01 概 述
Introduction

数据库

- Database
  - 长期储存在计算机内的、有组织的、可共享的、互相关联的数据的集合
  - 是一个持久数据的集合
  - A very large, integrated collection of data.

数据库管理系统

- DataBase Management System(DBMS)
- A software package designed to store and manage databases.
- Provide efficient, reliable, convenient, and safe multi-user storage of and access to massive amounts of persistent data.

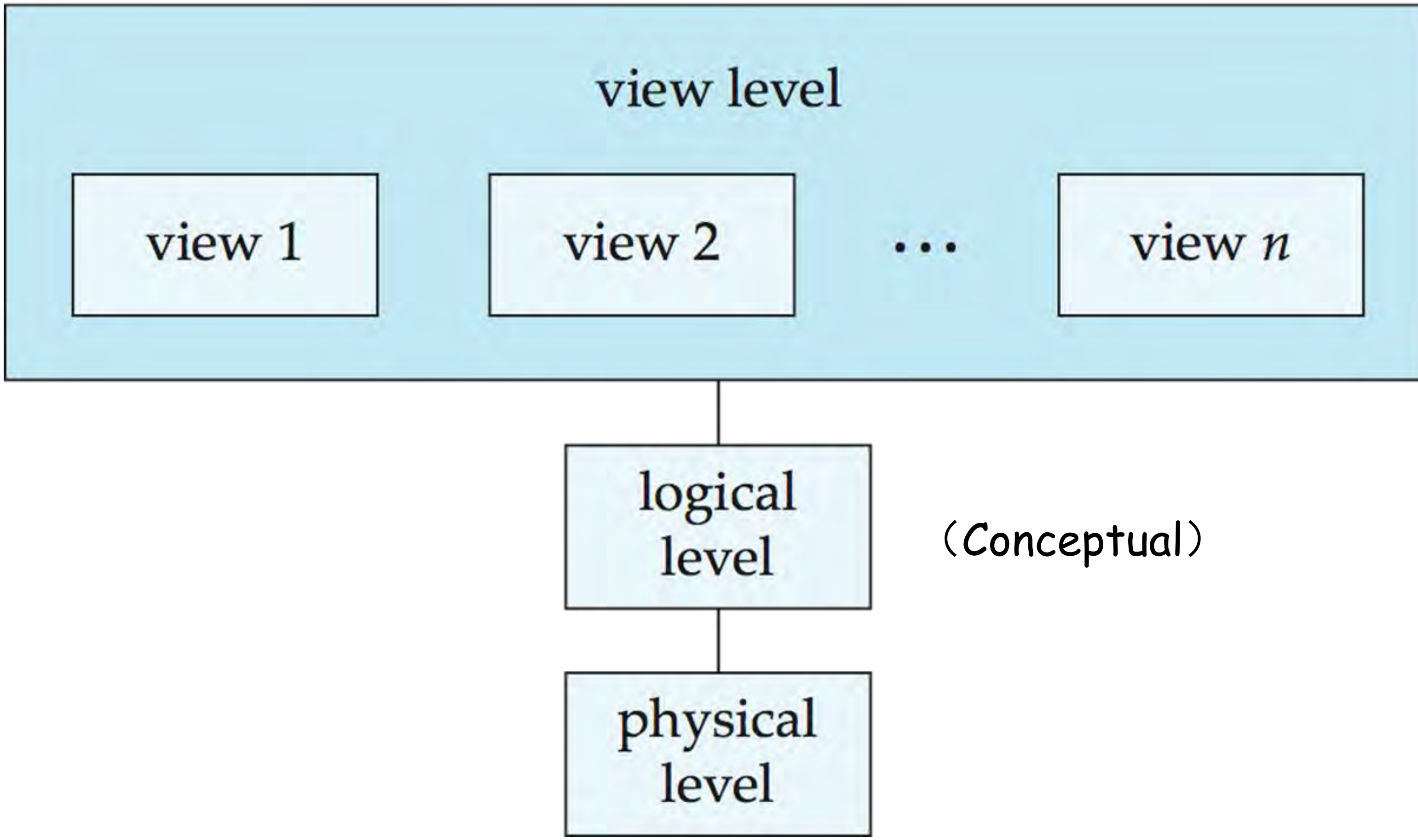
DBMS

- Massive
- Persistent
- Safe
- Multi-user
- Convenient
- Efficient
- Reliable

Key concepts

- Data model
  - A collection of concepts for describing data
  - A description of, in general, how the data is structured.
- Schema versus data
  - A description of a particular collection of data, using the given data model
  - The schema sets up the structure of the database.
- Data definition language (DDL)
- Data manipulation or query language (DML)

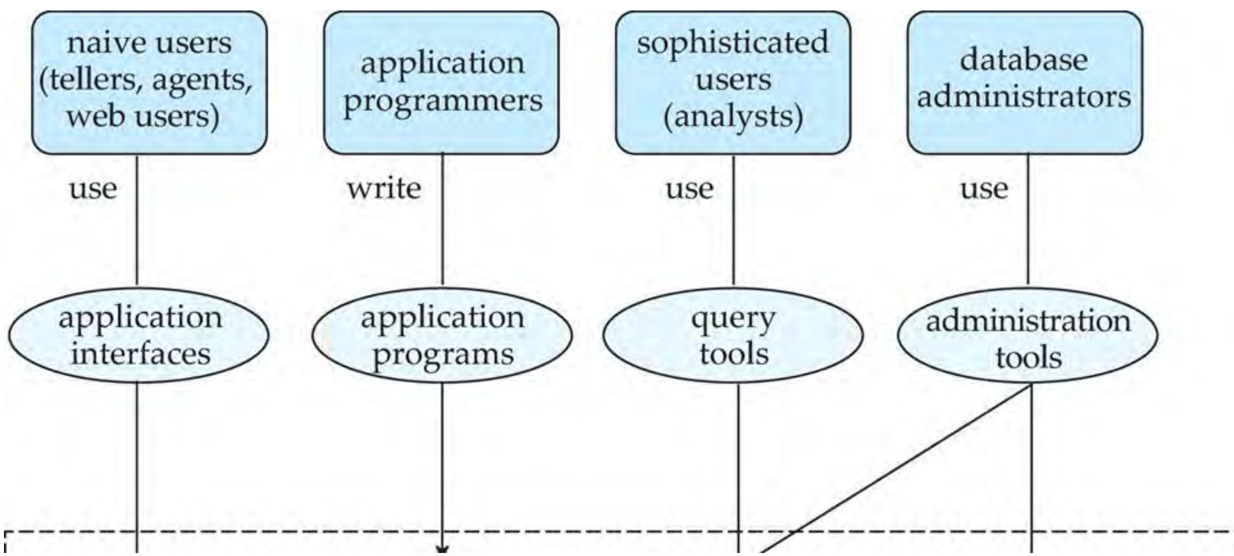
数据抽象



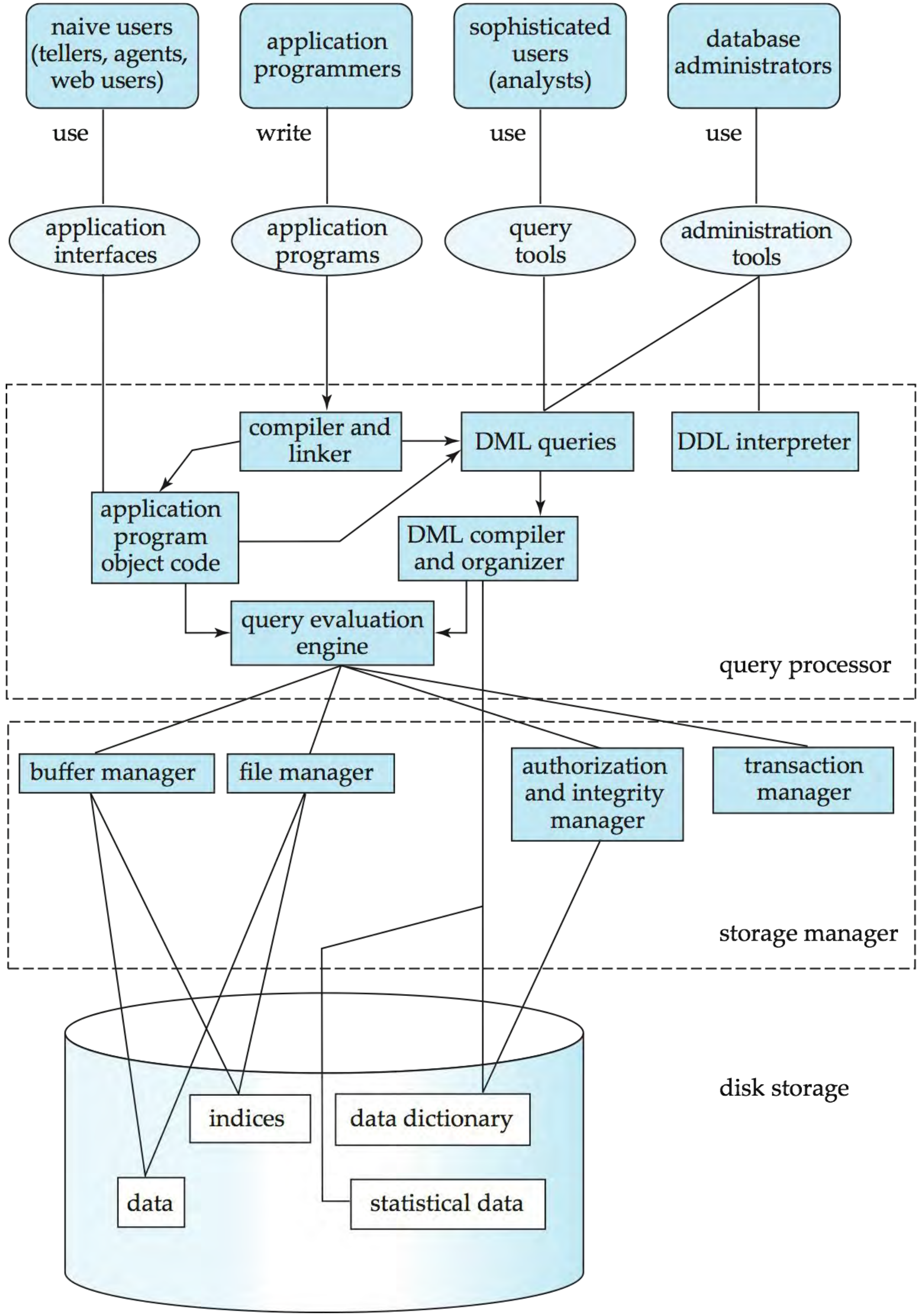
（Conceptual）

People

- DBMS implementer
- Database designer
- Database application developer
- Database administrator



DBMS系统结构


naive users (tellers, agents, web users)
application programmers
sophisticated users (analysts)
database administrators
use
write
use
use
application interfaces
application programs
query tools
administration tools
compiler and linker
DML queries
DDL interpreter
application program object code
DML compiler and organizer
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
disk storage
indices
data dictionary
data
statistical data

数据库应用

Whether you know it or not,
you're using a database every day

……

02 关系模型
Relational Model

关系数据模型基础

- 关系模型是一种基于表的数据模型
  - 用表的集合表示数据及数据间的联系

- **Schema**
  –structural description of relations in database

  Student(<u>sno</u>, sname, age, gender, dept）

- **Instance**
  –actual contents at given point in time

  Student(95001,"张三",21,"M","SE")

- Database = set of named **relations**(**tables**)

- Each relation has a set of named **attributes** (**columns**)

- Each **tuple**(**row**) has a value for each attribute

- Each attribute has a **type**(**domain**)

- **Key（键、码）**
  - attribute whose value is unique in each tuple
  - Or set of attributes whose combined values are unique
- Super Key（超码）
- Candidate Key（候选码）
- Primary Key（主码）
- Foreign key（外码）
  - Value in one relation must appear in another
  - Referencing relation
  - Referenced relation

student

sno	sname	age	gende	dept
001	张三	21	M	SE
002	李四	20	F	CS

course

cno	cname	credit
222	SE	3
223	DB	3

sc

sno	cno	grade
001	223	80
002	222	86
001	222	82

Example: University

- **大学分成多个系**
  - 每个系由自己唯一的名字（**dept_name**）来标识
  - 座落在特定的建筑物（**building**）中
  - 有它的经费预算（**budget**）
- **每一个系有一个开设课程列表**
  - 每门课程有课程号（**course id**）
  - 课程名（**title**）
  - 系名（**dept name**）
  - 学分（**credits**）
  - 还可能有先修要求（**prerequisites**）

Example: University

- **教师**由个人唯一的标识号（**ID**）来标识
  - 每位教师有姓名（**name**）
  - 所在的系（**dept name**）
  - 工资（**salary**）
- **学生**由个人唯一的标识号（**ID**）来标识
  - 每位学生有姓名（**name**）
  - 主修的系（**dept name**）
  - 已修学分数（**tot_cred**）

Example: University

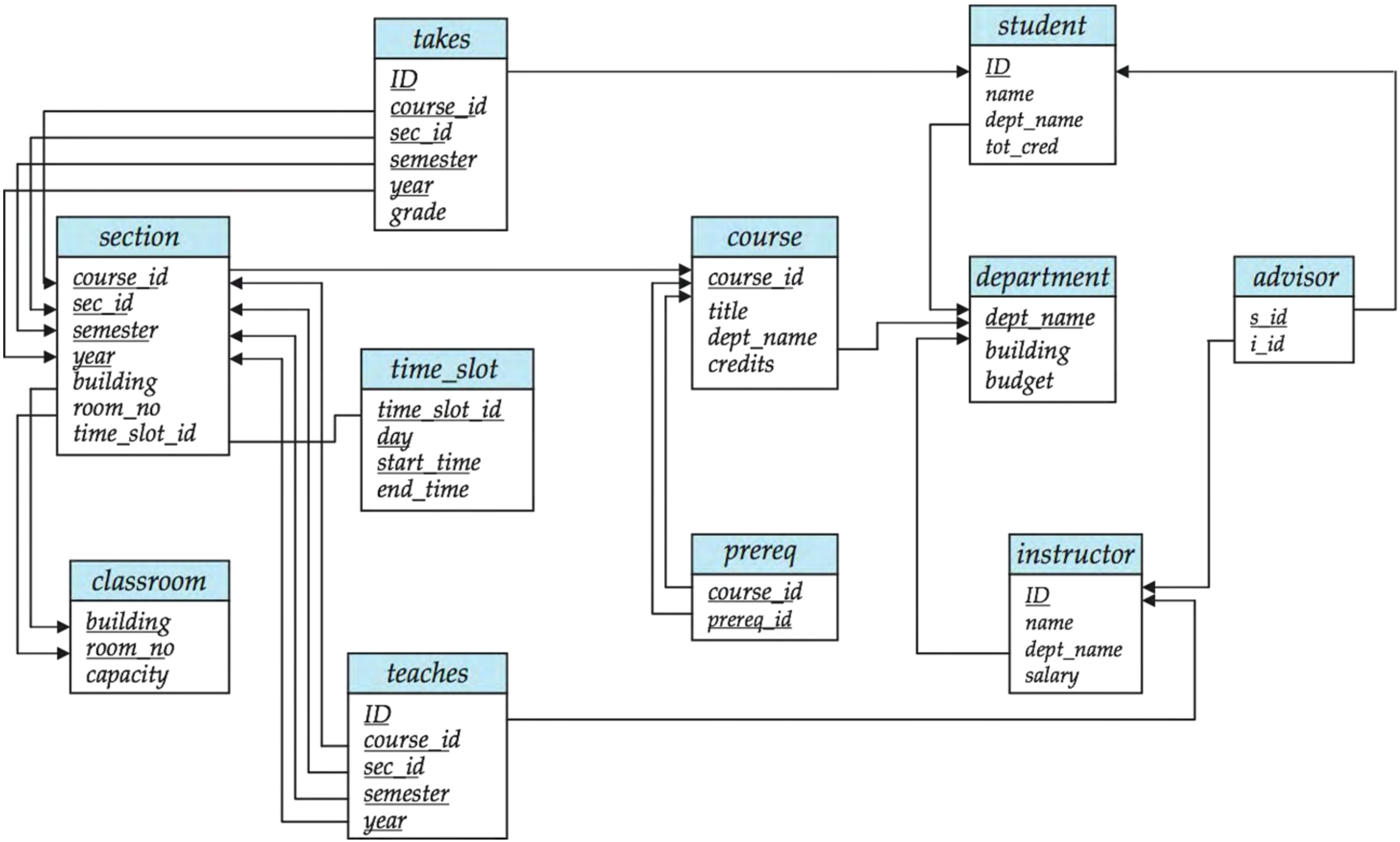
- 大学维护一个**教室**列表
  - 楼名（building）
  - 房间号（room_number）
  - 容量（capacity）。
- 大学维护开设的所有课程（**开课**）的列表
  - 每次开课由课程号（course id）、开课号（sec_id）、年（year）、学期（semester）来标识
  - 与之相关联的有学期（semester）、年（year）、楼名（building）、房间号（room_number）、时段号（time_slot_id，即上课的时间）。

Example: University

- 系有一个**教学任务**列表，说明每位教师的授课情况。
- 大学有一个所有**学生课程注册**的列表，说明每位学生在哪些课程的哪次开课中注册了。

- 一个真正的大学数据库会比上述的设计复杂得多。然而，我们就用这个简化了的模型来帮助你理解概念思想，避免你迷失在复杂设计的细节中。

Example: University



Example: University

Classroom (building, room number, capacity)
Department (dept_name, building, budget)
Course (course id, title, dept_name, credits)
Instructor (ID, name, dept_name, salary)
Section (course id, sec id, semester, year, building, room_number, time_slot_id)
Teaches (ID, course id, sec id, semester, year) **student** (ID, name, dept_name, tot_cred)
Takes (ID, course id, sec id, semester, year, grade)
Advisor (s_ID, i_ID)
time_slot (time slot id, day, start time, end_time)
Prereq (course id, prereq id)

关系代数

关系代数基础

- 关系数据模型的操作
  - 查询、修改
- 查询语言
  - 用户用来从数据库中请求获取信息的语言
  - 过程化语言（what & how）
  - 非过程化语言（what）
- 关系代数
  - 过程化查询语言

- Set
- Bag
- List

关系代数

- Six basic operators
  - select:
  - project:
  - union:
  - set difference: –
  - Cartesian product: x
  - rename:
- The operators take one or two relations as inputs and produce a new relation as a result.

- **select:**

$$_p(r) = \{t \mid t \quad r \text{ and } p(t)\}$$

- **project:**

$$\prod_{A_1, A_2, \ldots, A_k} (r)$$

- **Cartesian product: x**

$$r \times s = \{t\,q \mid t \quad r \text{ and } q \quad s\}$$

将多个relation组合（join）在一起

- **union:**

  r s = {t | t r or t s }

  在r中，或在s中的tuple

试列出所有师生员工的姓名（Student，Instructor）

X?

∪?

- set difference: –

$$r - s = \{t \mid t \quad\quad r \text{ and } t \quad\quad s\}$$

在r中，不在s中的tuple

试列出没有选课成绩的学生的学号

- **rename:**

关系代数

additional operations

- Set intersection $\cap$
- Natural join $\bowtie$
- Assignment
- Outer join ⟗

- Set intersection

$$E_1 - E_2 = E_1 - (E_1 - E_2)$$

- Natural join $\bowtie$ （自然连接）

$r \bowtie s$

$=\Pi_{schema(r) \cup schema(s)}\sigma_{r.a_1=s.a_1 \wedge r.a_2=s.a_2 \wedge \ldots}(r \times s)$

$schema(r) \cap schema(s)=(a_1,a_2,\ldots)$

（属性的交集上取等值）

- 试列出选过课程的学生的学号、姓名

- 试列出没有选课记录的学生的学号、姓名

- **Theta join $\bowtie_\theta$ or X_θ**

  $$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

  （条件连接）

- Outer join $⟗$

  外连接，匹配连接 及 不匹配连接

  左外 $⟕$

  右外 $⟖$

  全外 $⟗$

所有学生的选课情况

所有课程的被选情况

- **Assignment**

  赋值

  $p \qquad r \bowtie s$

  insert $r \qquad r \cup t_1$

  delete $r \qquad r - t1$

Aggregate Functions and Operations

- **Aggregation function (聚集函数)** takes a collection of values and returns a **single value** as a result.

avg:	average value
min:	minimum value
max:	maximum value
sum:	sum of values
count:	number of values

Aggregate Functions and Operations

- **Aggregate operation** in relational algebra

$$_{G_1, G_2, \ldots, G_n}\ \mathcal{G}\ _{F_1(A_1), F_2(A_2, \ldots, F_n(A_n)}(E)$$

  E is any relational-algebra expression
  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name
- Note: Some books/articles use instead of $\mathcal{G}$ (Calligraphic G)

Aggregate Functions and Operations

- 统计男女生人数
- 分别计算男女生平均年龄
- 列出学生数大于1000人的院系名称

Aggregate Functions and Operations

- Null,不参与
- for r(a,b),
  - sum(a) – sum(b) = sum(a – b) ?
  - count(*),count(a),count(b)?
- count(name),count(distinct name)?

基本关系运算

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary>=85000}}(instructor)$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,\ salary}(instructor)$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
⋈ (Natural Join)	$instructor \bowtie department$
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
× (Cartesian Product)	$instructor \times department$
	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
∪ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the two input relations.

SQL

(Structured Query Language)

SQL
Structured Query Language

- 读作“S.Q.L.” or “sequel”
- Supported by all major commercial database systems
- Standardized – many new features over time
- Interactive via GUI or prompt, or embedded in programs
- Declarative, based on relational algebra

History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86 (First formalized by ANSI)
  - SQL-89 (integrity constraints)
  - SQL-92 (Major revision)
  - SQL:1999 (language name became Y2K compliant! ORDB)
  - SQL:2003,SQL:2006 (XML)
  - SQL:2008 (INSTEAD OF,TRUNCATE)
  - SQL:2011 (temporal databases)
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First formalized by ANSI.
1989	SQL-89	FIPS127-1	Minor revision, in which the major addition were integrity constraints. Adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features (e.g. structured types). Support for embedding SQL in Java (SQL/OLB) and vice-versa (SQL/JRT).
2003	SQL:2003	SQL 2003	Introduced XML-related features (SQL/XML), *window functions*, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006	SQL 2006	ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it enables applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.
2008	SQL:2008	SQL 2008	Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers. Adds the TRUNCATE statement.
2011	SQL:2011	SQL 2011	One of the main new features is improved support for temporal databases

数据库交互工具

- Oracle
  - SQL*Plus (command line)
  - iSQL*Plus (Web-based)
- SQL Server
  - SQLCMD (command line)
  - iSQL,OSQL
- Sql server anywhere
  - Dbisqlc (command line style)
  - Dbisql (GUI)
- Mysql
  - MYSQL (command line)

SQL

- DDL
  - For Schema
  - Create, Alter, Drop
- DML
  - For Data
  - Modification: Insert, Update, Delete
  - Query: Select
  - Authorization: Grant, Revoke

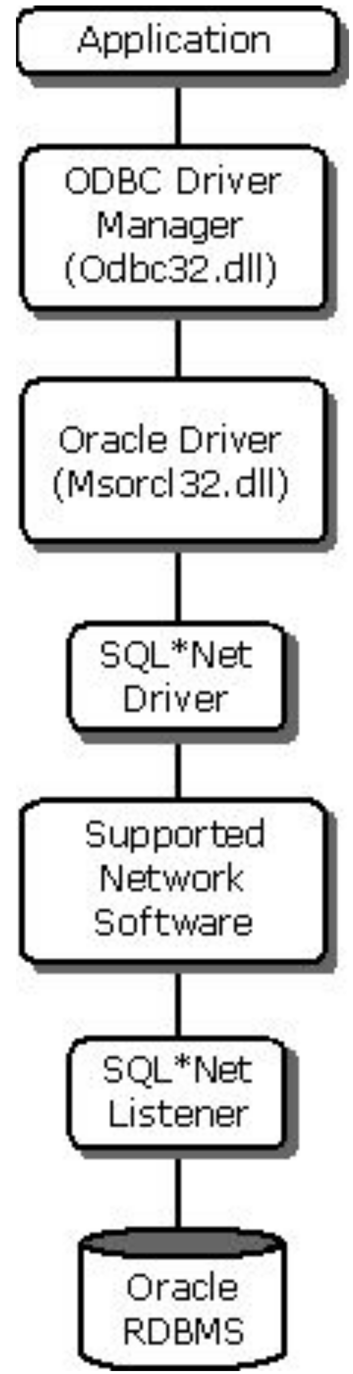
ODBC

- The Microsoft Open Database Connectivity (ODBC,1992) interface is a C programming language interface that makes it possible for applications to access data from a variety of database management systems (DBMSs).
- ODBC is a low-level, high-performance interface that is designed specifically for relational data stores.
- CLI(Call Level Interface)

ODBC:Architecture

- Application
- ODBC driver manager
- ODBC driver
- DBMS

- Windows ODBC
- UnixODBC
- iODBC



ODBC:DSN

- DSN（Data Source Name） is a symbolic name that represents the ODBC connection.
- It stores the connection details like database name, directory, database driver, UserID, password, etc. when making a connection to the ODBC.

iSQL Anywhere

Interactive SQL

DDL

DDL

For Schema (Create/Alter/Drop)

```
Create table Student(
      sno         char(10) primary key,
      sname       varchar(10),
      age         smallint check(age>=0),
      gende       char(1),
      dept        char(2)
);
```

DDL

```
Alter table student add GPA int;
Alter table student drop GPA;

Drop table student;
```

DDL: data type

- **char(n)**
  - Fixed length character string, with user-specified length *n*.
- **varchar(n)**
  - Variable length character strings, with user-specified maximum length *n*.
- **Int**
  - Integer (a finite subset of the integers that is machine-dependent).
- **Smallint**
  - Small integer (a machine-dependent subset of the integer domain type).

DDL: data type

- **numeric(p,d)**
  - Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point.
  - Eg. numeric(3,1), 44.5
- **real, double precision**
  - Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n)**
  - Floating point number, with user-specified precision of at least *n* digits.

DDL: data type

- Date
  - Dates, containing a (4 digit) year, month and date
  - Example: date ‘2005-7-27’
- Time
  - Time of day, in hours, minutes and seconds.
  - Example: time ‘09:00:30’ time ‘09:00:30.75’
- Timestamp
  - Date plus time of day
  - Example: timestamp ‘2005-7-27 09:00:30.75’

DDL: data type

- Interval
  - Period of time
  - Example: interval ‘1’ day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values

DDL: data type

- Large objects (photos, videos, CAD files, etc.) are stored as a *large object*:
  - Blob
    - binary large object
    - object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  - Clob
    - character large object
    - object is a large collection of character data

DDL: data type

- **Large objects (photos, videos, CAD files, etc.) are stored as a *large object*:**
  - **When a query returns a large object, a pointer is returned rather than the large object itself.**

Query

DML: Query

```
SELECT  sno,sname,gende
FROM    student
WHERE   dept = 'SE'
```

$\Pi_{sno,sname,gende}\ \sigma_{dept='SE'}\ (student)$

DML: Query

```
SELECT  *
FROM    student
WHERE   dept = 'SE'

SELECT  DISTINCT sname,dept
FROM    student
WHERE   age >= 21
```

DML: Query

```
SELECT  *
FROM    student
WHERE   sname LIKE '%明_'

SELECT  sno, cno, SQRT(grade)*10
FROM    SC
```

DML：Query

列出软件学院所有学生的姓名、课名和成绩

```
SELECT  sname,cname,grade
FROM    student,SC,course
WHERE   dept = 'SE'
?
```

DML: Query

找出与95001同龄的学生的学号、姓名、院系

```
SELECT  sno,sname,dept
FROM    student s1,student s2
WHERE   s1.sno = '95001' and
        s1.age = s2.age
```

?

元组变量：s1,s2

DML: Query

- **join-expression :**

table-expr. join-operator table-expr. [**ON** *join-condition*]

  - **join-operator :**

[**KEY** | **NATURAL**] [*join-type*] **JOIN** | **CROSS JOIN**

    - **join-type :**

INNER
| **LEFT** [**OUTER**]
| **RIGHT** [**OUTER**]
| **FULL** [**OUTER**]

DML：Query,Aggregation

- 统计软件学院男女生平均年龄及人数

```
SELECT  gende,AVG(age),COUNT(*)
FROM    student
WHERE   dept = 'SE'
GROUP BY dept,gende
```

DML：Query,Aggregation

- 统计人数大于1000人的院系的男女生平均年龄

```
SELECT  dept,gende,AVG(age)
FROM    student
WHERE   dept = 'SE'
GROUP BY dept,gende
HAVING COUNT(*)>1000
```

DML：Query,Aggregation

- 容易混淆的问题
  - Where？ Having?
  - 那些原始属性可以出现在 select 子句中？
  - Null的角色

DML：Query,SET

- 列出所有师生的编号（工号或学号）、姓名

```
SELECT  id,name
FROM    insturctor
Union
SELECT id,name
FROM    student
```

DML：Query,MultiSet

- 列出所有师生的编号（工号或学号）、姓名

```
SELECT  id,name
FROM    insturctor
Union ALL
SELECT id,name
FROM    student
```

DML：Query,MultiSet

r有m个t元组，s有n个t元组，
则结果集中t元组数：

r union s

$m+n$

r intersect s

$\min(m,n)$

r except s

$\max(0,m-n)$.

Subquery

SubQuery

回忆一下：

找出与95001同龄的学生的学号、姓名、院系

Tuple varible

自然思维方式？

SubQuery

- Select-from-where嵌入where,from中

找出与95001同龄的学生的学号、姓名、院系

```
SELECT  sno,sname,dept
FROM    student
WHERE   age = (  SELECT  age
                 FROM    student
                 WHERE   sno = '95001')
```

SubQuery

```
SELECT  sno,sname,dept
FROM    student
WHERE   age = (  SELECT  age
                 FROM    student
                 WHERE   sno = '95001')
```

结果集中如何去除95001本人？

关联变量（Correlation Variables）

SubQuery

列出选过02号课程的学生的姓名

```
SELECT  sname
FROM    student
WHERE   sno in (  SELECT  sno
                  FROM    SC
                  WHERE   cno = '02' )
```

SubQuery

找出年龄最大的学生

```
SELECT  *
FROM    student
WHERE   age >= ALL ( SELECT  age
                     FROM    student )
```

SubQuery

找出年龄不是最大的学生

```
SELECT  *
FROM    student
WHERE   age <= SOME (SELECT  age
                     FROM  student )
```

SOME = ANY

SubQuery

找出选修了SE开设的所有课程的学生

```
SELECT   *
FROM     student s
WHERE    not exist (SELECT   cno
                     FROM      course
                     WHERE dept = 'SE' )
                     except
                    (SELECT    cno
                     FROM       SC
                     WHERE SC.sno = s.sno )
```

SubQuery

Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester.

```
select course_id
from  section as S
where semester = 'Fall' and year= 2009 and
      exists ( select *
               from section as T
               where semester = 'Spring' and
                     year= 2010 and
                     S.course_id= T.course_id);
```

SubQuery

Select-from-where,可以出现在哪里？

- **Where**
- **From**

Find the average instructors' salaries of those departments where the average salary is greater than $42,000.

```
	selectdept_name, avg_salary from
		(select	dept_name, avg (salary) as
avg_salary
		from	instructor
		group by dept_name
		) as dept_avg (dept_name, avg_salary)
	where avg_salary > 42000;
```

SubQuery

Select-from-where,可以出现在哪里？

- **With**

Find the average instructors' salaries of those departments where the average salary is greater than $42,000.

```
With dept_avg (dept_name, avg_salary) as
( select      dept_name, avg (salary) as
avg_salary
  from        instructor
  group by dept_name)
select dept_name, avg_salary
from        dept_avg
where avg_salary > 42000;
```

DML: Modification

DML：Modification

- INSERT
  - 单行整行
  - 单行部分属性
  - 多行

DML: Modification

```
INSERT INTO student Values
      ('98001','张三',21,'M','SE')

INSERT INTO  student (sno,sname)
      Values('99001','李四')

INSERT INTO SC
      SELECT  sno,cno
      FROM student as s,course as c
      WHERE c.dept = 'SE'
'
```

DML: Modification

DELETE 整元组删除

```
DELETE
    FROM student
    WHERE dept = 'SE'

DELETE
    FROM      student
    FROM      SC
    WHERE     student.sno = SC.sno and
              grade < 60
```

DML: Modification

UPDATE

修改相关元组中的指定属性

```
UPDATE student
    SET age  = age+1,
        dept = 'SE'
    WHERE dept = 'MA'
```

Authorization

Authorization

- DCL
- Grant授权
- Revoke回收

grant select on *instructor* to U_1, U_2, U_3

revoke select on *branch* from U_1, U_2, U_3

Authorization

grant select on department to Amit
with grant option;

revoke select on department from Amit, Satoshi cascade;

revoke select on department from Amit, Satoshi restrict;

View

View(视图)

- 仅允许用户察看部分数据（瞎子摸象）
- 提高数据安全性

```
CREATE VIEW se_name_list as
     SELECT    sno,sname
     FROM      student
     WHERE     dept = 'SE'
```

View

- Create a view of department salary totals

```
create view
        departments_total_salary(dept_name,
                        total_salary) as
    select dept_name, sum (salary)
    from instructor
    group by dept_name;
```

View的存在形式及操作

- Virtual Table   Base Table
- View的定义被存储在Meta Data中

- 操作时，被转换成对BASE TABLE的处理

View的使用

```
create view
   departments_total_salary(dept_name,
     total_salary) as
  select dept_name, sum (salary)
  from instructor
  group by dept_name;
```

```
SELECT    dept_name,total_salary
FROM      departments_total_salary
WHERE     total_salary > 2,000,000
```

查询：如同BaseTable，转换成对BaseTable的查询

View的使用

```
UPDATE   departments_total_salary
     SET  total_salary = tatal_salary*1.5
     WHERE    total_salary < 2,000,000
```

能成功转换成对BaseTable的运算吗？

对View的更新操作，必须慎重！

View的使用

- Most SQL implementations allow updates only on simple views
  - The FROM clause has only one database relation.
  - The SELECT clause contains only attribute names of the relation, and does not have any expressions, aggregates, or DISTINCT specification.
  - Any attribute not listed in the SELECT clause can be set to NULL
  - The query does not have a GROUP BY or HAVING clause.

Integrity Constraints

Integrity Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
  - A checking account must have a balance greater than $10,000.00.
  - A salary of a bank employee must be at least $4.00 an hour.
  - A customer must have a (non-null) phone number

Constraints on a Single Relation

- not null
- primary key
- unique
- check (P)

- unique (A1, A2, …, Am)
  - The unique specification states that the attributes A1, A2, … Am form a candidate key.
  - Candidate keys are permitted to be null (in contrast to primary keys).

Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.

Cascading Actions in Referential Integrity

```
create table course (
    course_id   char(5) primary key,
    title       varchar(20),
    dept_name   varchar(20) references department
)
create table course (
    …
    dept_name varchar(20),
    foreign key (dept_name) references department
                on delete cascade
                on update cascade,
    …
)
```

alternative actions to cascade: set null, set default

Complex Check Clauses

- check (time_slot_id in (select time_slot_id from time_slot))
  - why not use a foreign key here?
- Every section has at least one instructor teaching the section.
  - how to write this?
- Unfortunately: subquery in check clause not supported by pretty much any database
  - Alternative: triggers (later)
- create assertion <assertion-name> check <predicate>;
  - Also not supported by anyone

参照完整性中的违例及处理方式

Trigger

Trigger

- "Event-Condition-Action Rules"
  - When event occurs
  - Check condition; if true, do action

Trigger

Create Trigger name
Before|After|Instead Of events
[referencing-variables]
[For Each Row]
When (condition)
    action

Trigger

Use triggers on section and time_slot to enforce integrity constraints

```
Create Trigger timeslot_check1 after insert on section
referencing new row as nrow
for each row
when (nrow.time_slot_id not in (select time_slot_id
                                 from time_slot))
begin
    rollback
end;
```

Trigger

```
create trigger timeslot_check2 after delete on time_slot
referencing old row as orow
for each row
when (orow.time_slot_id not in (
                select time_slot_id
                from time_slot)
        and orow.time_slot_id in (
                select time_slot_id
                from section))
begin
    rollback
end;
```

Trigger

```
create trigger credits_earned after update of takes on (grade)
      referencing new row as nrow
      referencing old row as orow
      for each row
      when nrow.grade <> 'F' and nrow.grade is not null
            and (orow.grade = 'F' or orow.grade is null)
      begin atomic
            update student
                  set tot_cred= tot_cred +
                  (select credits
                   from course
                  where course.course_id= nrow.course_id)
            where student.id = nrow.id;
      end;
```

Function &
Stored procedure

Procedure

- SQL provides a module language
  - Permits definition of procedures in SQL, with if-then-else statements, for and while loops, etc.
- Stored Procedures
  - Can store procedures in the database
  - then execute them using the call statement
  - permit external applications to operate on the database without knowing about internal details

Procedure

- SQL:1999 supports functions and procedures
  - Functions/procedures can be written in SQL itself, or in an external programming language.
  - Functions are particularly useful with specialized data types such as images and geometric objects.
    - Example: functions to check if polygons overlap, or to compare images for similarity.
  - Some database systems support table-valued functions, which can return a relation as a result.
- SQL:1999 also supports a rich set of imperative constructs, including
  - Loops, if-then-else, assignment

Function

- Define a function that, given the name of a department, returns the count of the number of instructors in that department.

```
Create Function dept_count (dept_name varchar(20))
  returns integer
  begin
       declare d_count integer;
      select count (* ) into d_count
       from instructor
       where instructor.dept_name = dept_name
      return d_count;
   end
```

Function

Find the department name and budget of all departments with more that 12 instructors.

```
Select dept_name, budget
From   department
Where dept_count (dept_name ) > 12
```

Function

•SQL:2003 added functions that return a relation as a result

```
Create Function instructors_of (dept_name char(20))
        returns table (ID varchar(5),
                       name varchar(20),
                         dept_name varchar(20),
                       salary numeric(8,2))
        return table(Select ID, name, dept_name, salary
                        From instructor
                        Where instructor.dept_name =
                                     instructors_of.dept_name)

Select *
From table (instructors_of ('Music'))
```

Stored Procedure

- The dept_count function could instead be written as procedure:

```
Create Procedure dept_count_proc (
      in   dept_name varchar(20),
     out  d_count integer)
Begin
      select count(*) into d_count
      from  instructor
      where instructor.dept_name =
              dept_count_proc.dept_name
End
```

Stored Procedure

- Procedures can be invoked either from an SQL procedure or from embedded SQL, using the call statement.

```
Declare d_count integer;
Call dept_count_proc( 'Physics', d_count);
```

- Procedures and functions can be invoked also from dynamic SQL

Cursor

Cursor

- A cursor is used to retrieve rows from a query that has multiple rows in its result set. A cursor is a handle or an identifier for the SQL query and a position within the result set.
- Cursors can be positioned in the following places:
  - Before the first row of the result set.
  - On a row in the result set.
  - After the last row of the result set.

Cursor

```
CREATE PROCEDURE get_table_name(
IN id_value INT, OUT tabname CHAR(128))
BEGIN
  DECLARE qry LONG VARCHAR;
  SET qry = 'SELECT table_name    FROM
      SYS.SYSTAB ' || 'WHERE table_id=' ||
      string(id_value);
  BEGIN
      DECLARE crsr CURSOR USING qry;
      OPEN crsr;
      FETCH crsr INTO tabname;
      CLOSE crsr;
  END
END;
```

Cursor

```
CREATE FUNCTION GetRowCount( IN qry LONG VARCHAR )
        RETURNS INT
BEGIN
  DECLARE crsr CURSOR USING qry;
  DECLARE rowcnt INT;
  SET rowcnt = 0;
  OPEN crsr;
  lp: LOOP
      FETCH crsr;
      IF SQLCODE <> 0 THEN LEAVE lp END IF;
      SET rowcnt = rowcnt + 1;
      END LOOP;
   CLOSE crsr;
   RETURN rowcnt;
END;
```

Updates Through Cursors

- Can update tuples fetched by cursor by declaring that the cursor is for update

```
declare c cursor for
 select *
 from instructor
 where dept_name = 'Music'
 for update
```

To update tuple at the current location of cursor c

```
 update instructor
set salary = salary + 100
where current of c
```

API & Application

JDBC and ODBC

- API (Application-Program Interface) for a program to interact with a database server
- Application makes calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- ODBC (Open Database Connectivity) works with C, C++, C#, and Visual Basic
  - Other API's such as ADO.NET sit on top of ODBC
- JDBC (Java Database Connectivity) works with Java

ODBC

- Open DataBase Connectivity(ODBC) standard
- standard for application program to communicate with a database server.
- application program interface (API) to
  - open a connection with a database,
  - send queries and updates,
  - get back results.
- Applications such as GUI, spreadsheets, etc. can use ODBC

ODBC

```
int ODBCexample()
{
  RETCODE error;
  HENV    env;          /* environment */
  HDBC    conn;         /* database connection */
  SQLAllocEnv(&env);
  SQLAllocConnect(env, &conn);
  SQLConnect(conn, “db.yale.edu", SQL_NTS, "avi", SQL_NTS,
"avipasswd", SQL_NTS);
  { …. Do actual work … }

  SQLDisconnect(conn);
  SQLFreeConnect(conn);
  SQLFreeEnv(env);
}
```

ODBC

```
char deptname[80];
float salary;
int lenOut1, lenOut2;
HSTMT stmt;
char * sqlquery = "select dept_name, sum (salary)
                              from instructor
                              group by dept_name";
SQLAllocStmt(conn, &stmt);
error = SQLExecDirect(stmt, sqlquery, SQL NTS);
if (error == SQL SUCCESS) {
   SQLBindCol(stmt, 1, SQL C CHAR, deptname , 80,
&lenOut1);
   SQLBindCol(stmt, 2, SQL C FLOAT, &salary, 0 ,
&lenOut2);
   while (SQLFetch(stmt) == SQL SUCCESS) {
         printf (" %s %g\n", deptname, salary);
   }
}
SQLFreeStmt(stmt, SQL DROP);
```

JDBC

- JDBC is a Java API for communicating with database systems supporting SQL.
- JDBC supports a variety of features for querying and updating data, and for retrieving query results.
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes.
- Model for communicating with the database:
  - Open a connection
  - Create a “statement” object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors

SQLCA

SQL Communication Area

- is an area of memory that is used on every database request for communicating statistics and errors from the application to the database server and back to the application.
- The *SQLCA* is used as a handle for the application-to-database communication link. It is passed in to all database library functions that need to communicate with the database server. It is implicitly passed on all embedded SQL statements.

SQLCA Fields

- **sqlcode** A 32-bit integer that specifies the error code when the database detects an error on a request.
- Definitions for the error codes can be found in the header file *sqlerr.h*.
  - The error code is 0 (zero) for a successful operation
  - 100 for row not found
  - Positive for a warning
  - Negative for an error.

Chapter 7: Entity-Relationship Model

Database System Concepts, 6th Ed.

Chapter 7: Entity-Relationship Model

- Design Process
- Modeling
- Constraints
- E-R Diagram
- Design Issues
- Weak Entity Sets
- Extended E-R Features
- Design of the Bank Database
- Reduction to Relation Schemas
- Database Design
- UML

Modeling

- A *database* can be modeled as:
  - a collection of entities,
  - relationship among entities.
- An **entity** is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- Entities have **attributes**
  - Example: people have *names* and *addresses*
- An **entity set** is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays

Entity Sets *instructor* and *student*

instructor_ID	instructor_name
76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

student-ID	student_name
98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Relationship Sets

- A **relationship** is an association among several entities

  Example:

  44553 (Peltier) *advisor* 22222 (Einstein)
  student entity relationship set *instructor* entity

- A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

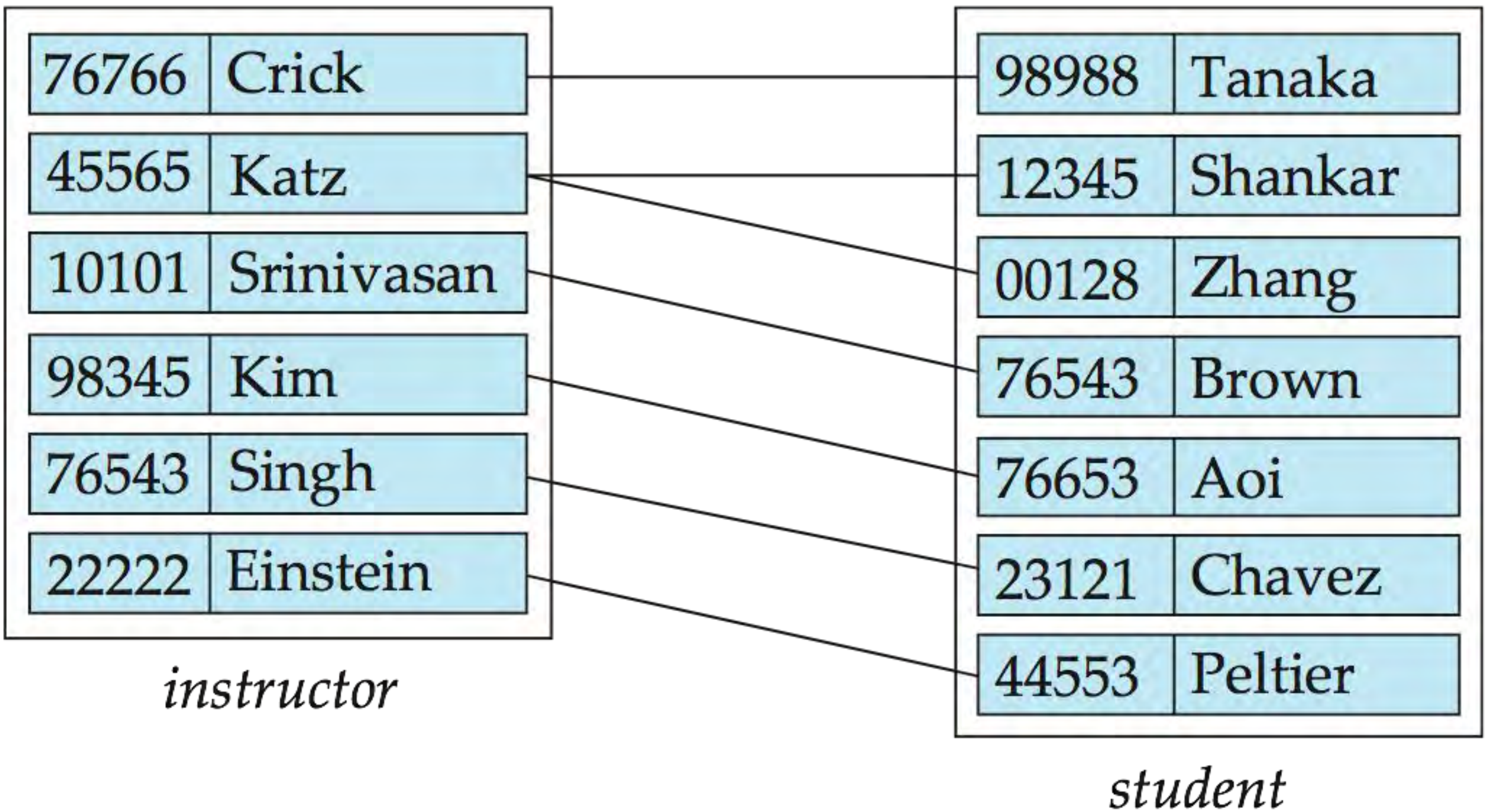
$$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

  where $(e_1, e_2, \ldots, e_n)$ is a relationship

- Example:

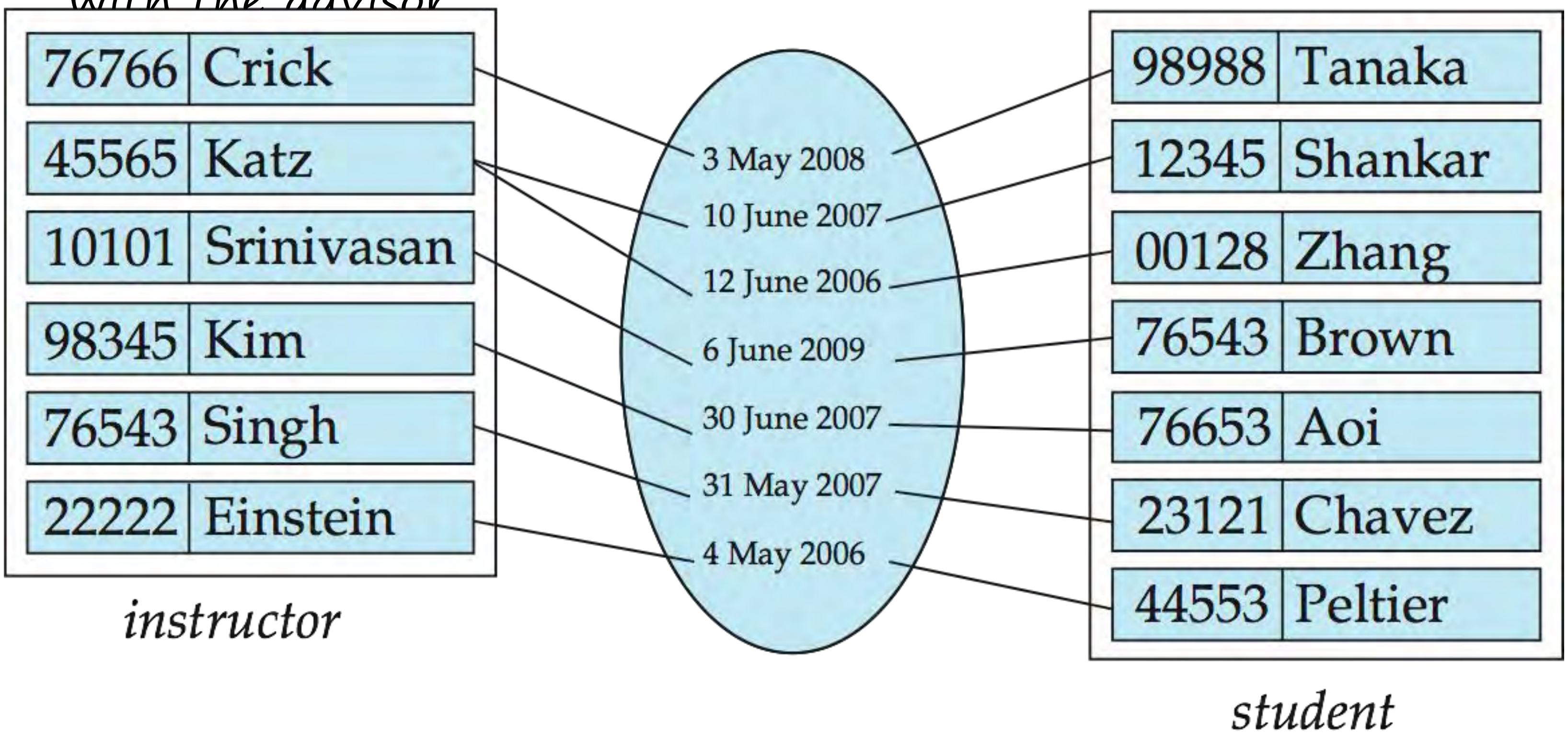
$$(44553,22222) \in advisor$$

Relationship Set *advisor*



Relationship Sets (Cont.)

- An **attribute** can also be property of a relationship set.
- For instance, the *advisor* relationship set between entity sets *instructor* and *student* may have the attribute *date* which tracks when the student started being associated with the advisor



Degree of a Relationship Set

- ***binary relationship***
  - involve two entity sets (or degree two).
  - most relationship sets in a database system are binary.
- Relationships between more than two entity sets are rare. Most relationships are binary. (More on this later.)
  - Example: *students* work on research *projects* under the guidance of an *instructor*.
  - relationship *proj_guide* is a ternary relationship between *instructor*, *student*, and *project*

Attributes

- An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.
  - Example:

    instructor = (*ID*, *name*, *street*, *city*, *salary*)
    course= (*course_id*, *title*, *credits*)

  Domain – the set of permitted values for each attribute

  Attribute types:

  Simple and **composite** attributes.

  Single-valued and **multivalued** attributes

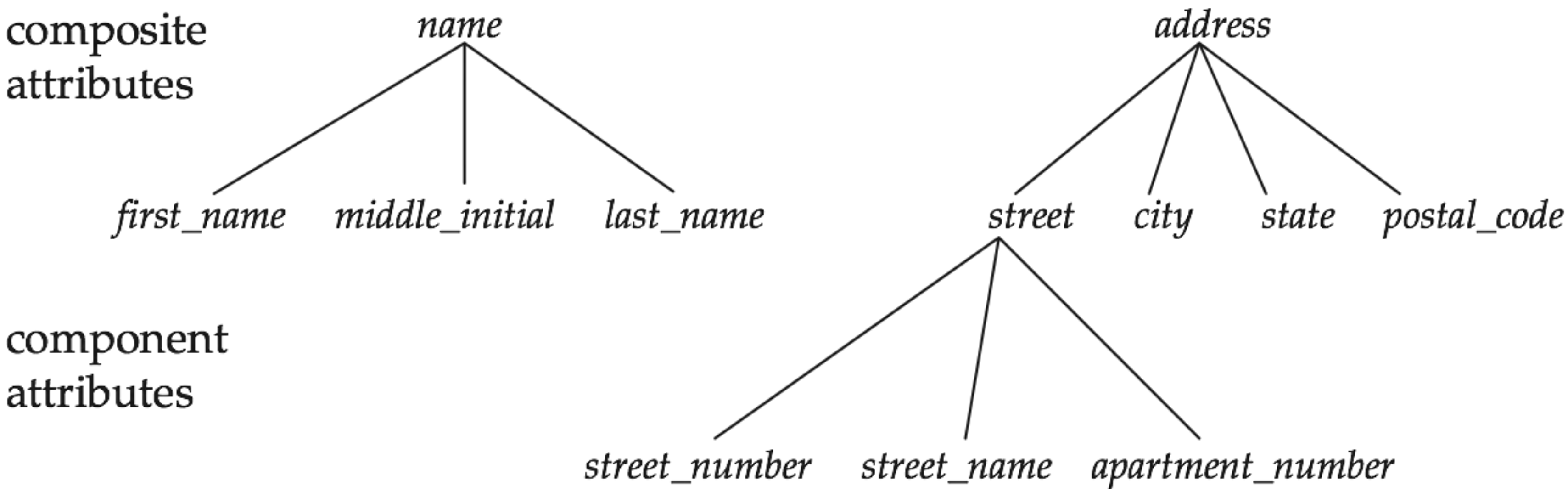
  Example: multivalued attribute: *phone_numbers*

  Derived attributes

  Can be computed from other attributes

  Example: age, given date_of_birth

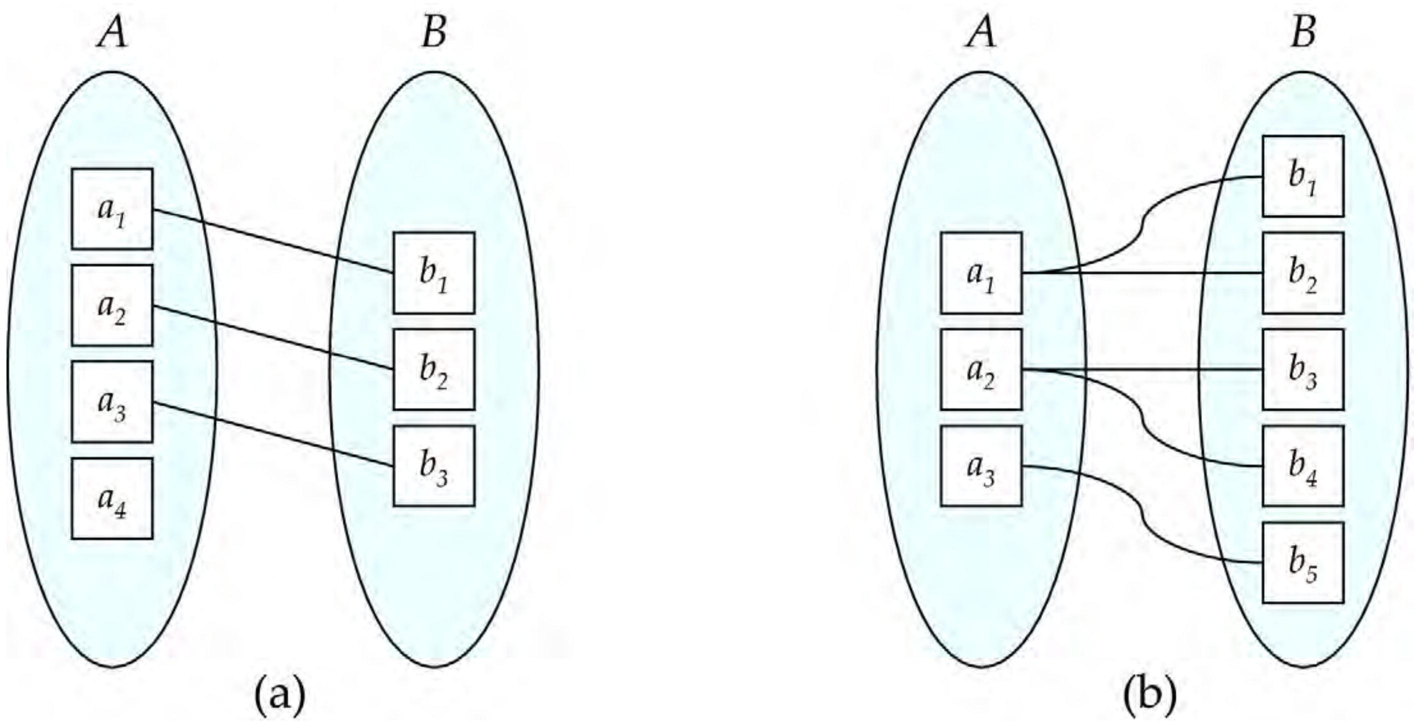
Composite Attributes



Mapping Cardinality Constraints

- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

Mapping Cardinalities

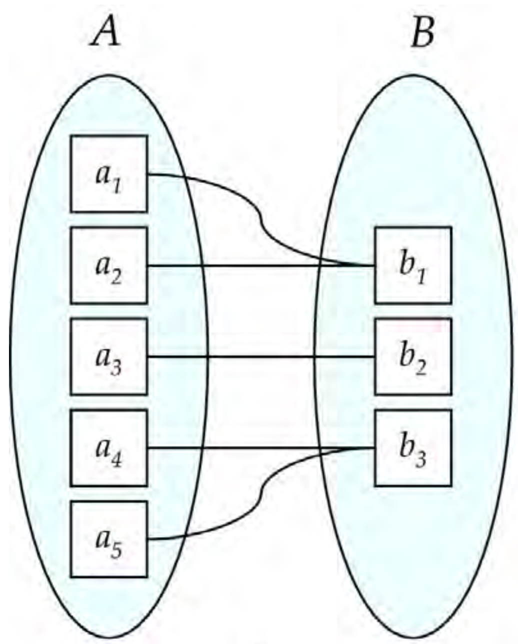


One to one

One to many

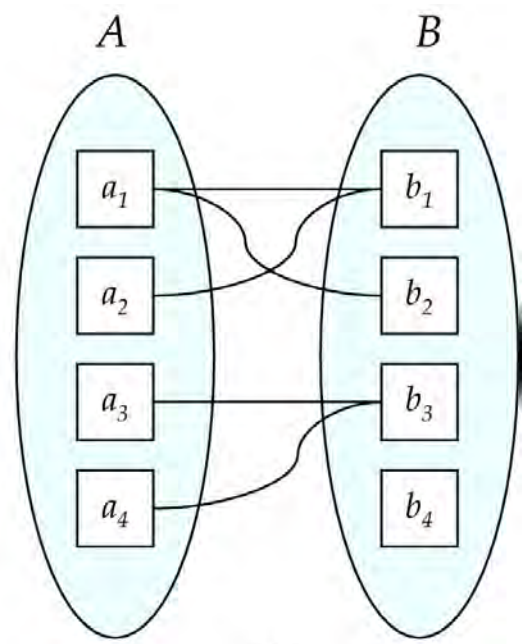
Note: Some elements in *A* and *B* may not be mapped to any elements in the other set

Mapping Cardinalities



(a)

Many to one



(b)

Many to many

Note: Some elements in A and B may not be mapped to any elements in the other set

Keys

- A **super key** of an entity set is a set of one or more attributes whose values uniquely determine each entity.
- A **candidate key** of an entity set is a minimal super key
  - *ID* is candidate key of *instructor*
  - *course_id* is candidate key of *course*
- Although several candidate keys may exist, one of the candidate keys is selected to be the **primary key**.

Keys for Relationship Sets

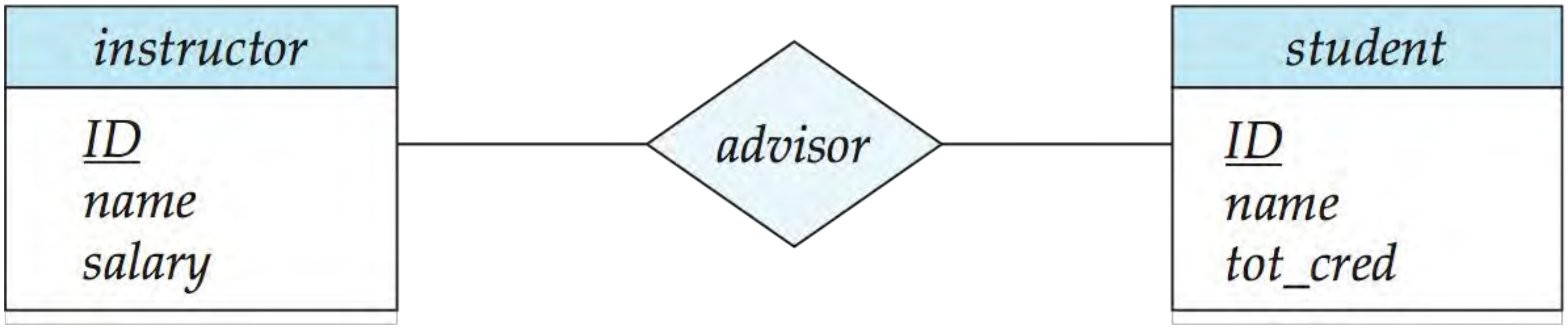
- The combination of primary keys of the participating entity sets forms a super key of a relationship set.
  - (*s_id, i_id*) is the super key of *advisor*
  - *NOTE: this means **a pair of entity sets can have at most one relationship in a particular relationship set.***
    - Example: if we wish to track multiple meeting dates between a student and her advisor, we cannot assume a relationship for each meeting. We can use a multivalued attribute though
- Must consider the mapping cardinality of the relationship set when deciding what are the candidate keys
- Need to consider semantics of relationship set in selecting the *primary key* in case of more than one candidate key

Redundant Attributes

- Suppose we have entity sets
  - *instructor*, with attributes including *dept_name*
  - *department*

  and a relationship
  - *inst_dept* relating *instructor* and *department*
- Attribute *dept_name* in entity *instructor* is redundant since there is an explicit relationship *inst_dept* which relates instructors to departments
  - The attribute replicates information present in the relationship, and should be removed from *instructor*
  - BUT: when converting back to tables, in some cases the attribute gets reintroduced, as we will see.

E-R Diagrams



- Peter Chen & IDEF1X

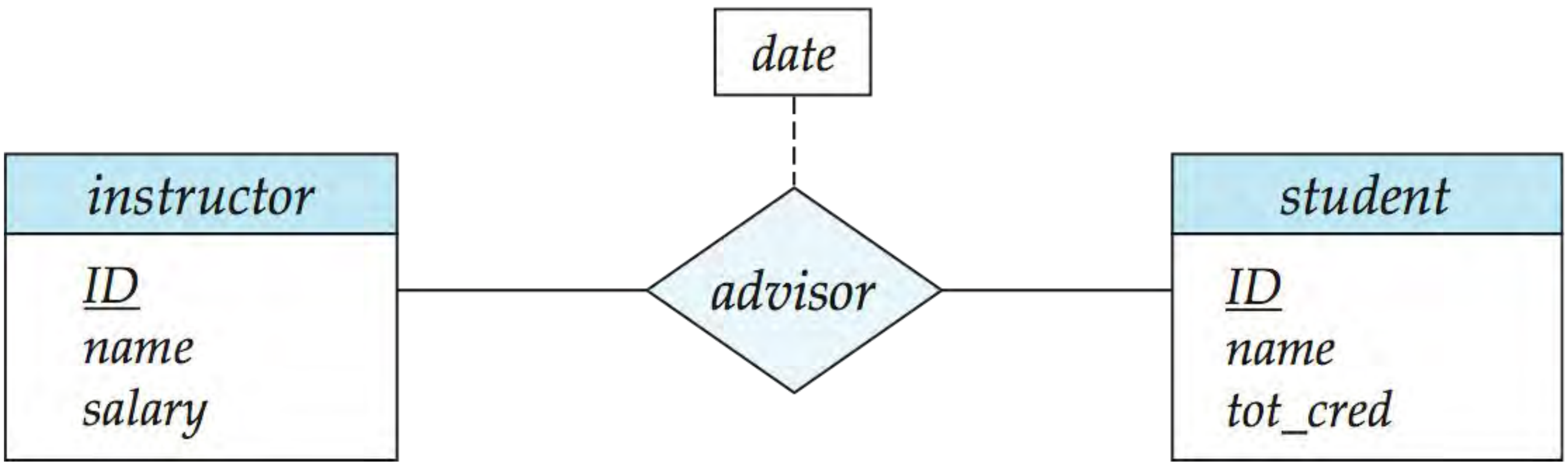
- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Attributes listed inside entity rectangle
- Underline indicates primary key attributes

Entity With Composite, Multivalued, and Derived Attributes

instructor

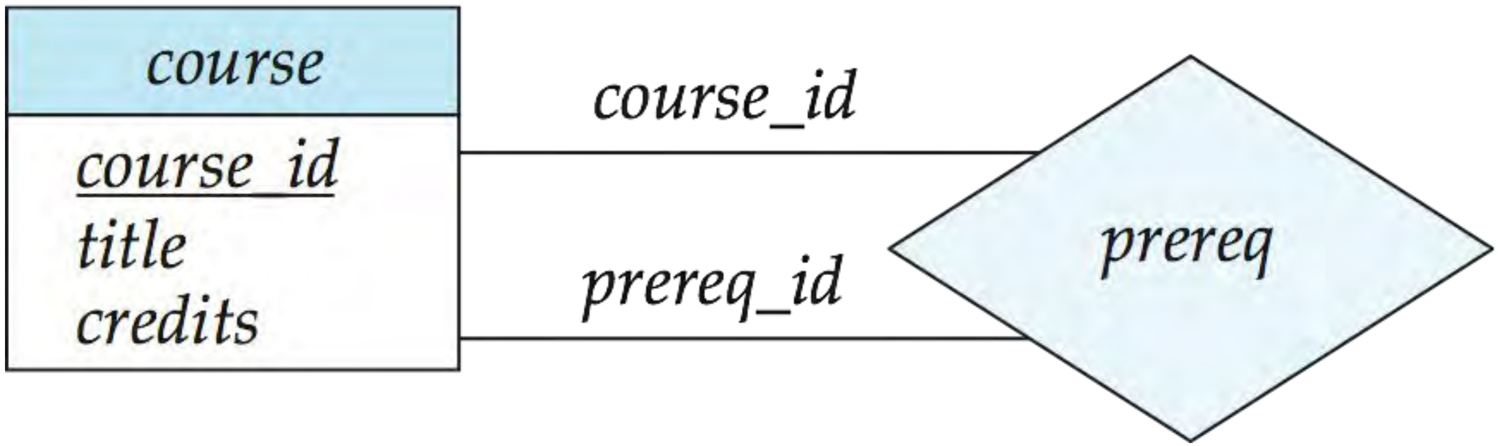
- *ID*
- *name*
  - *first_name*
  - *middle_initial*
  - *last_name*
- *address*
  - *street*
    - *street_number*
    - *street_name*
    - *apt_number*
  - *city*
  - *state*
  - *zip*
- *{ phone_number }*
- *date_of_birth*
- *age ( )*

Relationship Sets with Attributes



Roles

- Entity sets of a relationship need not be distinct
  - Each occurrence of an entity set plays a "role" in the relationship
- The labels "*course_id*" and "*prereq_id*" are called **roles**.



Cardinality Constraints

- We express cardinality constraints by drawing either a directed line ($\rightarrow$), signifying "one," or an undirected line ($—$), signifying "many," between the relationship set and the entity set.
- One-to-one relationship:
  - A student is associated with at most one *instructor* via the relationship *advisor*
  - A *student* is associated with at most one *department* via *stud_dept*

One-to-One Relationship

- one-to-one relationship between an *instructor* and a *student*
  - an instructor is associated with at most one student via *advisor*
  - and a student is associated with at most one instructor via *advisor*

One-to-Many Relationship

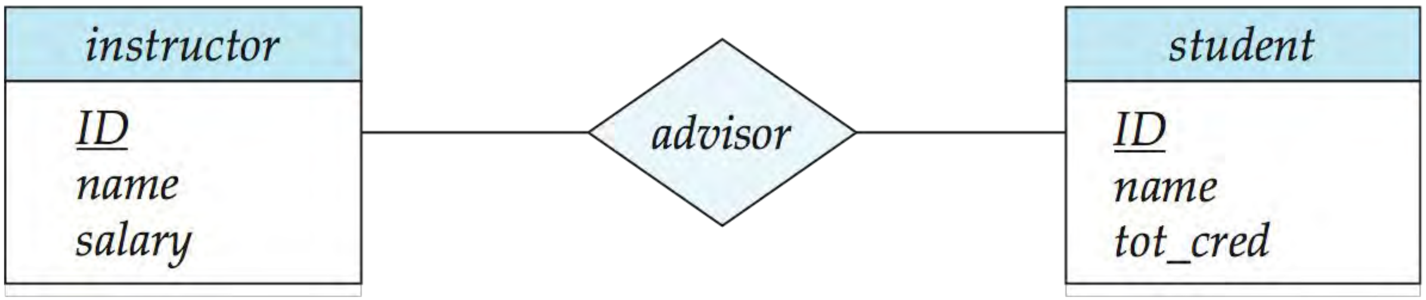
- one-to-many relationship between an *instructor* and a *student*
  - an instructor is associated with several (including 0) students via *advisor*
  - a student is associated with at most one instructor via advisor,

Many-to-One Relationships

- In a many-to-one relationship between an *instructor* and a *student*,
  - an instructor is associated with at most one student via *advisor*,
  - and a student is associated with several (including 0) instructors via *advisor*

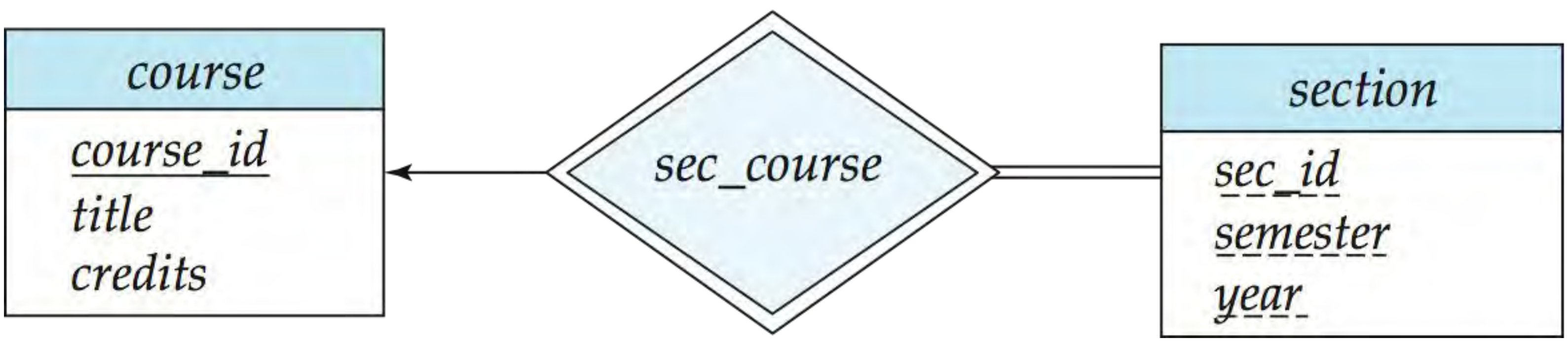
Many-to-Many Relationship

- An instructor is associated with several (possibly 0) students via *advisor*
- A student is associated with several (possibly 0) instructors via *advisor*



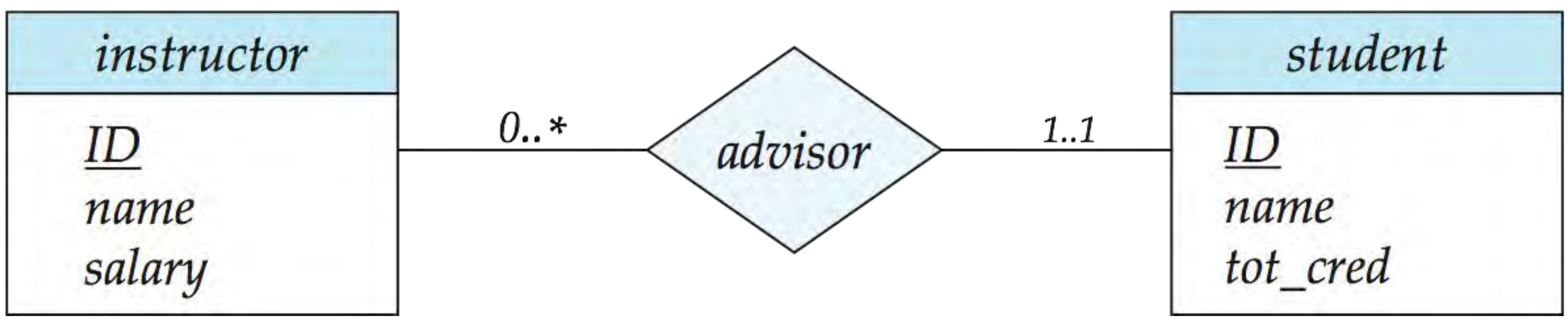
Participation of an Entity Set in a Relationship Set

- Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set
  - E.g., participation of *section* in *sec_course* is total
    - every *section* must have an associated course
- Partial participation: some entities may not participate in any relationship in the relationship set
  - Example: participation of *instructor* in *advisor* is partial

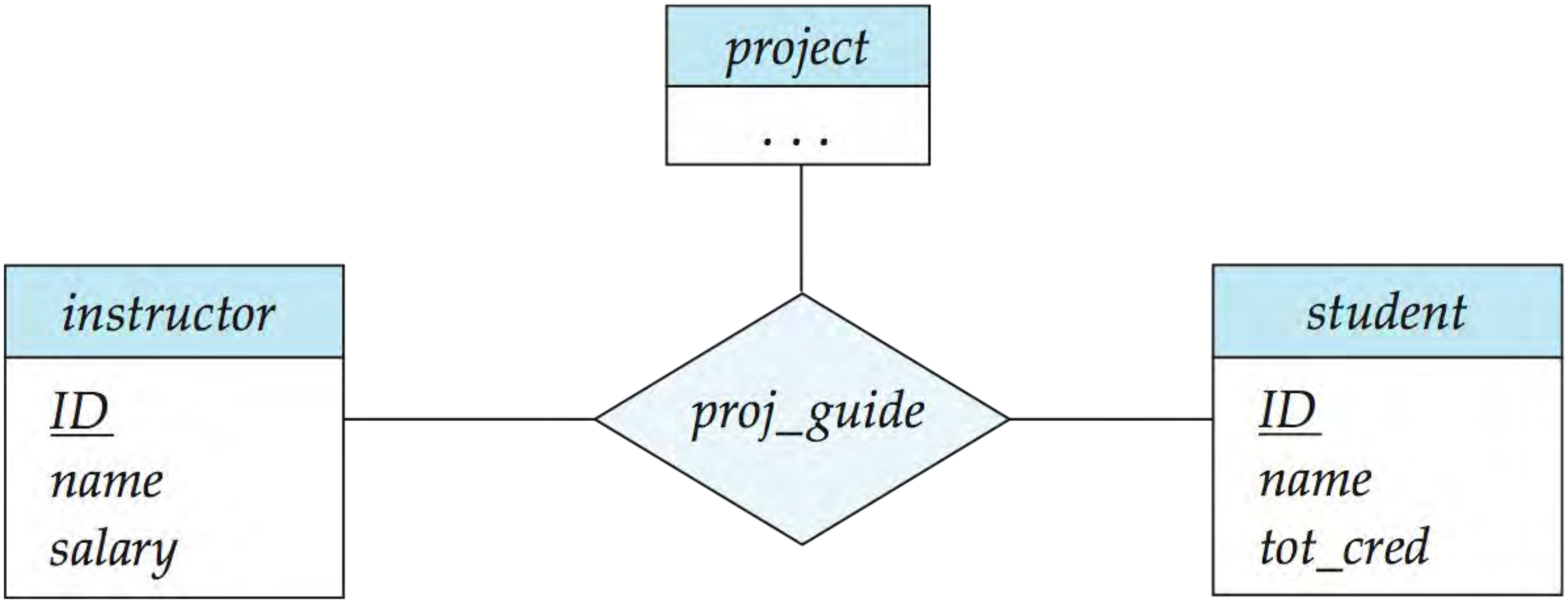


Alternative Notation for Cardinality Limits

- Cardinality limits can also express participation constraints



E-R Diagram with a Ternary Relationship



Cardinality Constraints on Ternary Relationship

- We allow at most one arrow out of a ternary (or greater degree) relationship to indicate a cardinality constraint
- E.g., an arrow from *proj_guide* to *instructor* indicates each student has at most one guide for a project
- If there is more than one arrow, there are two ways of defining the meaning.
  - E.g., a ternary relationship *R* between *A*, *B* and *C* with arrows to *B* and *C* could mean

    1. each *A* entity is associated with a unique entity from *B* and *C* or

    2. each pair of entities from (*A*, *B*) is associated with a unique *C* entity, and each pair (*A*, *C*) is associated with a unique *B*
  - Each alternative has been used in different formalisms
  - To avoid confusion we outlaw more than one arrow

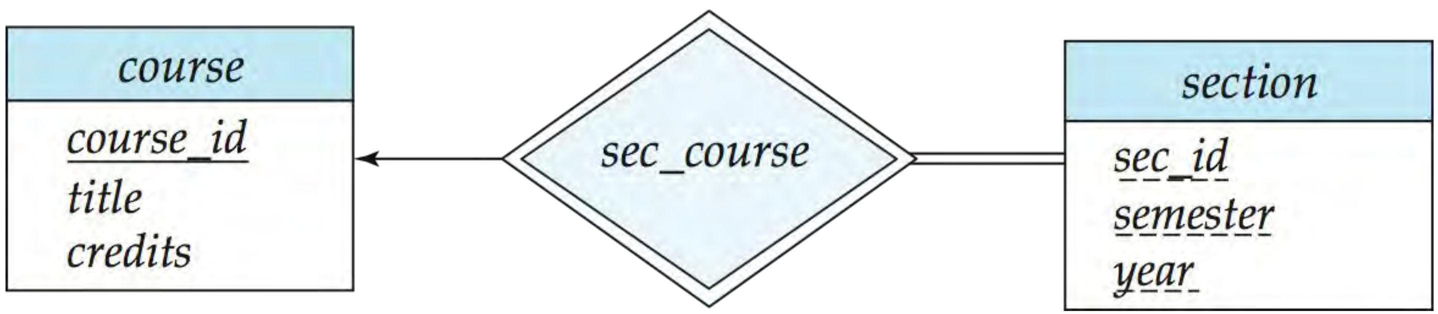
How about doing an ER design interactively on the board? Suggest an application to be modeled.

Weak Entity Sets

- An entity set that does not have a primary key is referred to as a **weak entity set**.
- The existence of a weak entity set depends on the existence of a **identifying entity set**
  - It must relate to the identifying entity set via a total, one-to-many relationship set from the identifying to the weak entity set
  - **Identifying relationship** depicted using a double diamond
- The **discriminator** (*or partial key*) of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
- The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator.

Weak Entity Sets (Cont.)

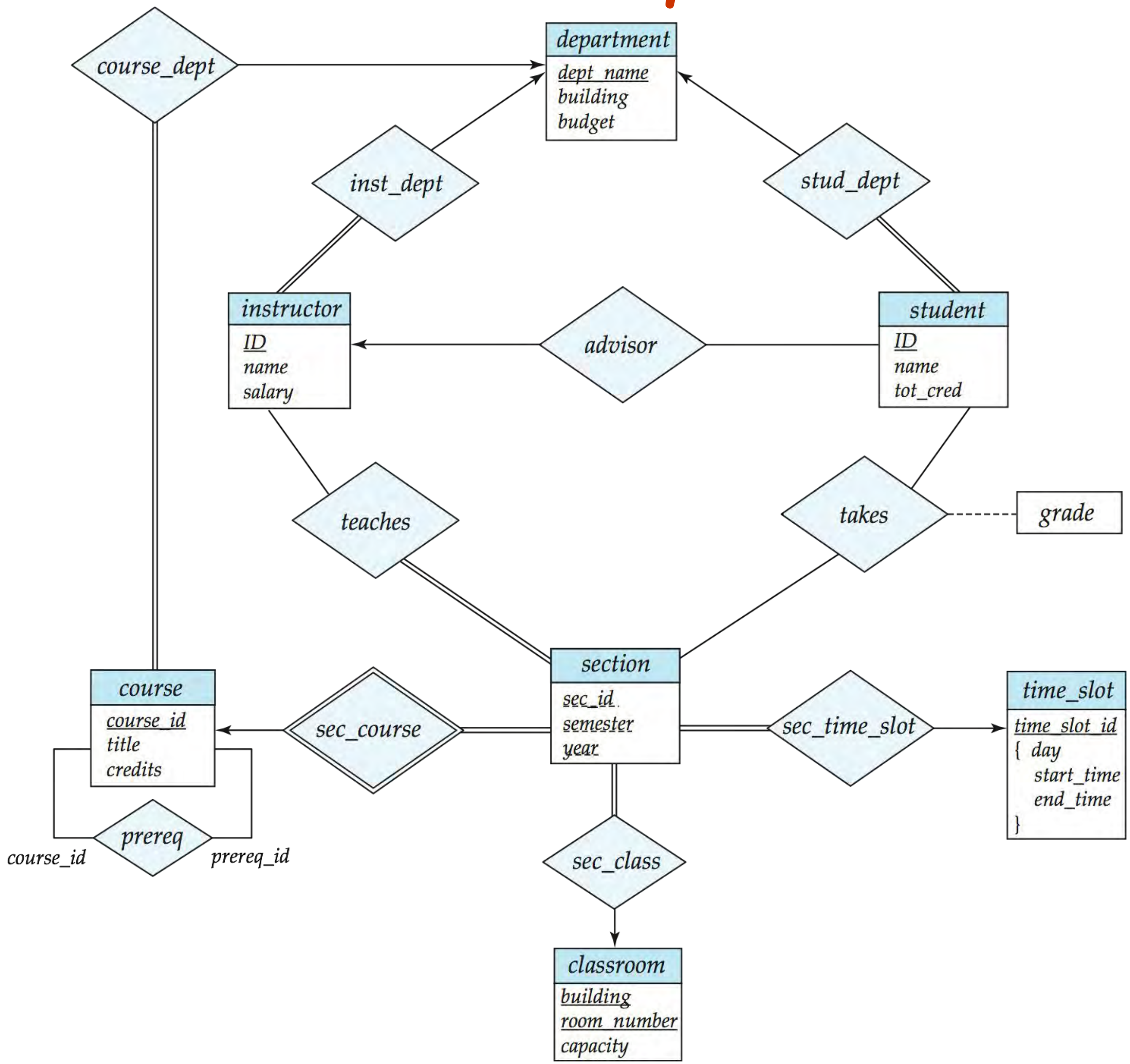
- We underline the discriminator of a weak entity set with a dashed line.
- We put the identifying relationship of a weak entity in a double diamond.
- Primary key for *section* – (*course_id, sec_id, semester, year*)



Weak Entity Sets (Cont.)

- Note: the primary key of the strong entity set is not explicitly stored with the weak entity set, since it is implicit in the identifying relationship.
- If *course_id* were explicitly stored, *section* could be made a strong entity, but then the relationship between *section* and *course* would be duplicated by an implicit relationship defined by the attribute *course_id* common to *course* and *section*

E-R Diagram for a University Enterprise



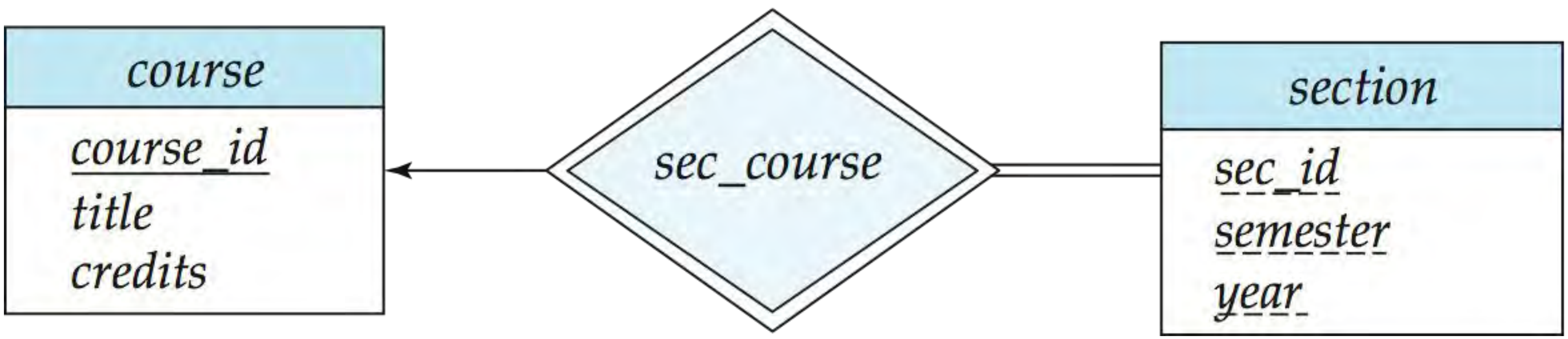
Reduction to Relational Schemas

Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as *relation schemas* that represent the contents of the database.
- A database which conforms to an E-R diagram can be represented by a collection of schemas.
- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.
- Each schema has a number of columns (generally corresponding to attributes), which have unique names.

Representing Entity Sets With Simple Attributes

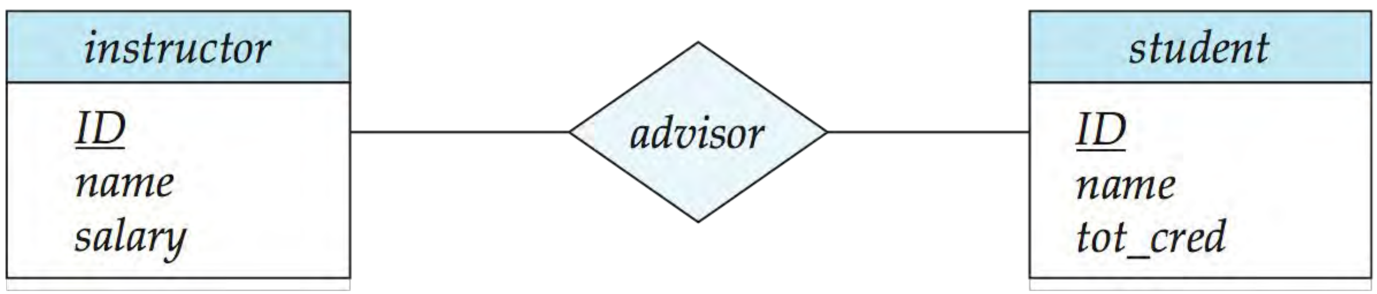
- A strong entity set reduces to a schema with the same attributes
  student(ID, name, tot_cred)
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set
  section (course_id, sec_id, sem, year)



Representing Relationship Sets

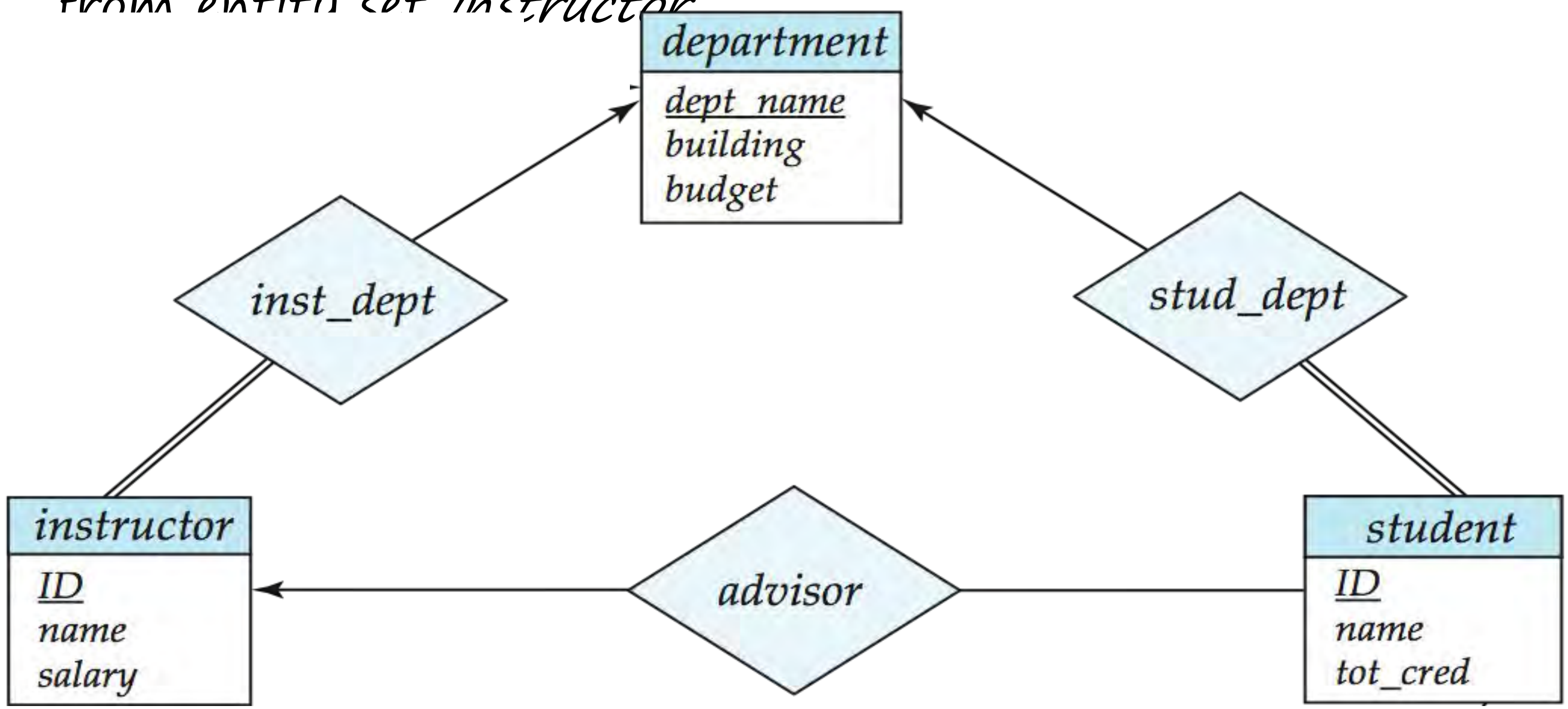
- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set *advisor*

  advisor = (*s_id, i_id*)



Redundancy of Schemas

- Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the "many" side, containing the primary key of the "one" side
- Example: Instead of creating a schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema arising from entity set *instructor*



Redundancy of Schemas (Cont.)

- For one-to-one relationship sets, either side can be chosen to act as the "many" side
  - That is, extra attribute can be added to either of the tables corresponding to the two entity sets
- If participation is *partial* on the "many" side, replacing a schema by an extra attribute in the schema corresponding to the "many" side could result in null values
- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.
  - Example: The *section* schema already contains the attributes that would appear in the *sec_course* schema

Composite and Multivalued Attributes

instructor

<u>ID</u>
name
    first_name
    middle_initial
    last_name
address
    street
        street_number
        street_name
        apt_number
    city
    state
    zip
{ phone_number }
date_of_birth
age ( )

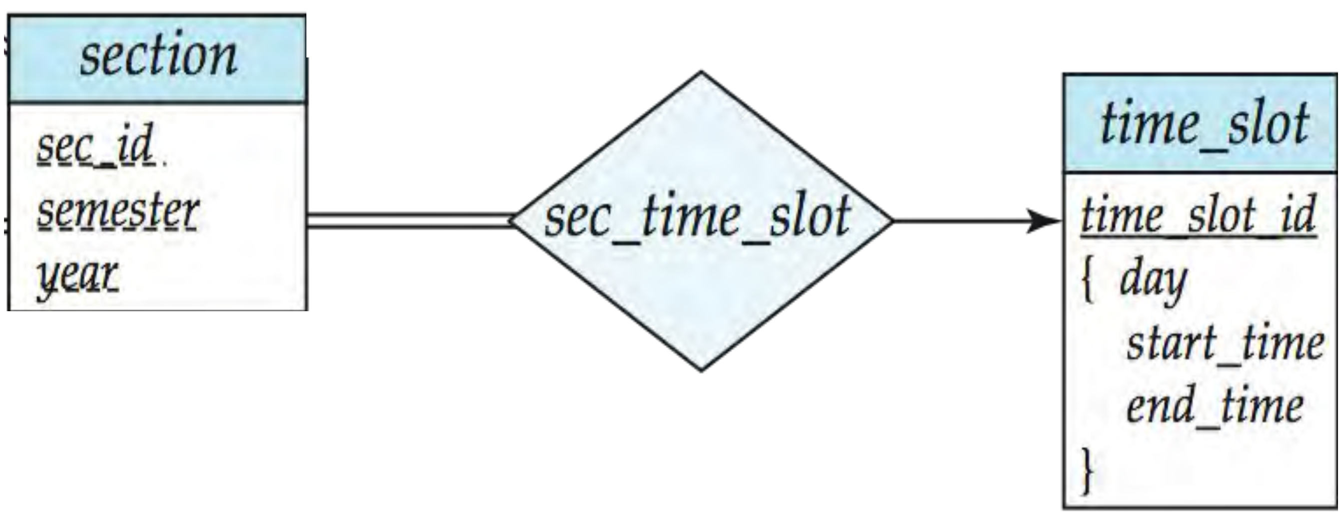
- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - Example: given entity set *instructor* with composite attribute *name* with component attributes *first_name* and *last_name* the schema corresponding to the entity set has two attributes *name_first_name* and *name_last_name*
    - *Prefix omitted if there is no ambiguity*
- Ignoring multivalued attributes, extended instructor schema is
  - *instructor(ID, first_name, middle_initial, last_name, street_number, street_name, apt_number, city, state, zip_code, date_of_birth)*

Composite and Multivalued Attributes

- A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*
  - Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*
  - Example: Multivalued attribute *phone_number* of *instructor* is represented by a schema:
    inst_phone= (*ID*, *phone_number*)
  - Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
    - For example, an *instructor* entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:
      (22222, 456-7890) and (22222, 123-4567)

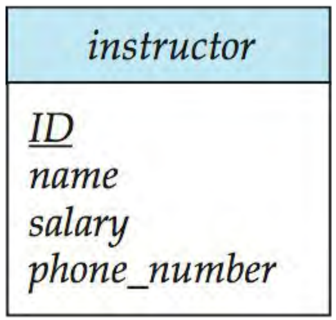
Multivalued Attributes (Cont.)

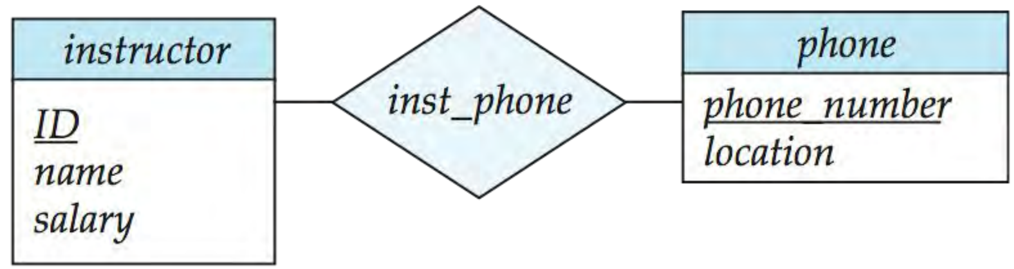
- Special case:entity *time_slot* has only one attribute other than the primary-key attribute, and that attribute is multivalued
  - Optimization: Don't create the relation corresponding to the entity, just create the one corresponding to the multivalued attribute
  - *time_slot*(<u>*time_slot_id, day, start_time*</u>, *end_time*)
  - Caveat: *time_slot* attribute of *section* (from *sec_time_slot*) cannot be a foreian key due to this optimization



Design Issues

- **Use of entity sets vs. attributes**

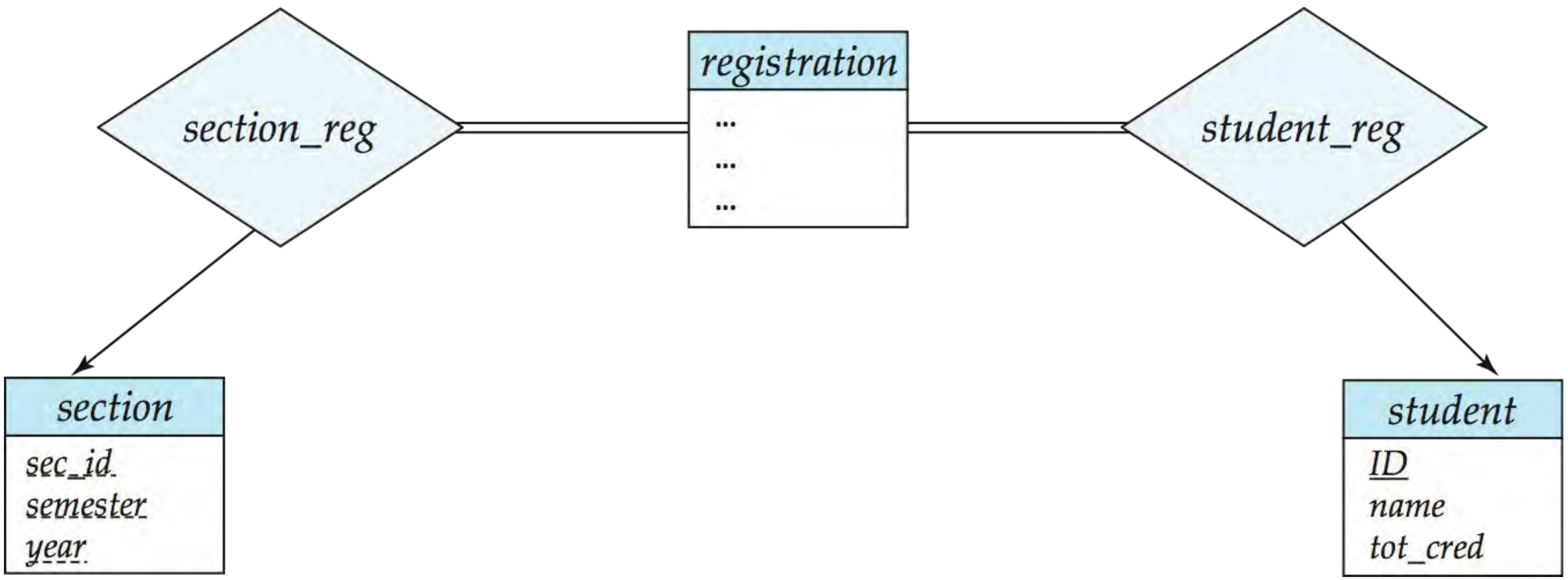




- Use of phone as an entity allows extra information about phone numbers (plus multiple phone numbers)

Design Issues

- ***Use of entity sets vs. relationship sets***
  Possible guideline is to designate a relationship set to describe an action that occurs between entities



Design Issues

- **Binary versus n-ary relationship sets**
  Although it is possible to replace any nonbinary (*n*-ary, for *n* > 2) relationship set by a number of distinct binary relationship sets, a *n*-ary relationship set shows more clearly that several entities participate in a single relationship.
- **Placement of relationship attributes**
- e.g., attribute *date* as attribute of *advisor* or as attribute of *student*

Binary Vs. Non-Binary Relationships

- Some relationships that appear to be non-binary may be better represented using binary relationships
  - E.g., A ternary relationship *parents*, relating a child to his/her father and mother, is best replaced by two binary relationships, *father* and *mother*
    - Using two binary relationships allows partial information (e.g., only mother being know)
  - But there are some relationships that are naturally non-binary
    - Example: *proj_guide*

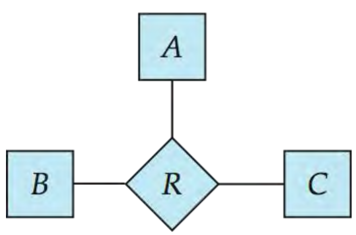
Converting Non-Binary Relationships to Binary Form

- In general, any non-binary relationship can be represented using binary relationships by creating an artificial entity set.
  - Replace *R* between entity sets A, B and C by an entity set *E*, and three relationship sets:

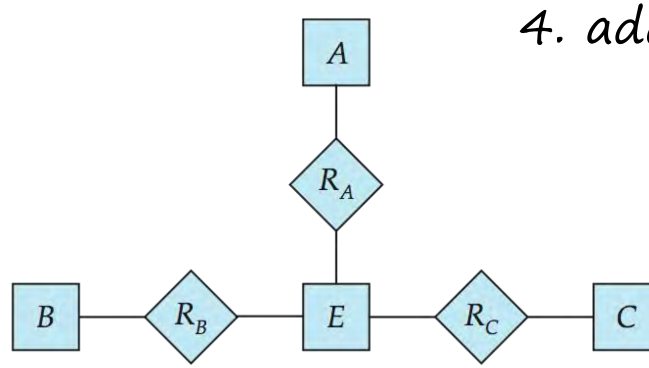
    1. R_A, relating *E* and *A*   2. R_B, relating *E* and *B*
    3. R_C, relating *E* and *C*
  - Create a special identifying attribute for *E*
  - Add any attributes of *R* to *E*
  - For each relationship (a_i, b_i, c_i) in *R*, create

    1. a new entity e_i in the entity set *E*   2. add (e_i, a_i) to R_A

    3. add (e_i, b_i) to R_B 4. add (e_i, c_i) to R_C



(a)



(b)

Converting Non-Binary Relationships (Cont.)

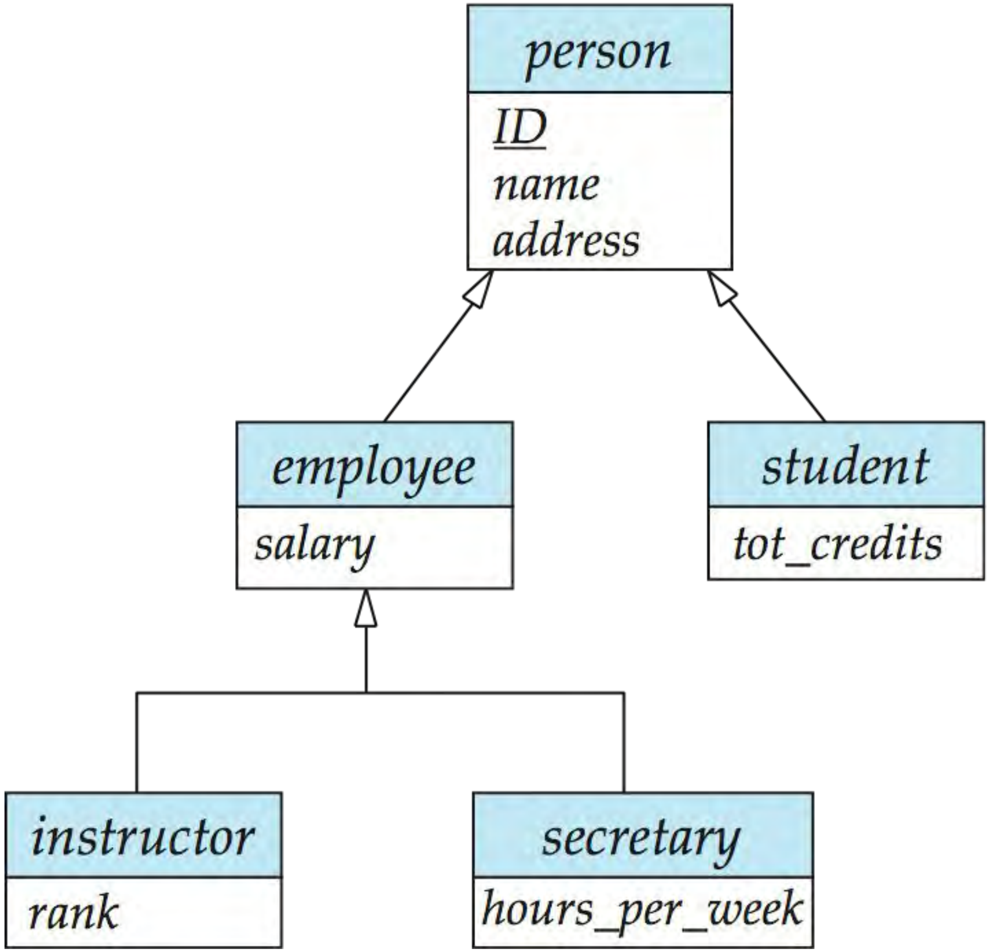
- Also need to translate constraints
  - Translating all constraints may not be possible
  - There may be instances in the translated schema that
    cannot correspond to any instance of *R*
    - Exercise: *add constraints to the relationships R_A, R_B and R_C to ensure that a newly created entity corresponds to exactly one entity in each of entity sets A, B and C*
  - We can avoid creating an identifying attribute by making E a weak entity set (described shortly) identified by the three relationship sets

Extended ER Features

Extended E-R Features: Specialization

- Top-down design process; we designate subgroupings within an entity set that are distinctive from other entities in the set.
- These subgroupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (E.g., *instructor* "is a" *person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

Specialization Example



Extended ER Features: Generalization

- **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

Specialization and Generalization (Cont.)

- Can have multiple specializations of an entity set based on different features.
- E.g., *permanent_employee* vs. *temporary_employee*, in addition to *instructor* vs. *secretary*
- Each particular employee would be
  - a member of one of *permanent_employee* or *temporary_employee*,
  - and also a member of one of *instructor, secretary*
- The ISA relationship also referred to as **superclass - subclass** relationship

Design Constraints on a Specialization/Generalization

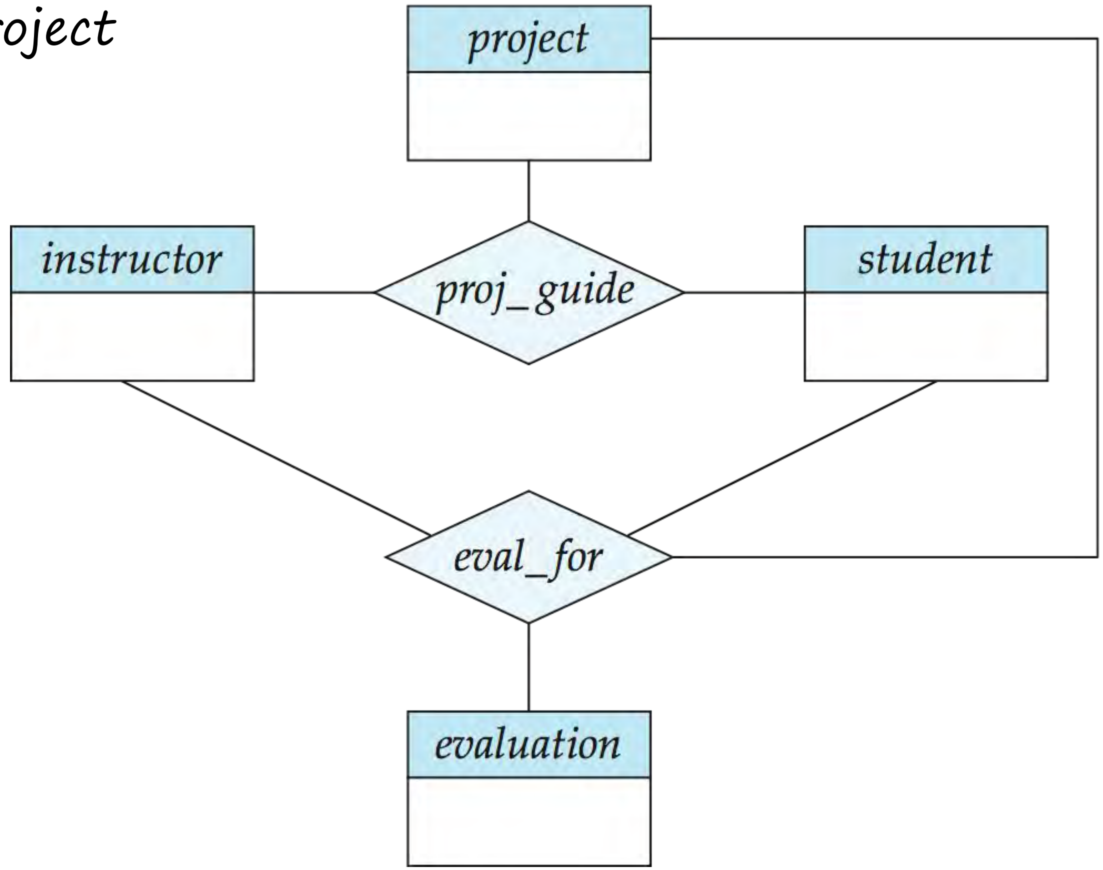
- Constraint on which entities can be members of a given lower-level entity set.
  - condition-defined
    - Example: all customers over 65 years are members of *senior-citizen* entity set; *senior-citizen* ISA *person*.
  - user-defined
- Constraint on whether or not entities may belong to more than one lower-level entity set within a single generalization.
  - **Disjoint**
    - an entity can belong to only one lower-level entity set
    - Noted in E-R diagram by having multiple lower-level entity sets link to the same triangle
  - **Overlapping**
    - an entity can belong to more than one lower-level entity set

Design Constraints on a Specialization/Generalization (Cont.)

- **Completeness constraint** -- specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization.
  - **total**: an entity must belong to one of the lower-level entity sets
  - **partial**: an entity need not belong to one of the lower-level entity sets

Aggregation

- Consider the ternary relationship *proj_guide*, which we saw earlier
- Suppose we want to record evaluations of a student by a guide on a project

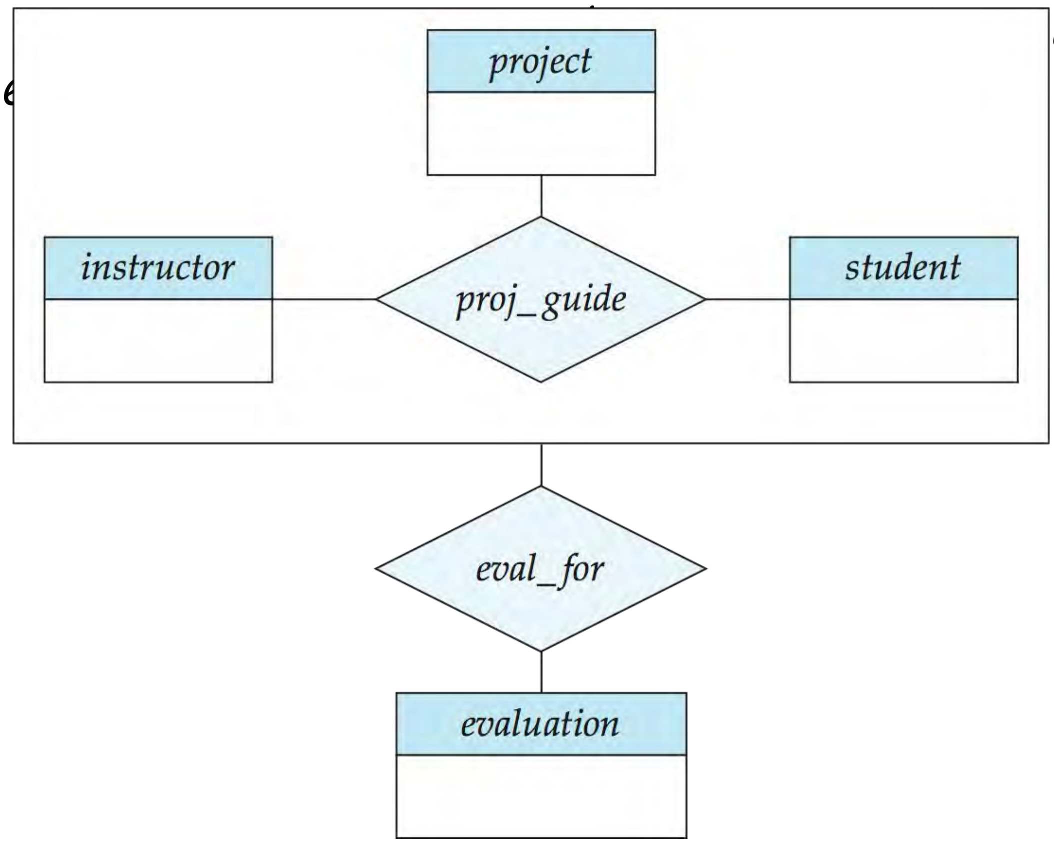


Aggregation (Cont.)

- Relationship sets *eval_for* and *proj_guide* represent overlapping information
  - Every *eval_for* relationship corresponds to a *proj_guide* relationship
  - However, some *proj_guide* relationships may not correspond to any *eval_for* relationships
    - So we can't discard the *proj_guide* relationship
- Eliminate this redundancy via *aggregation*
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity

Aggregation (Cont.)

- Without introducing redundancy, the following diagram represents:
  - A student is guided by a particular instructor on a particular project
  - A student, an
    associated e



Representing Specialization via Schemas

- Method 1:
  - Form a schema for the higher-level entity
  - Form a schema for each lower-level entity set, include primary key of higher-level entity set and local attributes

schema	attributes
person	*ID, name, street, city*
student	*ID, tot_cred*
employee	*ID, salary*

  - Drawback: getting information about, an *employee* requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema

Representing Specialization as Schemas (Cont.)

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes

schema	attributes
person	*ID, name, street, city*
student	*ID, name, street, city, tot_cred*
employee	*ID, name, street, city, salary*

  - If specialization is total, the schema for the generalized entity set (*person*) not required to store information
  - Can be defined as a "view" relation containing union of specialization relations
  - But explicit schema may still be needed for foreign key constraints
  - Drawback: *name, street* and *city* may be stored redundantly for people who are both students and employees

Schemas Corresponding to Aggregation

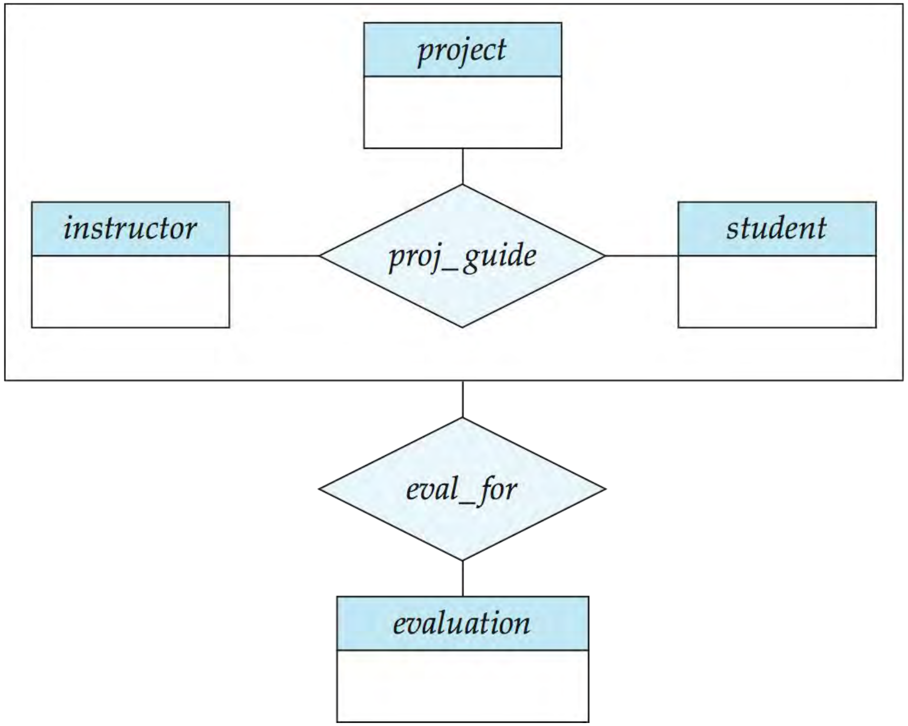
- To represent aggregation, create a schema containing
  - primary key of the aggregated relationship,
  - the primary key of the associated entity set
  - any descriptive attributes

Schemas Corresponding to Aggregation (Cont.)

- For example, to represent aggregation manages between relationship works_on and entity set manager, create a schema

  eval_for (s_ID, project_id, i_ID, evaluation_id)

- Schema *proj_guide* is redundant provided we are willing to store null values for attribute *manager_name* in relation on schema *manages*



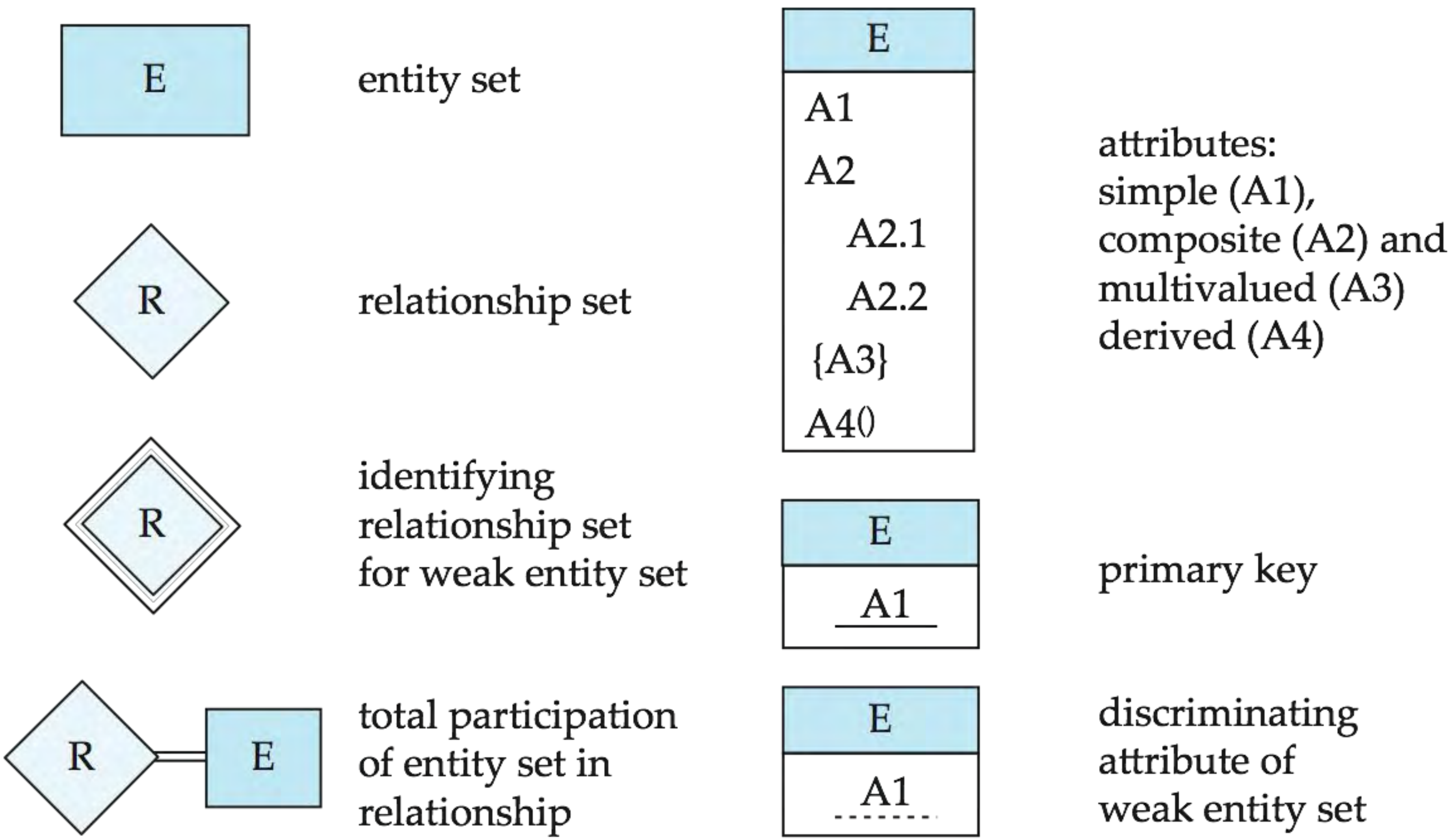
E-R Design Decisions

- The use of an attribute or entity set to represent an object.
- Whether a real-world concept is best expressed by an entity set or a relationship set.
- The use of a ternary relationship versus a pair of binary relationships.
- The use of a strong or weak entity set.
- The use of specialization/generalization – contributes to modularity in the design.
- The use of aggregation – can treat the aggregate entity set as a single unit without concern for the details of its internal structure.

How about doing another ER design interactively on the board?

Database System Concepts, 6th Ed.

Summary of Symbols Used in E-R Notation

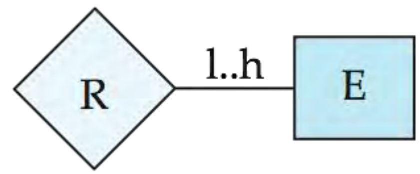


Symbols Used in E-R Notation (Cont.)

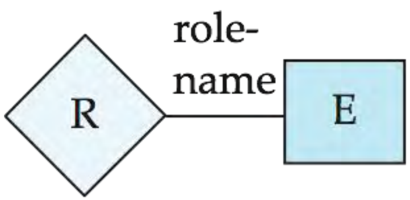
many-to-many relationship

many-to-one relationship

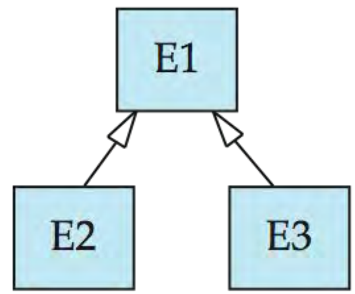
one-to-one relationship



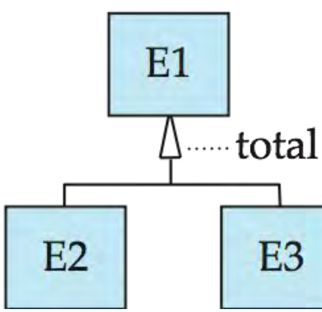
cardinality limits



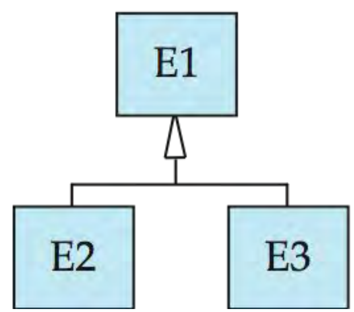
role indicator



ISA: generalization or specialization



total (disjoint) generalization

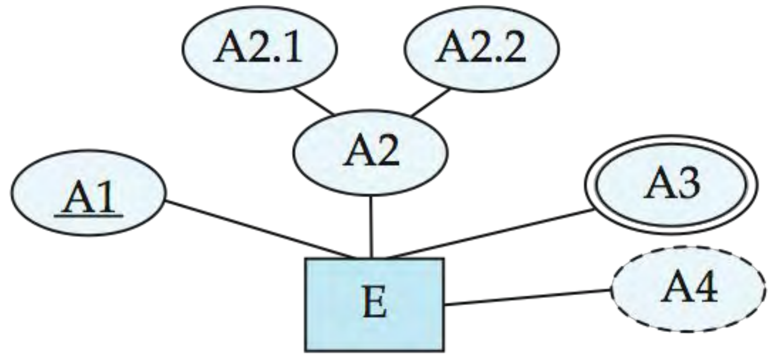


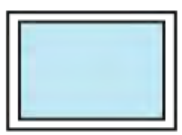
disjoint generalization

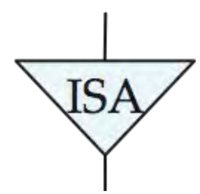
Alternative ER Notations

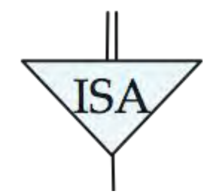
- Chen, IDE1FX, …

entity set E with
simple attribute A1,
composite attribute A2,
multivalued attribute A3,
derived attribute A4,
and primary key A1



weak entity set 

generalization 

total generalization 

Alternative ER Notations

Chen

IDE1FX

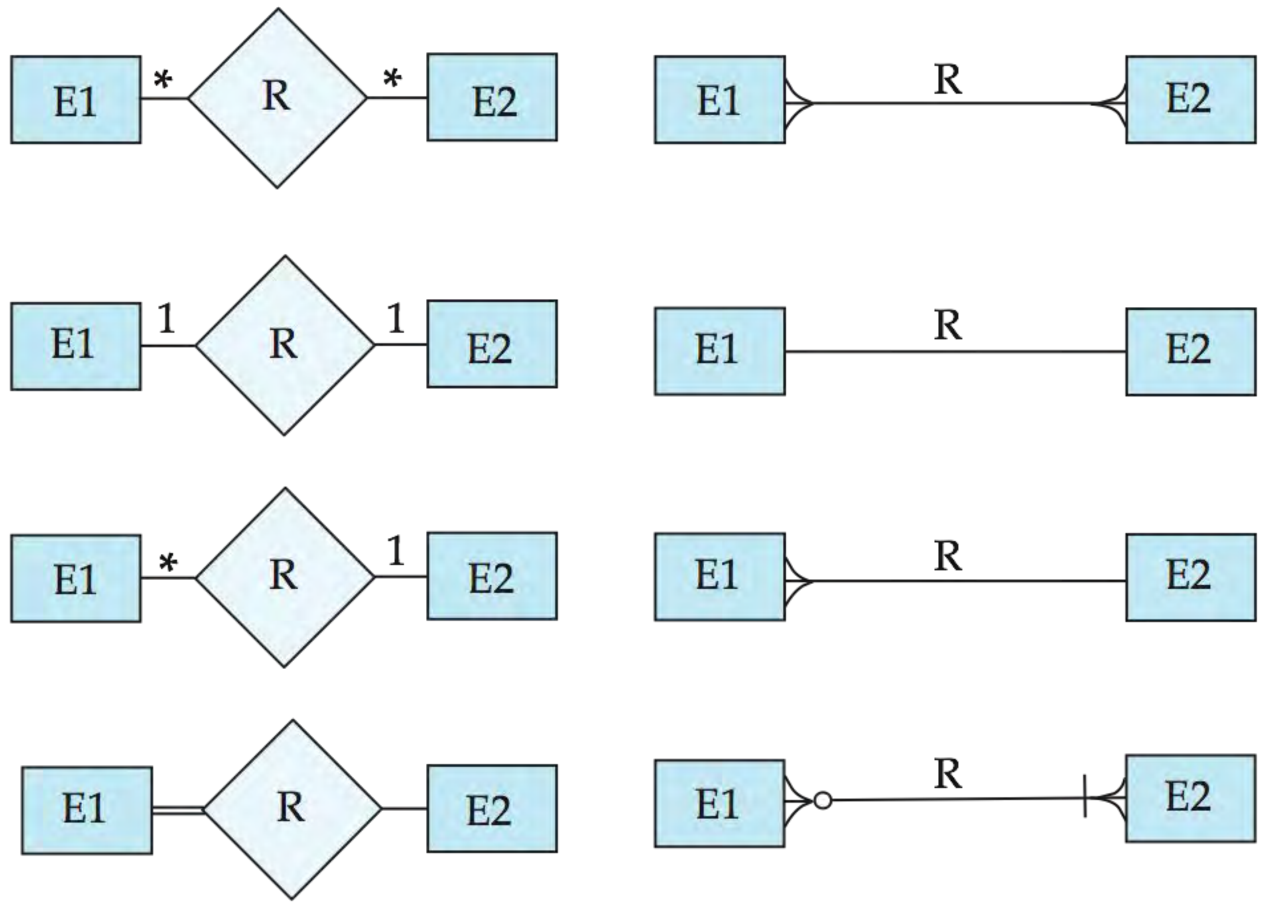
(Crows feet notation)

many-to-many relationship

one-to-one relationship

many-to-one relationship

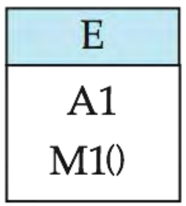
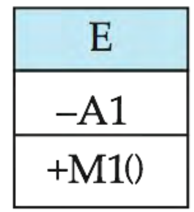
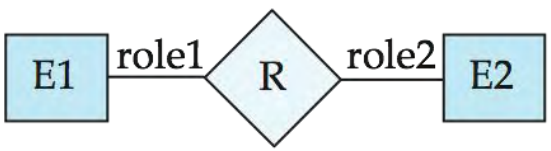
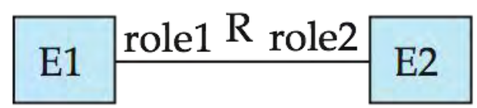
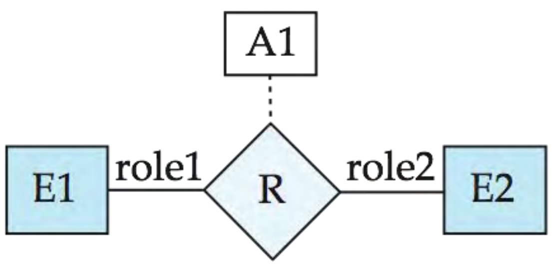
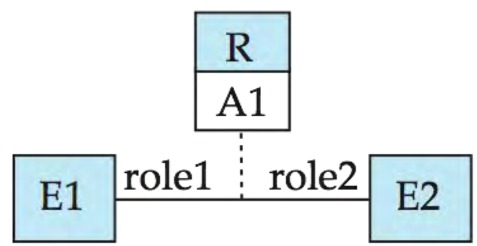
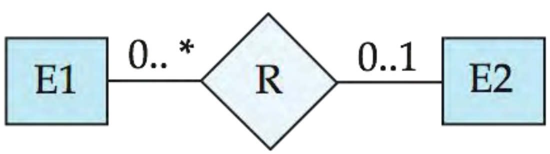
participation in R: total (E1) and partial (E2)



UML

- **UML**: Unified Modeling Language
- UML has many components to graphically model different aspects of an entire software system
- UML Class Diagrams correspond to E-R Diagram, but several differences.

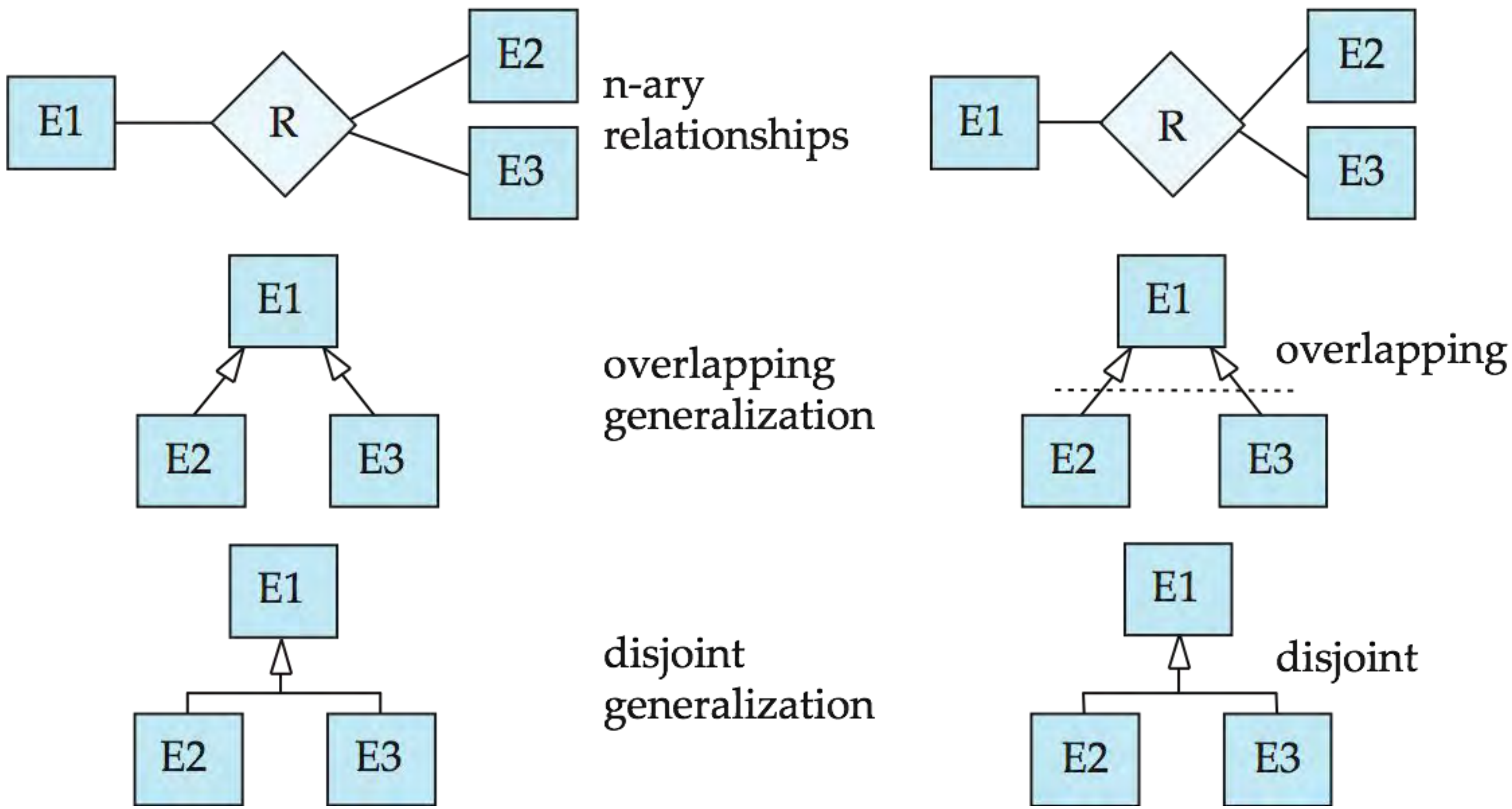
ER vs. UML Class Diagrams

ER Diagram Notation		Equivalent in UML	
	entity with attributes (simple, composite, multivalued, derived)		class with simple attributes and methods (attribute prefixes: + = public, – = private, # = protected)
	binary relationship		
	relationship attributes		
	cardinality constraints		

*Note reversal of position in cardinality constraint depiction

ER vs. UML Class Diagrams

ER Diagram Notation | **Equivalent in UML**



*Generalization can use merged or separate arrows independent of disjoint/overlapping

UML Class Diagrams (Cont.)

- Binary relationship sets are represented in UML by just drawing a line connecting the entity sets. The relationship set name is written adjacent to the line.
- The role played by an entity set in a relationship set may also be specified by writing the role name on the line, adjacent to the entity set.
- The relationship set name may alternatively be written in a box, along with attributes of the relationship set, and the box is connected, using a dotted line, to the line depicting the relationship set.

End of Chapter 7

Database System Concepts, 6th Ed.

Figure 7.01

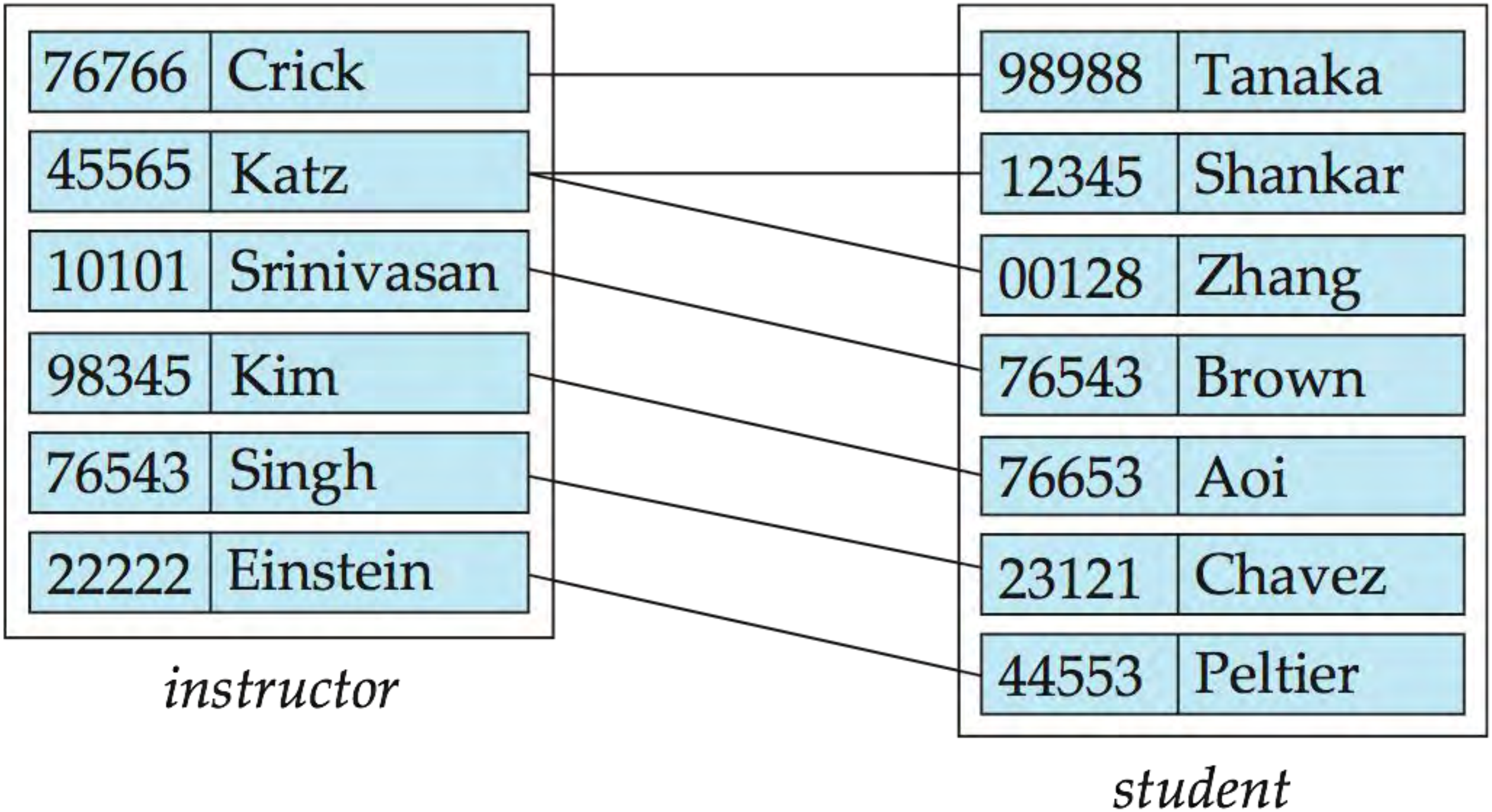
76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Figure 7.02



76766 Crick
45565 Katz
10101 Srinivasan
98345 Kim
76543 Singh
22222 Einstein
instructor
98988 Tanaka
12345 Shankar
00128 Zhang
76543 Brown
76653 Aoi
23121 Chavez
44553 Peltier
student

Figure 7.03

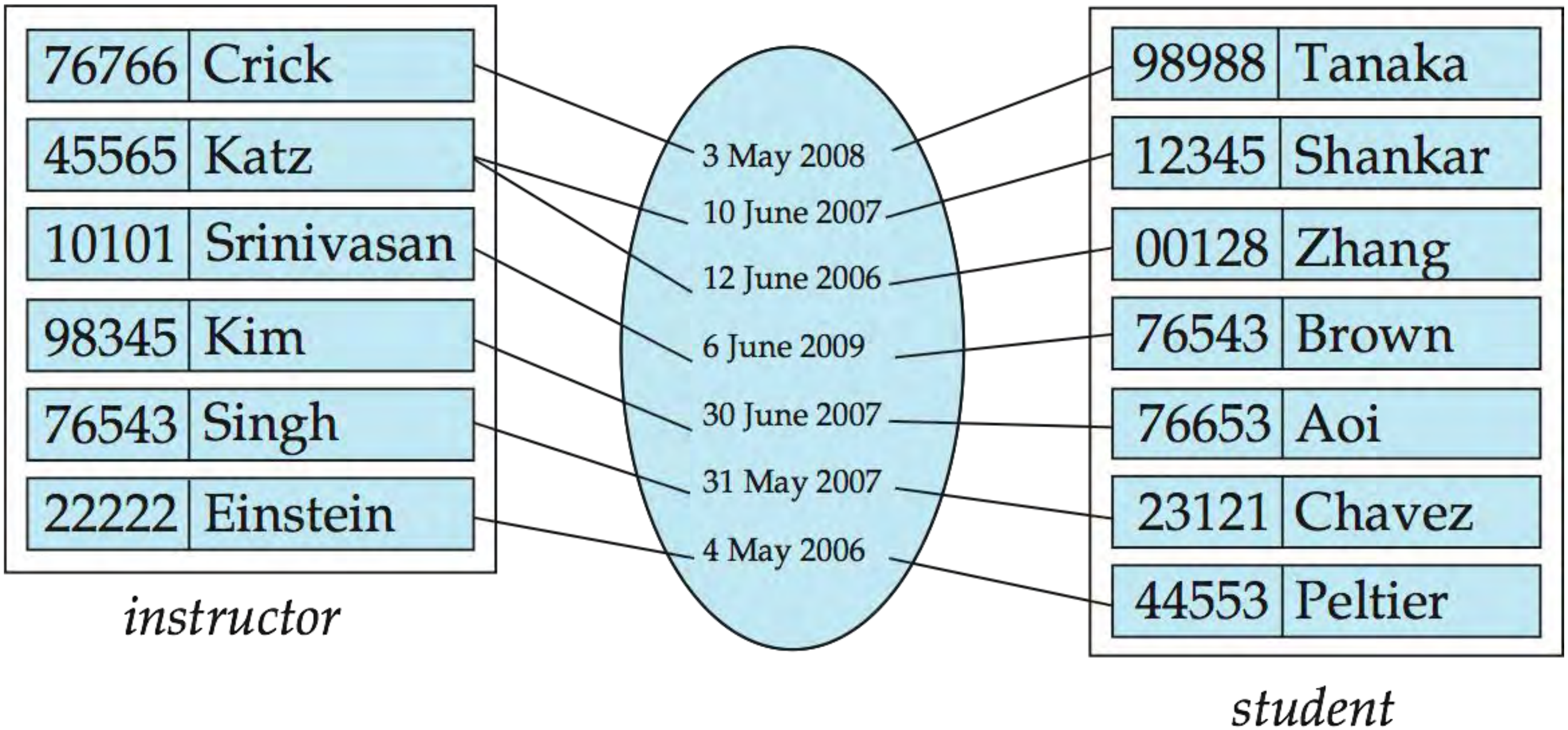

76766 Crick
45565 Katz
10101 Srinivasan
98345 Kim
76543 Singh
22222 Einstein
instructor
3 May 2008
10 June 2007
12 June 2006
6 June 2009
30 June 2007
31 May 2007
4 May 2006
98988 Tanaka
12345 Shankar
00128 Zhang
76543 Brown
76653 Aoi
23121 Chavez
44553 Peltier
student

Figure 7.04

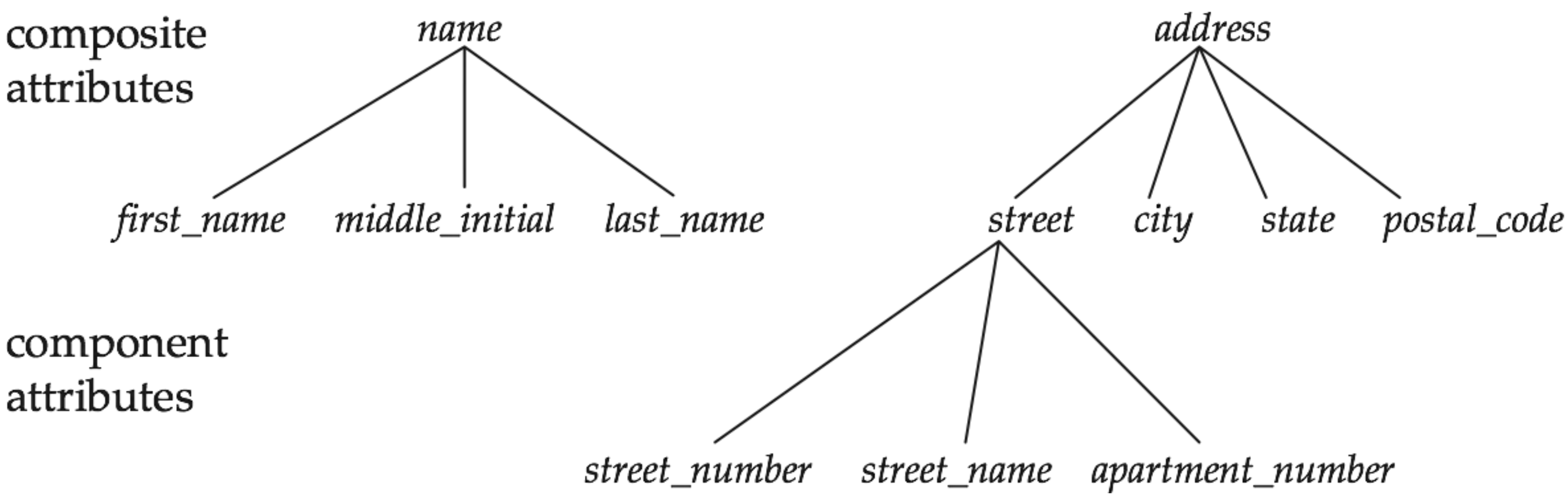
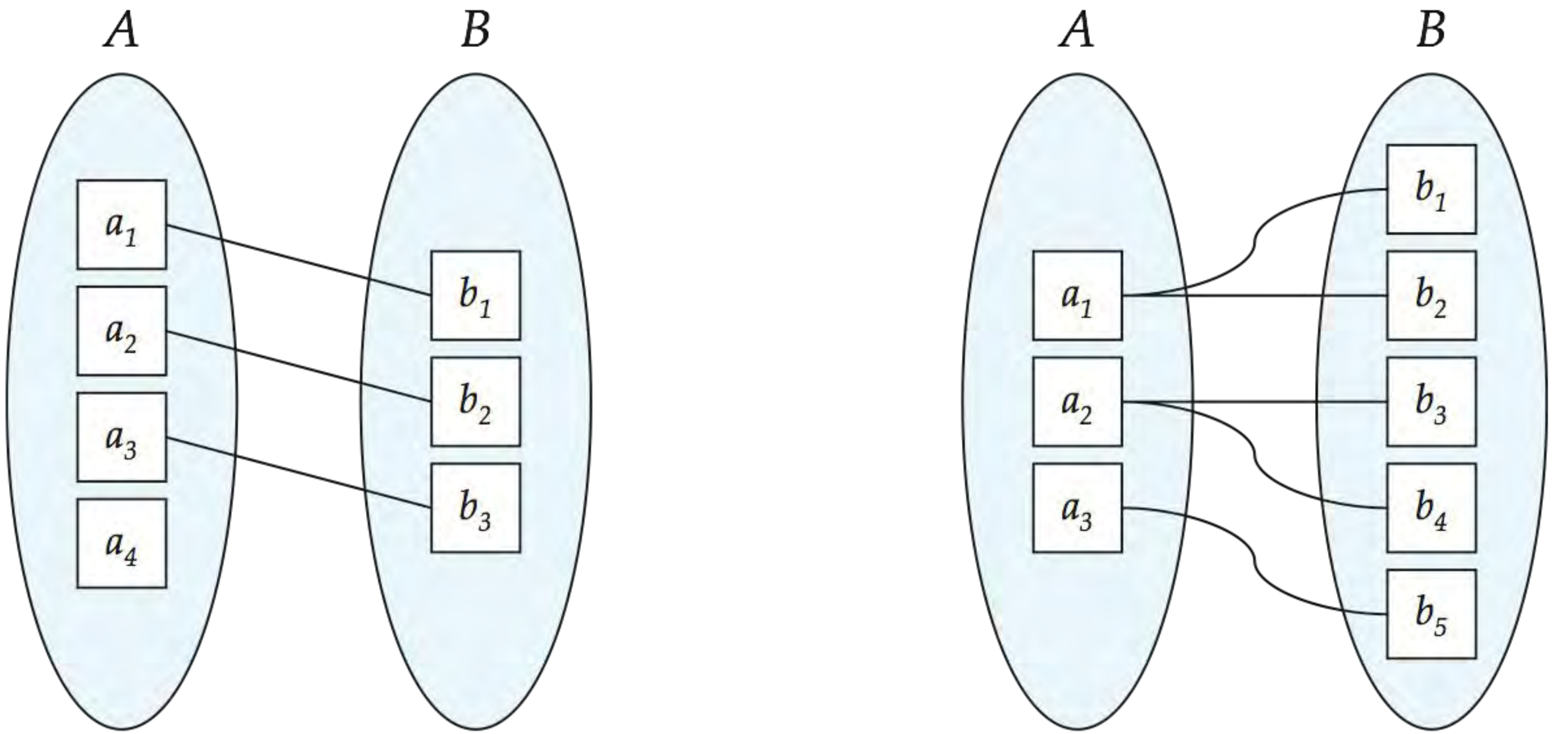

composite
attributes
component
attributes
name
first_name
middle_initial
last_name
address
street
city
state
postal_code
street_number
street_name
apartment_number

Figure 7.05



(a) (b)

Figure 7.06

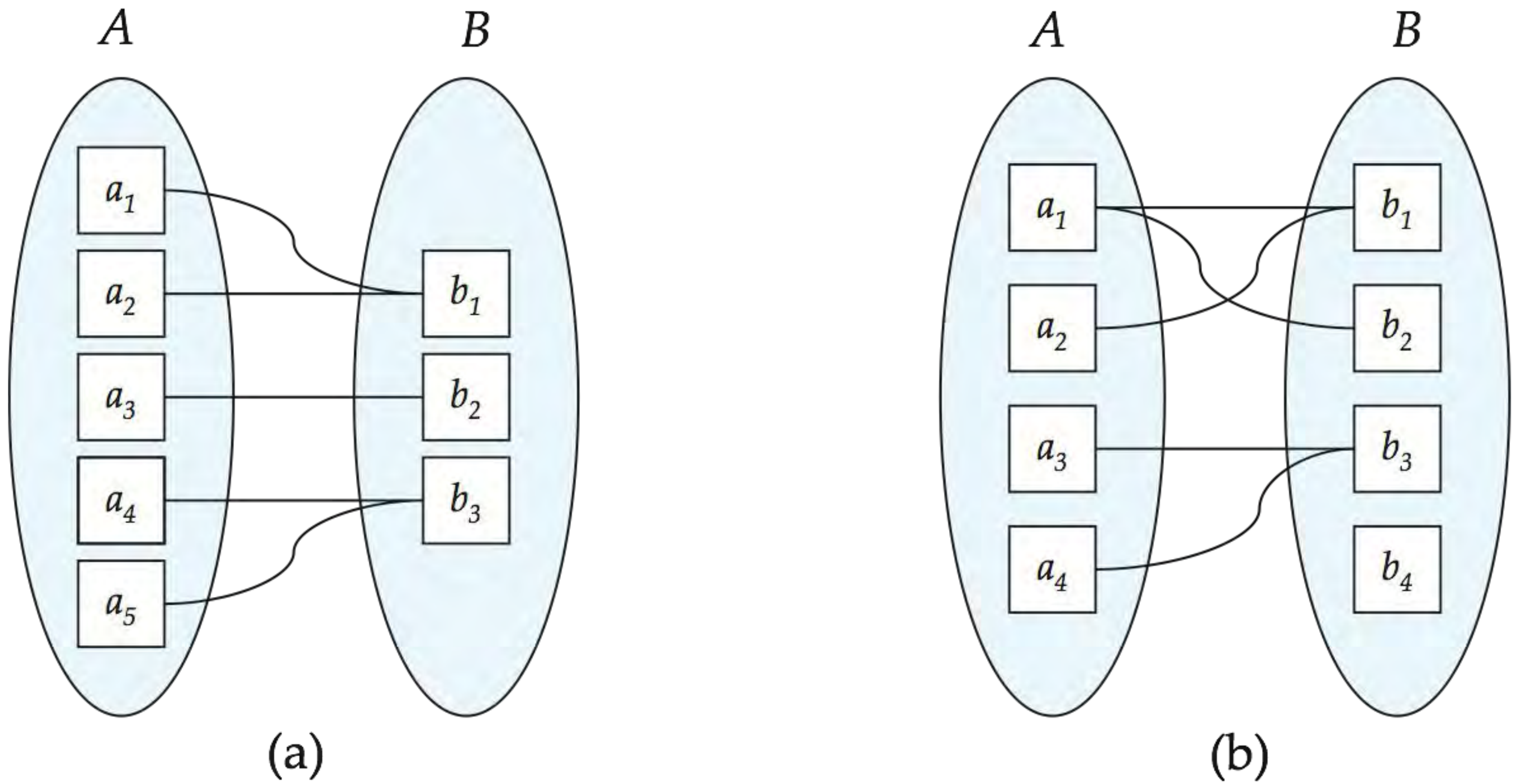


Figure 7.07

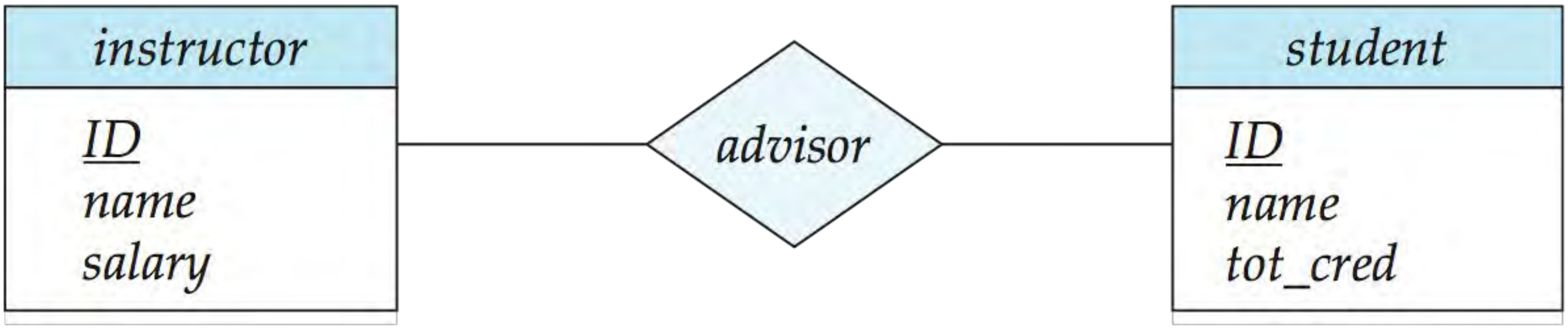


Figure 7.08

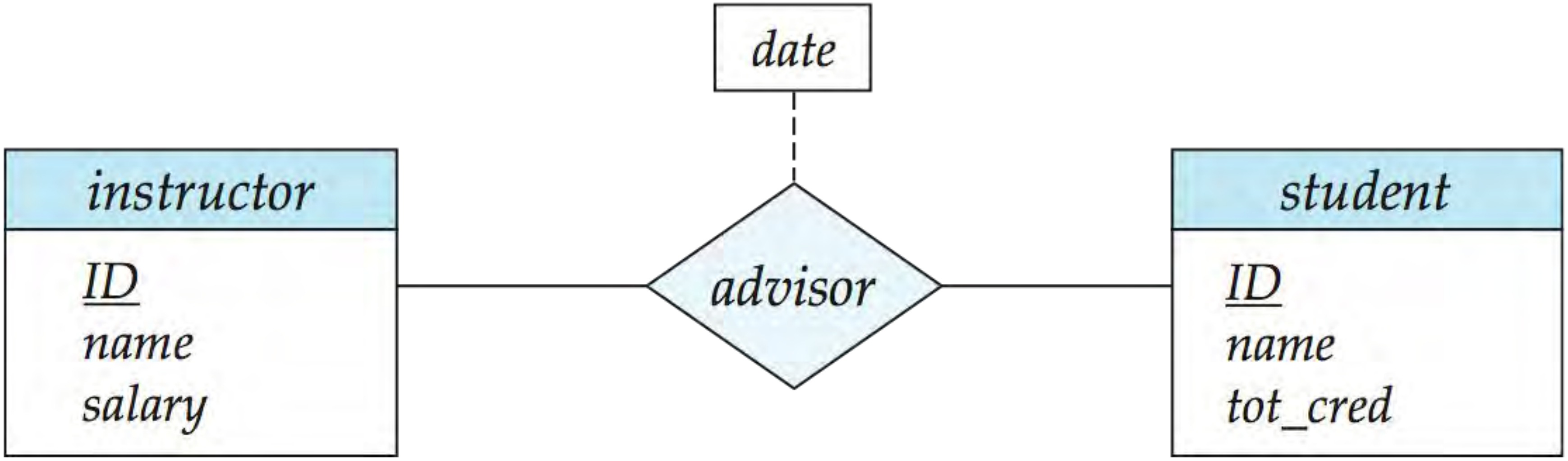

date
instructor
ID
name
salary
advisor
student
ID
name
tot_cred

Figure 7.09

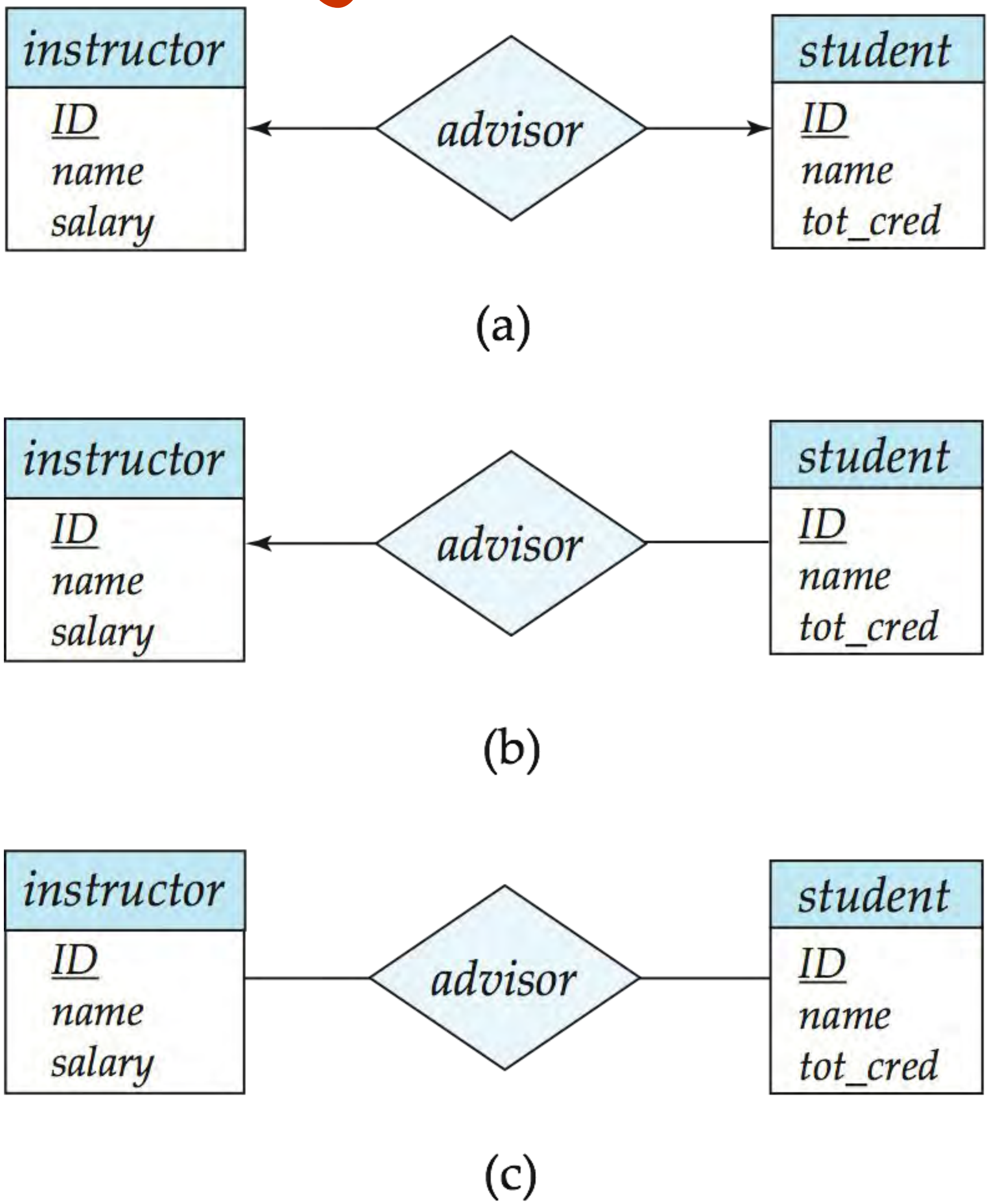

instructor
ID
name
salary
advisor
student
ID
name
tot_cred
(a)
instructor
ID
name
salary
advisor
student
ID
name
tot_cred
(b)
instructor
ID
name
salary
advisor
student
ID
name
tot_cred
(c)

Figure 7.10

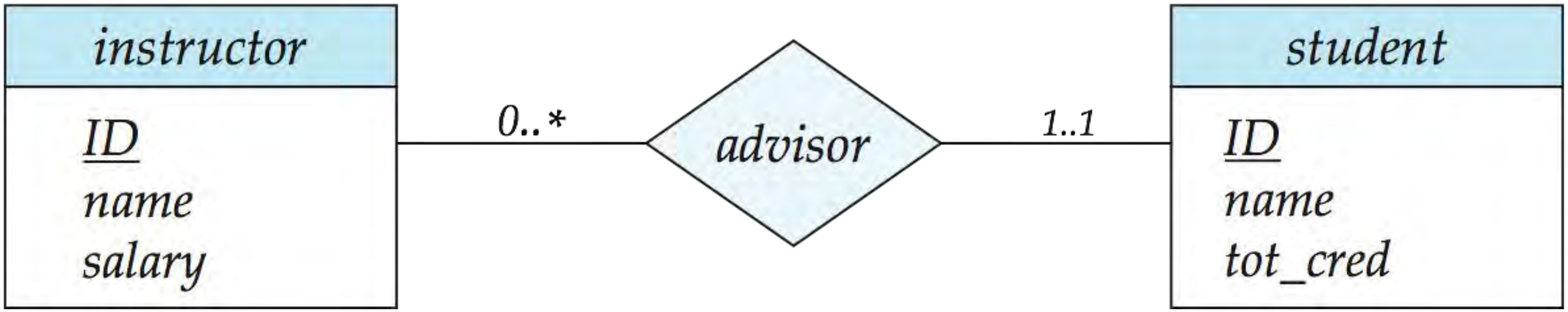

instructor
ID
name
salary
0..*
advisor
1..1
student
ID
name
tot_cred

Figure 7.11

instructor

- <u>ID</u>
- name
  - first_name
  - middle_initial
  - last_name
- address
  - street
    - street_number
    - street_name
    - apt_number
  - city
  - state
  - zip
- { phone_number }
- date_of_birth
- age ( )

Figure 7.12

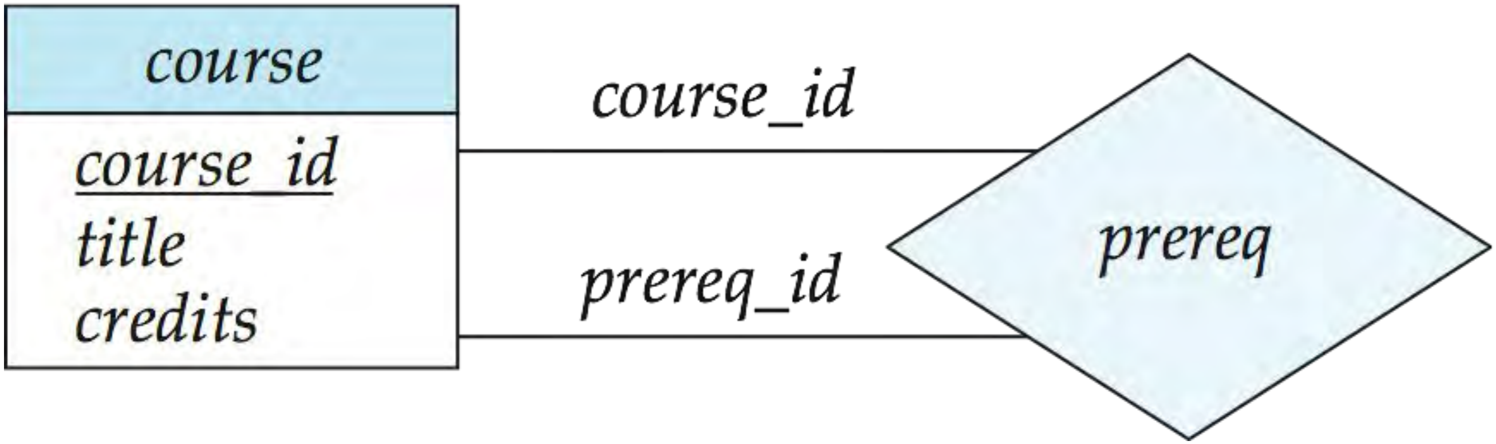


Figure 7.13

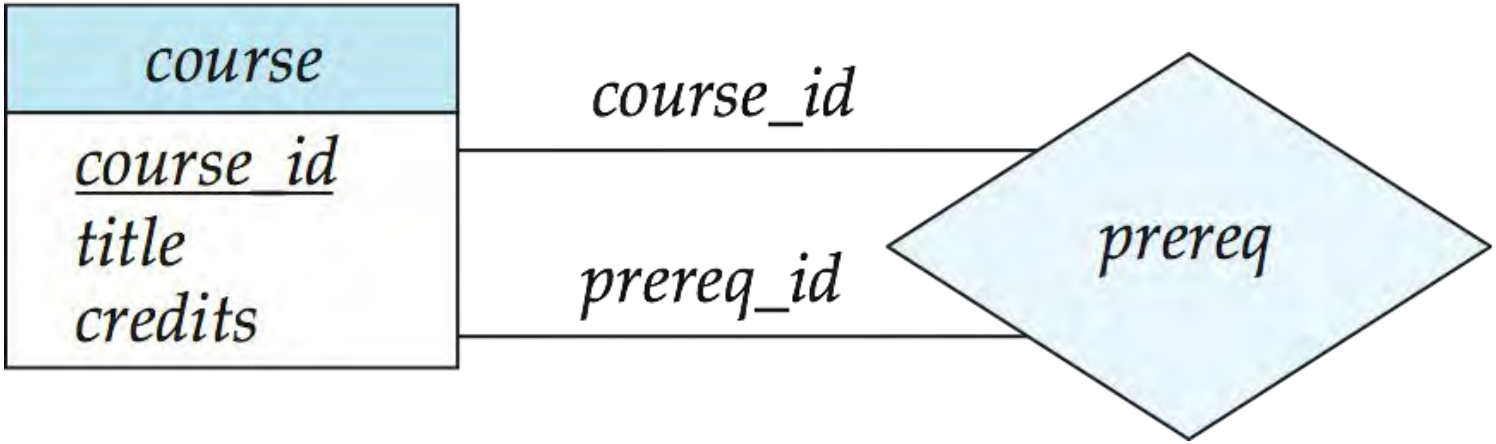


Figure 7.14

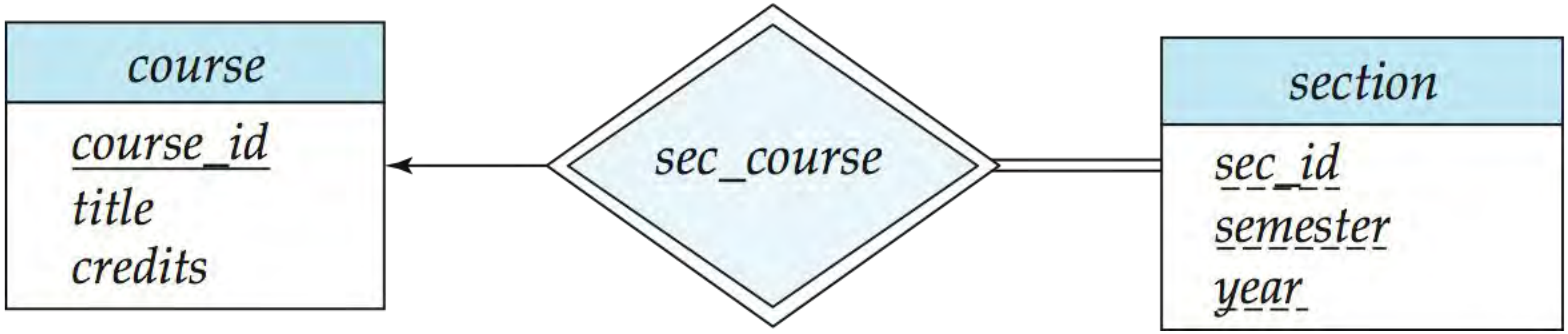
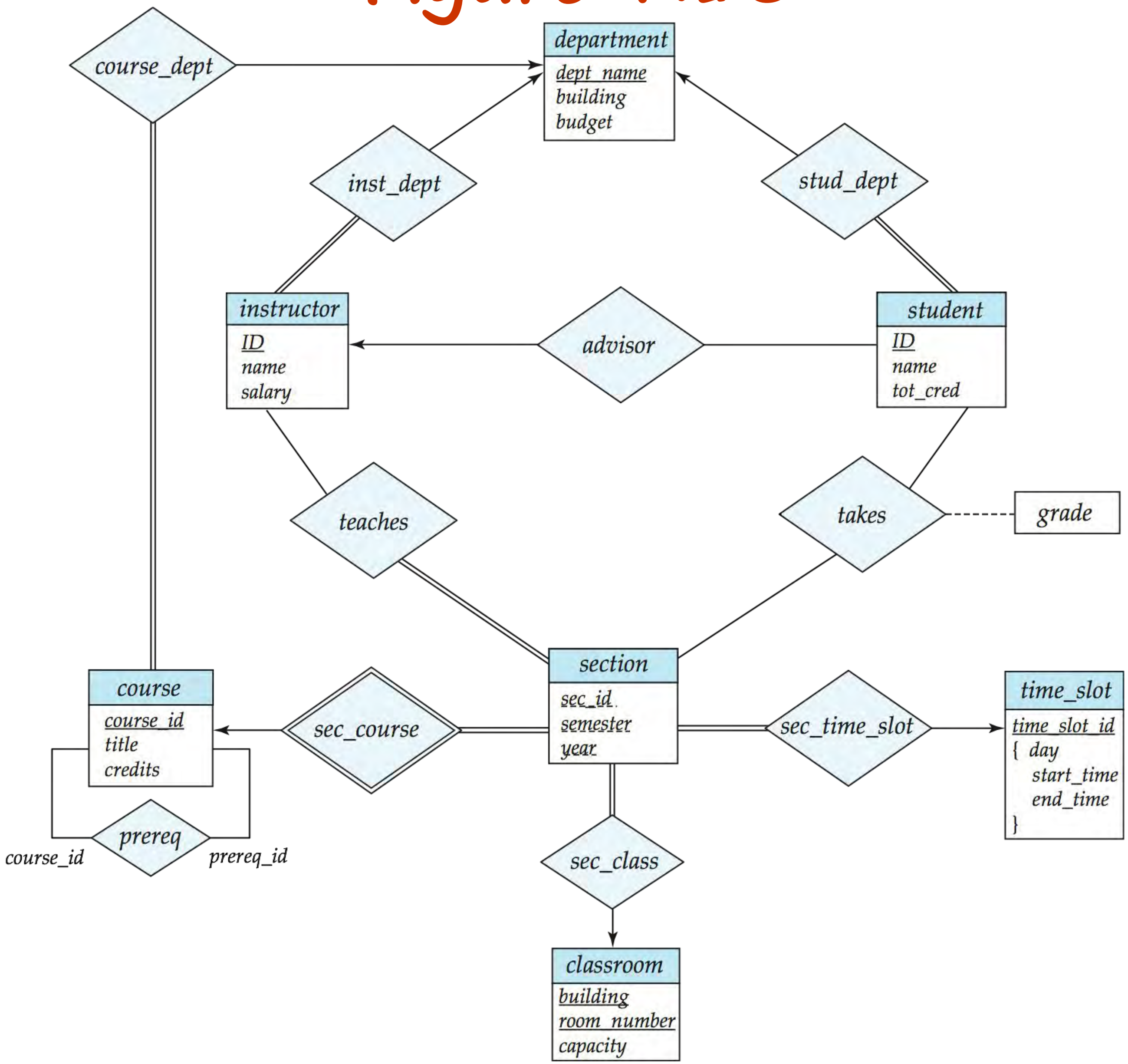

course
course_id
title
credits
sec_course
section
sec_id
semester
year

Figure 7.15



course_dept
department
dept_name
building
budget
inst_dept
stud_dept
instructor
ID
name
salary
advisor
student
ID
name
tot_cred
teaches
takes
grade
section
sec_id
semester
year
course
course_id
title
credits
sec_course
sec_time_slot
time_slot
time_slot_id
{ day
start_time
end_time
}
prereq
course_id
prereq_id
sec_class
classroom
building
room_number
capacity

Figure 7.17

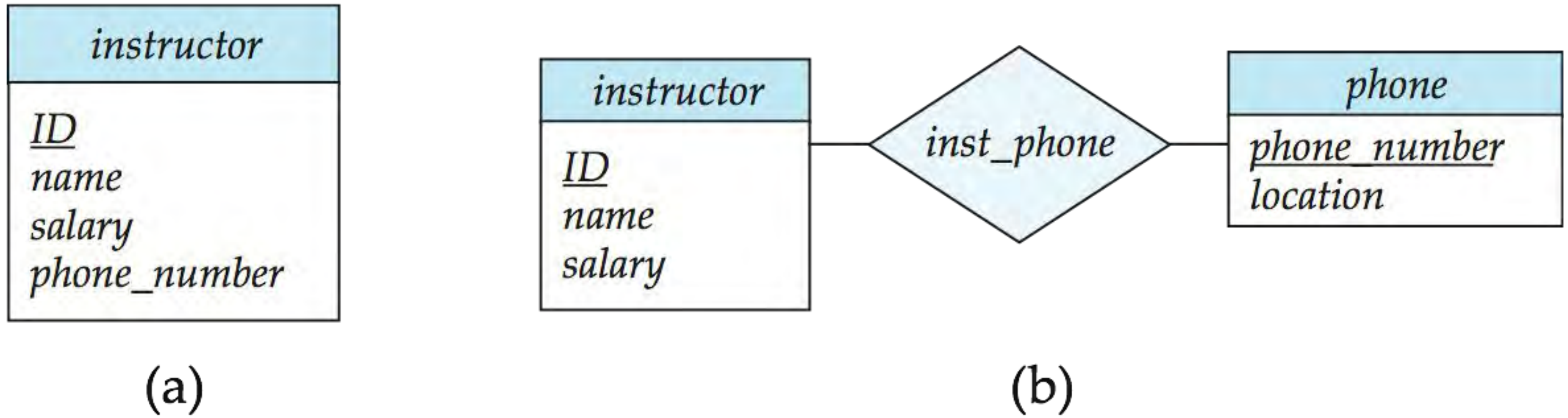
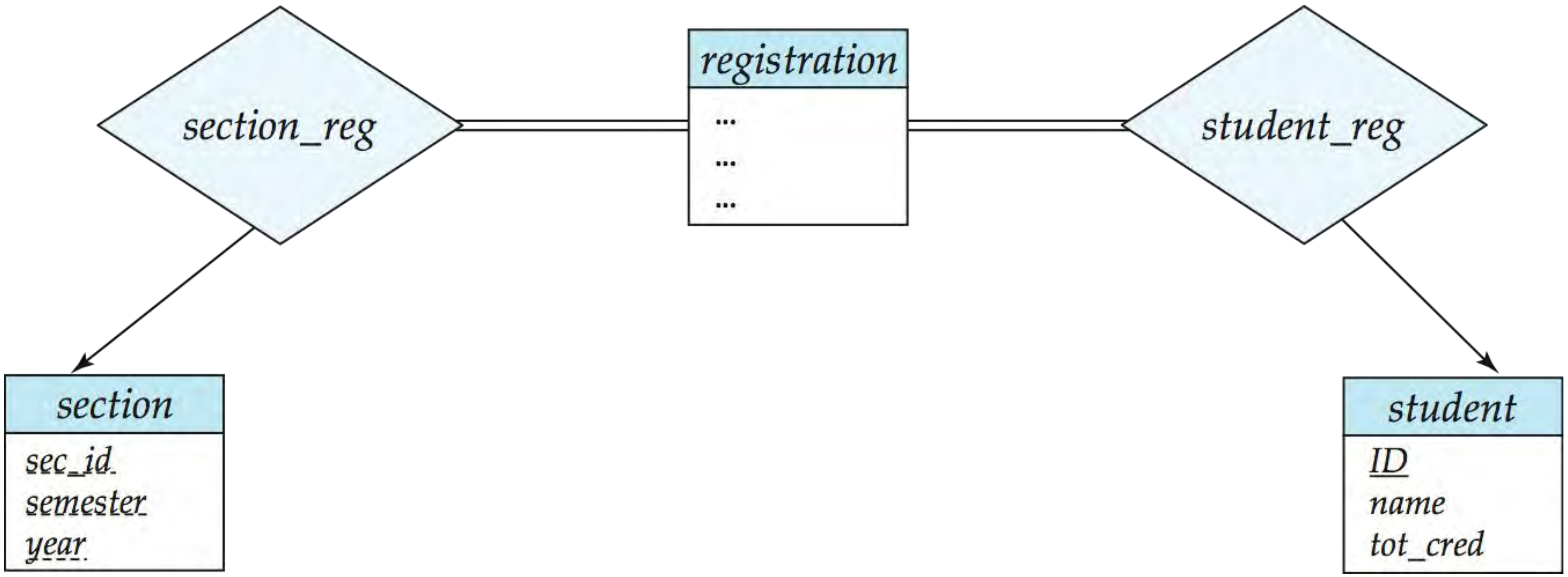
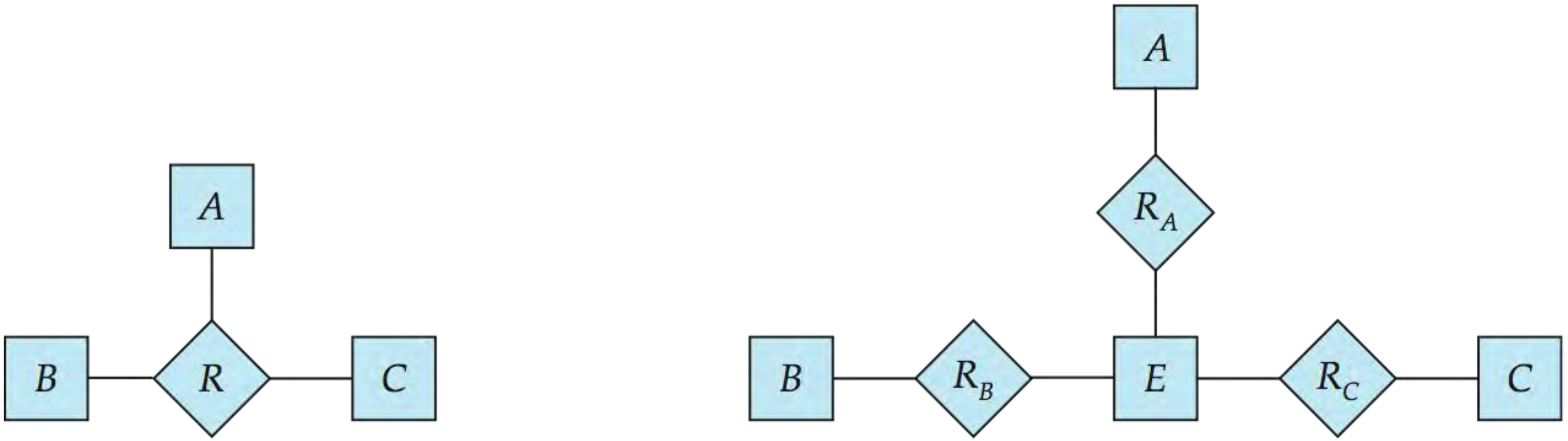

instructor
ID
name
salary
phone_number
(a)
instructor
ID
name
salary
inst_phone
phone
phone_number
location
(b)

Figure 7.18



section_reg
registration
...
...
...
student_reg
section
sec_id
semester
year
student
ID
name
tot_cred

Figure 7.19



(a) (b)

Figure 7.20

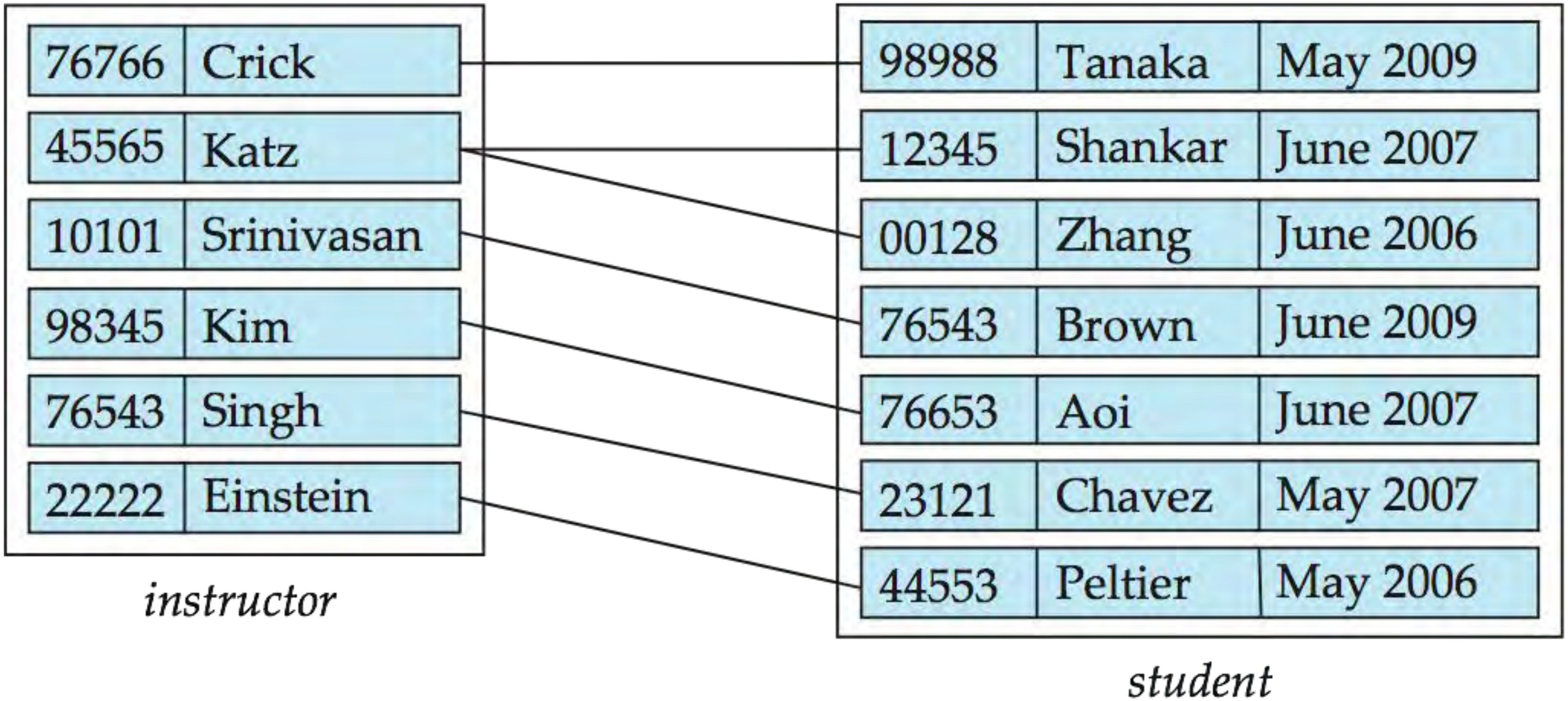

76766 Crick
45565 Katz
10101 Srinivasan
98345 Kim
76543 Singh
22222 Einstein
instructor
98988 Tanaka May 2009
12345 Shankar June 2007
00128 Zhang June 2006
76543 Brown June 2009
76653 Aoi June 2007
23121 Chavez May 2007
44553 Peltier May 2006
student

Figure 7.21

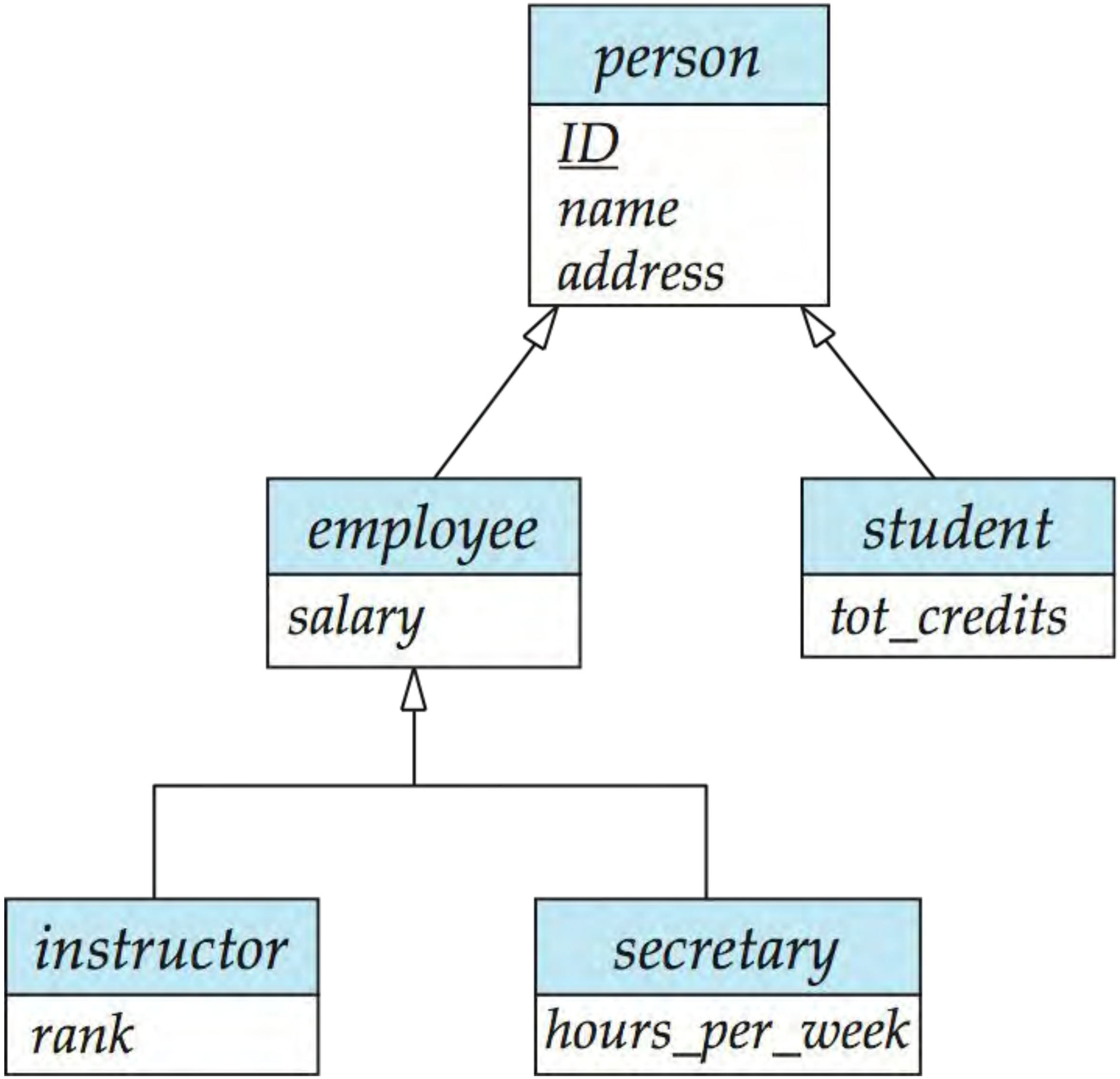

person
ID
name
address
employee
salary
student
tot_credits
instructor
rank
secretary
hours_per_week

Figure 7.22

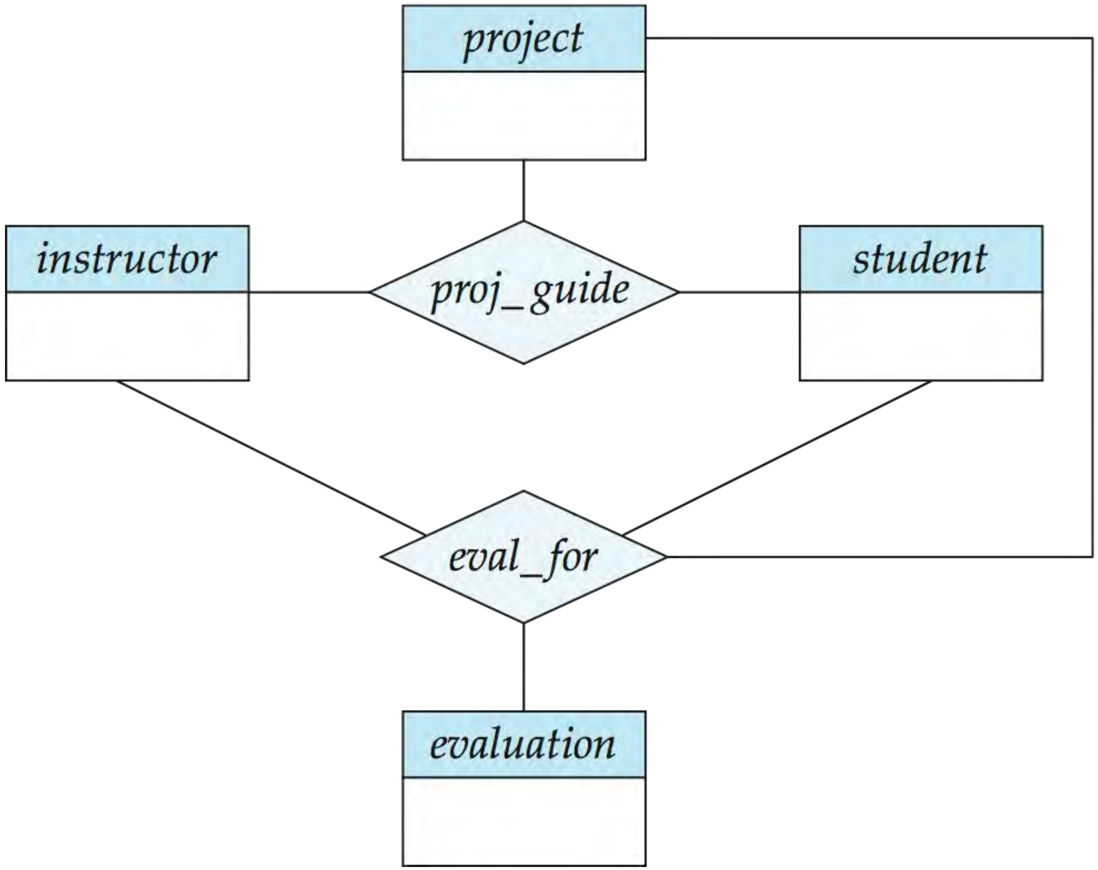

project
instructor
proj_guide
student
eval_for
evaluation

Figure 7.23

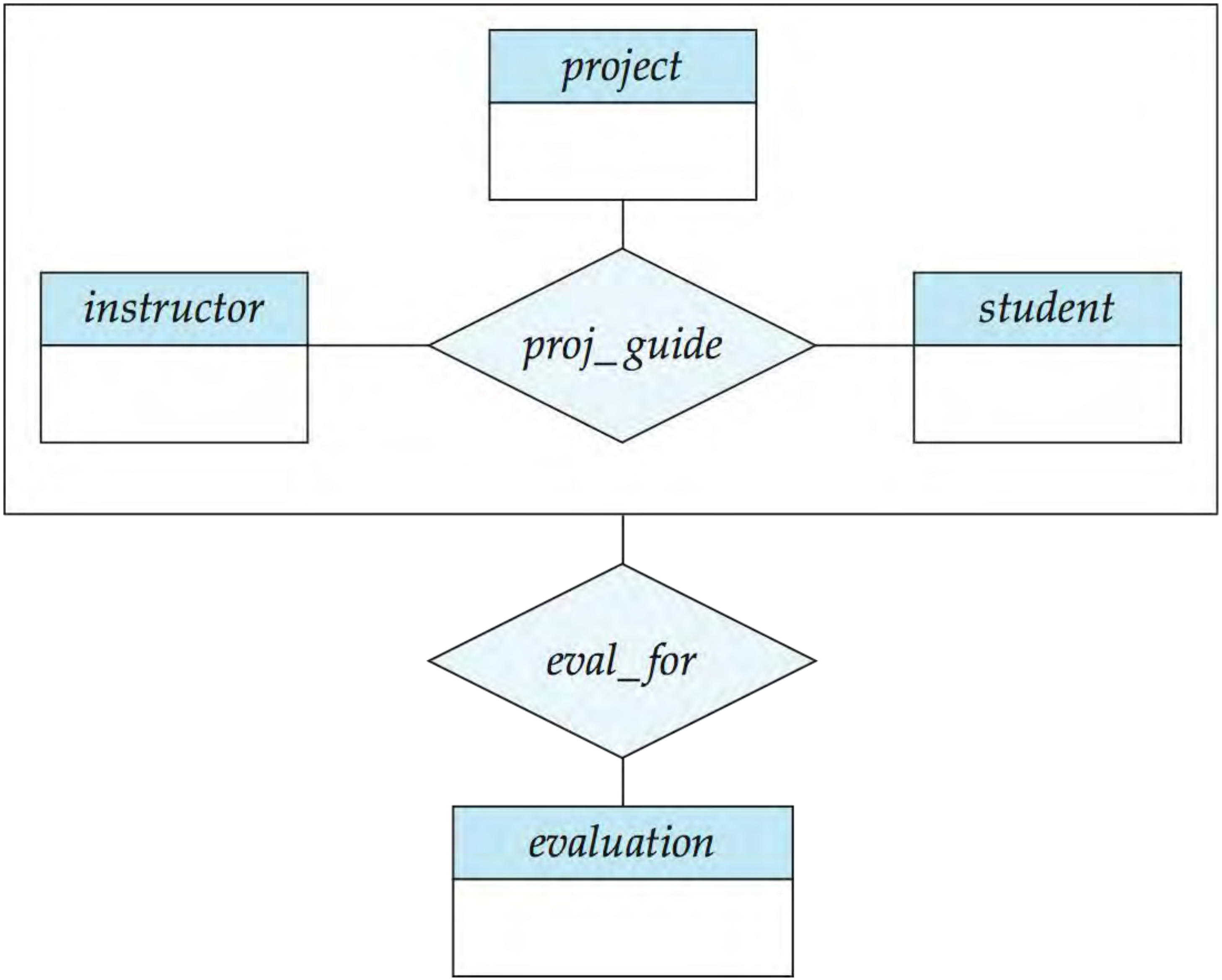

project
instructor
proj_guide
student
eval_for
evaluation

Figure 7.24

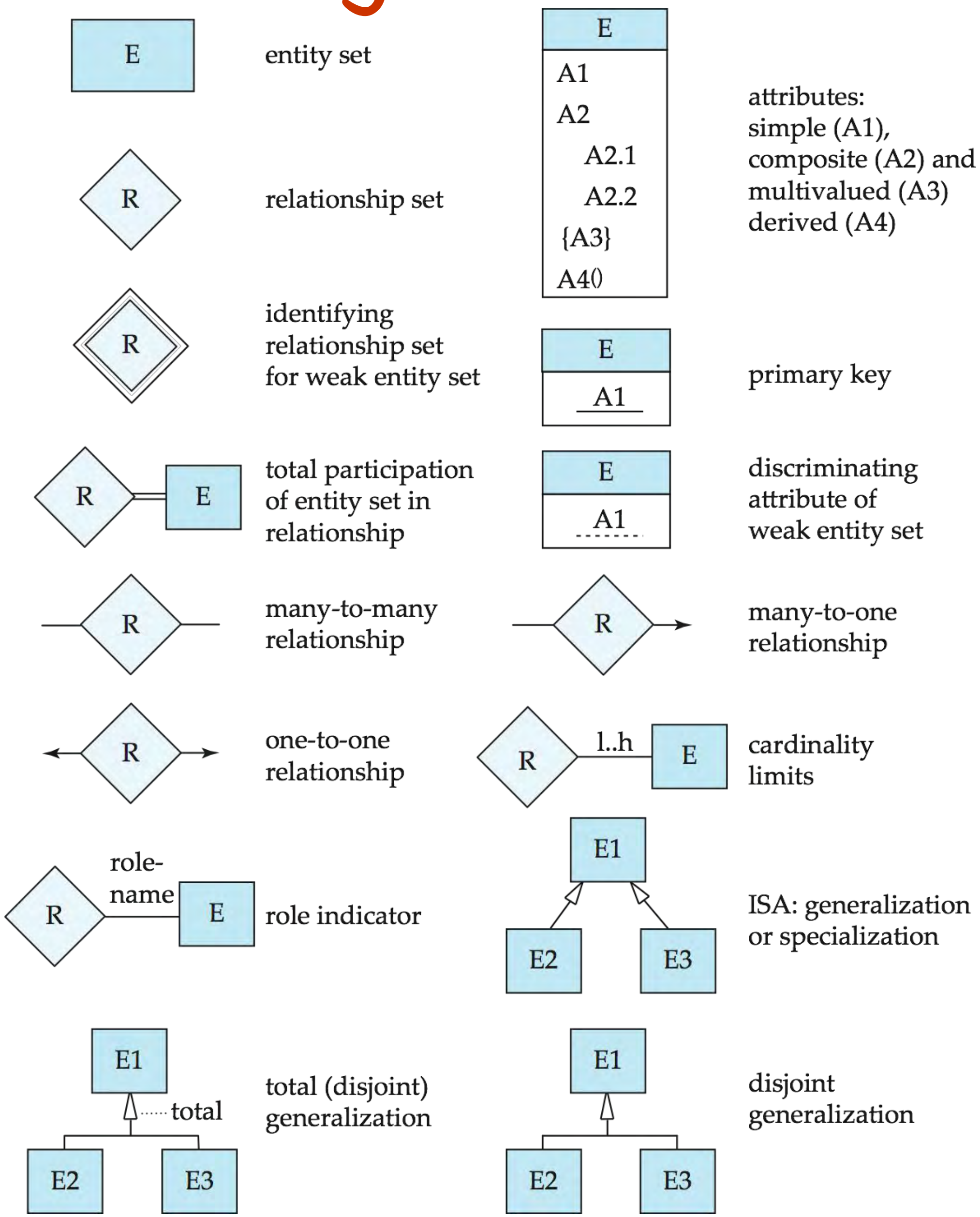
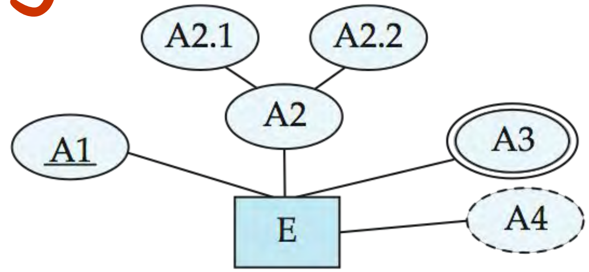

E
entity set
R
relationship set
R
identifying
relationship set
for weak entity set
R
E
total participation
of entity set in
relationship
R
many-to-many
relationship
R
one-to-one
relationship
R
role-
name
E
role indicator
E1
total
E2
E3
total (disjoint)
generalization
E
A1
A2
A2.1
A2.2
{A3}
A4()
attributes:
simple (A1),
composite (A2) and
multivalued (A3)
derived (A4)
E
A1
primary key
E
A1
discriminating
attribute of
weak entity set
R
many-to-one
relationship
R
l..h
E
cardinality
limits
E1
E2
E3
ISA: generalization
or specialization
E1
E2
E3
disjoint
generalization

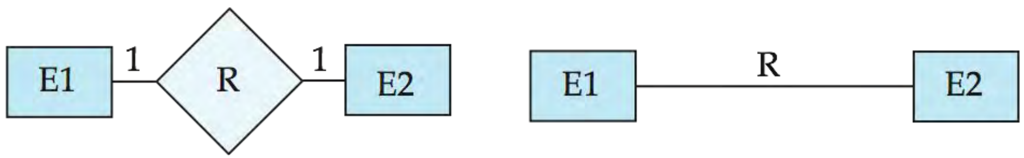
Figure 7.25

entity set E with
simple attribute A1,
composite attribute A2,
multivalued attribute A3,
derived attribute A4,
and primary key A1



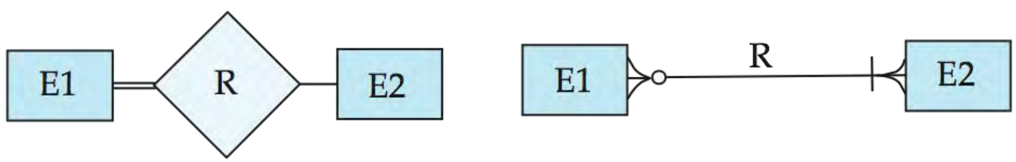
many-to-many
relationship

one-to-one
relationship



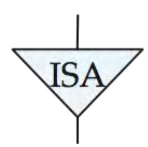
many-to-one
relationship

participation
in R: total (E1)
and partial (E2)



weak entity set

generalization



total
generalization

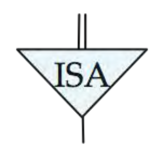


Figure 7.26

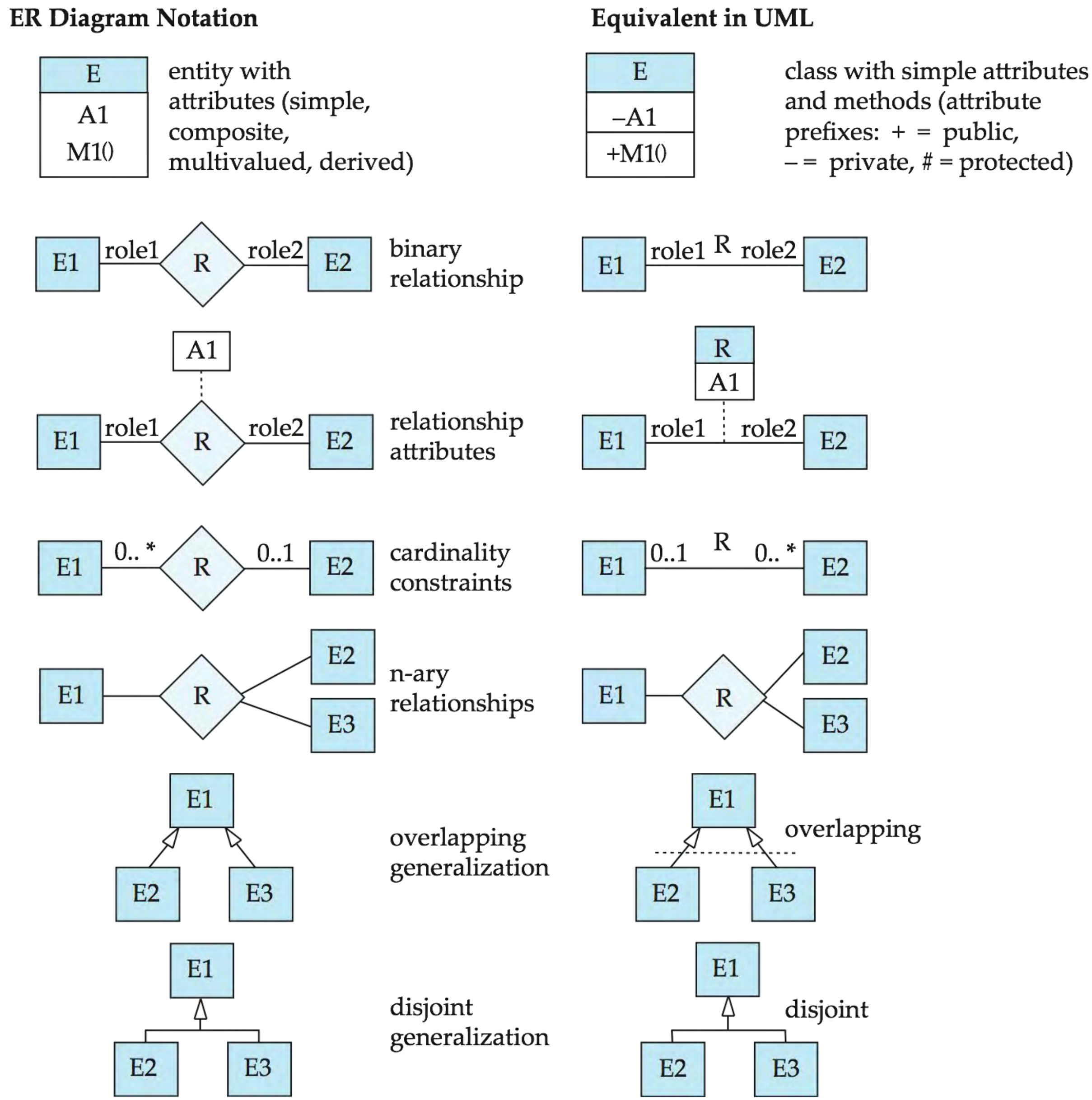

ER Diagram Notation
Equivalent in UML
E
A1
M1()
entity with attributes (simple, composite, multivalued, derived)
E
–A1
+M1()
class with simple attributes and methods (attribute prefixes: + = public, – = private, # = protected)
E1
role1
R
role2
E2
binary relationship
E1
role1
R
role2
E2
A1
E1
role1
R
role2
E2
relationship attributes
R
A1
E1
role1
role2
E2
E1
0.. *
R
0..1
E2
cardinality constraints
E1
0..1
R
0.. *
E2
E1
R
E2
E3
n-ary relationships
E1
R
E2
E3
E1
E2
E3
overlapping generalization
E1
E2
E3
overlapping
E1
E2
E3
disjoint generalization
E1
E2
E3
disjoint

Figure 7.27

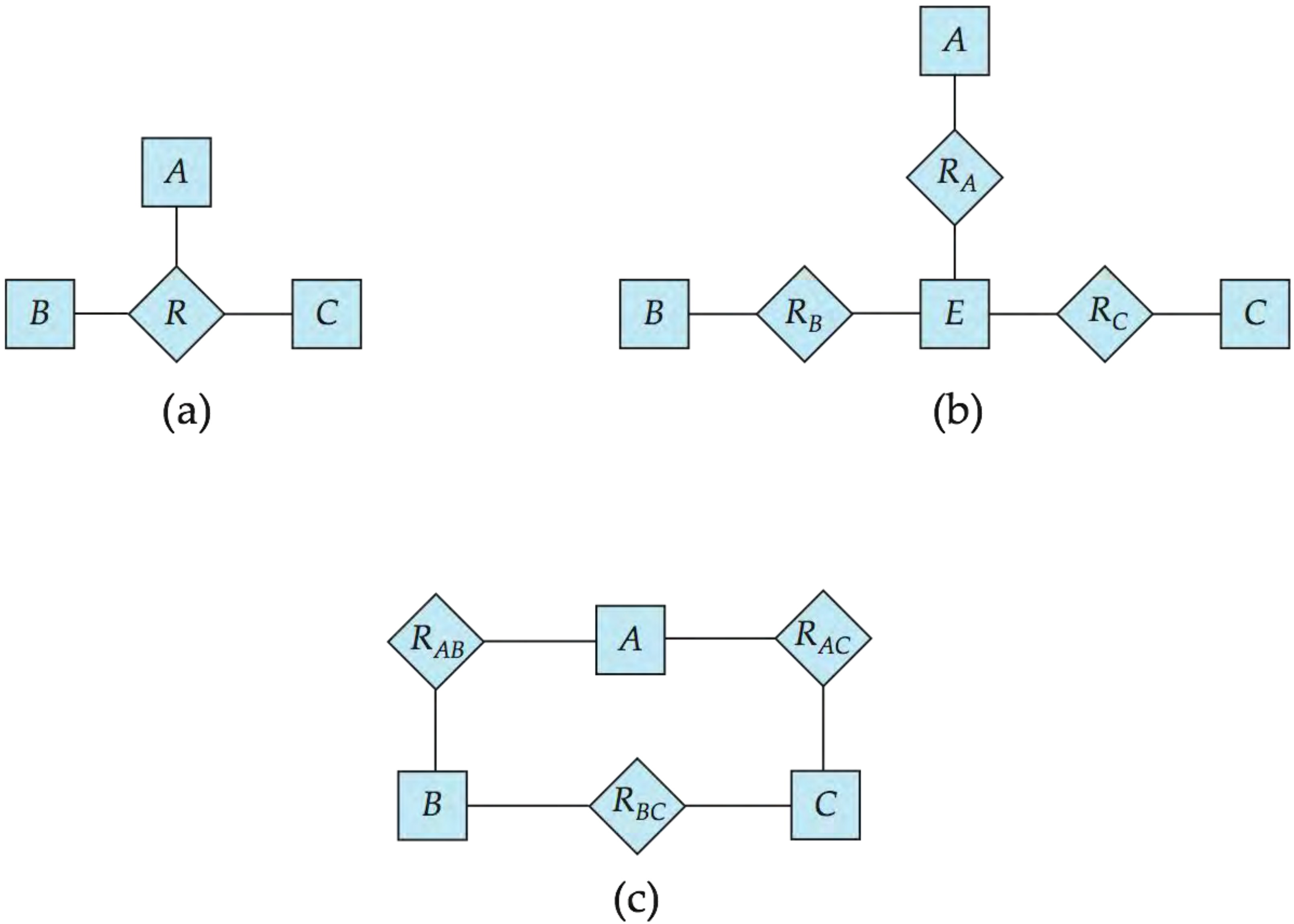


Figure 7.28

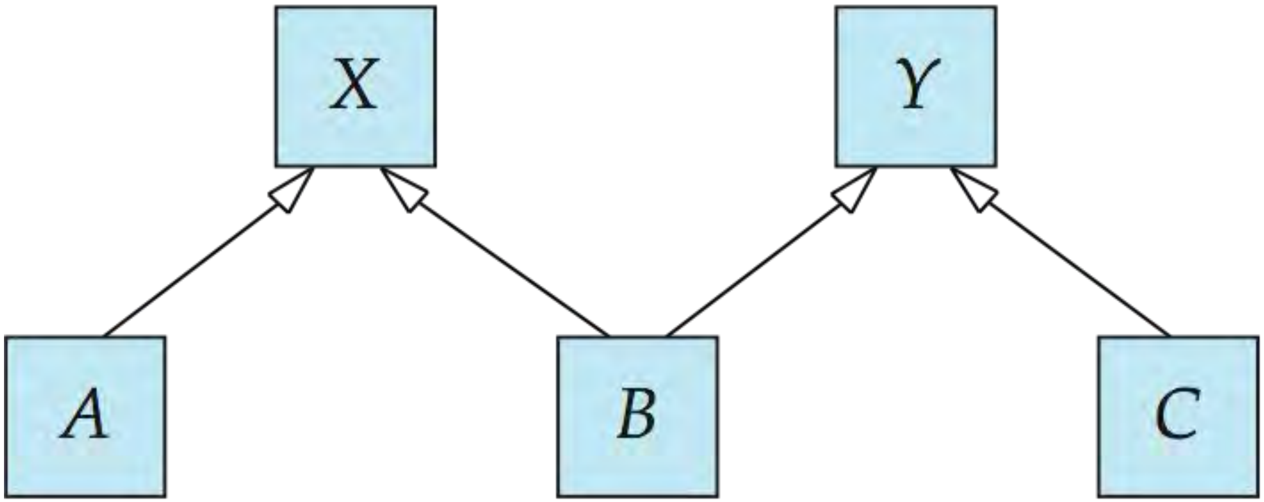
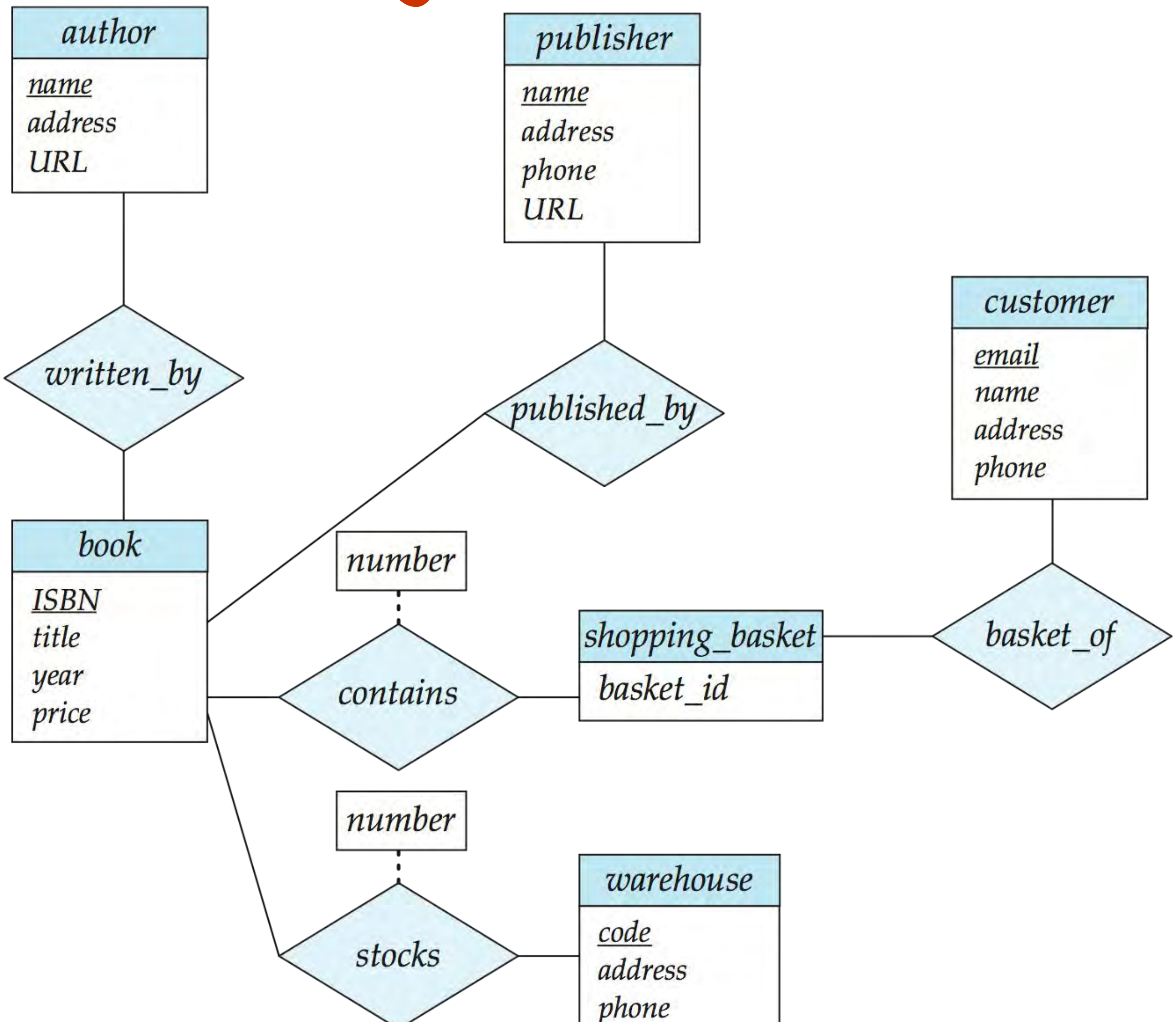


Figure 7.29


author
name
address
URL
publisher
name
address
phone
URL
written_by
published_by
customer
email
name
address
phone
book
ISBN
title
year
price
number
contains
shopping_basket
basket_id
basket_of
number
stocks
warehouse
code
address
phone

Relational Database Design

BAD Design

- Design “anomalies”
  - eg. student-course
  - Redundancy
  - Update anomaly
  - Deletion anomaly

GOOD Design

- Decompose into “GOOD”
  - Functional Dependency (FD)
  - Multivalued Dependency (MVD)
- Normal Form
  - FD,1NF-2NF-3NF-BCNF
  - MVD,4NF

FD

Let R be a relation schema

$$\alpha \subseteq R \text{ and } \beta \subseteq R$$

The functional dependency $\alpha \to \beta$ holds on R if and only if for any legal relations r(R), whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β . That is,

$$t_1[\alpha] = t_2 [\alpha]$$
$$t_1[\beta] = t_2 [\beta]$$

FD, Example

- Consider *r* (*A*, *B*) with the following instance of *r*.

1	4
1	
5	
3	
7	

- On this instance, *A*   *B* does NOT hold, but *B*   *A* does hold
- FD实质是约束，描述属性间的约束

FD, Example

- **Stu(sno,sname,age,gende,ID,dept,cno, cname,grade)**

Key

- K R
- K is a superkey for relation schema R if and only if K R
- K is a candidate key for R if and only if
  - K R, and
  - for no K, R

Trivial(平凡) FD

- A functional dependency is **trivial** if it is satisfied by all instances of a relation
- Example:

  ID, name   ID

  name   name

- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Armstrong's Axioms

- if $\beta \subseteq$, then $\alpha \rightarrow \beta$ (自反律)
- if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$ (增补律)
- if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (传递律)

- Additional rules:
- If and , then (合并律)
- If , then and (分解律)
- If and , then (伪传递律)

Closure(闭包)

- 根据相关定理，找出已知的FD所蕴涵的函数依赖
  - 函数依赖集的闭包
  - 属性的闭包

Normal Form

- 1NF
- 2NF
- 3NF
- BCNF

NF的分解

- 分解的要求
  - Lossless-join decomposition

    $R_1 \cap R_2 \rightarrow R_1$

    $R_1 \cap R_2 \rightarrow R_2$
  - Dependency preserving

    $(F_1 \cup F_2 \cup \cdots \cup F_n)^+ = F^+$
  - NF要求

Canonical Cover(正则覆盖)

- a canonical cover of F is a “minimal” set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies
- Canonical Cover的特征
  - 左边是唯一的，右边多属性合并
  - 两边均不含无关属性(extraneous)
- 用于校验依赖保持

无关属性的判断

- For
- To test if attribute *A* is extraneous in

  1. compute ({ } – *A*)$^+$ using the dependencies in *F*
  2. check that ({ } – *A*)$^+$ contains ; if it does, *A* is extraneous in
- To test if attribute *A* is extraneous in

  1. compute $^+$ using only the dependencies in
     F' = (F – { }) { (– A)},
  2. check that $^+$ contains *A*; if it does, *A* is extraneous in

Canonical Cover(正则覆盖)

- R = (A, B, C)
  F = {A   BC, B   C, A   B, AB   C}

- F = {A   BC, B   C, AB   C}

- F = {A   BC, B   C}

- F = {A   B, B   C}

范式分解

1）For R，求Key，找出违反范式要求的FD，$A \quad B$;

2）求A^+

3）$R1 = \{A^+\}$ √

$R2 = \{A \cup (R-A^+)\}$

4）对R2重复1-3，直至均满足范式要求

- $R1 \cap R2 \quad R1$,无损连接
- 验证 $(F_1 \quad F_2 \quad \ldots \quad F_n)^+ = F^+$
- 亦可验证Fc

范式分解

R = (A, B, C)
F = {A→B, B→C)

Can be decomposed in two different ways

- R1 = (A, B), R2 = (B, C)
  - Lossless-join decomposition:
    - $R1 \cap R2 = \{B\}$ and $B \rightarrow BC$
  - Dependency preserving
- R1 = (A, B), R2 = (A, C)
  - Lossless-join decomposition:
    - $R1 \cap R2 = \{A\}$ and $A \rightarrow AB$
  - Not dependency preserving

(cannot check B→C without computing $R1 \bowtie R2$)

范式分解

- R = (A, B, C)

F = {A→B, B→C, B→A)

- R1 = (A, B), R2 = (A, C)

  Lossless-join decomposition:

  $R1 \cap R2 = \{A\}$ and $A \rightarrow AB$

  dependency preserving?

  $B \rightarrow C$?

$$(F1 \quad F2 \quad \ldots \quad Fn)^{+} = F^{+}$$

范式分解

R(T,S,C),每位教师T只上一门课C，每门课可能有多个教学班，S为学生。

T　C,SC　T

Key

　SC,TS

分解

　R1(T,C),R2(T,S)

　无损连接，依赖保持？

例子

- 图书馆
  - ISBM，分类号：TP311.13

MVD

MVD

- inst_info(ID, child_name, phone_number)

ID	Child_name	Phone_number
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

- BCNF, why?
- 仍有冗余，仍有异常
- MVD
  - ID→→child_name
  - ID→→phone_number

MVD

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

MVD

- 平凡的MVD，$\alpha \rightarrow\rightarrow \beta$

= R

- If $\alpha \rightarrow \beta$ then $\alpha \rightarrow\rightarrow \beta$

- If $\rightarrow$ then $\rightarrow\rightarrow$ R - -

4NF

- 对所有的非平凡多值依赖，其决定因子均来自超码。

ID	Child_name	Phone_number
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

- ID→→child_name
- ID→→phone_number
- R1(ID,child_name),R2(ID,phone_number)

ER Model and Normalization

In a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity

- Example: an *employee* entity with attributes *department_name* and *building*,
  and a functional dependency *department_name* *building*
- Good design would have made department an entity

Denormalization for Performance

- May want to use non-normalized schema for performance
- For example, displaying prereqs along with course_id, and title requires join of course with prereq
- Use denormalized relation containing attributes of course as well as prereq with all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code

Transaction
（事务）

Transaction Concept

- A **transaction** is a unit of program execution that accesses and possibly updates various data items
- SQL DML/C++ Java
- 操作的集合、逻辑工作单元
- 可能由多条语句构成

Example of transaction

E.g. transaction to transfer \$50 from account *A* to account B:

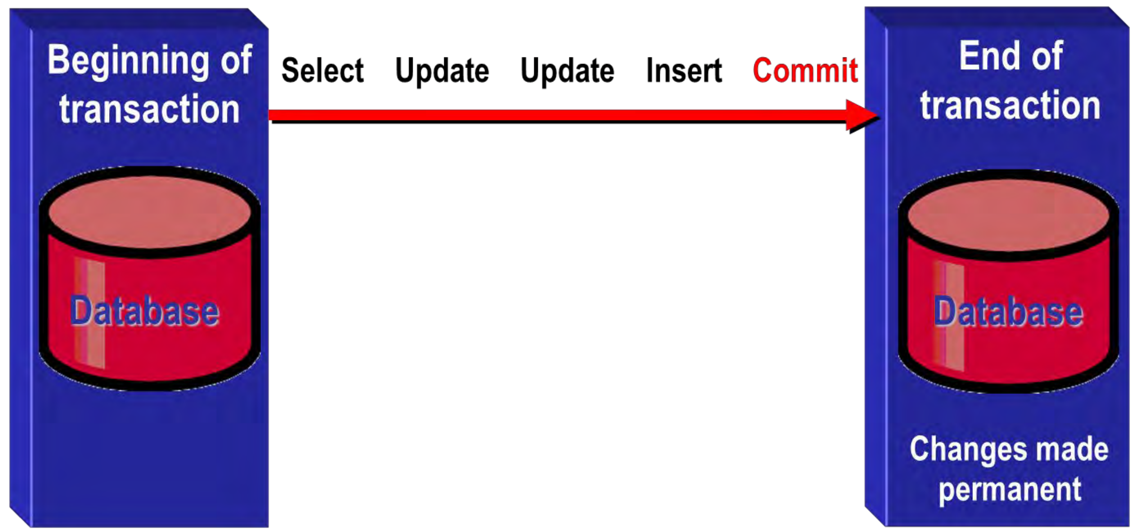
1. read(*A*)
2. *A* := *A* – 50
3. write(*A*)
4. read(B)
5. B := B + 50
6. write(B)

ACID Properties

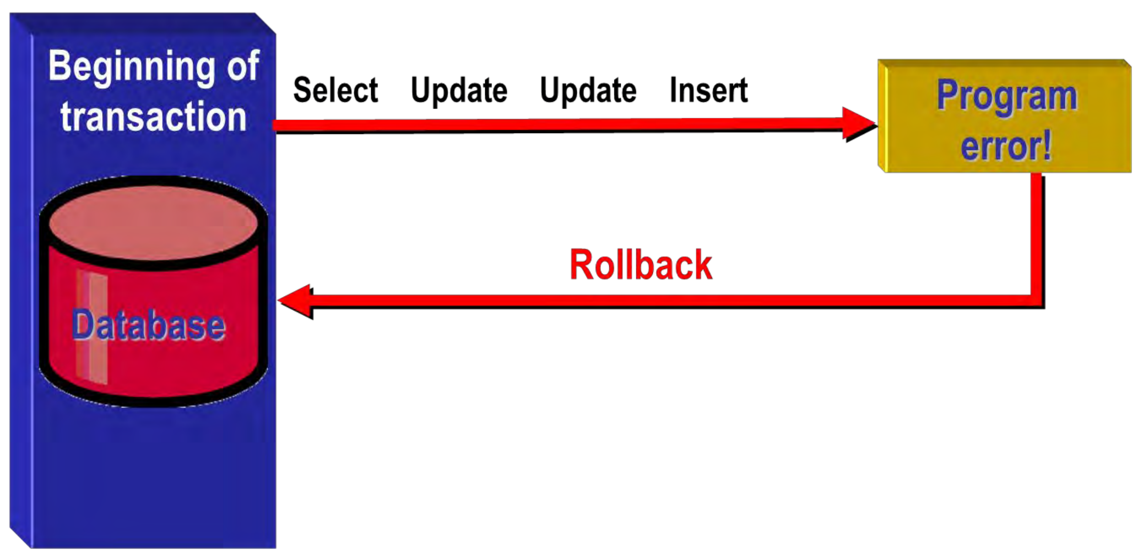
- Atomicity
- Consistency
- Isolation
- Durability

Transaction

- Commit

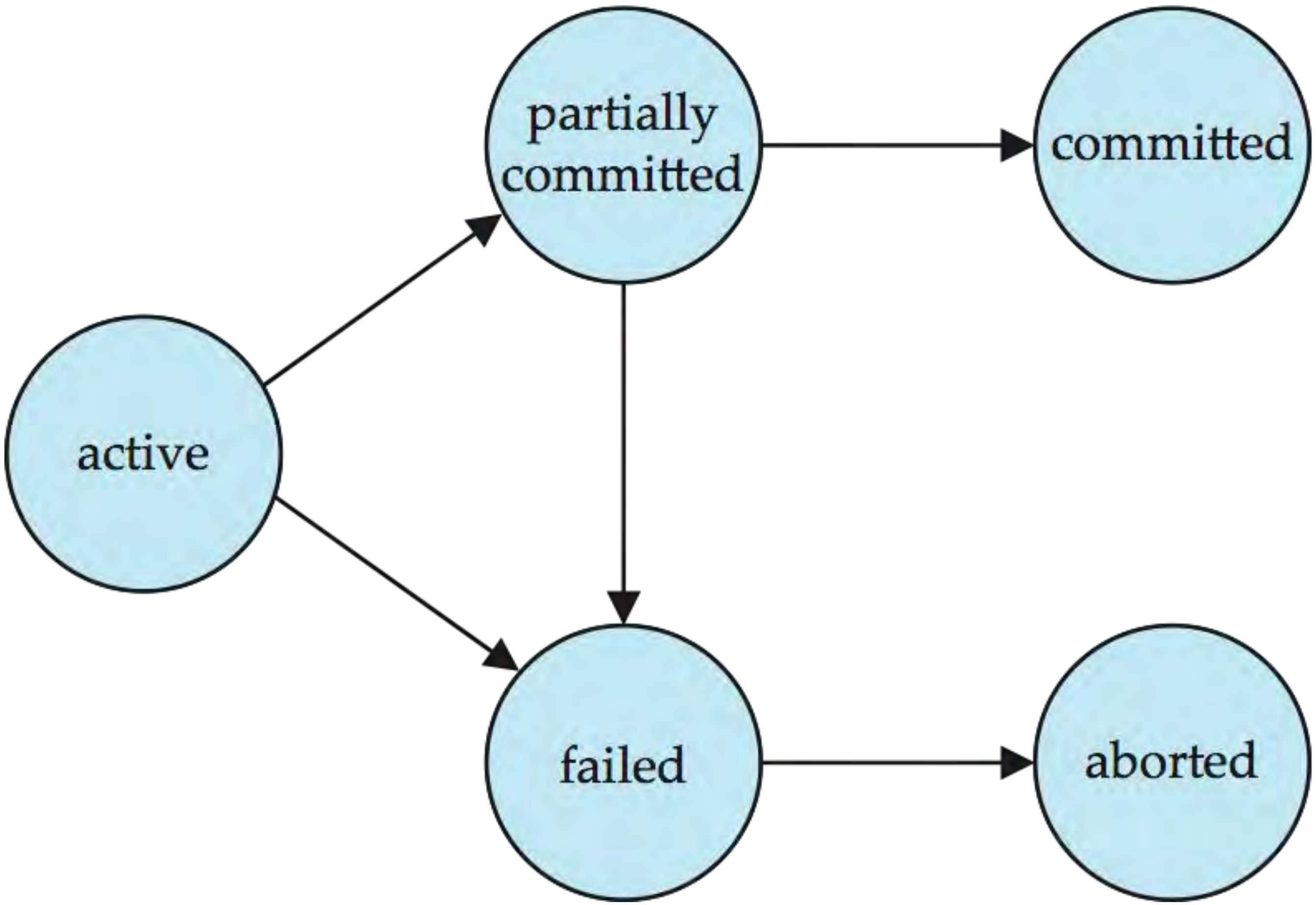


- Rollback



Transaction process

Transaction State



Tracsaction State

- **Active** – the initial state; the transaction stays in this state while it is executing
- **Partially committed** – after the final statement has been executed.
- **Failed** -- after the discovery that normal execution can no longer proceed.
- **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - restart the transaction

    can be done only if no internal logical error
  - kill the transaction
- **Committed** – after successful completion.

Schedules

- 指令执行的时间顺序
- 必须包含被调度的事务的所有指令，并保持指令在各事务中的原顺序
- 调度，实现事务的并发运行
- 可串行调度，多事务并发运行的结果，与串行运行一致

Schedule 1

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

A serial schedule in which T_1 is followed by T_2

Schedule 2

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A$ - $temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A – 50$ write (A) read (B) $B := B + 50$ write (B) commit	

A serial schedule where T_2 is followed by T_1

Schedule 3

T_1	T_2
read (A) $A := A - 50$ write (A)	
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A)
read (B) $B := B + 50$ write (B) commit	
	read (B) $B := B + temp$ write (B) commit

In Schedules 1, 2 and 3, the sum A + B is preserved.

Schedule 4

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

This concurrent schedule does not preserve the value of $(A + B)$.

Lock

锁的机制

- 共享锁（Shared）、排它锁（exclusive）

	S	X
S	true	false
X	false	false

Lock & Transaction

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions.
- Any number of transactions can hold shared locks on an item,
  - but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

Lock

T_3	T_4
lock-x (B)	
read (B)	
$B := B - 50$	
write (B)	
	lock-s (A)
	read (A)
	lock-s (B)
lock-x (A)	

- Neither T_3 nor T_4 can make progress — executing **lock-S**(B) causes T_4 to wait for T_3 to release its lock on B, while executing **lock-X**(A) causes T_3 to wait for T_4 to release its lock on A.
- Such a situation is called a **deadlock**.
  - To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.

Isolation level

Isolation level

- 事务的隔离层次（隔离级别）
  - 并发性能vs完整一致之间的一种弱化、折中
  - Read uncommitted Isolation_Level = 3
  - Read committed Isolation_Level = 2
  - Repeatable read Isolation_Level = 1
  - Serializable Isolation_Level = 0

0 - Read uncommitted

- Read permitted on row with or without write lock
- No read locks are applied
- No guarantee that concurrent transaction will not modify row or roll back changes to row
- Corresponds to table hints NOLOCK and READUNCOMMITTED
- Allow dirty reads, non-repeatable reads, and phantom rows

1 - Read committed

- Read only permitted on row with no write lock
- Read lock acquired and held for read on current row only, but released when cursor moves off the row
- No guarantee that data will not change during transaction
- Corresponds to table hint READCOMMITTED
- Prevent dirty reads
- Allow non-repeatable reads and phantom rows

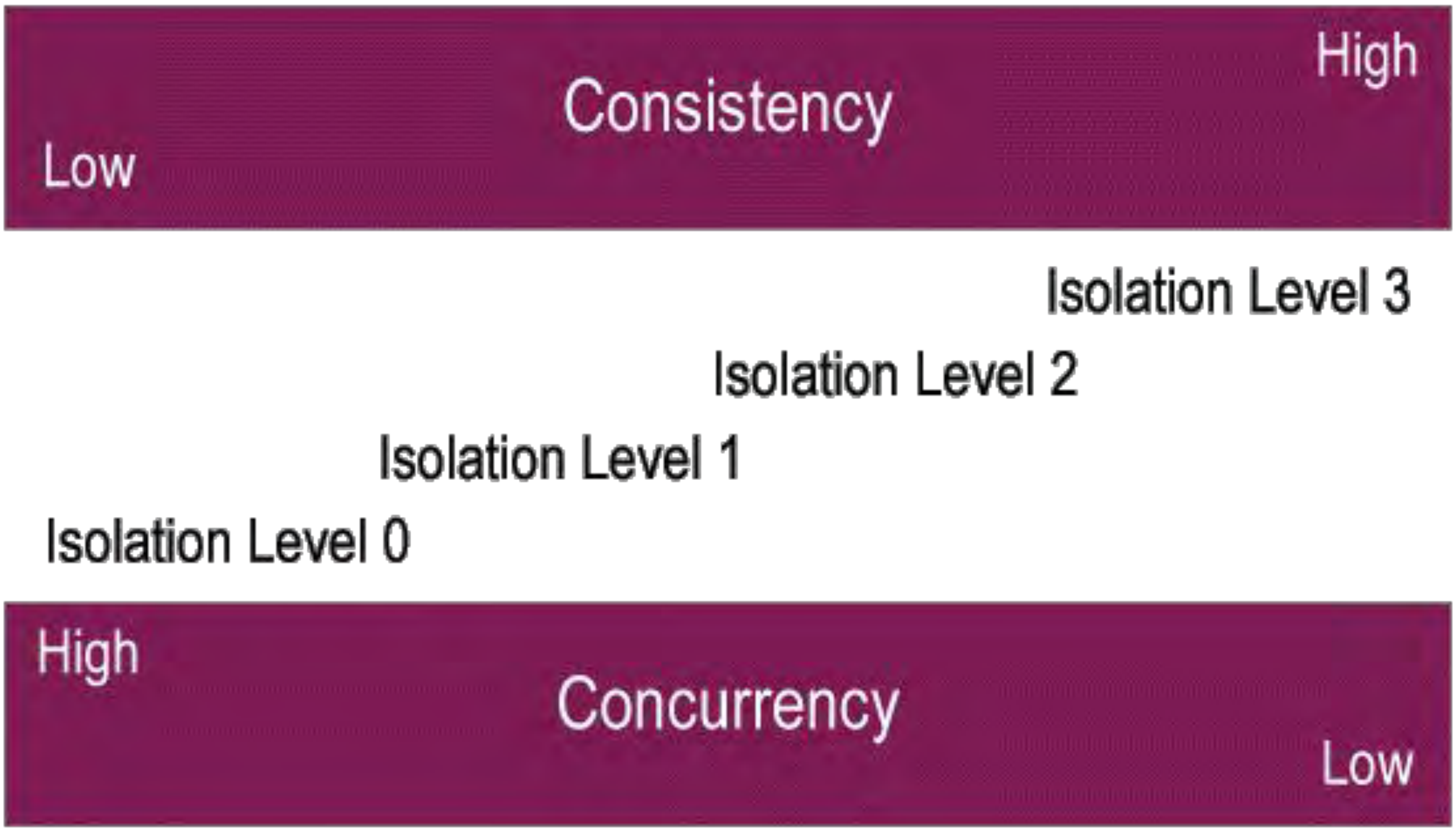
2 - Repeatable

- Read only permitted on row with no write lock
- Read lock acquired as each row in the result set is read, and held until transaction ends
- Corresponds to table hint REPEATABLEREAD
- Prevent dirty reads and non-repeatable reads
- Allow phantom rows

3 - serializable

- Read only permitted on rows in result without write lock
- Read locks acquired when cursor is opened and held until transaction ends
- Corresponds to table hints HOLDLOCK and SERIALIZABLE
- Prevent dirty reads, non-repeatable reads, and phantom rows

Isolation Level



Recovery System

Failure Classification

- Transaction failure:
  - Logical errors: transaction cannot complete due to some internal error condition
  - System errors: the database system must terminate an active transaction due to an error condition (e.g., deadlock)
- System crash: a power failure or other hardware or software failure causes the system to crash.

- Disk failure: a head crash or similar disk failure destroys all or part of disk storage

恢复策略

- 即使发生故障，仍需保持数据库的基本特性ACID
- 在正常事务处理时采取措施，保障有足够的信息可用于故障恢复；
- 故障发生后，使数据库恢复到某一保证ACID的状态。

Classification of Physical Storage Media

- Speed with which data can be accessed
- Cost per unit of data
- Reliability
  - data loss on power failure or system crash
  - physical failure of the storage device
- Can differentiate storage into:
  - volatile storage:
    - loses contents when power is switched off
  - non-volatile storage:
    - Contents persist even when power is switched off.
    - Includes secondary and tertiary storage, as well as batter-backed up main-memory.

Physical Storage Media

- **Cache – fastest and most costly form of storage; volatile; managed by the computer system hardware.**

Physical Storage Media

- **Main memory:**
  - fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} seconds)
  - generally too small (or too expensive) to store the entire database
    - capacities of up to a few Gigabytes widely used currently
    - Capacities have gone up and per-byte costs have decreased steadily and rapidly (roughly factor of 2 every 2 to 3 years)
  - Volatile — contents of main memory are usually lost if a power failure or system crash occurs.

Physical Storage Media (Cont.)

- Flash memory
  - Data survives power failure
  - Data can be written at a location only once, but location can be erased and written to again
  - Reads are roughly as fast as main memory
  - But writes are slow (few microseconds), erase is slower
  - Widely used in embedded devices such as digital cameras, phones, and USB keys
  - SSD (Solid-State Disk)

Physical Storage Media (Cont.)

- Magnetic-disk
  - Data is stored on spinning disk, and read/written magnetically
  - Primary medium for the long-term storage of data; typically stores entire database.
  - Data must be moved from disk to main memory for access, and written back for storage
  - direct-access possible to read data on disk in any order, unlike magnetic tape

Physical Storage Media (Cont.)

- **Magnetic-disk**
  - Capacities range up to roughly 1.5 TB as of 2009
    - Much larger capacity and cost/byte than main memory/flash memory
    - Growing constantly and rapidly with technology improvements (factor of 2 to 3 every 2 years)
  - Survives power failures and system crashes
    - disk failure can destroy data, but is rare

Physical Storage Media (Cont.)

- Optical storage
  - non-volatile, data is read optically from a spinning disk using a laser
  - CD-ROM (640 MB) and DVD (4.7 to 17 GB) most popular forms
  - Blu-ray disks: 27 GB to 54 GB

Physical Storage Media (Cont.)

- Optical storage
  - Write-one, read-many (WORM) optical disks used for archival storage (CD-R, DVD-R, DVD+R)
  - Multiple write versions also available (CD-RW, DVD-RW, DVD+RW, and DVD-RAM)
  - Reads and writes are slower than with magnetic disk
  - Juke-box systems, with large numbers of removable disks, a few drives, and a mechanism for automatic loading/unloading of disks available for storing large volumes of data

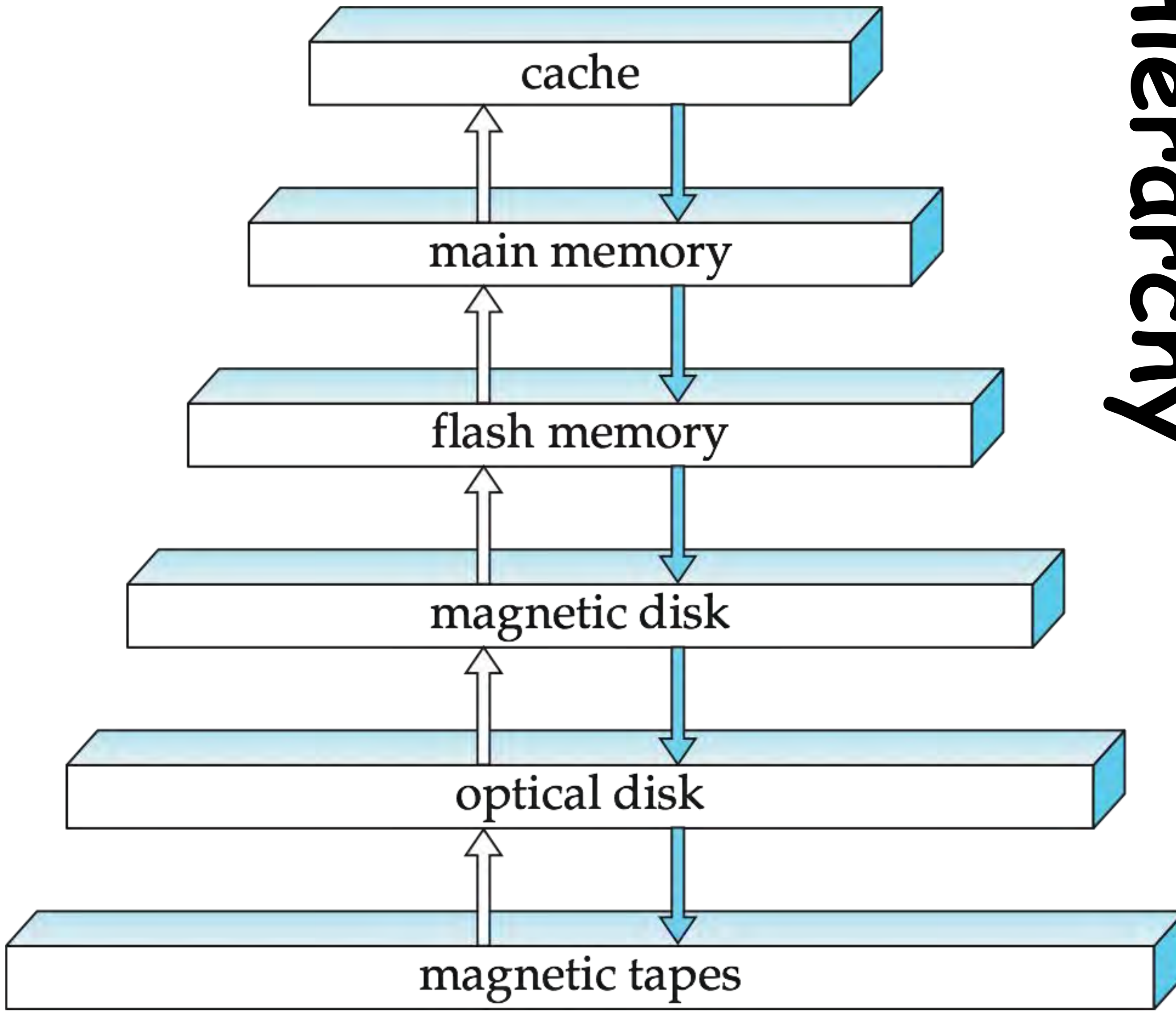
Physical Storage Media (Cont.)

- Tape storage
  - non-volatile, used primarily for backup (to recover from disk failure), and for archival data
  - sequential-access – much slower than disk
  - very high capacity (40 to 300 GB tapes available)

Physical Storage Media (Cont.)

- Tape storage
  - tape can be removed from drive storage costs much cheaper than disk, but drives are expensive
  - Tape jukeboxes available for storing massive amounts of data
    - hundreds of terabytes (1 terabyte = 10^9 bytes) to even multiple petabytes (1 petabyte = 10^{12} bytes)

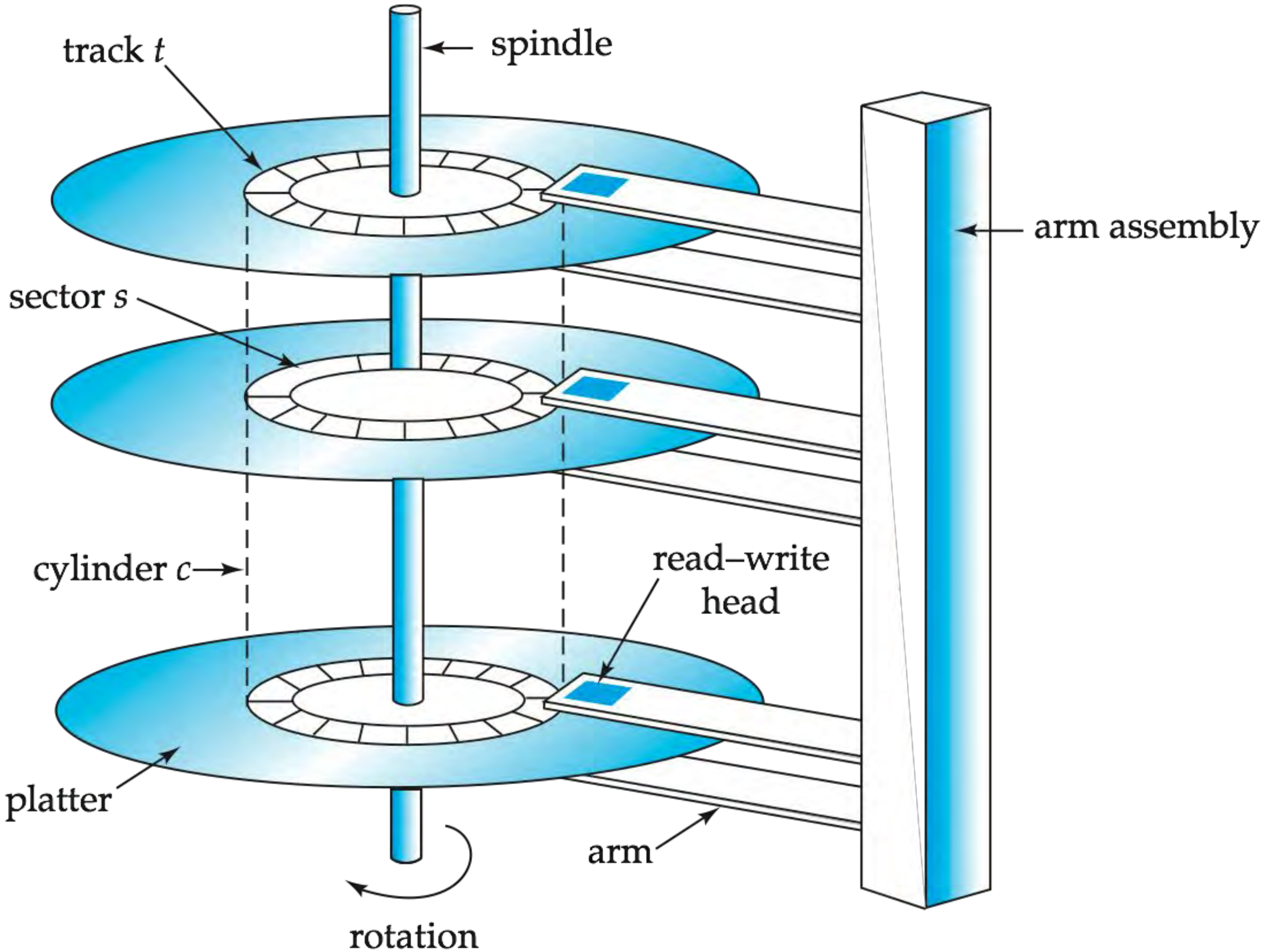
Storage Hierarchy



Storage Hierarchy (Cont.)

- primary storage: Fastest media but volatile (cache, main memory).
- secondary storage: next level in hierarchy, non-volatile, moderately fast access time
  - also called on-line storage
  - E.g. flash memory, magnetic disks
- tertiary storage: lowest level in hierarchy, non-volatile, slow access time
  - also called off-line storage
  - E.g. magnetic tape, optical storage

Magnetic Hard Disk Mechanism



NOTE: Diagram is schematic, and simplifies the structure of actual disk drives

Magnetic Disks

- Read-write head
  - Positioned very close to the platter surface (almost touching it)
  - Reads or writes magnetically encoded information.
- Surface of platter divided into circular tracks
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into sectors.
  - A sector is the smallest unit of data that can be read or written.
  - Sector size typically 512 bytes
  - Typical sectors per track: 500 to 1000 (on inner tracks) to 1000 to 2000 (on outer tracks)
- To read/write a sector
  - disk arm swings to position head on right track
  - platter spins continually; data is read/written as sector passes under head
- Head-disk assemblies
  - multiple disk platters on a single spindle (1 to 5 usually)
  - one head per platter, mounted on a common arm.
- Cylinder i consists of ith track of all the platters

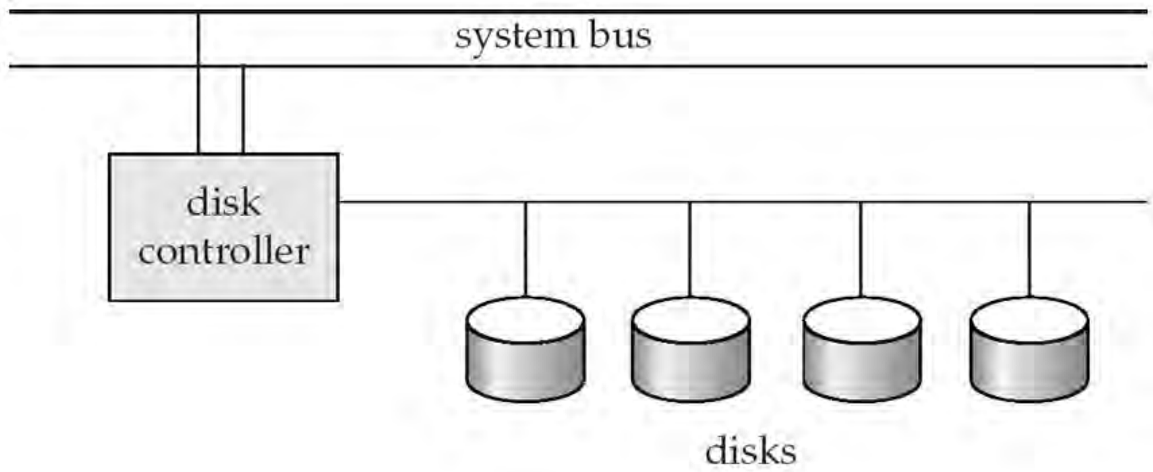
Magnetic Disks (Cont.)

- Earlier generation disks were susceptible to head-crashes
  - Surface of earlier generation disks had metal-oxide coatings which would disintegrate on head crash and damage all data on disk
  - Current generation disks are less susceptible to such disastrous failures, although individual sectors may get corrupted

Magnetic Disks (Cont.)

- Disk controller – interfaces between the computer system and the disk drive hardware.
  - accepts high-level commands to read or write a sector
  - initiates actions such as moving the disk arm to the right track and actually reading or writing the data
  - Computes and attaches checksums to each sector to verify that data is read back correctly
    - If data is corrupted, with very high probability stored checksum won't match recomputed checksum
  - Ensures successful writing by reading back sector after writing it
  - Performs remapping of bad sectors

Disk Subsystem



- **Multiple disks connected to a computer system through a controller**
  - **Controllers functionality (checksum, bad sector remapping) often carried out by individual disks; reduces load on controller**
- **Disk interface standards families**
  - **ATA (AT adaptor) range of standards**
  - **SATA (Serial ATA)**
  - **SCSI (Small Computer System Interconnect) range of standards**
  - **SAS (Serial Attached SCSI)**
  - **Several variants of each standard (different speeds and capabilities)**

Disk Subsystem

- Disks usually connected directly to computer system
- In Storage Area Networks (SAN), a large number of disks are connected by a high-speed network to a number of servers
- In Network Attached Storage (NAS) networked storage provides a file system interface using networked file system protocol, instead of providing a disk system interface

Performance Measures of Disks

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins. Consists of:
  - **Seek time** – time it takes to reposition the arm over the correct track.
    - Average seek time is 1/2 the worst case seek time.
      - Would be 1/3 if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
    - 4 to 10 milliseconds on typical disks
  - **Rotational latency** – time it takes for the sector to be accessed to appear under the head.
    - Average latency is 1/2 of the worst case latency.
    - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
  - 25 to 100 MB per second max rate, lower for inner tracks
  - Multiple disks may share a controller, so rate that controller can handle is also important
    - E.g. SATA: 150 MB/sec, SATA-II 3Gb (300 MB/sec)
    - Ultra 320 SCSI: 320 MB/s, SAS (3 to 6 Gb/sec)
    - Fiber Channel (FC2Gb or 4Gb): 256 to 512 MB/s

Performance Measures (Cont.)

- **Mean time to failure (MTTF) – the average time the disk is expected to run continuously without any failure.**
  - Typically 3 to 5 years
  - Probability of failure of new disks is quite low, corresponding to a "theoretical MTTF" of 500,000 to 1,200,000 hours for a new disk
    - E.g., an MTTF of 1,200,000 hours for a new disk means that *given* 1000 relatively new disks, on an average one will fail every 1200 hours
  - MTTF decreases as disk ages

Optimization of Disk-Block Access

- Block – a contiguous sequence of sectors from a single track
  - data is transferred between disk and main memory in blocks
  - sizes range from 512 bytes to several kilobytes
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
    - Typical block sizes today range from 4 to 16 kilobytes
- Disk-arm-scheduling algorithms order pending accesses to tracks so that disk arm movement is minimized

Optimization of Disk Block Access(Cont.)

- File organization – optimize block access time by organizing the blocks to correspond to how data will be accessed
  - E.g. Store related information on the same or nearby cylinders.
  - Files may get fragmented over time
    - E.g. if data is inserted to/deleted from the file
    - Or free blocks on disk are scattered, and newly created file has its blocks scattered over the disk
    - Sequential access to a fragmented file results in increased disk arm movement
  - Some systems have utilities to defragment the file system, in order to speed up file access

Optimization of Disk Block Access(Cont.)

- Nonvolatile write buffers speed up disk writes by writing blocks to a non-volatile RAM buffer immediately
  – Non-volatile RAM: battery backed up RAM or flash memory
    - Even if power fails, the data is safe and will be written to disk when power returns
  – Controller then writes to disk whenever the disk has no other requests or request has been pending for some time
  – Database operations that require data to be safely stored before continuing can continue without waiting for data to be written to disk
  – Writes can be reordered to minimize disk arm movement

Optimization of Disk Block Access(Cont.)

- Log disk – a disk devoted to writing a sequential log of block updates
  - Used exactly like nonvolatile RAM
    - Write to log disk is very fast since no seeks are required
    - No need for special hardware (NV-RAM)
- File systems typically reorder writes to disk to improve performance
  - Journaling file systems write data in safe order to NV-RAM or log disk
  - Reordering without journaling: risk of corruption of file system data

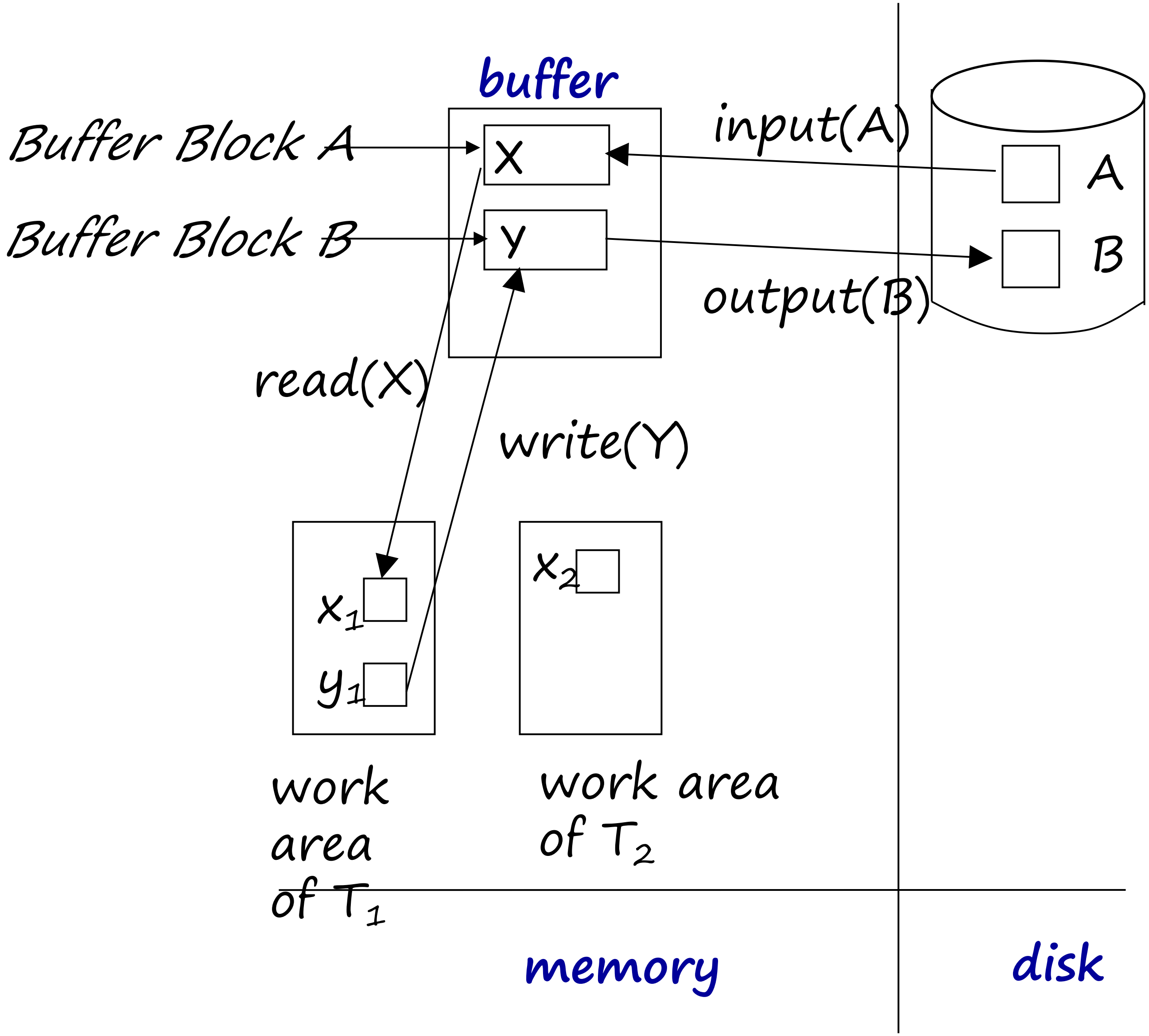
Flash Storage

- NOR flash vs NAND flash
- NAND flash
  - used widely for storage, since it is much cheaper than NOR flash
  - requires page-at-a-time read (page: 512 bytes to 4 KB)
  - transfer rate around 20 MB/sec
  - solid state disks: use multiple flash storage devices to provide higher transfer rate of 100 to 200 MB/sec

Flash Storage

- NOR flash vs NAND flash
- NAND flash
  - erase is very slow (1 to 2 millisecs)
    - erase block contains multiple pages
    - remapping of logical page addresses to physical page addresses avoids waiting for erase
      - translation table tracks mapping
        - » also stored in a label field of flash page
      - remapping carried out by flash translation layer
    - after 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
      - wear leveling

Example of Data Access



Data Access

- **Physical blocks** are those blocks residing on the disk.
- **Buffer blocks** are the blocks residing temporarily in main memory.
- Block movements between disk and main memory are initiated through the following two operations:
  - **input(B)** transfers the physical block B to main memory.
  - **output(B)** transfers the buffer block B to the disk, and replaces the appropriate physical block there.

Data Access

- Each transaction Ti has its private work-area in which local copies of all data items accessed and updated by it are kept.
  - Ti's local copy of a data item X is called xi.
- We assume, for simplicity, that each data item fits in, and is stored inside, a single block.

Data Access (Cont.)

- Transaction transfers data items between system buffer blocks and its private work-area using the following operations:
  - read(X) assigns the value of data item X to the local variable xi.
  - write(X) assigns the value of local variable xi to data item {X} in the buffer block.
  - Both these commands may necessitate the issue of an input(BX) instruction before the assignment, if the block BX in which X resides is not already in memory.

Data Access (Cont.)

- Transactions
  - Perform read(X) while accessing X for the first time;
  - All subsequent accesses are to the local copy.
  - After last access, transaction executes write(X).
- output(BX) need not immediately follow write(X). System can perform the output operation when it deems fit.

基于Log的备份

- To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself.
- log-based recovery

基于Log的恢复

- A log is kept on stable storage.
  - The log is a sequence of log records, and maintains a record of update activities on the database
- Two approaches using logs
  - Deferred database modification
  - Immediate database modification

Deferred Database Modification

$<T_0$ start$>$	$<T_0$ start$>$	$<T_0$ start$>$
$<T_0, A, 950>$	$<T_0, A, 950>$	$<T_0, A, 950>$
$<T_0, B, 2050>$	$<T_0, B, 2050>$	$<T_0, B, 2050>$
	$<T_0$ commit$>$	$<T_0$ commit$>$
	$<T_1$ start$>$	$<T_1$ start$>$
	$<T_1, C, 600>$	$<T_1, C, 600>$
		$<T_1$ commit$>$
(a)	(b)	(c)

Immediate database modification

$<T_0$ start$>$	$<T_0$ start$>$	$<T_0$ start$>$
$<T_0, A, 1000, 950>$	$<T_0, A, 1000, 950>$	$<T_0, A, 1000, 950>$
$<T_0, B, 2000, 2050>$	$<T_0, B, 2000, 2050>$	$<T_0, B, 2000, 2050>$
	$<T_0$ commit$>$	$<T_0$ commit$>$
	$<T_1$ start$>$	$<T_1$ start$>$
	$<T_1, C, 700, 600>$	$<T_1, C, 700, 600>$
		$<T_1$ commit$>$
(a)	(b)	(c)

Immediate database modification

Log	Write	Output
$\langle T_0$ start$\rangle$		
$\langle T_0$, A, 1000, 950$\rangle$		
T_o, B, 2000, 2050		
	A = 950	
	B = 2050	
$\langle T_0$ commit$\rangle$		
$\langle T_1$ start$\rangle$		
$\langle T_1$, C, 700, 600$\rangle$		
	C = 600	
		B_B, B_C
$\langle T_1$ commit$\rangle$		
		B_A

- Note: B_X denotes block containing X.

checkpoint

- Problems in recovery procedure
  1. searching the entire log is time-consuming
  2. we might unnecessarily redo transactions which have already
  3. output their updates to the database.
- Streamline recovery procedure by periodically performing checkpointing
  1. Output all log records currently residing in main memory onto stable storage.
  2. Output all modified buffer blocks to the disk.
  3. Write a log record < checkpoint> onto stable storage

日常备份策略

- Dump Database
- Dump Log
- Dump Log

- Recovery Database
- Recovery Log
- Recovery Log

Query Processing
&
Performance Tuning

INDEX

- Primary mechanism to get improved performance on a database
- Indexing mechanisms used to speed up access to desired data

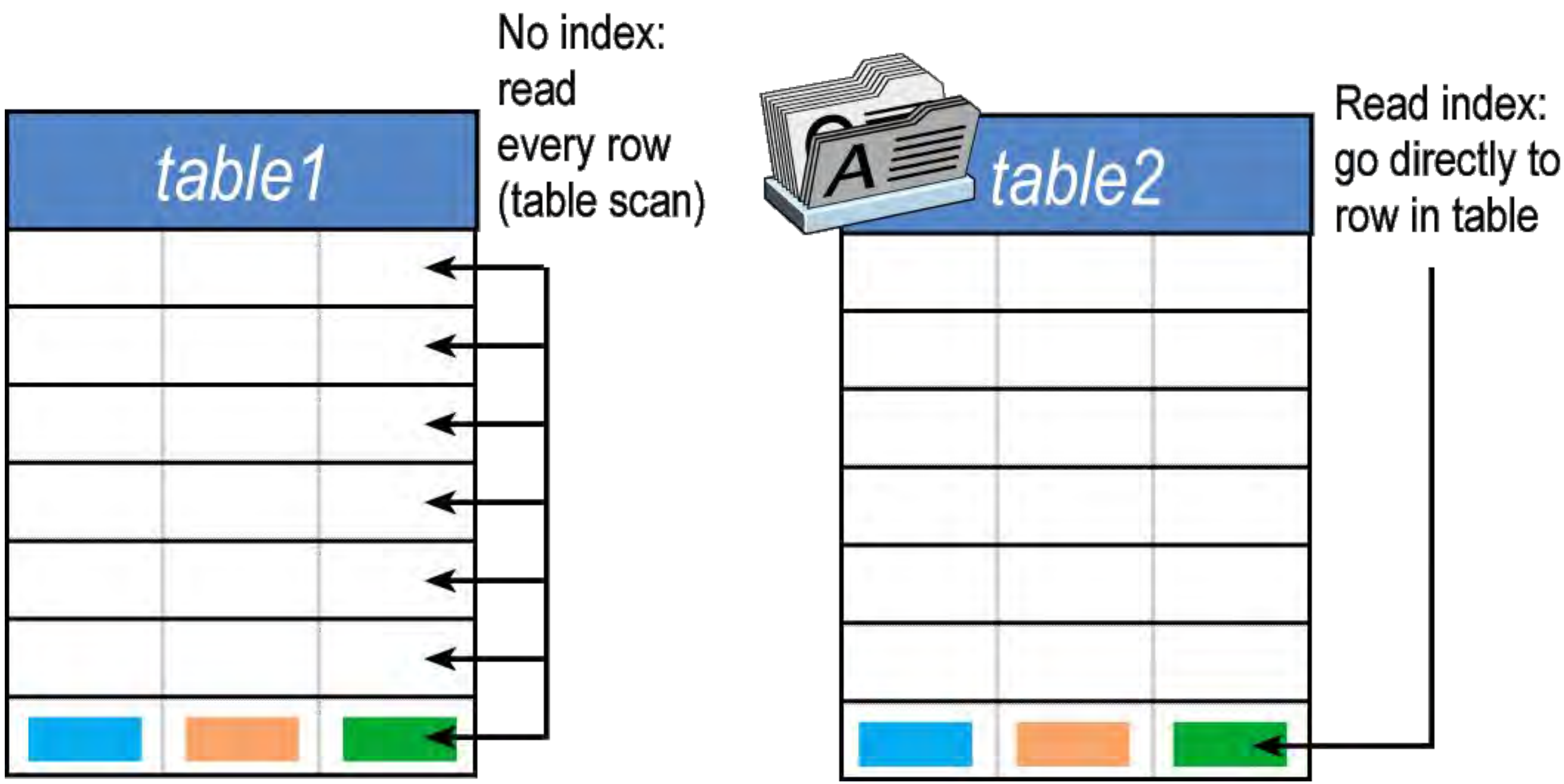
search-key	pointer

  - **Search Key** - attribute or set of attributes used to look up records in a file.

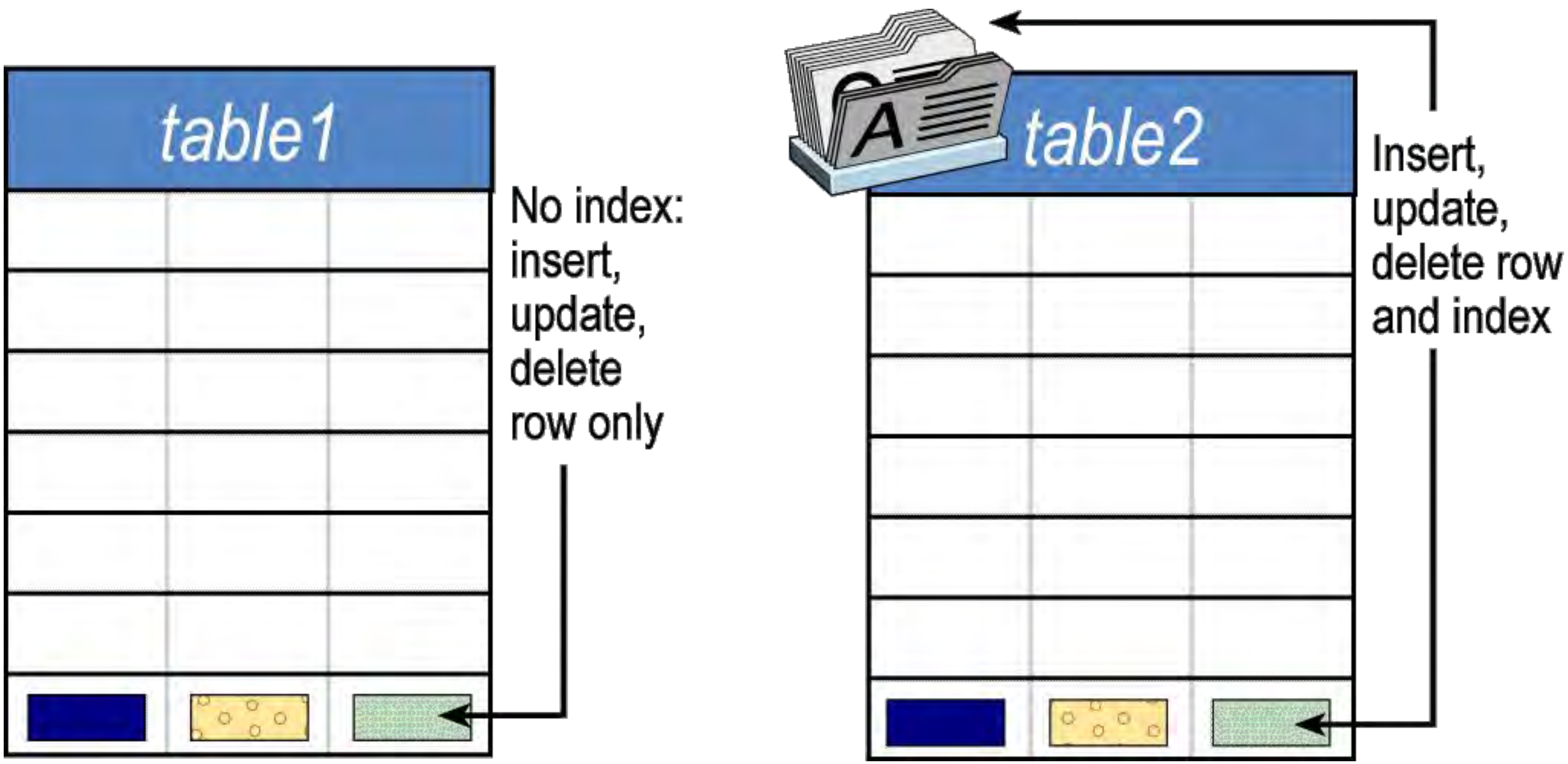
INDEX

- Two basic kinds of indices:
  - Ordered indices
    - search keys are stored in sorted order
    - Balanced trees
  - Hash indices
    - search keys are distributed uniformly across "buckets" using a "hash function"
    - Hash tables

INDEX - Query


No index:
read
every row
(table scan)
table1
table2
Read index:
go directly to
row in table

INDEX - Modification



Ordered Indices

- In an ordered index, index entries are stored sorted on the search key value.
- Primary index: in a sequentially ordered file, the index whose search key specifies the sequential order of the file.
  - Also called clustering index
  - The search key of a primary index is usually but not necessarily the primary key.

Ordered Indices

- **Secondary index**: an index whose search key specifies an order different from the sequential order of the file. Also called **non-clustering index**.
- **Index-sequential file**: ordered sequential file with a primary index.

Dense Index Files

- **Dense index** — Index record appears for every search-key value in the file.
- E.g. index on *ID* attribute of *instructor* relation

10101	
12121	
15151	
22222	
32343	
33456	
45565	
58583	
76543	
76766	
83821	
98345	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Dense Index Files (Cont.)

- **Dense index on *dept_name*, with *instructor* file sorted on *dept_name***

Biology	
Comp. Sci.	
Elec. Eng.	
Finance	
History	
Music	
Physics	

76766	Crick	Biology	72000	
10101	Srinivasan	Comp. Sci.	65000	
45565	Katz	Comp. Sci.	75000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	
12121	Wu	Finance	90000	
76543	Singh	Finance	80000	
32343	El Said	History	60000	
58583	Califieri	History	62000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
33465	Gold	Physics	87000	

Sparse Index Files

- **Sparse Index**: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key
- To locate a record with search-key value *K* we:
  - Find index record with largest search-key value <= *K*
  - Search file sequentially starting at the record to which the index record points

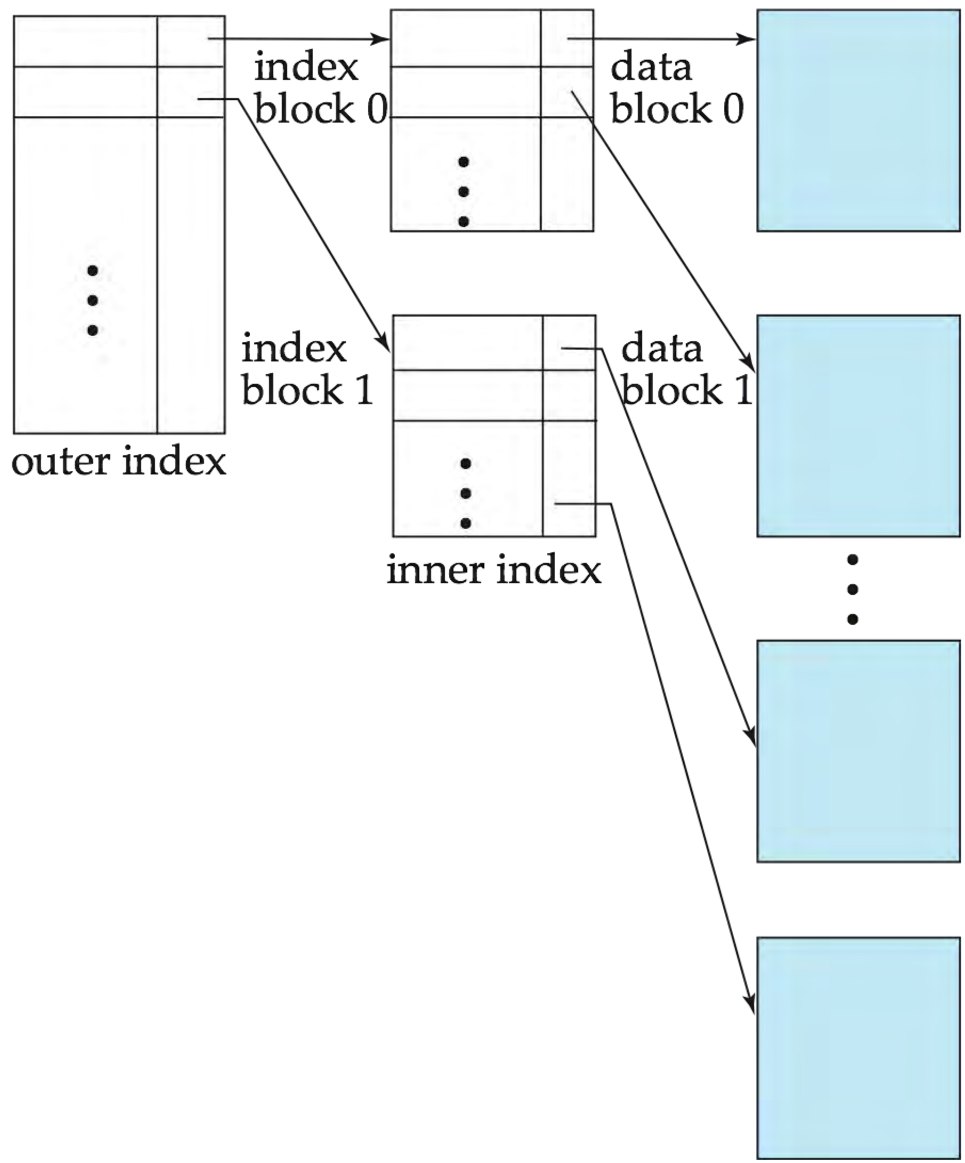
10101	
32343	
76766	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

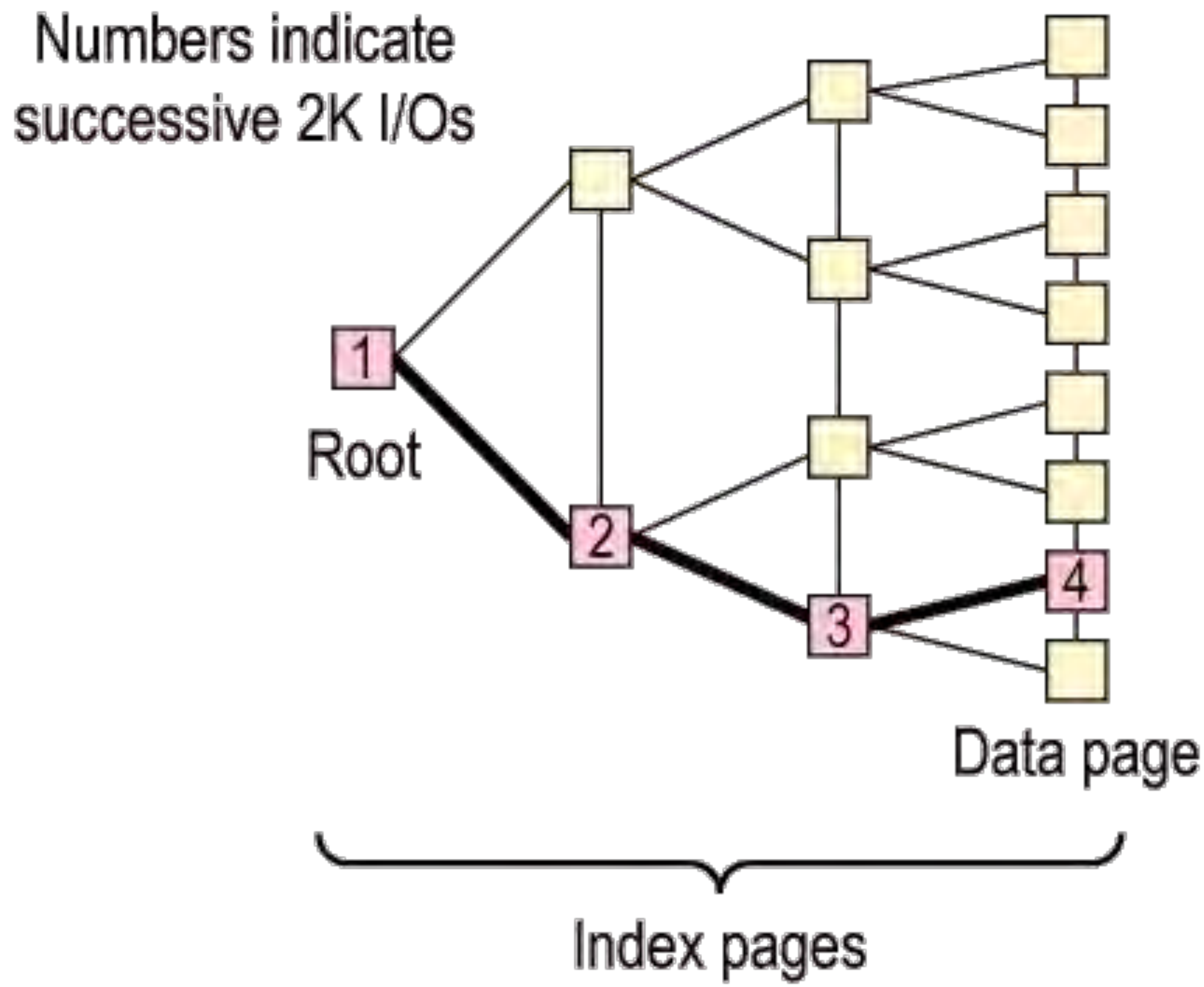
Multilevel Index

- If primary index does not fit in memory, access becomes expensive.
- Solution: treat primary index kept on disk as a sequential file and construct a sparse index on it.
  - outer index – a sparse index of primary index
  - inner index – the primary index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- Indices at all levels must be updated on insertion or deletion from the file.

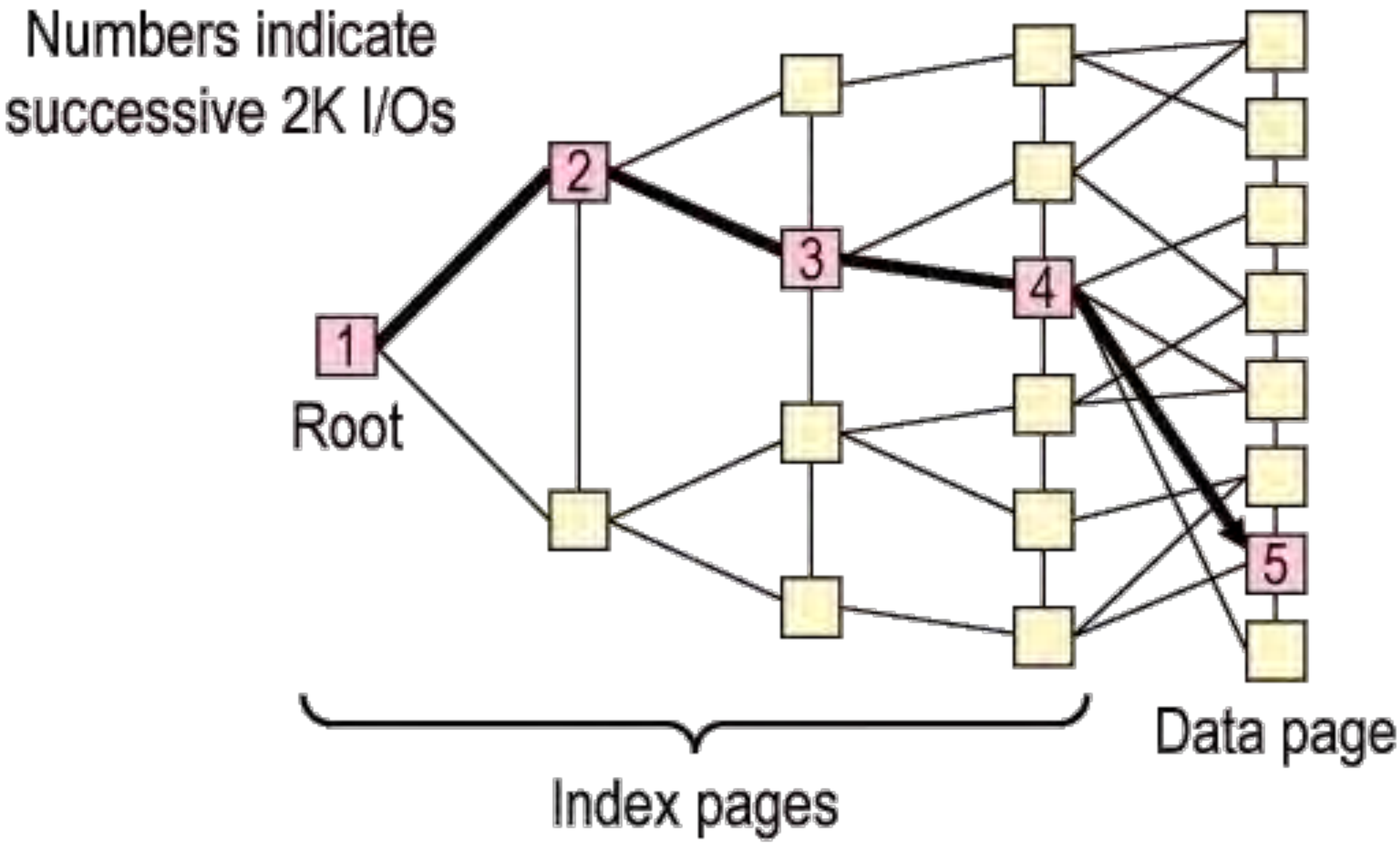
Multilevel Index (Cont.)



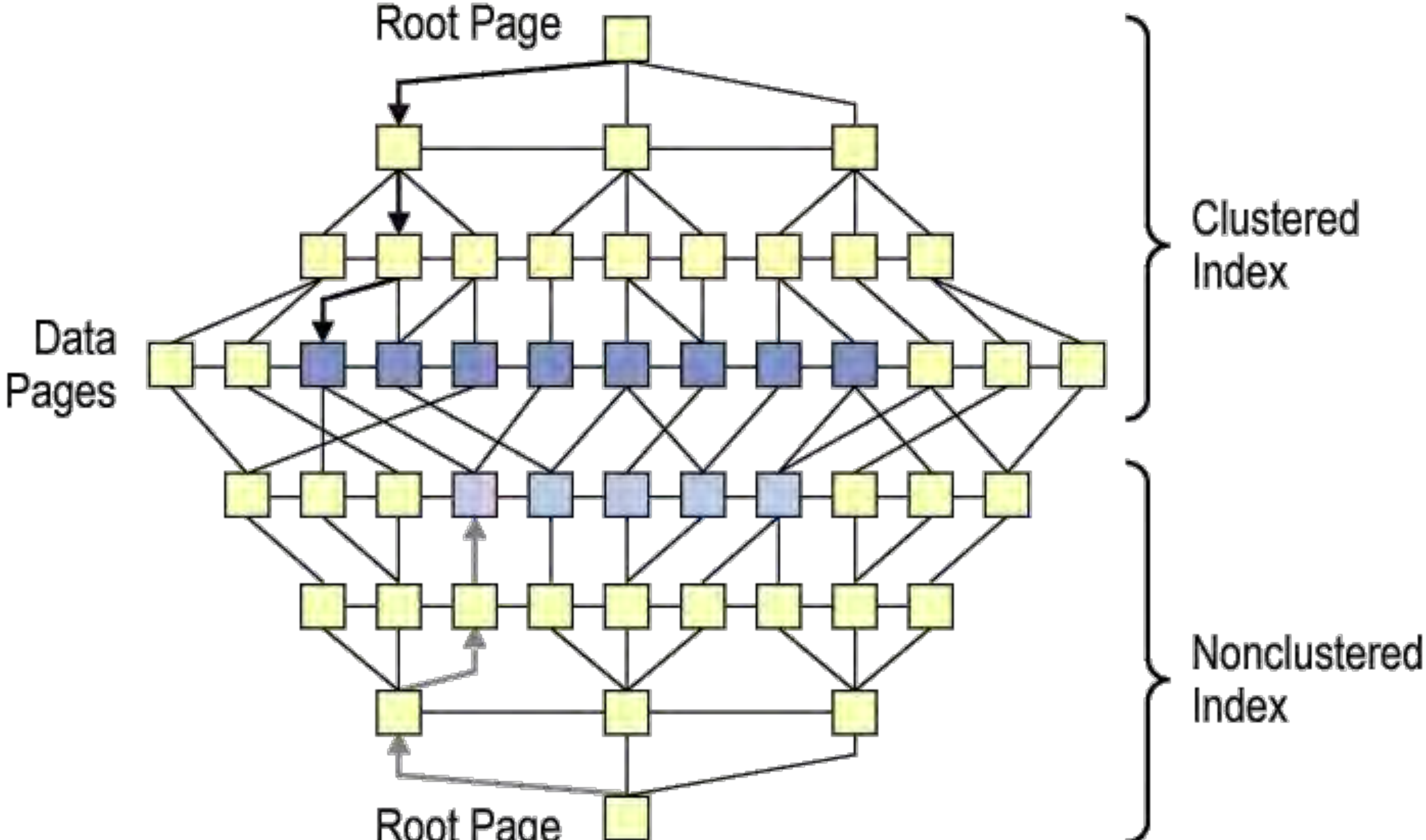
Multilevel Index (primary index)



Multilevel Index (secondary index)

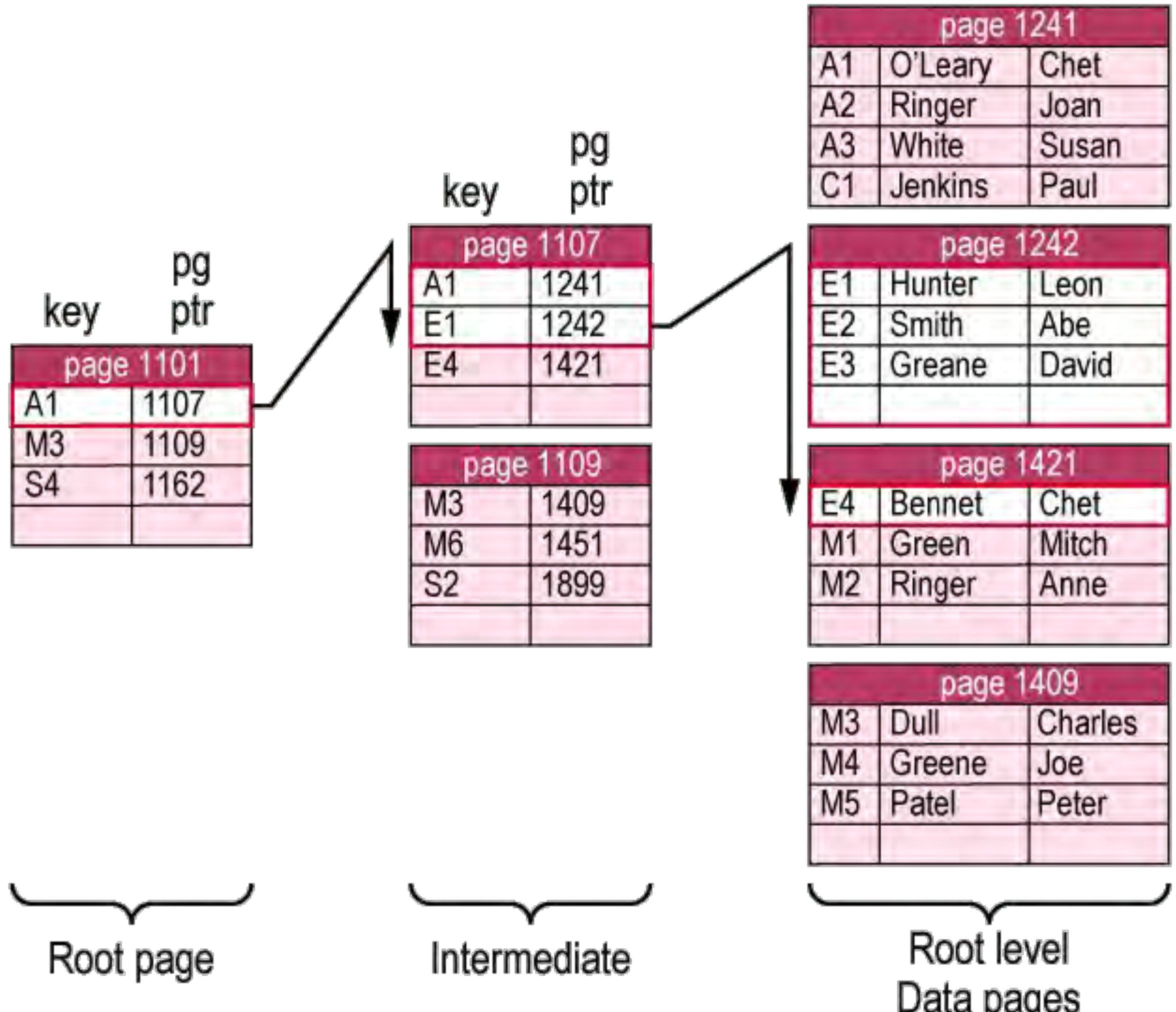


Multilevel Index (Cont.)

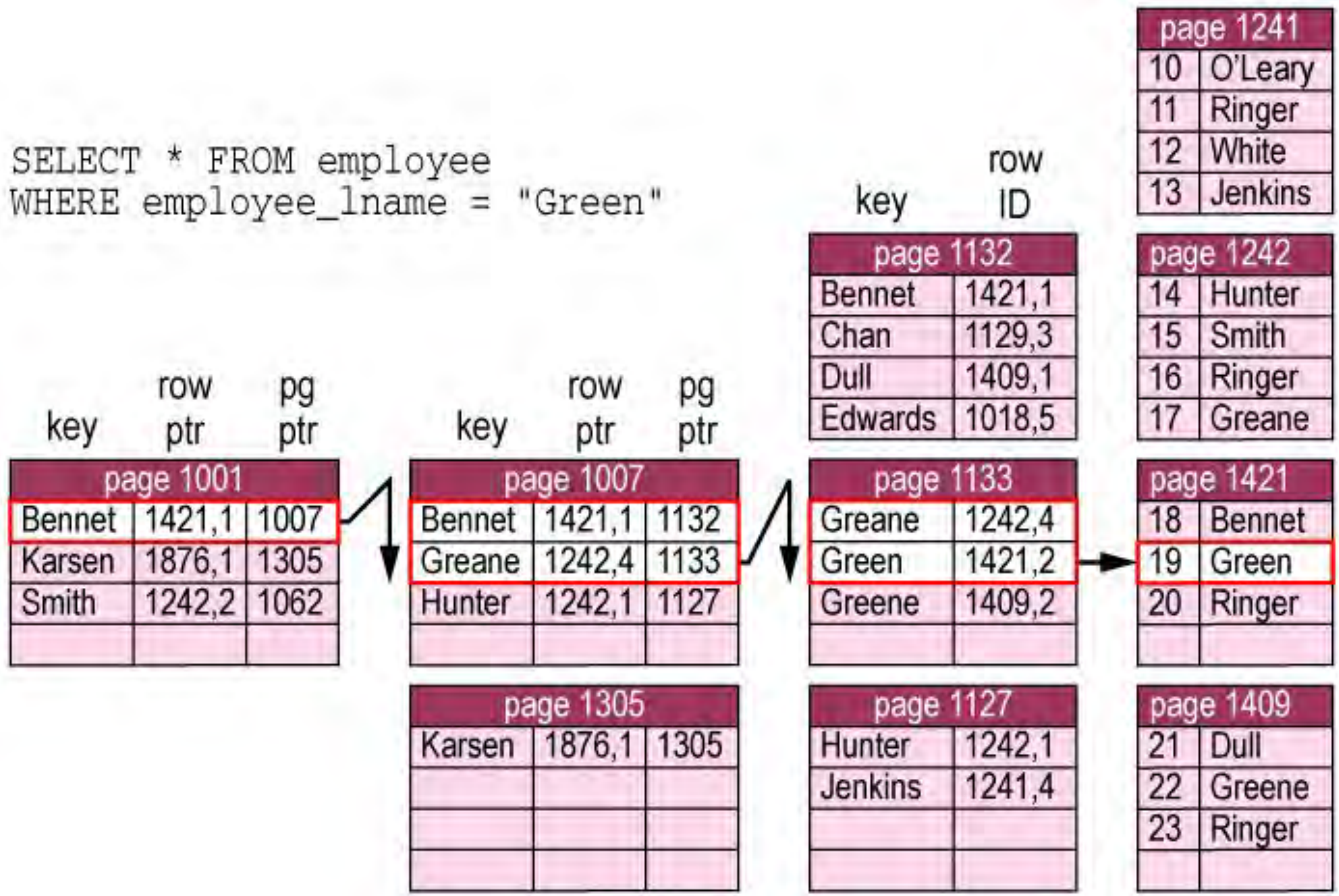


Covering indices (primary)

select * from employee
where emp_id like "E%"



Covering indices (secondary)



Covering indices (Multiple Attributes)

select emp_fname, emp_lname
from employee
where emp_lname like "Gr%"and
emp_fname like "M%"

Root page (key, row ID, pg ptr)

page 1001			
Bennet	Chet	1421,1	1007
Karsen	Kit	1876,4	1305
Smith	Abe	1242,2	1062

Intermediate (key, row ID, pg ptr)

page 1007			
Bennet	Chet	1421,1	1132
Fox	John	1317,3	1133
Hunter	Leon	1242,1	1127

page 1305			
Karsen	Kit	1876,1	1305
Lam	Pamela	1451,2	1321
Peters	Mary	1899,4	1350

Leaf level (key, row ID)

page 1132		
Bennet	Chet	1421,1
Chan	Greg	1129,3
Dull	Charles	1409,1

page 1133		
Fox	John	1317,3
Greane	David	1242,3
Green	Mitch	1421,2
Greene	Joe	1409,2

page 1127		
Hunter	Leon	1242,1
Jenkins	Paul	1241,4

Data pages

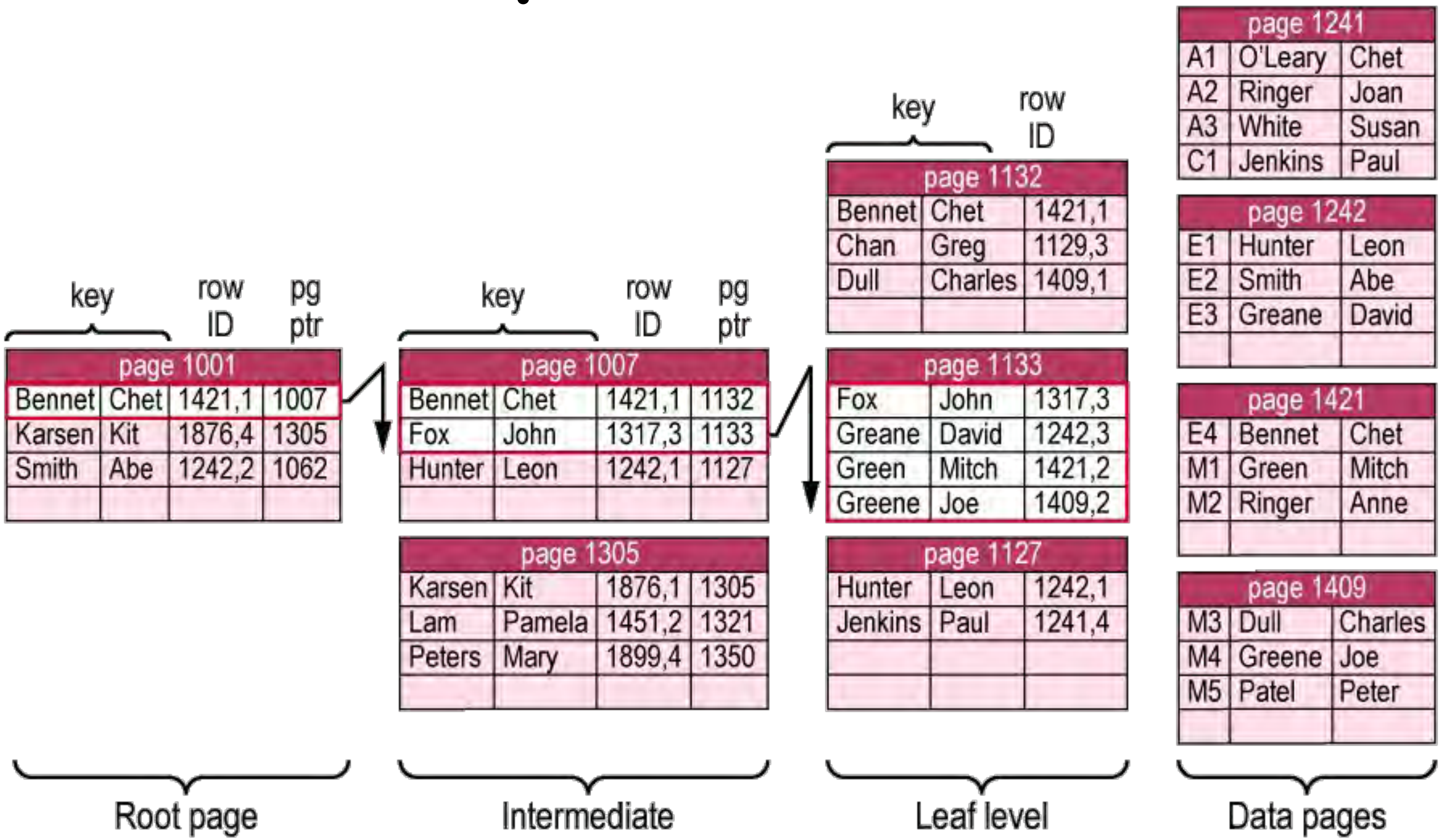
page 1241		
A1	O'Leary	Chet
A2	Ringer	Joan
A3	White	Susan
C1	Jenkins	Paul

page 1242		
E1	Hunter	Leon
E2	Smith	Abe
E3	Greane	David

page 1421		
E4	Bennet	Chet
M1	Green	Mitch
M2	Ringer	Anne

page 1409		
M3	Dull	Charles
M4	Greene	Joe
M5	Patel	Peter

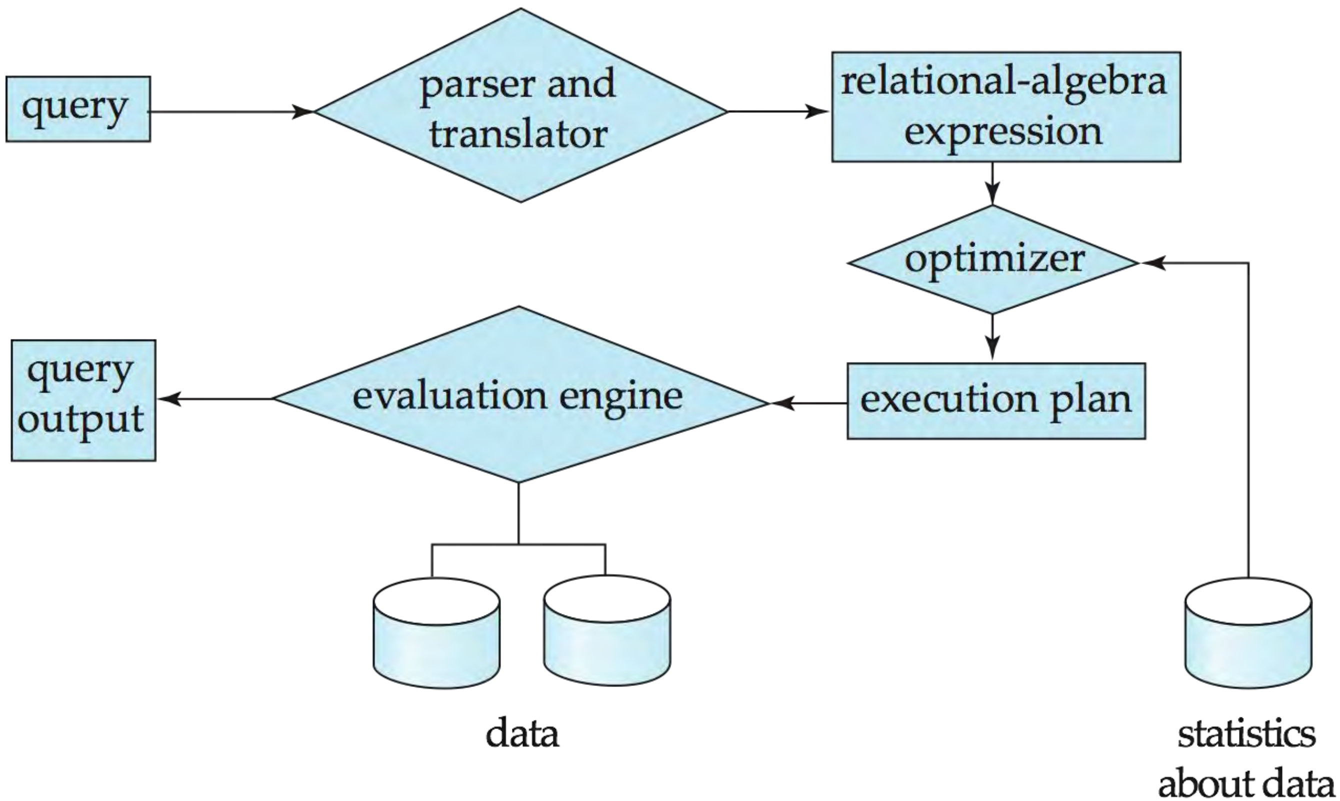
Covering indices (Multiple Attributes)



Query Processing

Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Measures of Query Cost

- **Cost is generally measured as total elapsed time for answering query**
  - **Many factors contribute to time cost**
    - ***disk accesses, CPU,* or even network *communication***
- **Typically disk access is the predominant cost, and is also relatively easy to estimate. Measured by taking into account**
  - **Number of seeks * average-seek-cost**
  - **Number of blocks read * average-block-read-cost**
  - **Number of blocks written * average-block-write-cost**
    - **Cost to write a block is greater than cost to read a block**
      - **data is read back after being written to ensure that the write was successful**

Measures of Query Cost (Cont.)

- For simplicity we just use the number of block transfers *from disk and the* number of seeks as the cost measures
  - t_T – time to transfer one block
  - t_S – time for one seek
  - Cost for b block transfers plus S seeks

    $$b * t_T + S * t_S$$
- We ignore CPU costs for simplicity
  - Real systems do take CPU cost into account
- We do not include cost to writing output to disk in our cost formulae

Measures of Query Cost (Cont.)

- Several algorithms can reduce disk IO by using extra buffer space
  - Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution
    - We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Required data may be buffer resident already, avoiding disk I/O
  - But hard to take into account for cost estimation

Selection Operation

File scan

- **Algorithm A1** (linear search). Scan each file block and test all records to see whether they satisfy the selection condition.
  - Cost estimate = b_r block transfers + 1 seek
    - b_r denotes number of blocks containing records from relation r
  - If selection is on a key attribute, can stop on finding record
    - cost = (b_r /2) block transfers + 1 seek

File scan

- Linear search can be applied regardless of
  - selection condition or
  - ordering of records in the file, or
  - availability of indices
- Note: binary search generally does not make sense since data is not stored consecutively
  - except when there is an index available,
  - and binary search requires more seeks than index search

Selections Using Indices

- Index scan – search algorithms that use an index
  - selection condition must be on search-key of index.
- *A2* (primary index, equality on key). Retrieve a single record that satisfies the corresponding equality condition
  - *Cost* = $(h_i + 1) * (t_T + t_S)$
- *A3* (primary index, equality on nonkey) Retrieve multiple records.
  - Records will be on consecutive blocks
    - Let b = number of blocks containing matching records
  - *Cost* = $h_i * (t_T + t_S) + t_S + t_T * b$

Selections Using Indices

- **A4** (secondary index, equality on nonkey).
  - Retrieve a single record if the search-key is a candidate key
    - Cost = $(h_i + 1) * (t_T + t_S)$
  - Retrieve multiple records if search-key is not a candidate key
    - each of n matching records may be on a different block
    - Cost = $(h_i + n) * (t_T + t_S)$
      - Can be very expensive!

Selections Involving Comparisons

- Can implement selections of the form $_{A}$ $_{V}$(r) or $_{A}$ $_{V}$(r) by using
  - a linear file scan,
  - or by using indices in the following ways:

- **A5** (primary index, comparison). (Relation is sorted on A)
    - For $_{A}$ $_{V}$(r) use index to find first tuple v and scan relation sequentially from there
    - For $_{A}$ $_{V}$(r) just scan relation sequentially till first tuple > v; do not use index

Selections Involving Comparisons

- A6 (secondary index, comparison).
  - For $\quad _{A} \quad _{V}(r)$ use index to find first index entry $\quad v$ and scan index sequentially from there, to find pointers to records.
  - For $\quad _{A} \; _{V}(r)$ just scan leaf pages of index finding pointers to records, till first entry > v
  - In either case, retrieve records that are pointed to
    - requires an I/O for each record
    - Linear file scan may be cheaper

Implementation of Complex Selections

- Conjunction: $_1$ $_2$ . . . $_n(r)$
- *A7* (conjunctive selection using one index).
  - Select a combination of $_i$ and algorithms A1 through A7 that results in the least cost for $_i$ (r).
  - Test other conditions on tuple after fetching it into memory buffer.
- *A8* (conjunctive selection using composite index).
  - Use appropriate composite (multiple-key) index if available.

Implementation of Complex Selections

- Conjunction: $_{1} \quad _{2} \quad . \ . \ . \quad _{n}(r)$

- **A9** (conjunctive selection by intersection of identifiers).
  - Requires indices with record pointers.
  - Use corresponding index for each condition, and take intersection of all the obtained sets of record pointers.
  - Then fetch records from file
  - If some conditions do not have appropriate indices, apply test in memory.

Algorithms for Complex Selections

- Disjunction: $_{1}$ $_{2}$ . . . $_{n}(r)$.
- A10 (disjunctive selection by union of identifiers).
  - Applicable if *all* conditions have available indices.
    - Otherwise use linear scan.
  - Use corresponding index for each condition, and take union of all the obtained sets of record pointers.
  - Then fetch records from file

Algorithms for Complex Selections

- **Negation:** (*r*)
  - Use linear scan on file
  - If very few records satisfy , and an index is applicable to
    - Find satisfying records using index and fetch from file

Sorting

- We may build an index on the relation, and then use the index to read the relation in sorted order. May lead to one disk block access for each tuple.
- For relations that fit in memory, techniques like quicksort can be used. For relations that don't fit in memory, external sort-merge is a good choice.

Join Operation

- Several different algorithms to implement joins
  - –Nested-loop join
  - –Block nested-loop join
  - –Indexed nested-loop join
  - –Merge-join
  - –Hash-join
- Choice based on cost estimate
- Examples use the following information
  - –Number of records of *student* : 5,000
    takes : 10,000
  - –Number of blocks of *student* : 100
    takes : 400

Nested-Loop Join

- To compute the theta join $r \bowtie s$

for each tuple t_r in r do begin
  for each tuple t_s in s do begin
    test pair(t_r, t_s) to see if they satisfy the join condition
      if they do, add $t_r \cdot t_s$ to the result.
  end
end

r is called the outer relation and s the inner relation of the join.

Requires no indices and can be used with any kind of join condition.

Expensive since it examines every pair of tuples in the two relations.

Nested-Loop Join (Cont.)

- **In the worst case, if there is enough memory only to hold one block of each relation, the estimated cost is**

  $n_r \quad b_s + b_r$ **block transfers, plus**

  $n_r + b_r$ **seeks**

- **If the smaller relation fits entirely in memory, use that as the inner relation.**
  - **Reduces cost to $b_r + b_s$ block transfers and 2 seeks**
- **Assuming worst case memory availability cost estimate is**
  - **with *student* as outer relation:**
    - **5000 400 + 100 = 2,000,100 block transfers,**
    - **5000 + 100 = 5100 seeks**
  - **with *takes* as the outer relation**
    - **10000 100 + 400 = 1,000,400 block transfers and 10,400 seeks**
- **If smaller relation (*student*) fits entirely in memory, the cost estimate will be 500 block transfers.**
- **Block nested-loops algorithm (next slide) is preferable.**

Block Nested-Loop Join

- **Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation.**

for each block B_r of r do begin
  for each block B_s of s do begin
    for each tuple t_r in B_r do begin
      for each tuple t_s in B_s do begin
        Check if (t_r, t_s) satisfy the join condition
        if they do, add $t_r \cdot t_s$ to the result.
      end
    end
  end
end

Block Nested-Loop Join (Cont.)

- Worst case estimate: $b_r * b_s + b_r$ block transfers + 2 * b_r seeks
  - Each block in the inner relation *s* is read once for each *block* in the outer relation
- Best case: $b_r + b_s$ block transfers + 2 seeks.
- Improvements to nested loop and block nested loop algorithms:
  - In block nested-loop, use $M - 2$ disk blocks as blocking unit for outer relations, where M = memory size in blocks; use remaining two blocks to buffer inner relation and output
    - Cost = $\lceil b_r / (M-2) \rceil * b_s + b_r$ block transfers + $2 \lceil b_r / (M-2) \rceil$ seeks
  - If equi-join attribute forms a key or inner relation, stop inner loop on first match
  - Scan inner loop forward and backward alternately, to make use of the blocks remaining in buffer (with LRU replacement)
  - Use index on inner relation if available (next slide)

Indexed Nested-Loop Join

- Index lookups can replace file scans if
  - join is an equi-join or natural join and
  - an index is available on the inner relation's join attribute
    - Can construct an index just to compute a join.
- For each tuple t_r in the outer relation r, use the index to look up tuples in s that satisfy the join condition with tuple t_r.
- Worst case: buffer has space for only one page of r, and, for each tuple in r, we perform an index lookup on s.
- Cost of the join: $b_r\ (t_T + t_S) + n_r \quad c$
  - Where c is the cost of traversing index and fetching all matching s tuples for one tuple or r
  - c can be estimated as cost of a single selection on s using the join condition.
- If indices are available on join attributes of both r and s, use the relation with fewer tuples as the outer relation.

Example of Nested-Loop Join Costs

- Compute *student* $\bowtie$ *takes,* with *student* as the outer relation.
- Let *takes* have a primary B^+–tree index on the attribute *ID,* which contains 20 entries in each index node.
- Since *takes* has 10,000 tuples, the height of the tree is 4, and one more access is needed to find the actual data
- *student* has 5000 tuples
- Cost of block nested loops join
  - 400*100 + 100 = 40,100 block transfers + 2 * 100 = 200 seeks
    - assuming worst case memory
    - may be significantly less with more memory
- Cost of indexed nested loops join
  - 100 + 5000 * 5 = 25,100 block transfers and seeks.
  - CPU cost likely to be less than that for block nested loops join