CS 4221: Database Design

Translating Relational Schema into OODB Schema

Ling Tok Wang
National University of Singapore

Topics

- **Inclusion dependency**
- **Identify object class**
- **Identify identifier dependency and complex object**
- **Identify ISA Hierarchy**
- **Identify inter-object Relationship**

Translating Relational Schema with FD's & Inclusion Dependencies into OODB Schema

Input information:

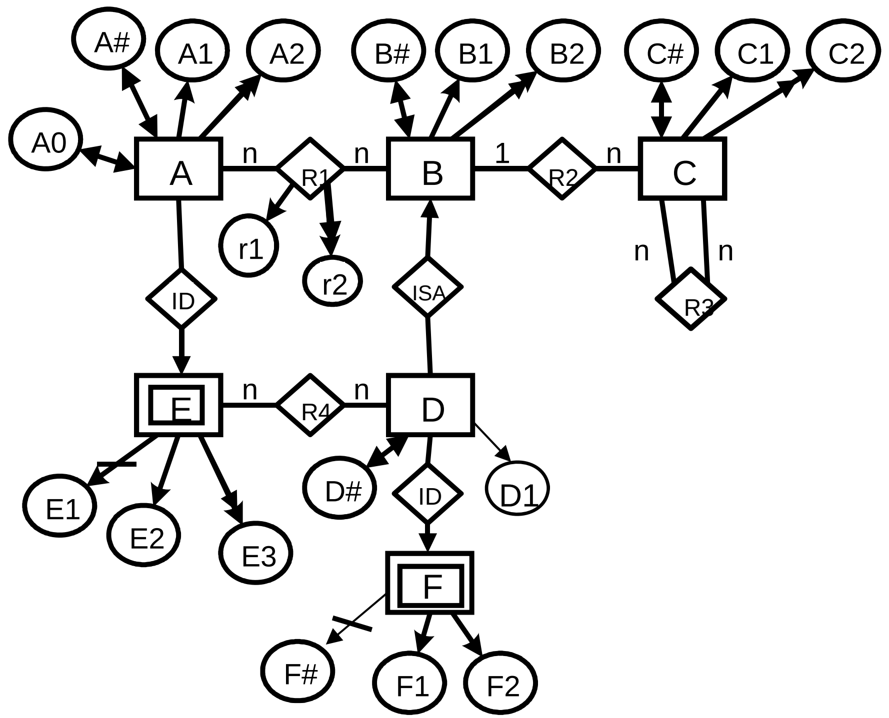
- a relational database schema
- primary/candidate keys
- inclusion dependencies (IND)

main features of the translator:

1. Identify clusters of relations that represent object classes and their attributes.
2. Identify Identifier Dependencies (ID) (like in ERD)
3. Identify ISA relationships among object classes.
4. Identify relationship types among object classes together with the relationship attributes.

Ref. (1) Ling Ling Yan and Tok Wang Ling, Translating Relational Schema with Constraints into OODB Schema, DS-5, Semantics of Interoperable Database Systems, Nov 1992, Lorne Australia.
(2) Wenyue Du, Mong-Li Lee, Tok Wang Ling, XML Structures for Relational Data, WISE (1) 2001: 151-160, 3-6 December 2001, Kyoto, Japan.

Main idea: We know how to map an ER diagram to relational schema. Question is how to construct the reverse mapping?



Some relations and constraints translated from the above ER diagram:

A(A#, A0, A1)
A′(A#, A2)
B(B#, B1)
B′(B#, B2)

R1(A#, B#, r1)
R1′(A#, B#, r2)

C(C#, C1, B#)
C′(C#, C2)
R3(C#1, C#2)

E(A#, E1, E2)
E′(A#, E1, E3)
R4(A#, E1, D#)

D# in D isa B# in B

C#1 in R3 isa C# in C
C#2 in R3 isa C# in C

Note: E1 is not a foreign key

Q: What are the object relations and relationship relations of the above relational schema? How to find them?

An **inclusion dependency** is denoted in the following form:

$$R_1 [P_1] \subseteq R_2 [P_2]$$

where R_1 and R_2 are relations, P_1 and P_2 are sequences of attributes in relation R_1 and R_2 , and $| P_1 | = | P_2 |$ (i.e. same no of attributes).

This inclusion dependency states that at any time if r_1 and r_2 are instances of relation schemas R_1 and R_2, resp, then the following always holds:

$$r_1 [P_1] \subseteq r_2 [P_2]$$

Defn: If there exists a key K of a relation R, and a sequence of attributes A' of another relation R', such that

$$R' [A'] \subseteq R [K]$$

We say R' **references** R. (i.e. A' is a foreign key in R' which references to a key K of R).

Moreover, A' in R' is a **foreign key**.

Note: **Referential constraints** are also inclusion dependencies.

(1) Identify object classes

An **object class** in the translated OODB schema corresponds to a **cluster of relations** from the underlying relational database.

An object class consists of a **core relation** (which represents the core part of an object class, i.e. all single valued attributes and the OID of the object class), and some **component relations** (which represent other properties of the object class, i.e. multi-valued attributes and the OID of the object class).

Example: staff (S#, name, DOB, sex, address)
staffHobby (S#, hobby)
staffQual (S#, degree, university, year)

staff is a core relation of the object class staff (i.e. employee).
staffHobby and staffQual are component relations of object class staff.

Defn: (Core relation or main class relation)

Consider a relation R, R is a core relation of some object class if one of the following case is true:

Case 1: R is not involved in any inclusion dependency.

(This is the case where a relation is stand-alone. Seldom)

Case 2: The followings hold:

(a) There is a relation R′ that references R.

(b) R does not contain more than one disjoint foreign key (i.e. R is not a relationship relation).

(c) There exists no inclusion dependency whose right side attribute set is proper subset of the primary key of relation R.

Case 3: R is identified as a core relation by **ID-dependency** identification rule as discussed later (see page 9).

E.g. Emp (<u>Eno</u>, Name, Dob, Dno)

Dept (<u>Dno</u>, Dname, Location)

Dept is core relation (by Case 2).

Q: How about relation EMP?

We will this discuss later. Emp will be classified as a mixed relation of a core relation and an inter-object relation.

Defn: (Component relation)

Let R be a main class relation. Relation R_1 is a **component relation** of R if the following hold:

(1) R_1 references R.

(2) No relation references R_1.

(3) R_1 does not contain more than one disjoint foreign key.

(4) The foreign key which references R is

- (i) part of the key of R_1, or
- (ii) a non prime of R_1, or
- (iii) the key of R_1.

Note: So, a component relation of R is formed by the primary key of R and a multi-valued attribute (m:m or 1:m, i.e. case 4(i) or 4(ii) resp.) or an optional m:1 attribute (case 4(iii)) of the object class of R.

Each core relation together with all its component relations form a cluster of relations that represents an object class. The name of the class will be the Name of the core relation.

Example 1. Consider the following RDB

Person (Pno, Name, Age)
PersonPhone (Pno, Phoneno)
PersonEmail (Email, Pno)
Parent (Pno, ChildPno)

with the following inclusion dependencies:

PersonPhone [Pno] $\subseteq$ Person[Pno]
PersonEmail [Pno] $\subseteq$ Person [Pno]
Parent [Pno] $\subseteq$ Person [Pno]
Parent [ChildPno] $\subseteq$ Person [Pno]

Clearly, person is a core relation with PersonPhone and PersonEmail are its component relations.
However parent is not a component relation of Person (since Parent consists 2 foreign keys). It is a (recursive) inter-object relationship relation (to be discussed later).

We have the following cluster of relations.

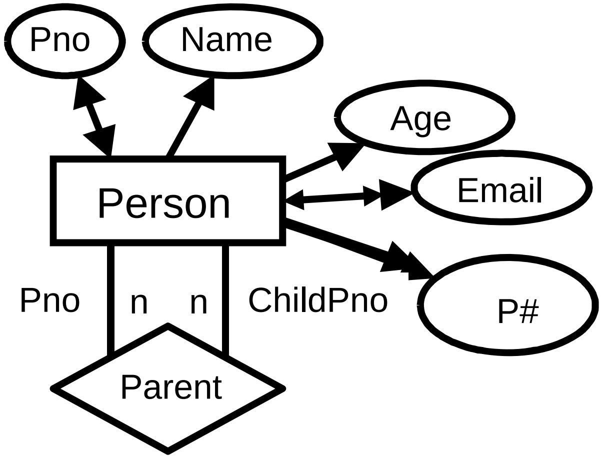
Person = { Person, PersonEmail, PersonPhone}

The object class Person has the following definition:

```
class Person {
        string Pno;
        string Name;
        integer Age;
        setof (string) Email;
        setof (integer) Phoneno;
};
```

where string, integer are data type and setof is a structure. Phoneno is a m:m multivalued attribute. Email is a 1:m multivalued attribute but cannot be expressed exactly. A person can have more than one email but an email is only owned by one person.

Note: The ER of the db is:



(2) Identifier dependency and complex object

- **ID-dependency** is a term from ER-approach.
- An entity type B is ID dependent on entity type A if B does not have its own key so that it has to depend on the identifier of A in order to be identified.

  e.g. Wards in hospital

- In OO term, B is a component object of the object A.

  e.g. Object Ward should be a component object of the object hospital. (Similar to ID weak entity type in ERD)

- In OODB, we say Ward **IS-PART-OF** hospital.

ID-Dependency Identification Rule

Let R_0 be a core relation with primary key K_0. Consider a relation R, with primary key K, that satisfies the followings:

(1) $K_0 \subset K$ and $R[K_0] \subseteq R_0[K_0]$.

(2) The primary key of R does not contain more than one disjoint foreign key.

(3) There exists a relation that references R or R has a non-prime attribute.

Then R is identified as a **core relation.** Moreover, object class of R is **ID-dependent** on object class of R_0 via the inclusion dependency

$$R[K'] \subseteq R0[K0].$$

Note: Without condition 3 in the above, the relation R will be taken as a component relation of relation R_0.
Condition 3 basically says that R qualifies to be an independent object class.

Example 2. We continue from Example 1 with the following extra relations and IND's:

Hospital (HName, address)
Ward (Hname, WardNo,#beds)
WardPatient (HName, Wardno, PatientPno)

Ward [HName] $\subseteq$ Hospital [HName]
WardPatient [HName, WardNo] $\subseteq$ Ward [HName, WardNo]
WardPatient [PatientPno] $\subseteq$ Person [Pno]

Apply the ID-dependency Rule: relations Ward and Hospital will be identified as core relations, and object class of Ward is ID-dependent object class of Hospital via

Ward [HName] $\subseteq$ Hospital [HName]

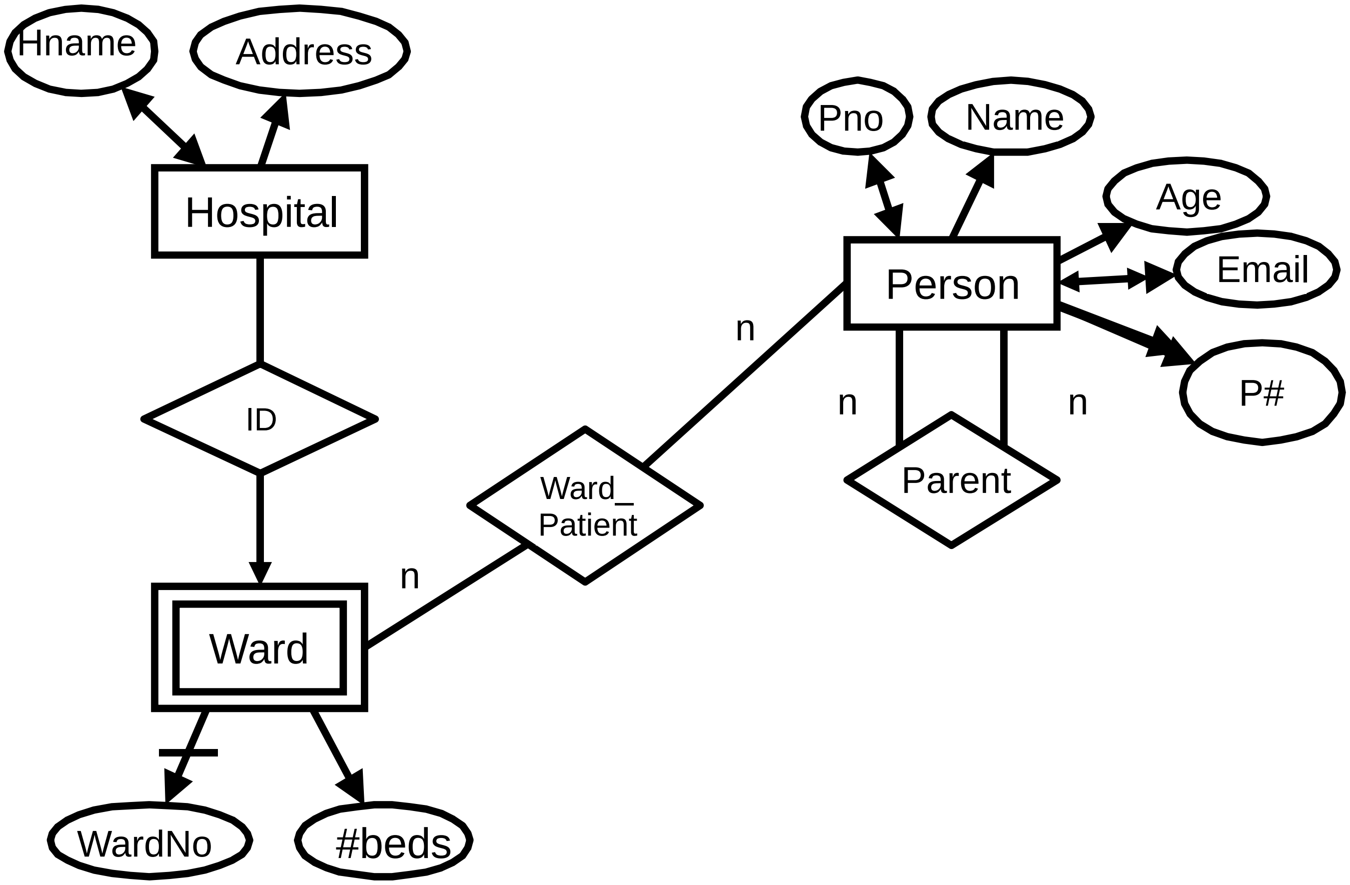
The translation will produce the following two class definitions:

```
class Hospital {
        string HName;
        string Address;
        own setof (Ward) WardNo;
        };

class Ward {
        Hospital HName;
        string WardNo;
        integer #beds;
        };
```

Note: The keyword "own" preceding the specification indicates the identifier dependency of Ward on Hospital. Also Wardno is a set of objects of Ward.

The corresponding ER diagram is:



(3) ISA Hierarchy and 1:1 relationship Type

After identifying all the core relations and their component relations, we want to identify the ISA relationship or 1:1 relationship type between object classes (core relations).

ISA Hierarchy and 1:1 relationship type Identification Rule

Consider two **core** relations R_1 and R_2. If there exist K_1 and K_2, keys in relations R_1 and R_2, resp., such that

$$R_1 [K_1] \subseteq R_2 [K_2]$$

then either one of the below cases is true for object classes R_1 and R_2.

(**Case 1**) R_1 ISA R_2 via $R_1 [K_1] \subseteq R_2 [K_2]$.

Note. If both R_1 ISA R_2 and R_2 ISA R_1 are true, then we should combine them to one object class.

(**Case 2**) There is a 1:1 relationship type between the object classes representing R1 and R2.

It is difficult to know which case is correct without additional information from user.

e.g. DB1={Person (nric, name, dob),
Student (s#, year, degree, nric)}
with Student[nric] $\subseteq$ Person[nric]

DB2={Mgr (m#, name, dob),
Dept (d#, name, location, m#)}
with Dept[m#] $\subseteq$ Mgr[m#]

These 2 RDBs and their inclusion dependencies (referential constraints) are isomorphic, cannot be distinguished between them without knowing the meanings of the attributes.

Note that DB1 is a ISA relationship however DB2 is a 1:1 relationship type.

Example 3. Consider the following relations and INDs.

Person (Pno, Name, Age)
PersonEmail (Email, Pno)
PersonPhone (Pno, Phoneno)
Employee (Eno, Pno, DateJoin)
Projstaff (ProjNo, Eno, Position)
SalaryHistory (Eno, Date, Amount)
Project (ProjNo, ProjName)

PersonEmail [Pno] $\subseteq$ Person [Pno]
PersonPhone [Pno] $\subseteq$ Person [Pno]
Employee [Pno] $\subseteq$ Person [Pno]
Projstaff [Eno] $\subseteq$ Employee [Eno]
Projstaff [ProjNo] $\subseteq$ Project [ProjNo]
SalaryHistory [Eno] $\subseteq$ Employee [Eno]

By applying the rules, we can identify 3 core relations with their component relations:

Person = { Person, PersonEmail, PersonPhone}
Project = { Project} /* No component relation */
Employee = { Employee, SalaryHistory}

and the following ISA relationship

Employee ISA Person

via Employee [Pno] $\subseteq$ Person [Pno].

Note: Projstaff is an inter-object relationship relation.

The object class definition are:

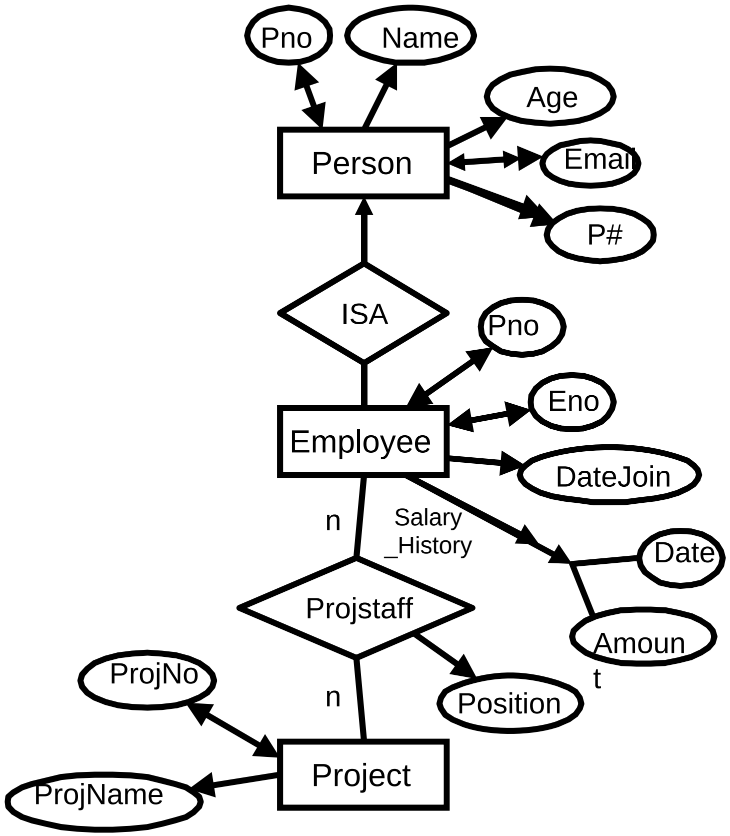
```
class Person {
        string Pno;
        string Name;
        integer Age;
        setof (string) Email;
        setof (integer) Phoneno;
        };

class Employee: isa Person {
        string Eno;
        Project ProjNo;
        DATE DateJoin;
        setof (tuple < DATE: Date, integer: Amount >)
                SalaryHistory;
};

class Project {
        string ProjNo;
        string ProjName;
        };
```

Note:
1. Employee ISA Person is represented in the class definition of Employee.
2. SalaryHistory is a set of tuples in Employee.

The corresponding ER diagram is:



Notes:

(1) Salary_History is a composite multivalued attribute of Employee. It could be represented as an weak entity set of Employee also.

(2) Projstaff is relationship type.

(4) Inter-object Relationships

Inter-object relationships may exist in relational database in 3 forms:

(1) A relation whose key consists of disjoint foreign keys.
(Similar to m:m relationship type in ER approach)

e.g. Projstaff (ProjNo, Eno, Position)

There is a m:m relationship type between Project and Employee.

(2) A relation which has non-prime(s) as a foreign key representing some object class.
(e.g. similar to m:1 relationship type in ER approach).

(1) **e.g.** Emp (Eno, Dno, DateJoinDept)

where non-prime Dno is a foreign key referencing Dept.

There is a m:1 relationship between Employee and Department.

Note that DateJoinDept is semantically dependent on Eno. It is a relationship attribute.

$$Eno \xrightarrow{sem} DateJoinDept$$

e.g. EMP' (Eno, Name, Dob, DateJoin, Dno) and Dno is a foreign key.

It is a **mixed relation** of a core relation and an inter-object relation.

Q: Is DateJoin an attribute of the employee or an attribute of the m:1 relationship type between employee and department?

e.g. EMP" (Eno, Name, Dob, Mgr#, Mname)

where Mgr# is a role name of Eno and Mname is a role name of Name, i.e. the name of the manager Mgr#.

EMP is a **mixed relation** of core relation (employee) and an inter-object relation (recursive relationship between employee and the manager).

(3) As all key relations (i.e. all attributes form the key of the relation) which contain more than one foreign key and some other attributes.

e.g. Progress (ProjNo, Eno, Date, ProgressReport)

where Date and ProgressReport is a composite multi-valued attribute of the relationship between Project and Employee.

Example 4. Consider the Projstaff in Example 3

Projstaff (ProjNo, Eno, Position)

ProjNo and Eno are foreign keys of object classes Project and Employee.

Relation Projstaff represents the **inter-object relationship** as follows:

```
class Projstaff {
        Project     ProjNo;
        Employee  Eno;
        string       Position;
      };
```

Note that the 2 attributes ProjNo and Eno are types Project and Employee resp., i.e., they are references (pointers, foreign keys). There is a m:m relationship type between Project and Employee.

Example 5. Consider the following

Student (Sno, Sname, Dno)

// this is a **mixed relation** of core and inter-object relations)

Department (Dno, DName)

StudentHobby (Sno, Hobby)

Student[Dno] $\subseteq$ Department [Dno]

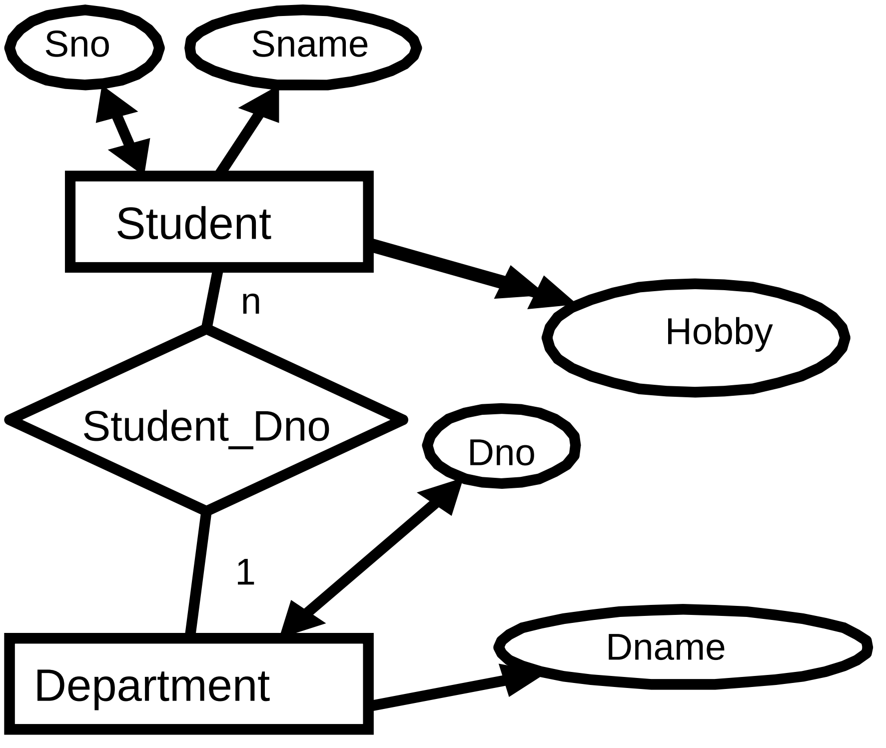
StudentHobby[Sno] $\subseteq$ Student[Sno]

The class definitions are:

```
class Student {
        string          Sno;
        string          Sname;
        setof (string)  Hobby;
        Department      Dno;
        };

class Department {
        string Dno;
        string DName;
        };
```

The ER diagram is:



Student_Dno is a m:1 relationship type.