*: Applications should not have to know the organization of the data or the access strategy employed
    *Need query processing facility, which generates automatically*
    *an access plan, given a query*
- Databases also founded on the concept of *data sharing*: Applications should be able to work on the same data concurrently, without knowing of each others' existence.

    *Database procedures defined in terms of atomic operations called*
    *transactions*

# *Conventional Files vs Databases*

**Databases**
Advantages -- Good for data integration; allow for more flexible formats (not just records)

Disadvantages -- high cost; drawbacks in a centralized facility

**Files**
Advantages -- many already exist; good for simple applications; very efficient

Disadvantages -- data duplication; hard to evolve; hard to build for complex applications

*The future is with databases!*

# *Database Concepts*

- *Data model* -- defines a set of data structures along with associated operations, for building and accessing a database
  - ✓ e.g., the relational model offers relations (tables) as data structure for building a database
- *Database management system* (DBMS) -- generic tool for building, accessing, updating and managing a database
  - ✓ E.g., Oracle, DB2, Access,… are all relational DBMSs
- *Database schema* -- describes the types and structure of the data stored in the database.
  - ✓ E.g., `Employee(emp#,name,addr,sal,dept,mngr)`
- *Transaction* -- an atomic operation on a database; looks like a procedure but has different semantics: when called, it either completes its execution, or aborts and undoes all changes it made to the database.
  - ✓ E.g., `TransferFunds(fromAcct#,toAcct#,amount,date)`

# *Types of Databases*

- **■** ***Conventional databases*** -- (relational, network, hierarchical) consist of records of many different record types (database looks like a collection of files)
- **■** ***Object-Oriented databases*** -- database consists of objects (and possibly associated programs); database schema consists of classes (which can be objects too).
- **■** ***Multimedia databases*** -- database can store formatted data (i.e., records) but also text, pictures,...
- **■** ***Active databases*** -- database includes event-condition-action rules
- **■** ***Deductive databases**** -- like large Prolog programs
- **■** ***Hypertext databases*** -- store and access efficiently HTML/XML documents; provide navigational facilities through a database, so that a user can retrieve and/or browse.
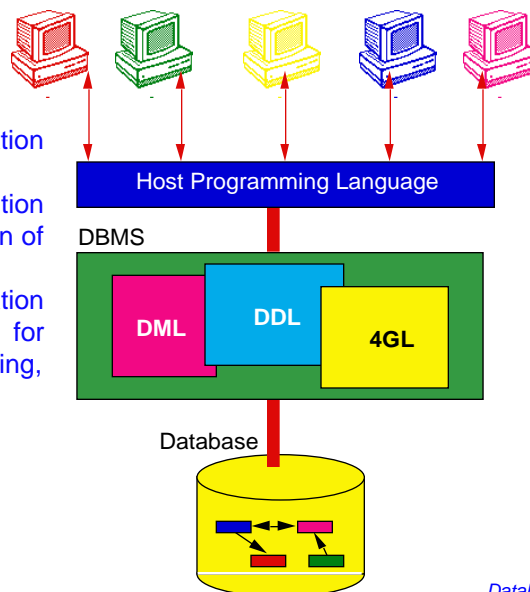- \* -- not available commercially

---

# *Database Management Systems*

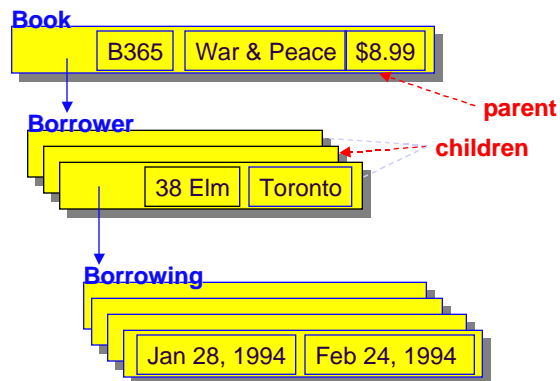**DML** -- data manipulation language
**DDL** -- data definition language (allows definition of database schema)
**4GL** -- fourth generation language, useful for declarative query processing, report generation,...

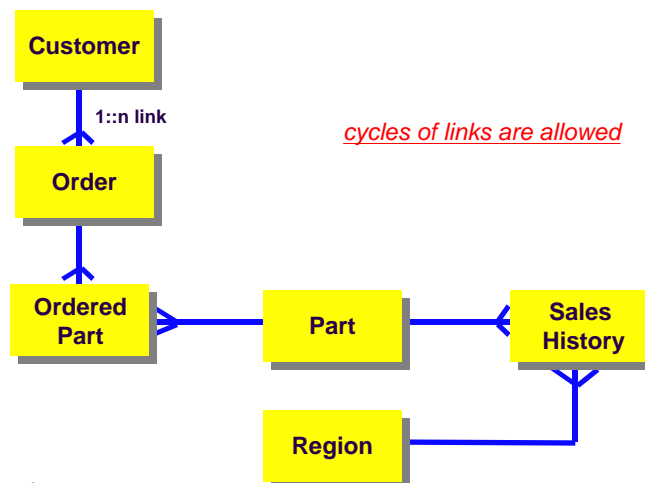Host Programming Language

DBMS

DML    DDL    4GL

Database

3

# *The Hieratchical Data Model*

■ Database consists of *hierarchical record structures*; a field may have as value a list of records; every record has at most one parent

**Book**

| | B365 | War & Peace | $8.99 |

**parent**

**Borrower**

**children**

| | 38 Elm | Toronto |

**Borrowing**

| | Jan 28, 1994 | Feb 24, 1994 |

---

# *The Network Data Model*

■ A database now consists of records with pointers (links) to other records. Offers a navigational view of a database.

**Customer**

**1::n link**

*cycles of links are allowed*

**Order**

**Ordered Part**        **Part**        **Sales History**

**Region**

4

# *The Relational Data Model*

- A database now consists of sets of records or (equivalently) sets of tuples (relations) or (equivalently) tables of tuples; no links allowed in the database.
- Every tuple is an element of exactly one relation and is identified uniquely by a primary key

**Customer**

| Cust# | Name | Address |
|-------|--------|-------------|
| 1127 | George | 25 Mars St |
| 1377 | Maria | 12 Low Av |
| 1532 | Manolis | 1 Bloor St. |
| ... | | |

**Order**

| Ord# | Date | Amour |
|------|----------|----------|
| 1997 | 11/3/93 | $65.87 |
| 4237 | 25/8/93 | $126.88 |
| 1552 | 12/12/93 | $284.21 |
| ... | | |

**Part**

| Part# | Desc | Quantity |
|-------|--------|-----------|
| 2397 | widget | 12,980 |
| 2908 | nut | 16,000doz. |
| 6590 | bolt | 14,340doz |
| ... | | |

**Ordered Part**

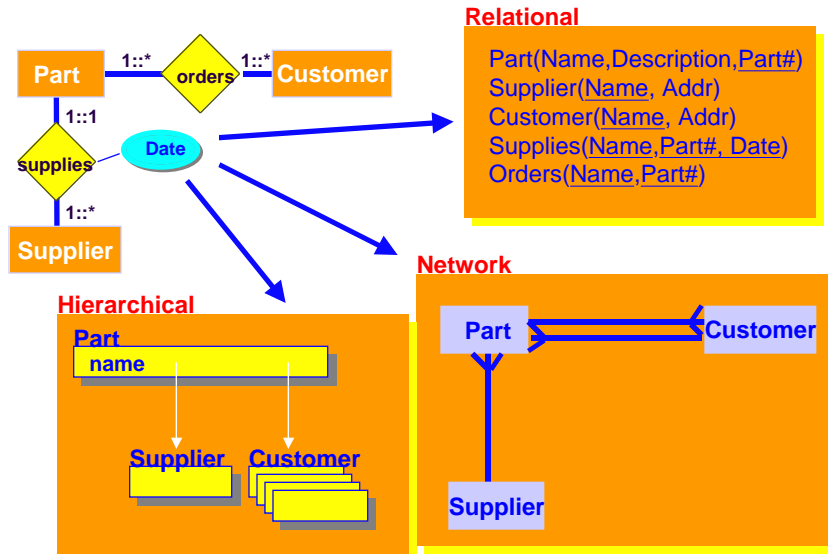| Part# | Ord# | Quantity |
|-------|------|----------|
| 2397 | 1997 | 980 |
| 2908 | 1997 | 100doz. |
| 6590 | 4237 | 40doz |
| ... | | |

---

# *Comparing Data Models*

- The oldest DBMSs were hierarchical, dating back to the mid-60s. IMS (IBM product) is the most popular among them. Many old databases are hierarchical.
- The network data model came next (early '70s). At the time of its proposal, it was viewed as a breakthrough. It emphasized the role of the database programmer as "navigator", chasing links (pointers, actually) around a database.
- But, the network model was found to be in many respects too implementation-oriented, not insulating sufficiently the programmer from implementation features of network DBMSs.
- The relational model is the most recent arrival (early '80s) and it has taken over the database market. Relational databases are considered simpler than their hierarchical and network cousins because they don't allow any links/pointers (which are necessarily implementation-dependent).
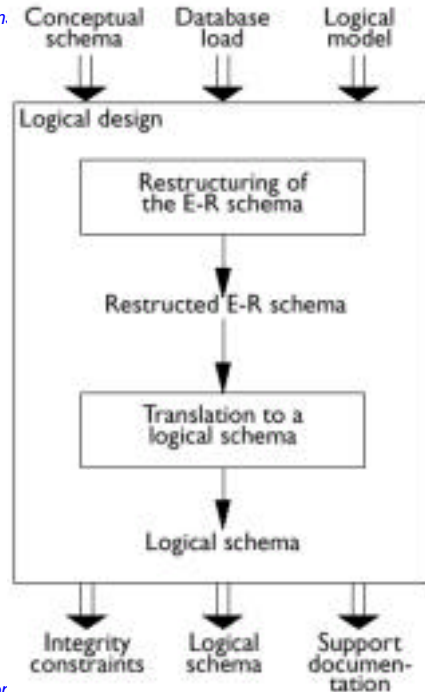
5

# *Designing a Database Schema*

**Relational**

| Part | 1::* | orders | 1::* | Customer |

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

1::1

Date

supplies

1::*

**Supplier**

**Network**

**Hierarchical**

**Part**
  name

**Part** **Customer**

**Supplier** **Customer**

**Supplier**

---

# *(Relational) Database Design*

- The aim of database design is to construct a relational schema that correctly and efficiently represents all of the information described by a class or Entity-Relationship diagram (or 'schema') produced during requirements analysis.

- From now, we'll only talk about transforming an E-R schema into a relational schema. Most of the transformation process applies for class diagrams as well.

- This is ***not*** just a simple translation from one model to another for two main reasons:
  - ✓ not all the constructs of the Entity-Relationship 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.

6

Conceptual schema  Database load  Logical model

Logical design

Restructuring of the E-R schema

Restructed E-R schema

Translation to a logical schema

Logical schema

Integrity constraints  Logical schema  Support documen- tation

# Database Design Process

---

## Logical Design Steps

It is helpful to divide the logical design into two steps:

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

# *Performance Analysis*

- An ER schema can be restructured to optimize two parameters:
  - ✓ *Cost of an operation* (evaluated in terms of the number of occurrences of entities and relationships that are visited during the execution of an operation on the database);
  - ✓ *Storage requirements* (evaluated in terms of number of bytes necessary to store the data described by the schema).
- In order to study these parameters, we need to know:
  - ✓ Projected volume of data;
  - ✓ Projected operation characteristics.

# *Cost Model*

- The cost of an operation is measured in terms of the number of disk accesses required. A disk access is, generally orders of magnitude more expensive than in-memory accesses, or CPU operations.
- For a coarse estimate of cost, we will assume that
  - ✓ a Read operation requires 1 disk access
  - ✓ A Write operation requires 2 disk accesses (read from disk, change, write back to disk)

# *Employee-Department Example*

---
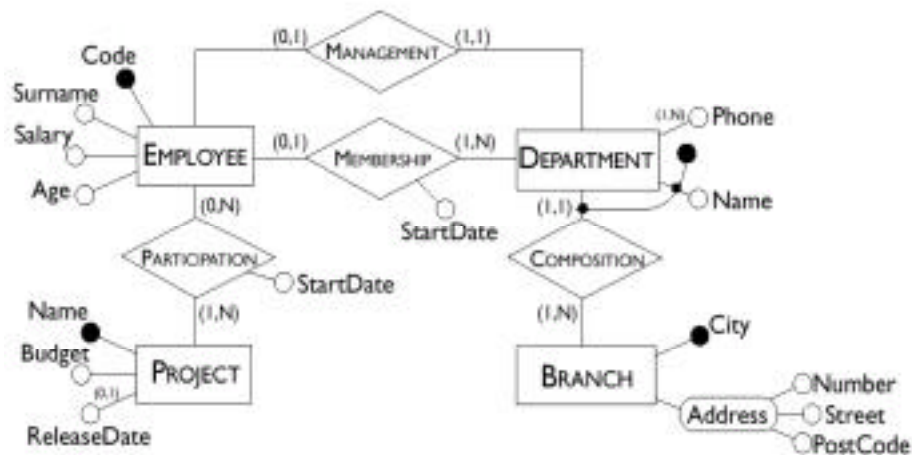
# *Typical Operations*

- Operation 1: Assign an employee to a project.
- Operation 2: Find the record of an employee, including the department where she works, and the projects she works for.
- Operation 3: Find the records of all employees for a given department.
- Operation 4: For each branch, retrieve its departments, and for each department, retrieve the last names of their managers, and the list of their employees.

- *Note:* For class diagrams, these would be operations associated with database classes.

9

# *Tables of Volumes and Operations*

The volume of data and the general characteristics of the operations can be summed up using two special tables.

**Table of volumes**

| Concept | Type | Volume |
|---|---|---|
| Branch | E | 10 |
| Department | E | 80 |
| Employee | E | 2000 |
| Project | E | 500 |
| Composition | R | 80 |
| Membership | R | 1900 |
| Management | R | 80 |
| Participation | R | 6000 |

**Table of operations**

| Operation | Type | Frequency |
|---|---|---|
| Operation 1 | I | 50 per day |
| Operation 2 | I | 100 per day |
| Operation 3 | I | 10 per day |
| Operation 4 | B | 2 per day |

**I - Interactive**

**B - Batch**

---

# *Navigation Schema for Operation 2*

A navigation schema starts from the inputs to an operation and moves (via arrows) towards its outputs.

10
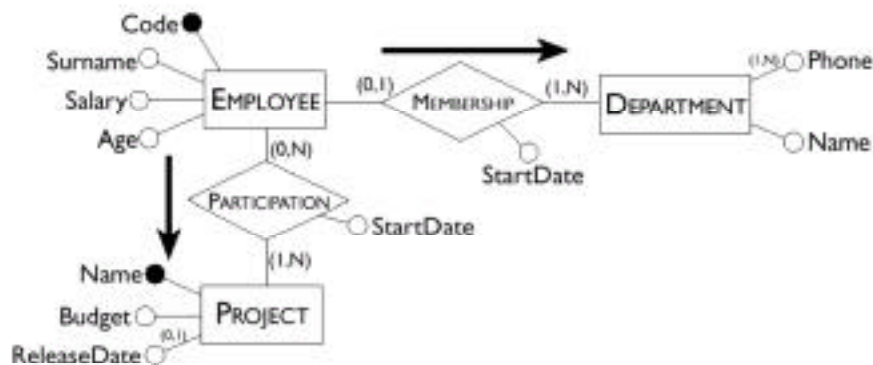
# *Table of Accesses*

This table evaluates the cost of an operation, using the table of volumes and the navigation schema.

**Operation 2**

| Concept | Type | Accesses | Type |
|---------|------|----------|------|
| Employee | Entity | 1 | R |
| Membership | Relationship | 1 | R |
| Department | Entity | 1 | R |
| Participation | Relationship | 3 | R |
| Project | Entity | 3 | R |

**Average # of participations and projects per employee**

**Type: R - Read. W - Write, RW - Read&Write**

# *Analysis Steps*

E-R schema    Database load

Restructuring of the E-R schema

Analysis of redundancies

Removing generalizations

Partitioning/merging of entities and relations

Selection of primary identifiers

Restructured E-R schema

11
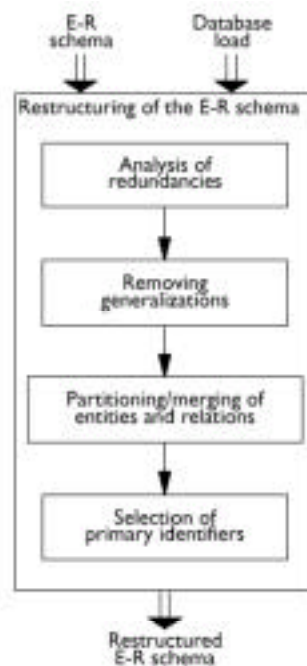
# *Analysis of Redundancies*

- A redundancy in a conceptual schema corresponds to a piece of information that can be derived (that is, obtained through a series of retrieval operations) from other data in the database.
- An Entity-Relationship schema can contain various forms of redundancy.

# *Examples of Redundancies*
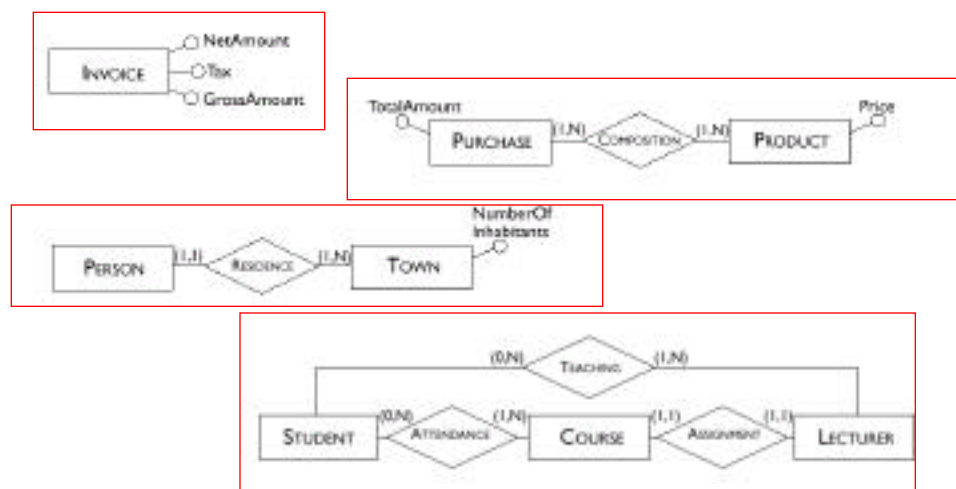
# *Deciding About Redundancies*

- The presence of a redundancy in a database may be
  - ✓ *an advantage*: a reduction in the number of accesses necessary to obtain the 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 delete 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.

# *Cost Comparison: An Example*
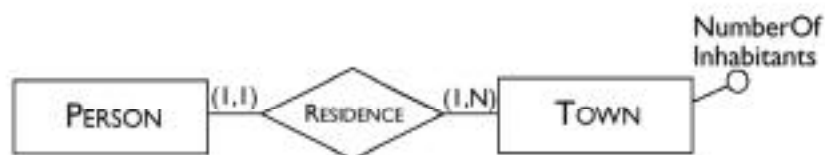


In this schema the attribute NumberOfInhabitants is redundant.

# *Load and Operations for the Example*

**Table of volumes**

| Concept | Type | Volume |
|---------|------|--------|
| Town | E | 200 |
| Person | E | 1000000 |
| Residence | R | 1000000 |

**Table of operations**

| Operation | Type | Frequency |
|-----------|------|-----------|
| Operation 1 | I | 500 per day |
| Operation 2 | I | 2 per day |

- **Operation 1**: add a new person with the person's town of residence.
- **Operation 2**: print all the data of a town (including the number of inhabitants).

---

# *Table of Accesses, with Redundancy*

**Operation 1**

| Concept | Type | Accesses | Type |
|---------|------|----------|------|
| Person | Entity | 1 | W |
| Residence | Relationship | 1 | W |
| Town | Entity | 1 | W |

**Operation 2**

| Concept | Type | Accesses | Type |
|---------|------|----------|------|
| Town | Entity | 1 | R |

# Table of Accesses, without Redundancy

**Operation 1**

| Concept | Type | Accesses | Type |
|---------|------|----------|------|
| Person | Entity | 1 | W |
| Residence | Relationship | 1 | W |

**Operation 2**

| Concept | Type | Accesses | Type |
|---------|------|----------|------|
| Town | Entity | 1 | R |
| Residence | Relationship | 5000 | R |

---

# Comparing the Cost of Operations

Presence of redundancy:
- ✓ Operation 1: 1,500 write accesses per day;
- ✓ The cost of operation 2 is almost negligible;
- ✓ Counting twice the write accesses, we have a total of 3,000 accesses a day.

Absence of redundancy.
- ✓ Operation 1: 1,000 write accesses per day;
- ✓ Operation 2 however requires a total of 10,000 read accesses per day;
- ✓ Counting twice the write accesses, we have a total of 12,000 accesses per day.

*Redundant data can improve performance, sometimes!*

# *Removing Generalizations*

■ The relational model does not allow the direct representation of generalizations that may be present in an E-R diagram.

■ We need, therefore, to transform these constructs into other constructs that are easier to translate: entities and relationships.

# *A Schema with Generalizations*

16
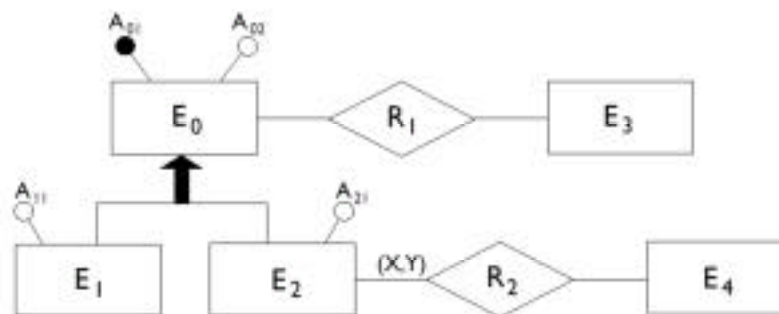
**Option 1**

**Note!**

*Possible Restructurings*
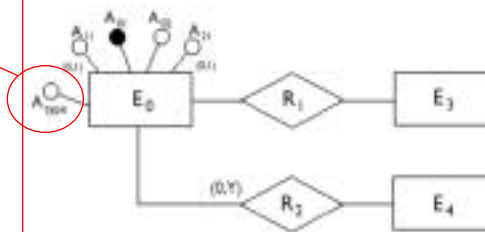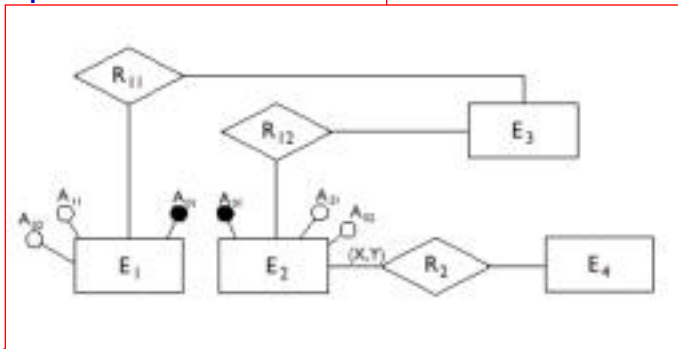
**Option 2**

**Option 3**

*...Two More...*
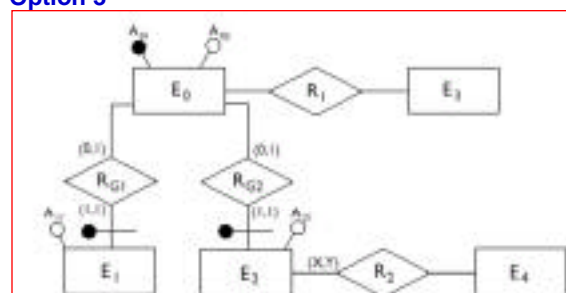
**Option 4 (combination)**
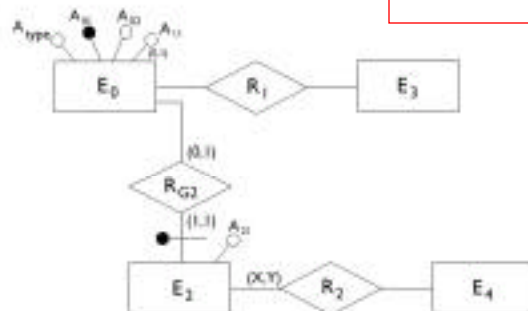
## *General Rules For Removing Generalization*

- Option 1 is convenient when the operations involve the occurrences and the attributes of $E_0$, $E_1$ and $E_2$ more or less in the same way.
- Option 2 is possible only if the generalization is total and is useful when there are operations that apply only to occurrences of $E_1$ or of $E_2$.
- Option 3 is useful when the generalization is not total and the operations refer to either occurrences and attributes of $E_1$ ($E_2$) or of $E_0$, and therefore make distinctions between child and parent entities.
- Available options can be combined (see option 4)

---

## *Partitioning and Merging of Entities and Relationships*

- Entities and relationships of an E-R schema can be partitioned or merged to improve the efficiency of operations, using the following principle: Accesses are reduced by separating attributes of the same concept that are accessed by different operations and by merging attributes of different concepts that are accessed by the same operations.
- The same criteria with those discussed for redundancies are valid in making a decision about this type of restructuring.

# *Example of Partitioning*
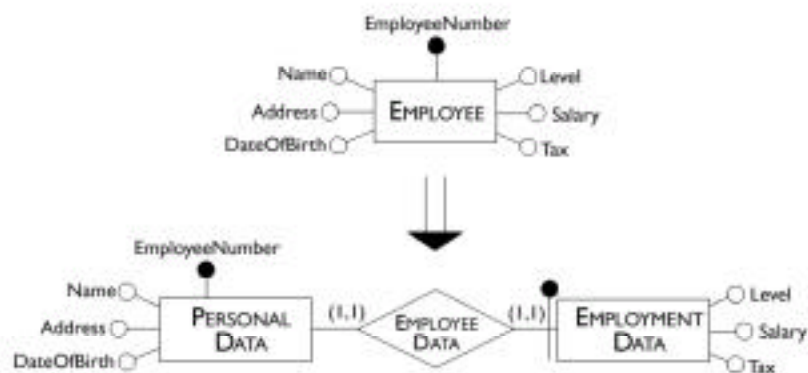
# *Deletion of Multi-Valued Attribute*
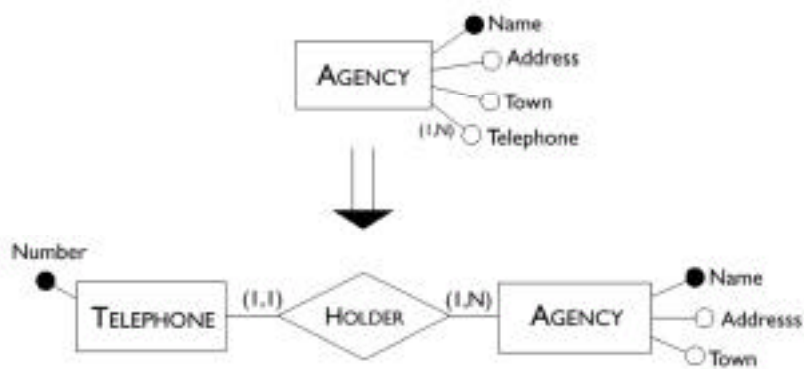
# *Merging of Entities*
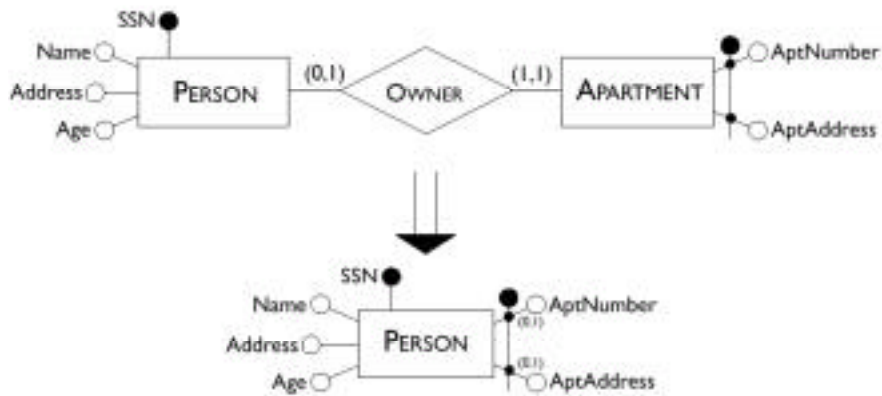
# *Partitioning of a Relationship*
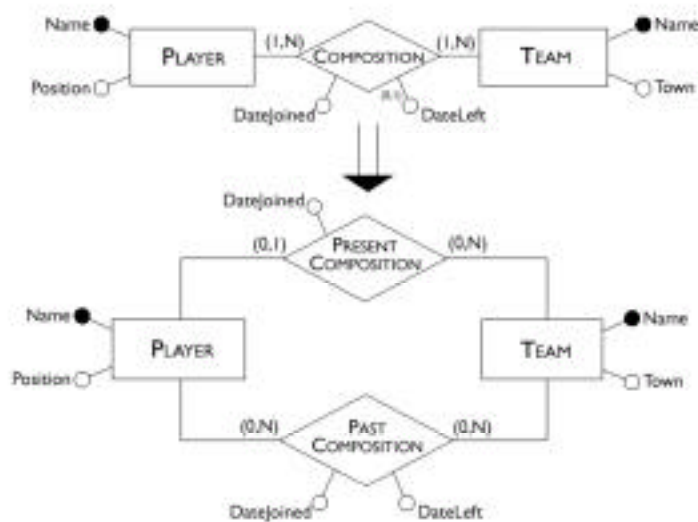
20

# *Selection of Primary Identifiers*

■ The criteria for this decision are as follows.
- ✓ Attributes with null values cannot form primary identifiers;
- ✓ One or few attributes are preferable to many attributes;
- ✓ An internal identifier with few attributes is preferable to an external one, possibly involving many entities;
- ✓ An identifier that is used by many operations to access the occurrences of an entity is preferable to others.

■ At this stage, if none of the candidate identifiers satisfies the above requirements, it is possible to introduce a further attribute to the entity. This attribute will hold special values (often called ***codes***) generated solely for the purpose of identifying occurrences of the entity.

# *Translation into a Logical Schema*

■ The second step of logical design corresponds to a translation between different data models.

■ Starting from an E-R schema, an equivalent relational schema is constructed. By "equivalent", we mean 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.
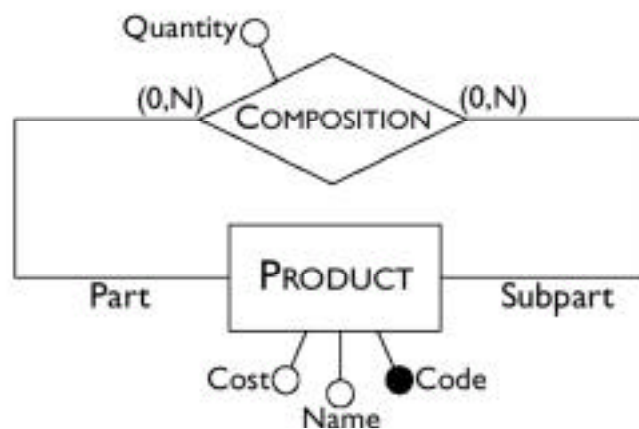
# *Many-to-Many Relationships*



Employee(<u>Number</u>, Surname, salary)
Project(<u>Code</u>, Name, Budget)
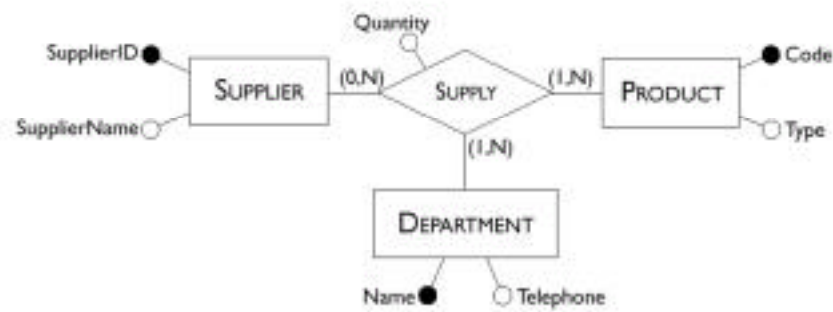Participation(<u>Number</u>, <u>Code</u>, StartDate)

---

# *Recursive Relationships*



Product(<u>Code</u>, Name, Cost)
Composition(<u>Part</u>, <u>SubPart</u>, Quantity)

22

# *Ternary Relationships*



Supplier(<u>SupplierID</u>, SupplierName)
Product(<u>Code</u>, Type)
Department(<u>Name</u>, Telephone)
Supply(<u>Supplier</u>, <u>Product</u>, <u>Department</u>, Quantity)

---

# *One-to-Many Relationships*



Player(<u>Surname</u>,<u>DateOfBirth</u>, Position)
Team(<u>Name</u>, Town, TeamColours)
Contract(<u>PlayerSurname</u>, <u>PlayerDateOfBirth</u>, Team, Salary)

23

# *Entities with External Identifiers*
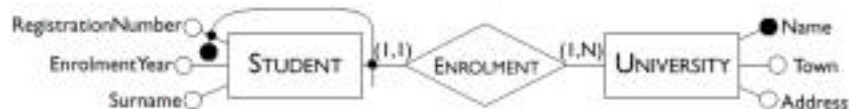


Student(<u>RegistrationNumber</u>, <u>University</u>, Surname, EnrolmentYear)
University(<u>Name</u>, Town, Address)

---

# *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*
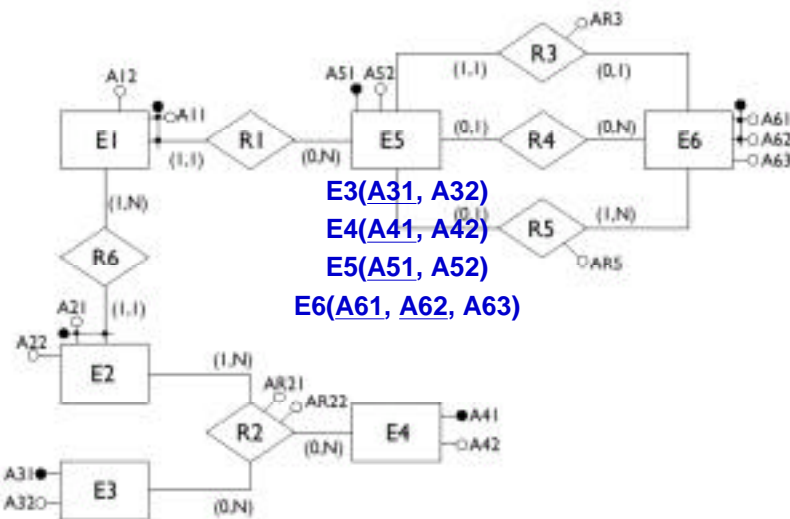Employee(<u>Number</u>, Name, Salary)
Department(<u>Name</u>, Telephone, Branch)
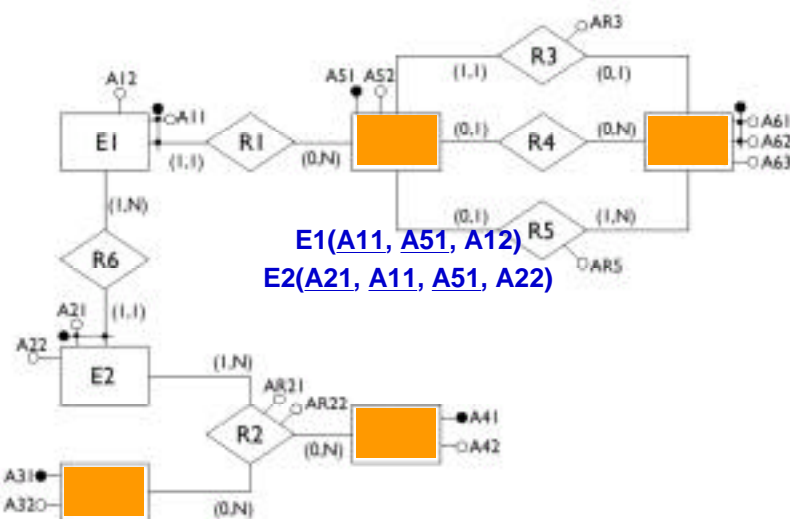Management(<u>Head</u>, Department, StartDate)

---

# *A Sample ER Schema*
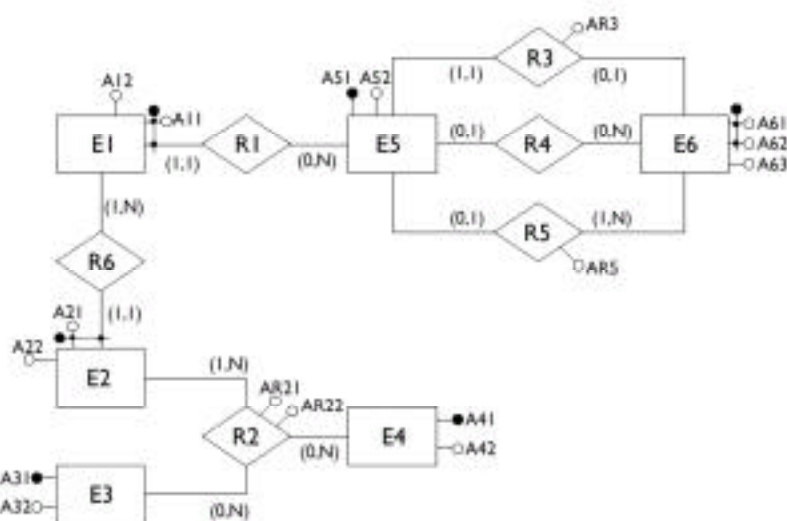
# Entities with Internal Identifiers



**E3(A31, A32)**
**E4(A41, A42)**
**E5(A51, A52)**
**E6(A61, A62, A63)**

# Entities with External Identifiers



**E1(A11, A51, A12)**
**E2(A21, A11, A51, A22)**

# Many-to-Many Relationships

**R2(A21, A11, A51, A31, A41, AR21, AR22)**

---
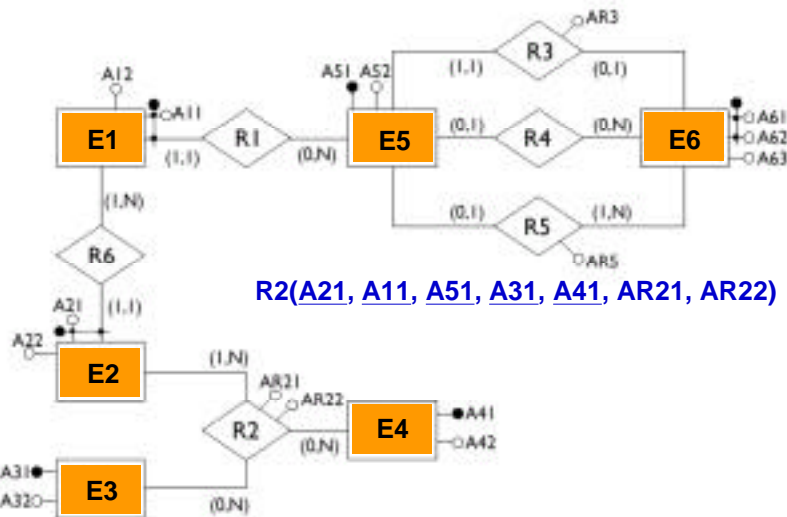
# Result of the Translation

**E1(A11, A51, A12)**

**E2(A21, A11, A51, A22)**

**E3(A31, A32)**

**E4(A41, A42)**

*1-1 or optional 1-1 relationships can lead to messy transformations*

**E5(A51, A52, A61R3, A62R3, AR3, A61R4, A62R4, A61R5, A62R5, AR5)**

**E6(A61, A62, A63)**

**R2(A21, A11, A51, A31, A41, AR21, AR22)**

# *Summary of Transformation Rules*

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

---

# *...More Rules...*

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

# *…Even More Rules...*

# *…and the Last One...*

# *The Training Company Revisited*

---
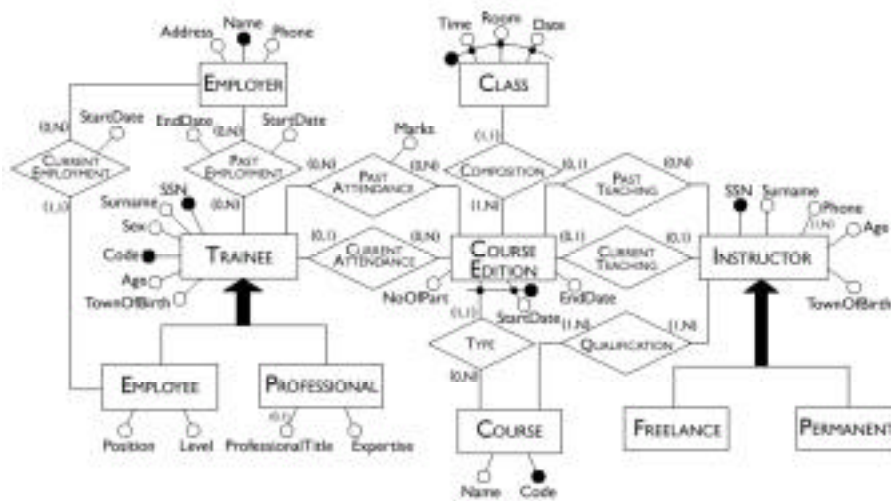
# *Operational Requirements, Revisited*

- **operation 1**: insert a new trainee including all his or her data (to be carried out approximately 40 times a day);
- **operation 2**: assign a trainee to an edition of a course (50 times a day);
- **operation 3**: insert a new instructor, including all his or her data and the courses he or she is qualified to teach (twice a day);
- **operation 4**: assign a qualified instructor to an edition of a course (15 times a day);
- **operation 5**: display all the information on the past editions of a course with title, class timetables and number of trainees (10 times a day);
- **operation 6**: display all the courses offered, with information on the instructors who are qualified to teach them (20 times a day);
- **operation 7**: for each instructor, find the trainees all the courses he or she is teaching or has taught (5 times a week);
- **operation 8**: carry out a statistical analysis of all the trainees with all the information about them, about the editions of courses they have attended and the marks obtained (10 times a month).

30

# *Database Load*

### Table of volumes

| Concept | Type | Volume |
|---|---|---|
| Class | E | 8000 |
| CourseEdition | E | 1000 |
| Course | E | 200 |
| Instructor | E | 300 |
| Freelance | E | 250 |
| Permanent | E | 50 |
| Trainee | E | 5000 |
| Employee | E | 4000 |
| Professional | E | 1000 |
| Employer | E | 8000 |
| PastAttendance | R | 10000 |
| CurrentAttendance | R | 500 |
| Composition | R | 8000 |
| Type | R | 1000 |
| PastTeaching | R | 900 |
| CurrentTeaching | R | 100 |
| Qualification | R | 500 |
| CurrentEmployment | R | 4000 |
| PastEmployment | R | 10000 |

### Table of operations

| Operation | Type | Frequency |
|---|---|---|
| Operation 1 | I | 40 per day |
| Operation 2 | I | 50 per day |
| Operation 3 | I | 2 per day |
| Operation 4 | I | 15 per day |
| Operation 5 | I | 10 per day |
| Operation 6 | I | 20 per day |
| Operation 7 | I | 5 per day |
| Operation 8 | B | 10 per month |

---

# *Access Tables*

The attribute `NumberOfParticipants` in `CourseEdition` can be derived from the relationships `CurrentAttendance` and `PastAttendance`.

### Operation 2 with redundancy

| Concept | Type | Acc | Type |
|---|---|---|---|
| Trainee | E | 1 | R |
| CurrentAtt'nce | R | 1 | W |
| CourseEdition | E | 1 | R |
| CourseEdition | E | 1 | W |

### Operation 2 without redundancy

| Concept | Type | Acc | Type |
|---|---|---|---|
| Trainee | E | 1 | R |
| CurrentAtt'nce | R | 1 | W |

### Operation 5 with redundancy

| Concept | Type | Acc | Type |
|---|---|---|---|
| CourseEdition | E | **5** | R |
| Type | R | **5** | R |
| Course | E | **1** | R |
| Composition | R | **40** | R |
| Class | E | **40** | R |

### Operation 5 without redundancy

| Concept | Type | Acc | Type |
|---|---|---|---|
| CourseEdition | E | **5** | R |
| Type | R | **5** | R |
| Course | E | **1** | R |
| Composition | R | **40** | R |
| Class | E | **40** | R |
| PastAtt'nce | E | **50** | R |

31

# *Analysis of Redundancy*

- ■ From the access tables we obtain (giving double weight to the write accesses):
  - ✓ presence of redundancy: for operation 2 we have 100 read accesses and 100 write accesses per day; for operation 5 we have 910 read accesses per day, for a total of 1,210 accesses per day;
  - ✓ without redundancy:  for operation 2 we have 50 read accesses per day and 100 write accesses per day; for operation 5, we have 1,410 read accesses per day, for a total of 1,660 accesses per day.
- ■ Thus, redundancy makes sense in this case, so we leave the attribute `NumberOfParticipants` as an attribute of the entity `CourseEdition.`

---

# *Removing Generalizations*

- ■ For the generalization on instructors:
  - ✓ the relevant operations make no distinction between the child entities and these entities have no specific attributes;
  - ✓ we can therefore delete the child entities and add an attribute Type to the parent entity.
- ■ For the generalization on trainees:
  - ✓ the relevant operations make no distinction between the child entities, but these entities have specific attributes;
  - ✓ we can therefore leave all the entities and add two relationships to link each child with the parent entity: in this way, we will have no attributes with possible null values on the parent entity and the dimension of the relations will be reduced.

# *Partitioning and Merging of Concepts*

- The relationships `PastTeaching` and `PresentTeaching` can be merged since they describe similar concepts between which the operations make no difference. A similar consideration applies to the relationships `PastAttendance` and `PresentAttendance`.
- The multi-valued attribute `Telephone` can be removed from the `Instructor` entity by introducing a new entity `Telephone` linked by a one-to-many relationship to the `Instructor` entity.

# *Choice of Main Identifiers*

- `Trainee` entity:
  - ✓ there are two identifiers: the social security number and the internal code;
  - ✓ it is far preferable to choose the latter: a social security number will require several bytes whereas an internal code, which serves to distinguish between 5000 occurrences, requires a few bytes.
- `CourseEdition` entity:
  - ✓ it is identified externally by the `StartDate` attribute and by the `Course` entity;
  - ✓ we can see however that we can easily generate for each edition a code from the course code: this code is simpler and can replace the external identifier.

33

# After Restructuring

---
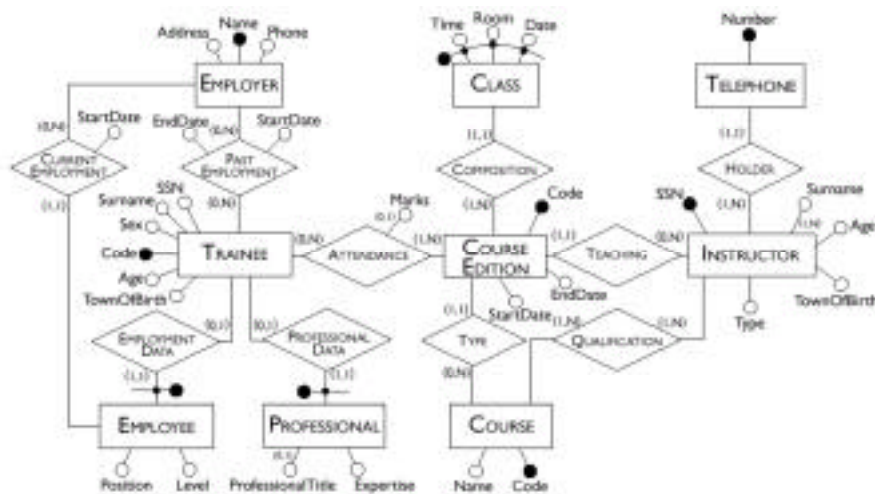
# Translation into the Relational Model

CourseEdition(Code, StartDate, EndDate, Course, Instructor)

Class(Time, Room, Date, Edition)

Instructor(SSN, Surname, Age, TownOfBirth, Type)

Telephone(Number, Instructor)

Course(Code, Name)

Qualification(Course, Instructor)

Trainee(Code, SSN, Surname, Age, TownOfBirth, Sex)

Attendance(Trainee, Edition, Marks*)

Employer(Name, Address, Telephone)

PastEmployment(Trainee, Employer, StartDate, EndDate)

Professional(Trainee, Expertise, ProfessionalTitle*)

Employee(Trainee, Level, Position, Employer, StartDate)

34

# *Logical Design Using CASE Tools*

- The logical design phase is partially supported by database design tools:
  - ✓ the translation to the relational model is carried out by such tools semi-automatically;
  - ✓ the restructuring step is difficult to automate and CASE tools provide little or no support for it.
- Most commercial CASE tools will generate automatically SQL code for the creation of the database.
- Some tools allow direct connection with a DBMS and can construct the corresponding database automatically.

*Logical Design with a CASE Tool*

35

# *What is a Good Relational Schema Like?*

n Some relational schemata are "better" representations than others. What are the criteria we can use to decide whether a diagram is better than another? Should we have more/fewer relations as opposed to attributes?

### *Enter normal forms*

n An attribute **a** *(functionally) depends* on a set of attributes **a$_1$,a$_2$,..., a$_n$** if these determine uniquely the value of **a** for every tuple of the relation where they appear together

$$a_1,a_2,..., a_n \text{ --> } a$$

n Example: For the relation

**Course(<u>name</u>,title,instrName,rmName,address),**

E.g. (csc340,"Analysis and Design",JM,RW117,"48 StG")

the **title** attribute depends on the **name** attribute. Likewise, the address attribute depends on the **rmName** attribute,

**name --> title** , also **rmName --> address**

---

# *Examples of Functional Dependencies*

■ Consider

Supplier(<u>S#</u>,SName, Status, Address)

■ Here SName,Status,Address functionally depend on S# because S# uniquely determines the values of the other attributes of the Supplier relation

S# --> SName, Status, Address

■ Likewise, assuming that Lastname, Firstname uniquely identify people, we have

Lastname, Firstname --> Salary, Address

36

## *What's Wrong with Un-Normalized Relations?*

- Normalization helps produce a schema that is not redundant and does not suffer from *anomalies*.
- Consider `Emp(Emp#,Ename,Address,Dept,Mngr#)`
  with `Empl# --> EName, Address, Dept, Mngr#,`
  `Dept --> Mngr#`
- *Insertion anomaly:* We can't add information about a new department and its manager until we have an employee in that department.
- *Deletion anomaly:* If we delete the only employee in a department, we lose information about the department (e.g., its manager)
- *Update anomaly:* If we update the `Mngr#` attribute of one tuple, we must do it for all, otherwise we have an inconsistent database.

*It's easy to get an inconsistent database*

*when it's not normalized*

---

## *How Do We Identify Functional Dependencies?*

- Think about the meaning of different attributes and try to think of situations where the value of $a$ is not determined by the values of $a_1$, $a_2, \ldots, a_n$
- Alternatively, if you are given sample values for the attributes of the relation (see below), check to ensure that every combination of values for $a_1, a_2, \ldots, a_n$ has the same associated value for $a$

| Name | Title | Instructor | Office | Tutors | Enrolment |
|------|-------|-----------|--------|--------|-----------|
| csc148 | "Intro.." | Reiter | LP290G | 4 | 133 |
| csc228 | "DP" | Clarke | SF285 | 3 | 124 |
| csc238 | "Logic" | Fich | SF254 | 3 | 85 |
| csc324 | "PLs" | Bonner | LP354 | 2 | 72 |
| csc340 | "SA" | Reiter | LP290G | 2 | 121 |
| csc408 | "SE" | Clarke | SF285 | 3 | 88 |
| csc434 | "DM" | Fich | SF254 | 3 | 107 |

*What functional dependencies are appropriate here?*

# *Normalizing Relational Schemas: 1NF*

n A relation is in *First Normal Form* (*1NF*) if it does not include any multi-valued attributes or any composite attributes.       *Multi-valued attributes*

  e.g., consider the relation

    Course(<u>name</u>,title,instrName*,studentNo*,addr)

  Course is not in 1NF because of two attribute groups that repeat (instructor and student groups)

n To place a relation in 1NF, take each multi-valued attribute or composite attribute and promote it into an relation in its own right.

---

# *An Example*

n For the Course(<u>name</u>,title,instrName*,studentNo*,addr), example, assume that addr is a composite attribute consisting of a streetNm, streetNo, city and postalCode):

                  ==> Course(<u>name</u>,title)
            CourseStudt(name,studentNo)
             CourseInstr(<u>name,instName</u>)
    CourseAddr(name,streetNm,streetNo,city,postalCode)

# *Normalizing Relational Schemas: 2NF*

- n An relation is in *Second Normal Form* (*2NF*) if it is in 1NF and, moreover, all non-key attributes depend on all elements of its key, rather than a subset.
- n Consider
  `Room(street,number,bldgNm,room#,capacity,AVEquip)`
- ■ `Room` is not in 2NF because its `address` attributes functionally depend on its `bldgNm` key element, rather than the combination (`room#,bldgNm`)
- n To place a 1NF relation into 2NF, take each non-key attribute that does not depend on the full key and move it to a relation identified by the partial key

      `==>` Room(**bldgNm,room#,capacity,AVEquip),**

            **Building(bldgNm,street,number)**

---

# *Normalizing Relational Schemas: 3NF*

- ■ A relation is in *Third Normal Form* (*3NF*) if it is in 2NF and none of its non-key attributes depends on any other non-key attribute.
- ■ Assuming that each course has only one instructor (*why do we need this assumption?*), `Course` is not in 3NF because `instrDept` depends on `instrNm`:

      **Course(name,year,sem,instrNm,instrDept,enrol#)**

- ■ To place a 2NF relation into 3NF, move each non-key attribute that depends on other non-key attributes to another relation

      `==>` **Course(name,year,sem,instrNm,enrol#)**

                  **Instructor(name,dept)**