Data Models
Past, Present, Future

Raj Sunderraman
Department of Computer Science
Georgia State University
June 2012

Data Models

1960s:

File systems, Access Methods

Hierarchical (IMS) and Network Data Models (IDMS)

1970s:

Relational Data Model (tables, SQL)

ER Conceptual Data Model

1980s:

Object-Oriented Data Model (ODL/OQL)

1990s:

XML Data Model (XPath, XQuery, XML Schema)

Late 2000s:

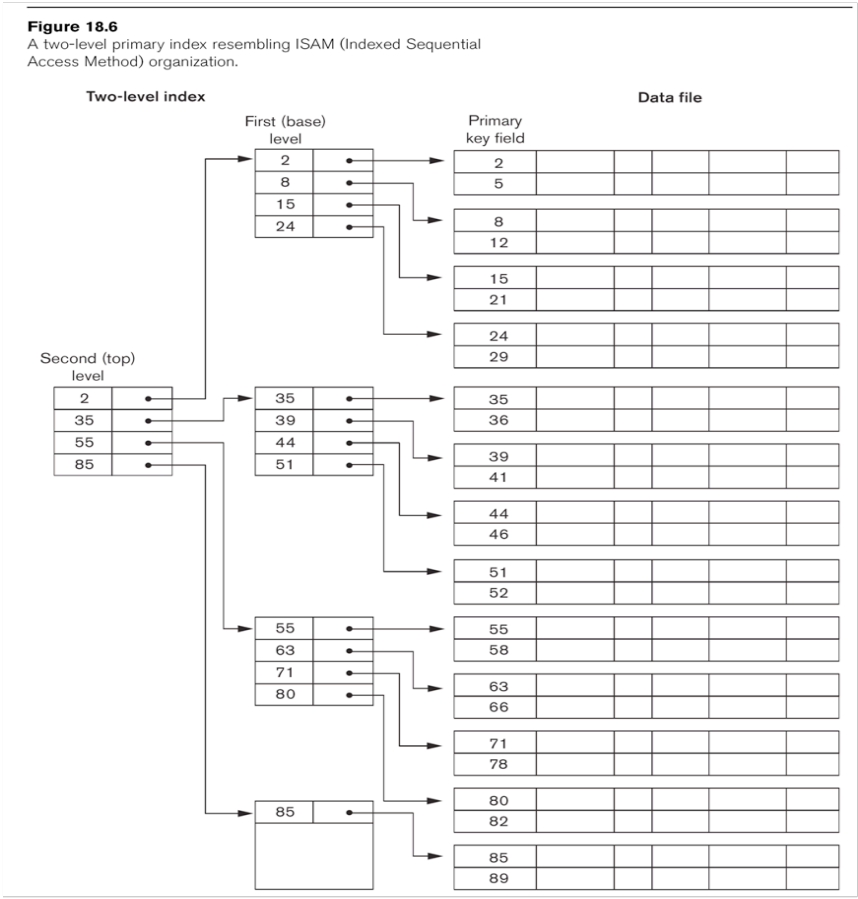
NoSQL (nosql-database.org)

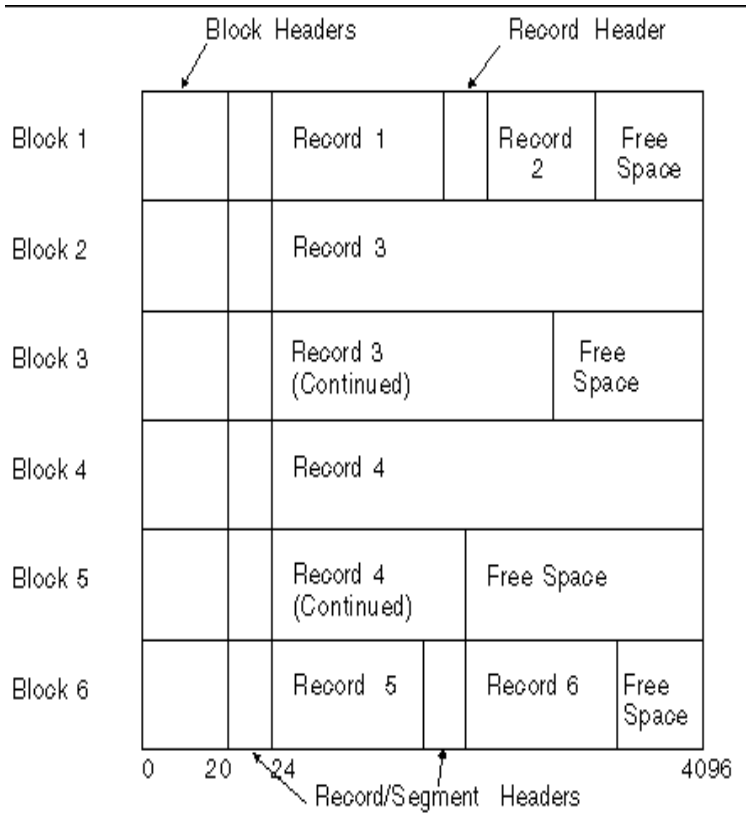
Google BigTable, Amazon Dynamo, Cassandra,

MongoDB, CouchDB

1960s – File Systems

IBM – ISAM (Indexed Sequential), VSAM (Virtual Storage)





1. Hierarchical Model

IMS (Information Management System) Data Model

(IBM product) **Ref**: CJ Date's book

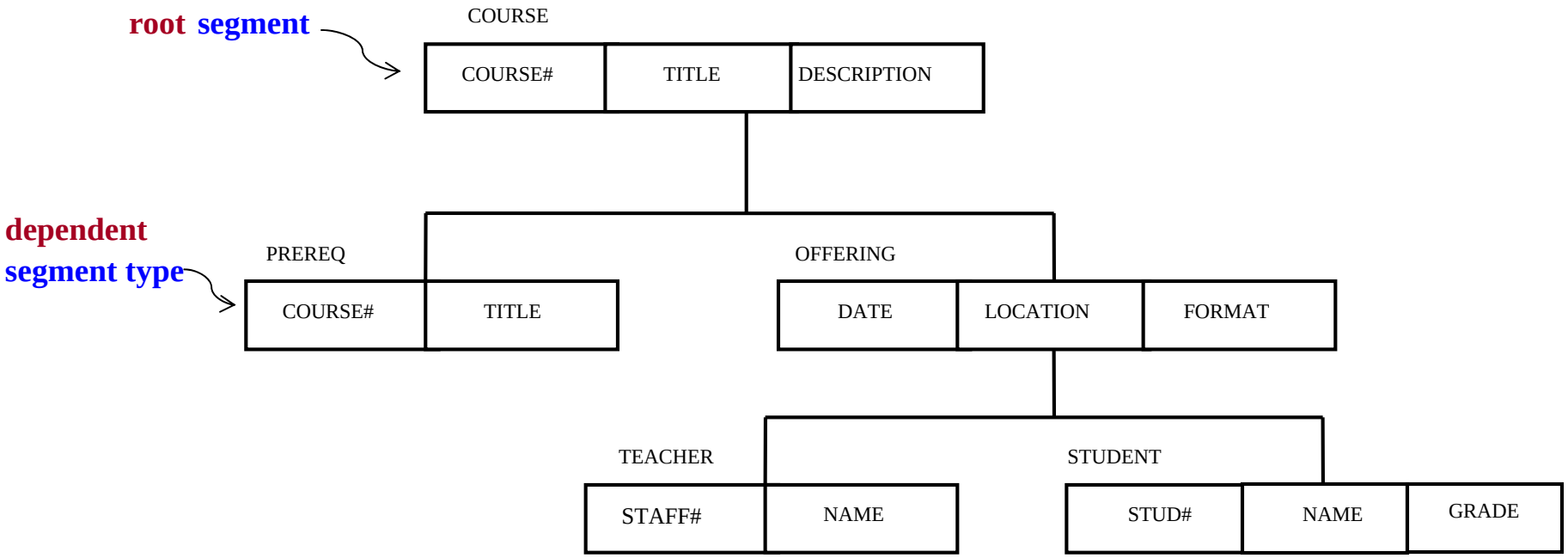


Fig. 1. PDBR (physical database record) type for the education database (schema)

IMS (Information Management System) Data Model (cont.)

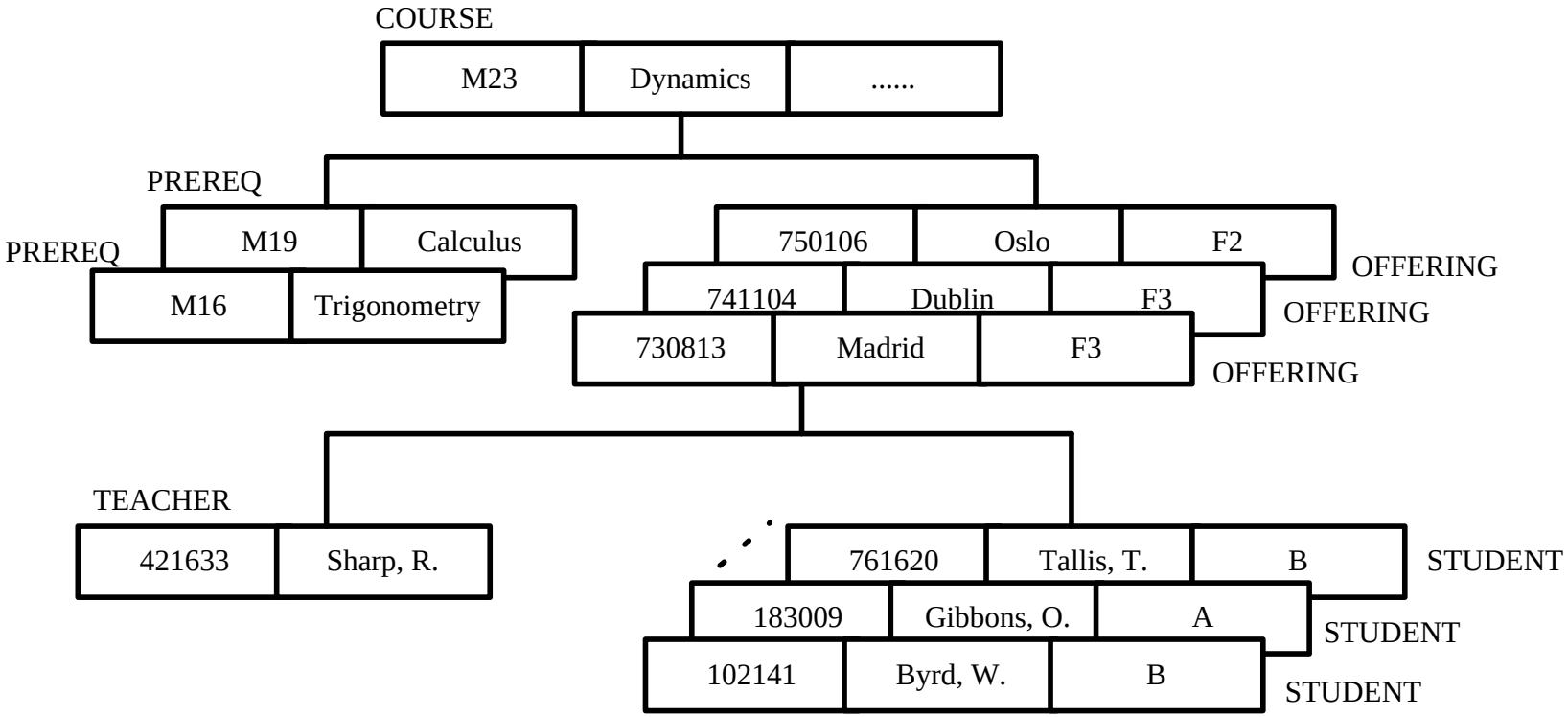


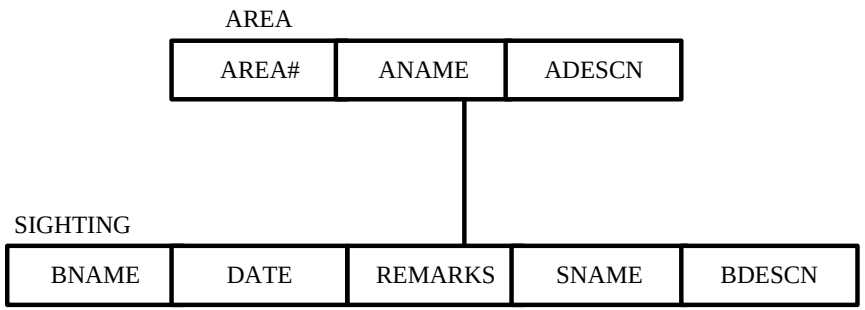
Fig. 2. Sample PDBR occurrence for the education database (database instance)

- **Note:** IMS is a hierarchical database model. It is similar to (but not exactly the same as) the XML data model.

Many-to-many (m:m) relationships

- Many-to-many relationships in hierarchical structure will contain redundant data
- IMS removes redundant data using logical parent pointers

Many-to-many Relationships using logical parent pointers



Note: The same type of birds may appear in different areas, so the relationship between AREA and SIGHTING is a m:m relationship. The SNAME and BDESCN of a bird BNAME will be replicated under different areas.

Fig. 3. Required record structure for the survey database (schema).

BNAME – name of a bird. SNAME – scientific name of the bird.

To remove redundant data, we first create another database to store the information of birds as shown below:

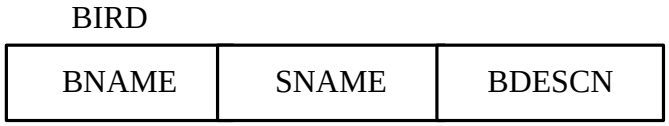


Fig. 4. Record structure of the bird database

Many-to-many Relationships using logical parent pointers (cont.)

We then redesign the schema in Fig 3 to the below using logical parent pointers.

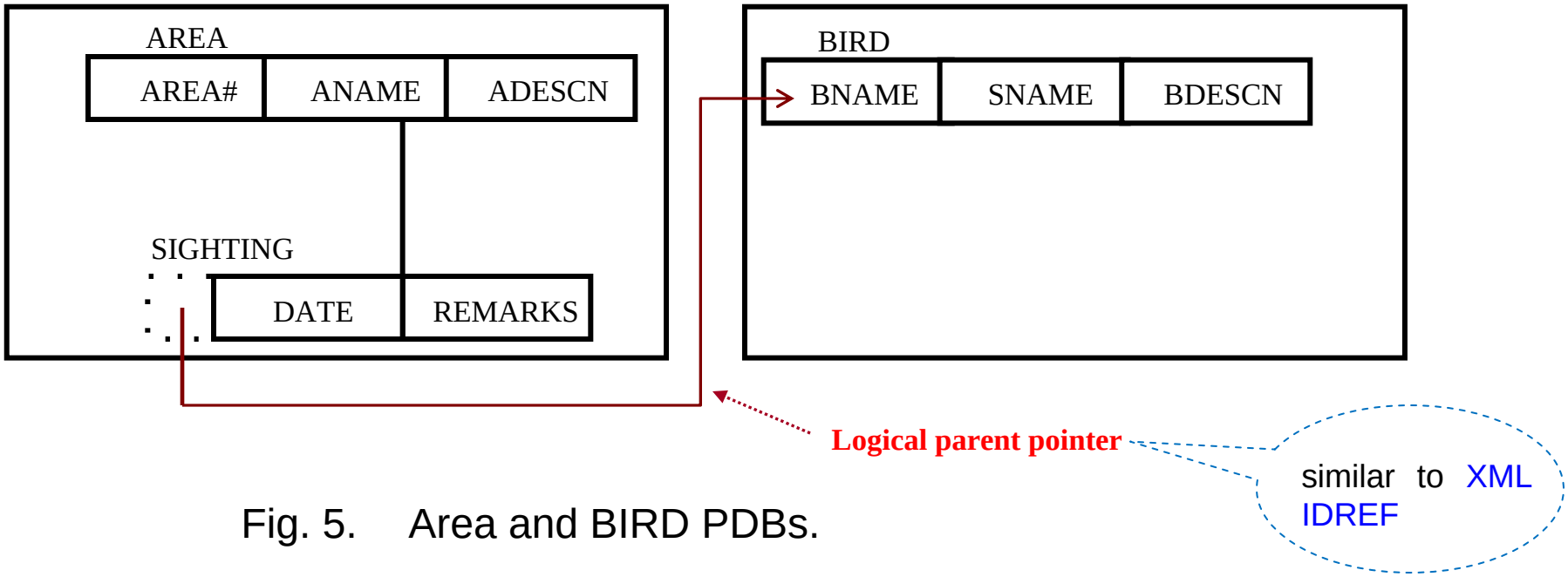


Fig. 5. Area and BIRD PDBs.

Note: DATE and REMARKS depend on AREA and SIGHTING

❖Note: Logical parent pointer is similar to XML IDREF.

2. Network Model

Network Model was proposed by **DBTG** (Database Task Group) in 1971.

Ref: CJ Dates' book. Software Product: IDMS

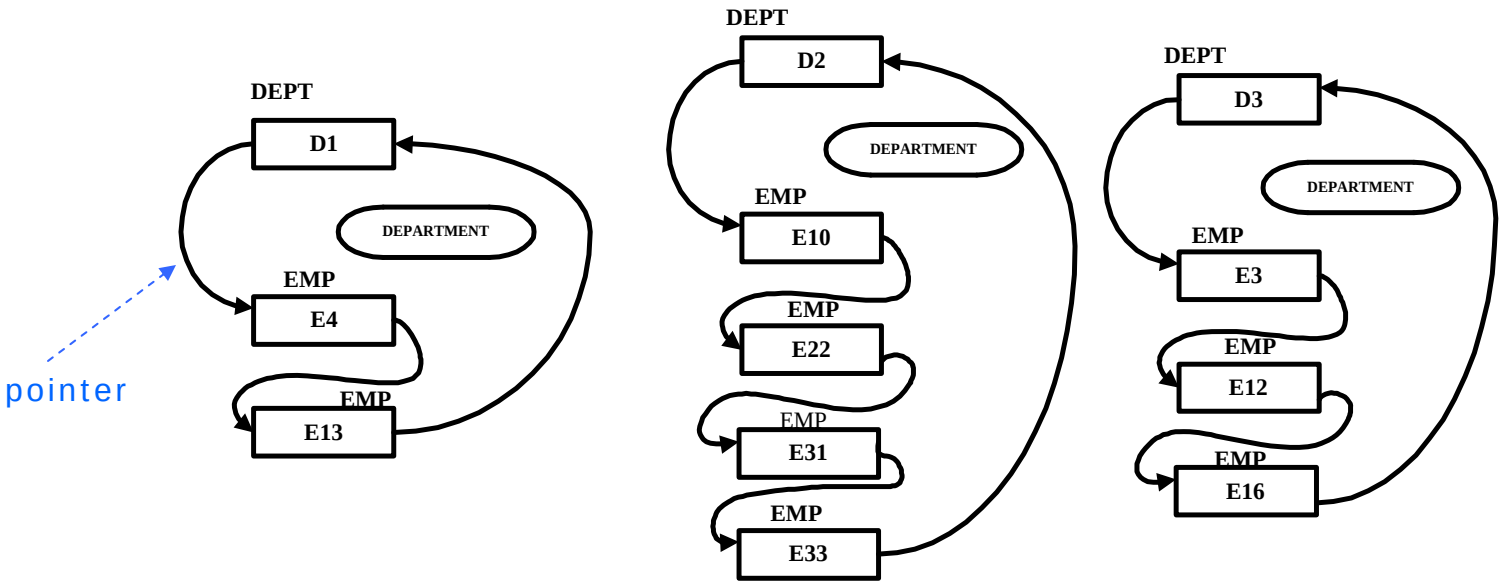


Fig. 9. A Department-employee database instance.

Note: Each employee only works for one department and a department may have many employees. The relationship between employee and department is a **many-to-one** relationships.

Network Model Data Structure

(schema diagram)

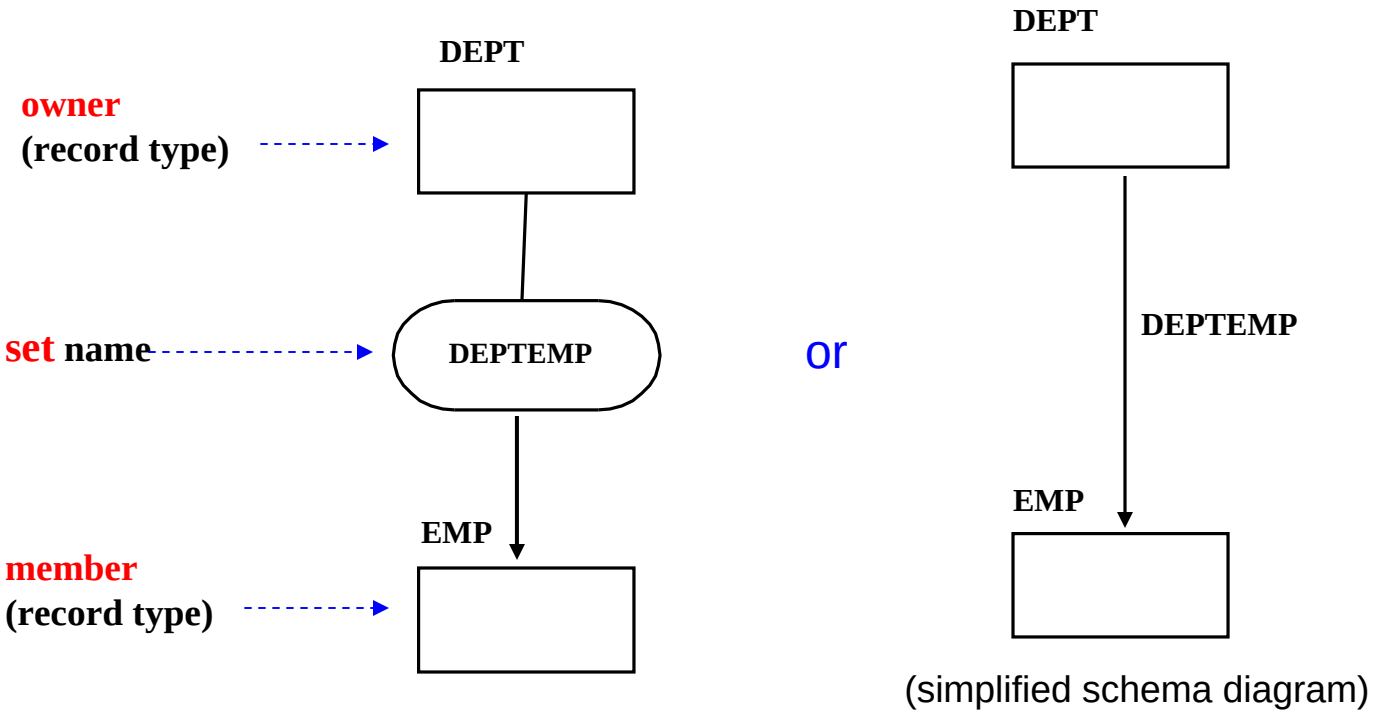


Fig. 10. Structure of the set DEPTEMP.

Each employee works for only one department

Note: A set is a **1:m relationship** from owner to member. E.g. a dept has many employees and each employee only works for one dept.

A three level network example

a record type may be both a **owner** and **member** of two **set types**

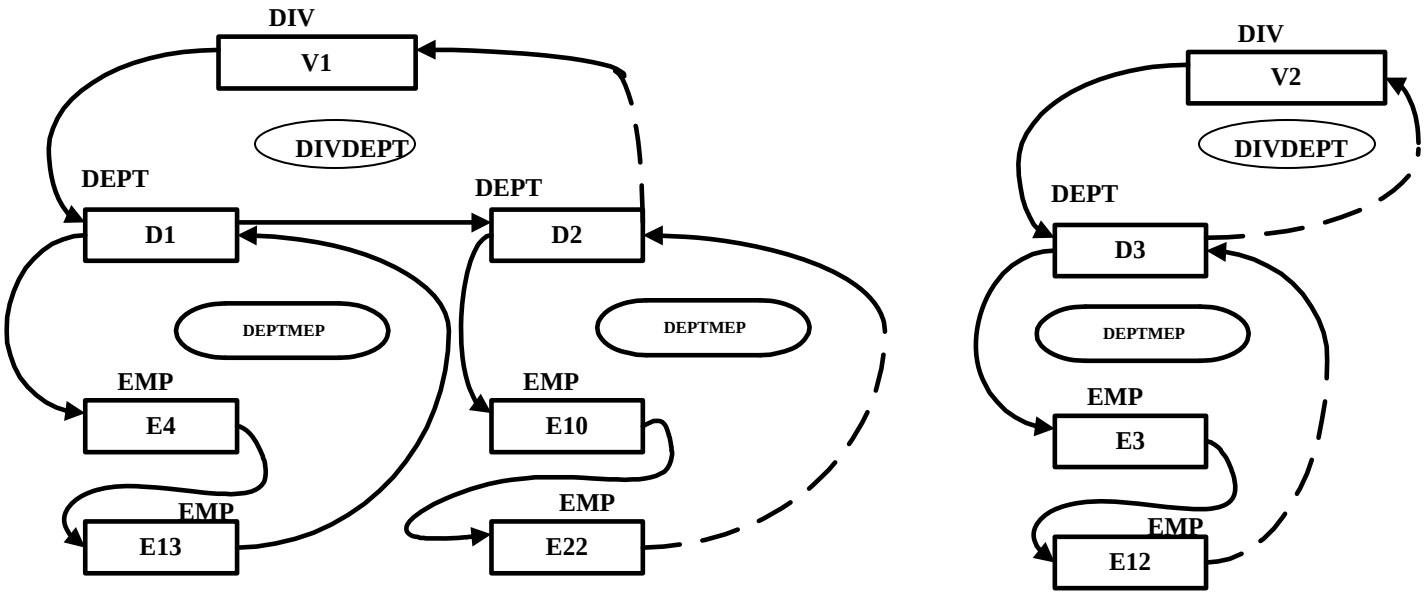


Fig. 11. A division-department-employee database instance.

Each department belongs to one Division. There are 2 divisions.

A three level network example (cont.)

A record type may be both a **owner** and **member** of two **set types**

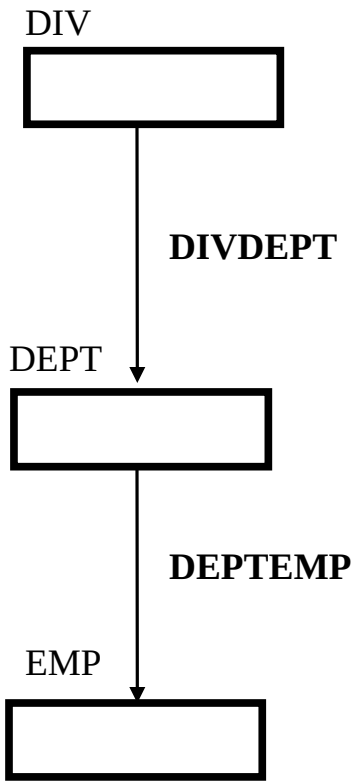


Fig. 12. Structure of the sets DIVDEPT and DEPTEMP

One owner with two members

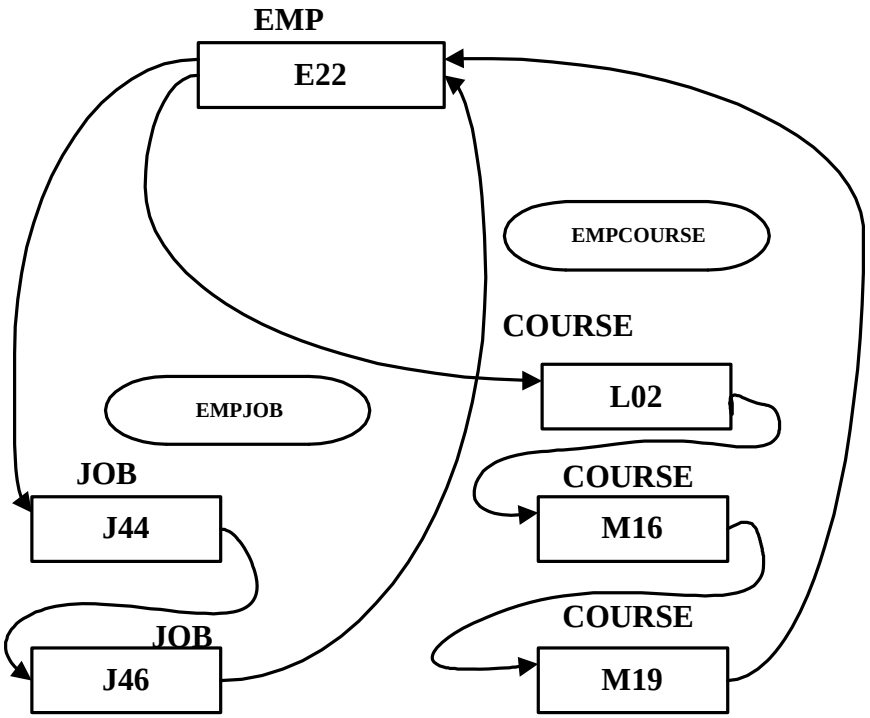


Fig. 13. An employee-history database instance

One owner with two members (cont.)

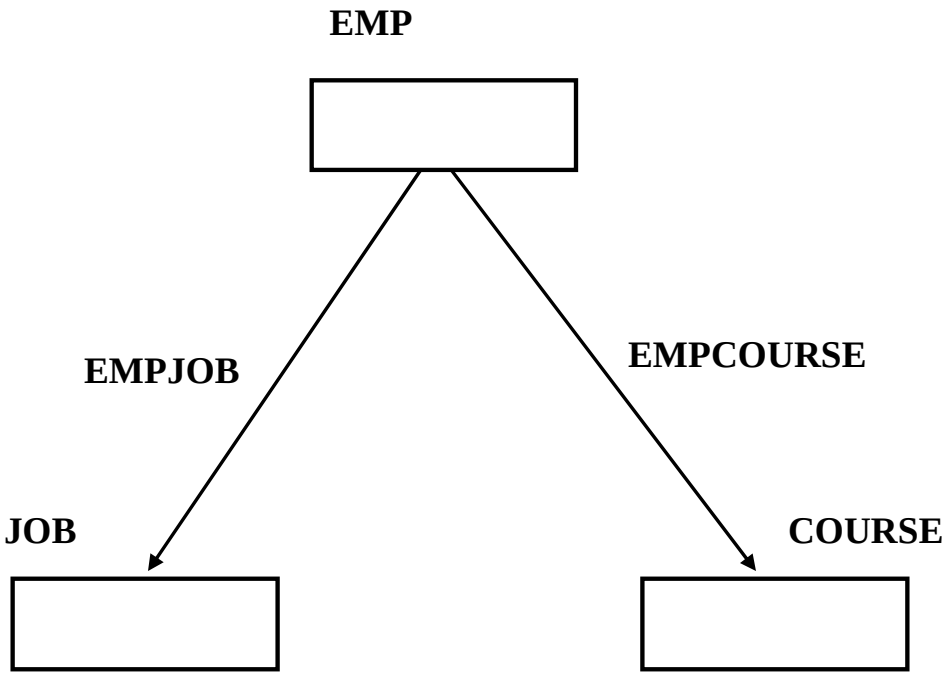


Fig. 14. Structure of the sets EMPJOB and EMPCOURSE

Many-to-many relationship

E.g. part and supplier relationship.

- Note: Network model cannot represent m:m relationships directly. A m:m relationship can be simulated by two 1:m relationships and a dummy record type (e.g. SP in Fig 15). Not a very nice solution!

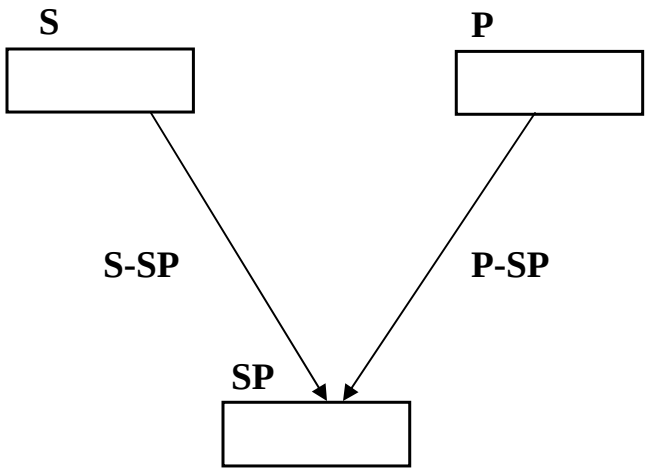


Fig. 15. Structure of the sets S-SP and P-SP.

Many-to-many relationship (cont.)

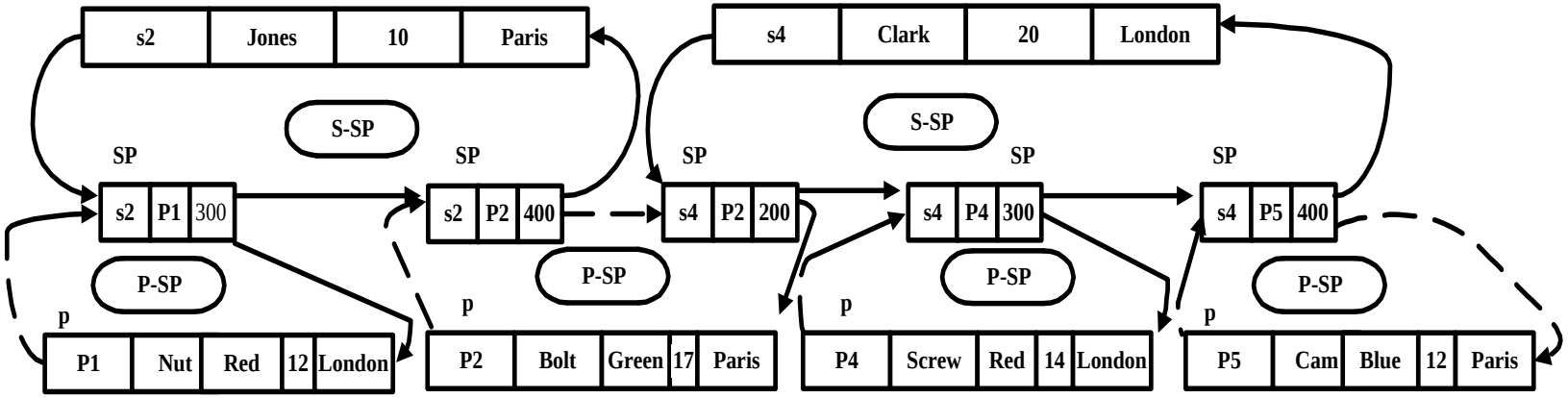


Fig. 16. A suppliers-and-parts database instance

- **Q:** How to represent 1:1 relationships? N-ary relationships?

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Figure 3.6
One possible database state for the COMPANY relational database schema.

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

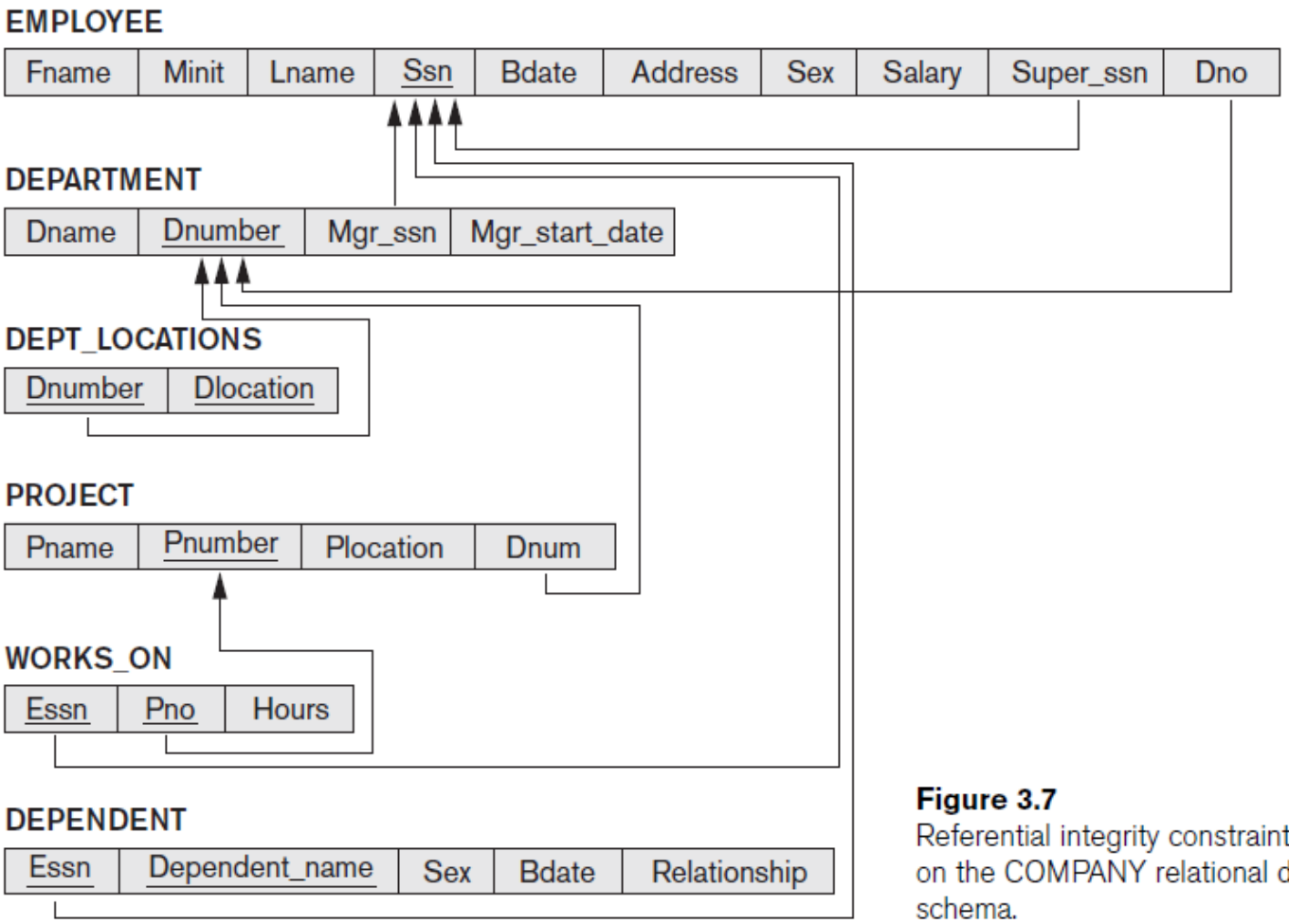


Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

SQL Queries

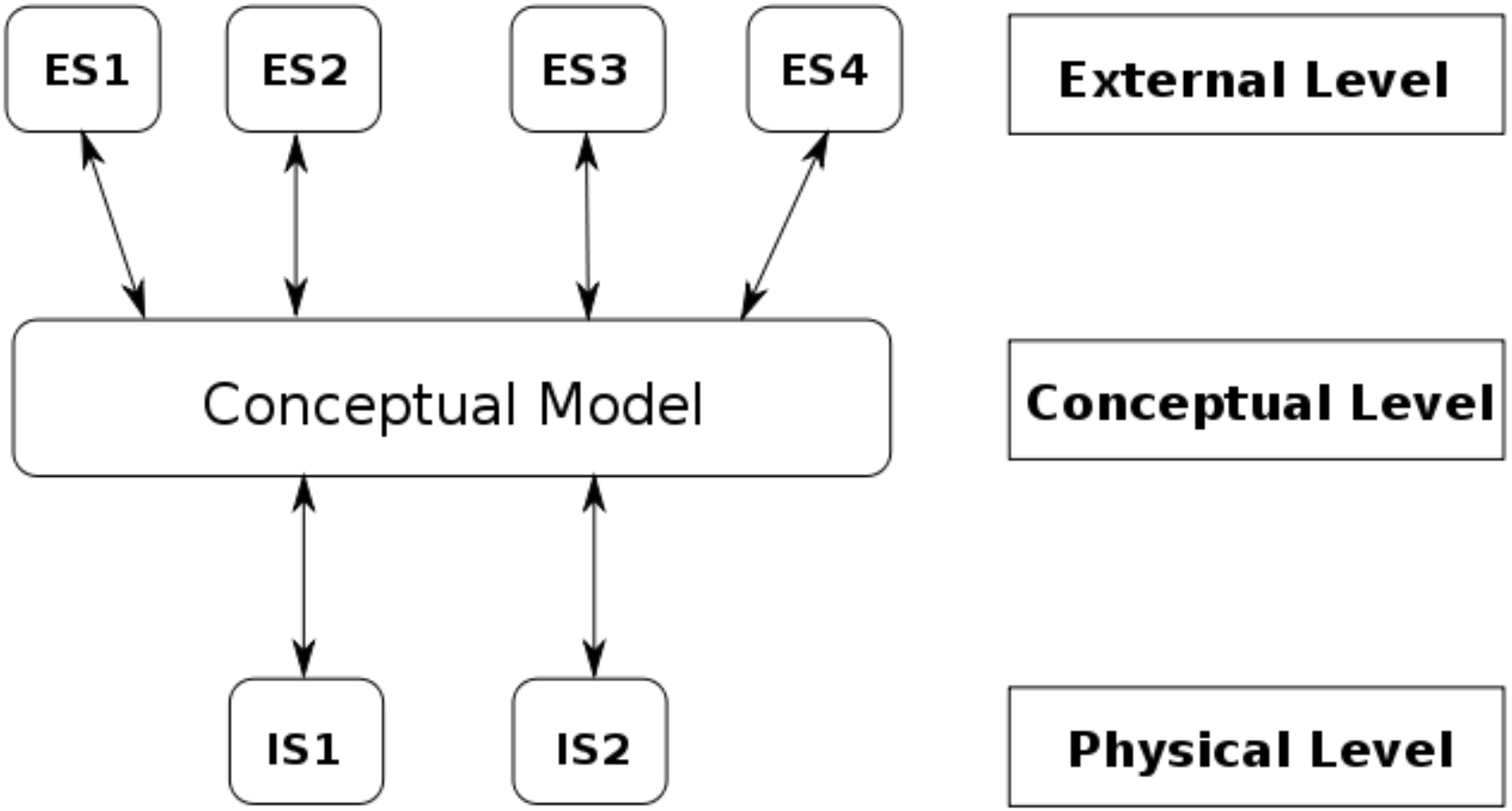
Find names of employees who work on every project controlled by department 5.

```
select lname,fname
from   employee
where  not exists (
      select *
      from   project b
      where  dnum=5 and not exists (
            select *
            from   works_on c
            where  ssn = c.essn and b.pno = c.pno));
```

For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

```
select pnumber,pname,count(*)
from   project,works_on
where  pnumber=pno
group by pnumber,pname
having count(*)>2;
```

3-Layered Architecture



Entity Relationship Model

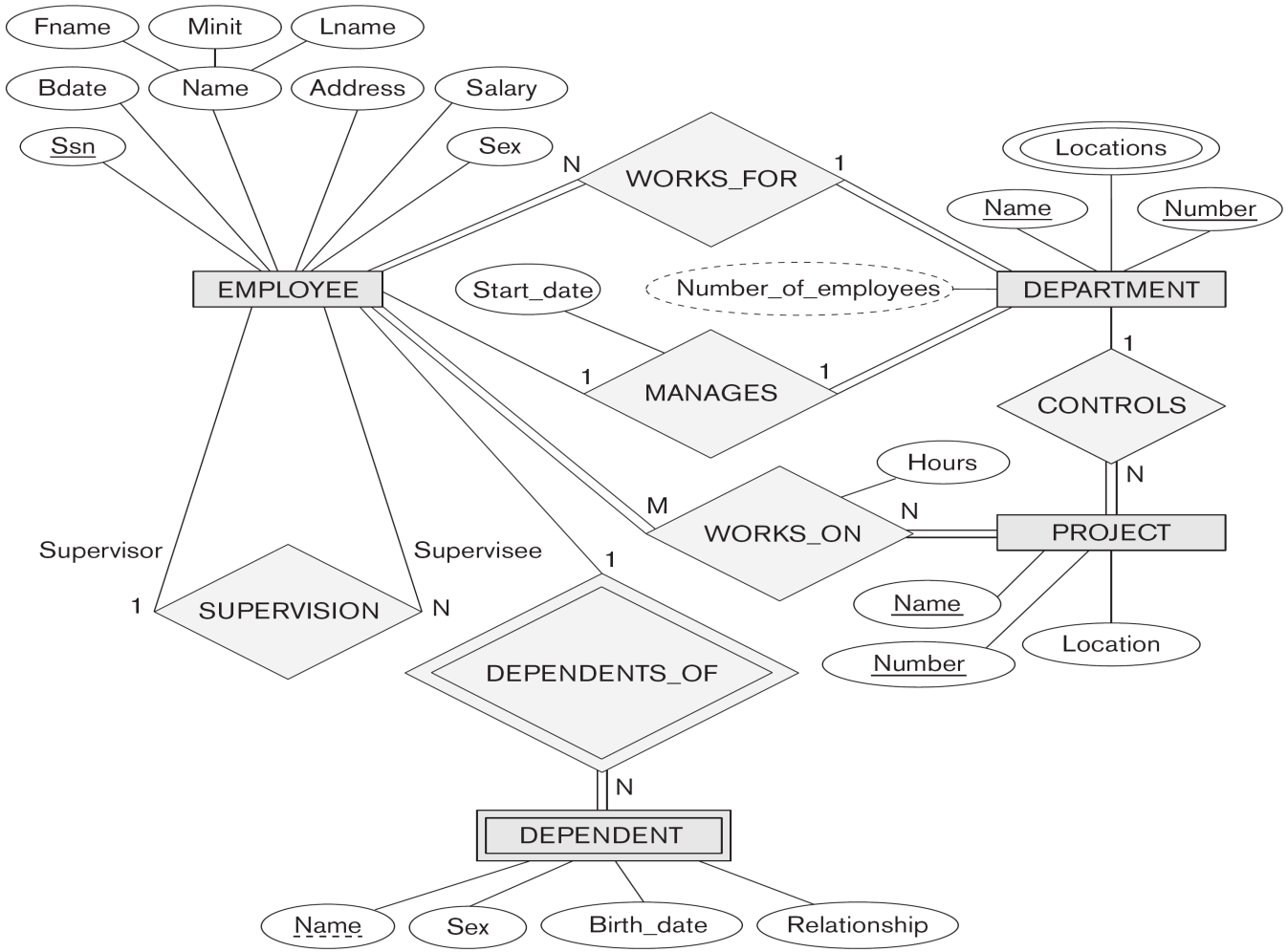


Figure 7.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.

ODMG, ODL and OQL

ODMG: Object Data Management Group (a standards group)

ODL: Object Description Language (schema description language)

OQL: Object Query Language (queries an OO database with an
ODL schema, similar to SQL)

ODL

ODL Classes (called interfaces): contain definitions for
- attributes
- relationships
- methods

Consider database about movies, stars and studios.

Movies have stars acting in them.
Stars may act in one or more movies.
Studios produce one or more movies.
Each movie is produced by one studio.

ODL Schema for Movie database

```
interface Movie {
  attribute string title;
  attribute integer year;
  attribute integer length;
  attribute enum Film {color,blackAndWhite} filmType;
  relationship Set<Star> stars inverse Star::starredIn;
  relationship Studio ownedBy inverse Studio::owns;
};
interface Star {
  attribute string name;
  attribute Struct Addr {string street, string city} address;
  relationship Set<Movie> starredIn inverse Movie::stars;
};
interface Studio {
  attribute string name;
  attribute strin address;
  relationship Set<Movie> owns inverse Movie::ownedBy;
};
```

OQL - continued

OQL Queries:

(1) Find the year of movie "Gone With the Wind"

```
select m.year
from Movies m
where m.title = "Gone With the Wind"
```

OQL - continued

Select-from-where statement in OQL is constructed as follows:

SELECT keyword followed by a list of expressions (using constants and variables defined in the FROM clause)

FROM keyword followed by a list of variable declarations
variable is declared by
- giving an expression whose value is a collection type (typically an extent; could be another select-from-where)
- followed by an optional AS keyword
- followed by the name of the variable

WHERE keyword followed by a boolean-valued expression (can use only constants and variables declared in the FROM clause);

The OQL query produces a bag of objects.

OQL - continued

(2) Find the names of the stars of "Casablanca"

```
select s.name
from  Movies m, m.stars s
where m.title = "Casablanca"
```

(3) Eliminating duplicates (distinct keyword)
Find the names of stars of "Disney" movies

```
select distinct s.name
from Movies m, m.stars s
where m.ownedBy.name = "Disney"
```

OQL - continued

Complex output type:

(4) Get set of pairs of stars living at the same address

```
select distinct Struct(star1: s1, star2: s2)
from Stars s1, Stars s2
where s1.addr = s2.addr and s1.name < s2.name
```

The result type of this query is

```
Set<Struct N {star1: Star, star2: Star}>
```

Note: Such a type cannot appear in an ODL declaration!

```
shortcut: select star1: s1, star2: s2
```

OQL - continued

Subqueries:

(5) Get the stars in movies made by Disney.

```
select m
from Movies m
where m.ownedBy.name = "Disney"
```

gives us the Disney Movies. This can be used in the from clause as follows:

```
select distinct s.name
from (select m
      from Movies m
      where m.ownedBy.name = "Disney") d, d.stars s
```

OQL - continued

Ordering the Result:

(6) Get Disney movies ordered by length (ties broken by title)

```
select m
from Movies m
where m.ownedBy.name = "Disney"
order by m.length, m.title
```

- asc or desc may be spevcified after order by (default is asc)

OQL - continued

```
Quantifier Expressions:
  for all x in S: C(x)
  exists x in S: C(x)

  (7) Get stars acting in Disney movies

    select s
    from Stars s
    where exists m in s.starredIn : m.ownedBy.name = "Disney"

  (8) Get stars who appear only in Disney movies

    select s
    from Stars s
    where for all m in s.starredIn : m.ownedBy.name = "Disney"
```

XML Data Model

Semi-Structured Data Model

Tree Structure

Querying via Xpath and Xquery

Go to http://tinman.cs.gsu.edu/~raj/elna-lab-2010

for examples

Late 2000s - NoSQL

- http://nosql-database.org/
- Non-relational
- Distributed
- Open source
- Horizontally scalable
- Schema-free
- Simple API
- Eventually consistent (BASE not ACID)
- Big Data

MongoDB

- A Mongo System holds a set of databases
- A database holds a set of collections
- A collection holds a set of documents
- A document is a set of fields
- A field is a key-value pair
- A key is a name (string)
- A value is a
  - Basic type (string, integer…)
  - A document
  - An array of values

```
> show dbs
    admin
    harmony-development
    harmony-test
    local
    ...
> use harmony-development
    switched to db harmony-development
> show collections
    accounts
    activities
    assets
    items
    ...
```

```
> db.accounts
harmony-development.accounts

> db.accounts.count()
1
```

```
> db.accounts.find().forEach(function(doc) {
  print(tojson(doc));
});
```

```
{
  "_id"          : ObjectId("4be97eaebcd1b30e86000003"),
  "title"        : "Ordered List",
  "creator_id"   : ObjectId("4be97eadbcd1b30e86000001"),
  "memberships" : [
    ObjectId("4be97eadbcd1b30e86000001"),
    ObjectId("4be97eaebcd1b30e86000002")
  ]
}
```

```
> use testing
switched to db testing

> db.colors.insert({name:'red',    primary:true})
> db.colors.insert({name:'green',  primary:true})
> db.colors.insert({name:'blue',   primary:true})
> db.colors.insert({name:'purple', primary:false})
> db.colors.insert({name:'orange', primary:false})
> db.colors.insert({name:'yellow', primary:false})
```

```
> var cursor = db.colors.find()
> cursor.next()
{
    "_id" : ObjectId("4bed7aeb0b4acd070c593ba6"),
    "name" : "red",
    "primary" : true
}
```

```
> cursor
{ "_id" : ObjectId("4bed7af40b4acd070c593ba7"), "name" : "green", "primary" : true }
{ "_id" : ObjectId("4bed7af80b4acd070c593ba8"), "name" : "blue", "primary" : true }
{ "_id" : ObjectId("4bed7b570b4acd070c593ba9"), "name" : "purple", "primary" : false }
{ "_id" : ObjectId("4bed7b6a0b4acd070c593baa"), "name" : "orange", "primary" : false }
{ "_id" : ObjectId("4bed7b7d0b4acd070c593bab"), "name" : "yellow", "primary" : false }
```

```
SELECT * from colors WHERE name = 'green'
```

```
> db.colors.find({name:'green'})
{ "_id" : ObjectId("4bed7af40b4acd070c593ba7"), "name" : "green", "primary" : true }
```

```
SELECT name from colors WHERE primary = 1
```

```
> db.colors.find({primary:true}, {name:true})
{ "_id" : ObjectId("4bed7aeb0b4acd070c593ba6"), "name" : "red" }
{ "_id" : ObjectId("4bed7af40b4acd070c593ba7"), "name" : "green" }
{ "_id" : ObjectId("4bed7af80b4acd070c593ba8"), "name" : "blue" }
```

```
> db.colors.find({name:/l/})
{ "_id" : ObjectId("4bed7af80b4acd070c593ba8"), "name" : "blue", "primary" : true }
{ "_id" : ObjectId("4bed7b570b4acd070c593ba9"), "name" : "purple", "primary" : false }
{ "_id" : ObjectId("4bed7b7d0b4acd070c593bab"), "name" : "yellow", "primary" : false }
```

```
> db.colors.find({primary:true}).sort({name:1}).limit(1)
{ "_id" : ObjectId("4bed7af80b4acd070c593ba8"), "name" : "blue", "primary" : true }
```

```
> db.colors.find({primary:true}).sort({name:-1}).limit(1)
{ "_id" : ObjectId("4bed7aeb0b4acd070c593ba6"), "name" : "red", "primary" : true }
```

```
> db.colors.find({primary:true}).sort({name:1}).skip(1).limit(1)
{ "_id" : ObjectId("4bed7af40b4acd070c593ba7"), "name" : "green", "primary" : true }
```

```
> db.people.insert({name:'John',  age:28})
> db.people.insert({name:'Steve', age:29})
> db.people.insert({name:'Steph', age:27})
```

```
SELECT * from people WHERE age > 27
```

```
> db.people.find({age: {$gt: 27}})
{ "_id" : ObjectId("4bed80b20b4acd070c593bac"), "name" : "John", "age" : 28 }
{ "_id" : ObjectId("4bed80bb0b4acd070c593bad"), "name" : "Steve", "age" : 29 }
```

```
SELECT * from people WHERE age <= 27
```

```
> db.people.find({age: {$lte: 27}})
{ "_id" : ObjectId("4bed80c10b4acd070c593bae"), "name" : "Steph", "age" : 27 }
```

$gt
$gte
$lt
$lte
$ne
$in
$nin
$mod
$all
$size
$exists
$type
$elemMatch
$not
$where

Aggregation

http://www.mongodb.org/display/DOCS/Aggregation

```
> db.colors.count()
6
> db.colors.count
({primary:true})
3
```

```
> db.colors.distinct('name')
[ "blue", "green", "orange", "purple", "red", "yellow" ]
```

```
> db.people.distinct('name', {age:28})
[ "John" ]
```

```
> db.items.insert({title:'Home',         template:'home'})
> db.items.insert({title:'What We Do',   template:'page'})
> db.items.insert({title:'Our Writing', template:'page'})
> db.items.insert({title:'Who We Are',   template:'page'})
> db.items.insert({title:'Hire Us',      template:'page'})
```

```
> var key     = {template: true};
> var initial = {count:0};
> var reduce  = function(obj, prev) { prev.count += 1; };

> db.items.group({key:key, initial:initial, reduce:reduce})
[
  {"template" : "home", "count" : 1},
  {"template" : "page", "count" : 4}
]
```

```
> db.items.insert({tags: ['dog', 'cat']})
> db.items.insert({tags: ['dog']})
> db.items.insert({tags: ['dog', 'mouse']})
> db.items.insert({tags: ['dog', 'mouse', 'hippo']})
> db.items.insert({tags: ['dog', 'mouse', 'hippo']})
> db.items.insert({tags: ['dog', 'hippo']})
```

```
> var map = function() {
    this.tags.forEach(function(t) {
      emit(t, {count: 1});
    });
}
```

```
> var reduce = function(key, values) {
    var count = 0;
    for(var i=0, len=values.length; i<len; i++) {
      count += values[i].count;
    }
    return {count: count};
}
```

```
> var result = db.items.mapReduce(map, reduce);
```

```
> var result = db.items.mapReduce(map, reduce);
> result
{
    "ok"           : 1,
    "timeMillis"   : 86,
    "result"       : "tmp.mr.mapreduce_1273861517_683",
    "counts"       : {
        "input"    : 6,
        "emit"     : 13,
        "output"   : 4
    }
}
```

```
> db[result.result].find()
{ "_id" : "cat",   "value" : { "count" : 1 } }
{ "_id" : "dog",   "value" : { "count" : 6 } }
{ "_id" : "hippo", "value" : { "count" : 3 } }
{ "_id" : "mouse", "value" : { "count" : 3 } }
```