IT244 Final Summary

Fall 2015-2016

Eng/ Waleed Omar
0509114811 –Waledbak@hotmail.com

Database Management System (DBMS)

A database-management system (DBMS) is a collection of interrelated data (database) and a set of programs to access those data. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

Levels of Abstraction

➢ **Physical level:** The lowest level of abstraction describes *how* the data are actually stored. The physical level describes complex low-level data structures in detail.

➢ **Logical level:**
describes *what* data are stored in the database, and what relationships exist among those data. Describes the entire database in terms of a small number of relatively simple structures. The user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**

➢ **View level:** (The highest level of abstraction) describes only part of the entire database. Application programs hide details of data types. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

The data models can be classified into four different categories:

1. ***Relational model:*** uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as **relations**.

2. ***Entity-Relationship data model*** (mainly for database design): uses a collection of basic objects, called *entities*, and *relationships* among these objects.

3-***Object-based data models***:
• **object-oriented model** (by Java, C++, or C#) the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, method (functions), and object identity.

4- ***Semistructured Data Model***. The semistructured data model permits the specification of data where individual data items of the same type may have different sets of attributes.
• **Extensible Markup Language (XML)**
• **network data model** and **hierarchical data model**

DB relatonal model components and modeling tools (ERD)

➢ A relational database consists of a collection of **tables**, each of which is assigned a unique name.

➢ In the relational model the term ***relation*** is used to refer to a table, while the term ***tuple*** is used to refer to a *row*. Similarly, the term ***attribute*** refers to a *column* of a table

➢ A row in a table represents a *relationship* among a set of values. Since a table is a collection of such relationships.

➢ For each attribute of a relation, there is a set of permitted values(allowed values), called the **domain** of that attribute

➢ **A domain** is **atomic** if elements of the domain are considered to be indivisible units

Database Schema

➢ The **database schema**, which is the logical design of the database,

➢ The **database instance**, which is a snapshot of the data in the database at a given instant in time.

➢ a **relation schema** corresponds to the programming-language notion of type definition. Relation schema consists of a *list of attributes* and their corresponding domains

➢ **Relation instance** to refer to a specific *instance of a relation*, i.e., containing a specific set of rows.

An E-R diagram consists of the following major components:

➢ **Rectangles divided into two parts** represent entity sets. The first part, which in this textbook is shaded blue, contains the *name of the entity set*. The second part contains the names of *all the attributes* of the entity set.

➢ **Diamonds** represent relationship sets.

➢ **Undivided rectangles** represent the attributes of a relationship set. Attributes that are part of the primary key are underlined.

➢ **Lines** link entity sets to relationship sets.

➢ **Dashed lines** link attributes of a relationship set to the relationship set.

➢ **Double lines** indicate *total participation* of an entity in a relationship set.

➢ **Double diamonds** represent identifying relationship sets linked to *weak entity sets*.

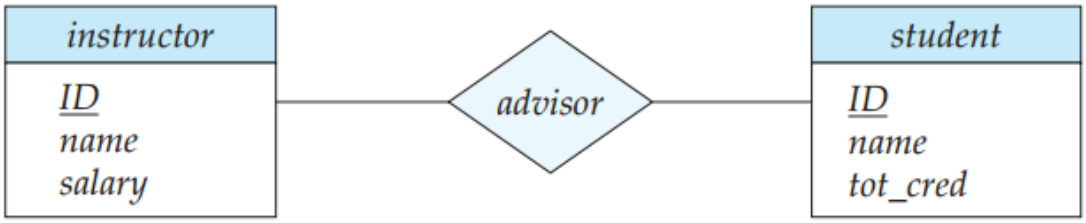


Figure 7.7 E-R diagram corresponding to instructors and students.

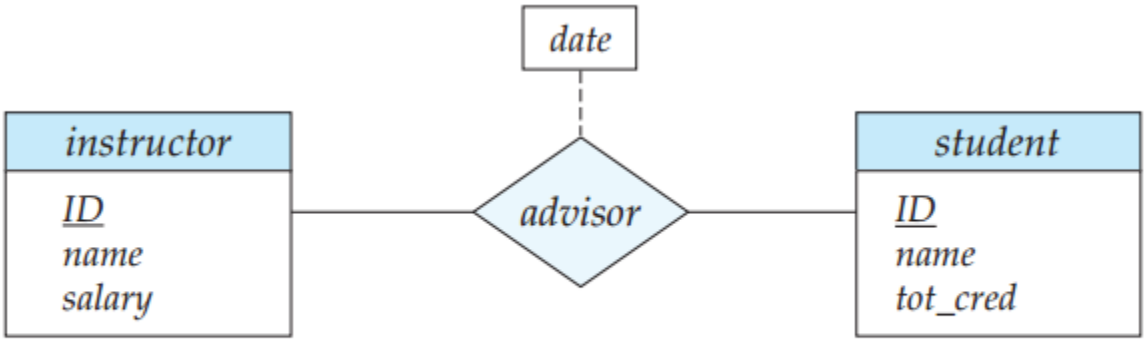
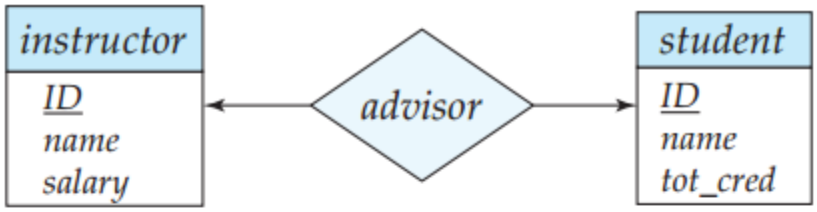
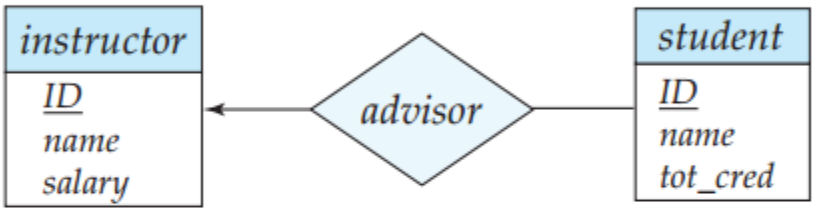


Figure 7.8 E-R diagram with an attribute attached to a relationship set.



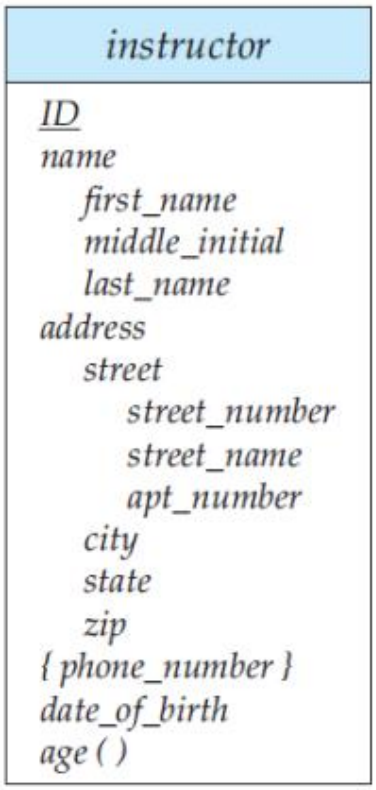
(a)



(b)

(c)

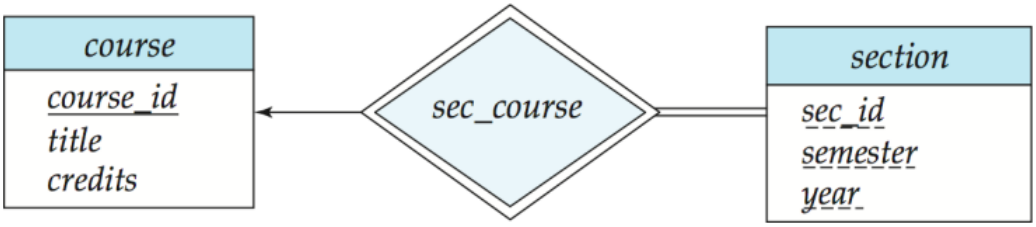
Relationships. (a) One-to-one. (b) One-to-many. (c) Many-to-many.



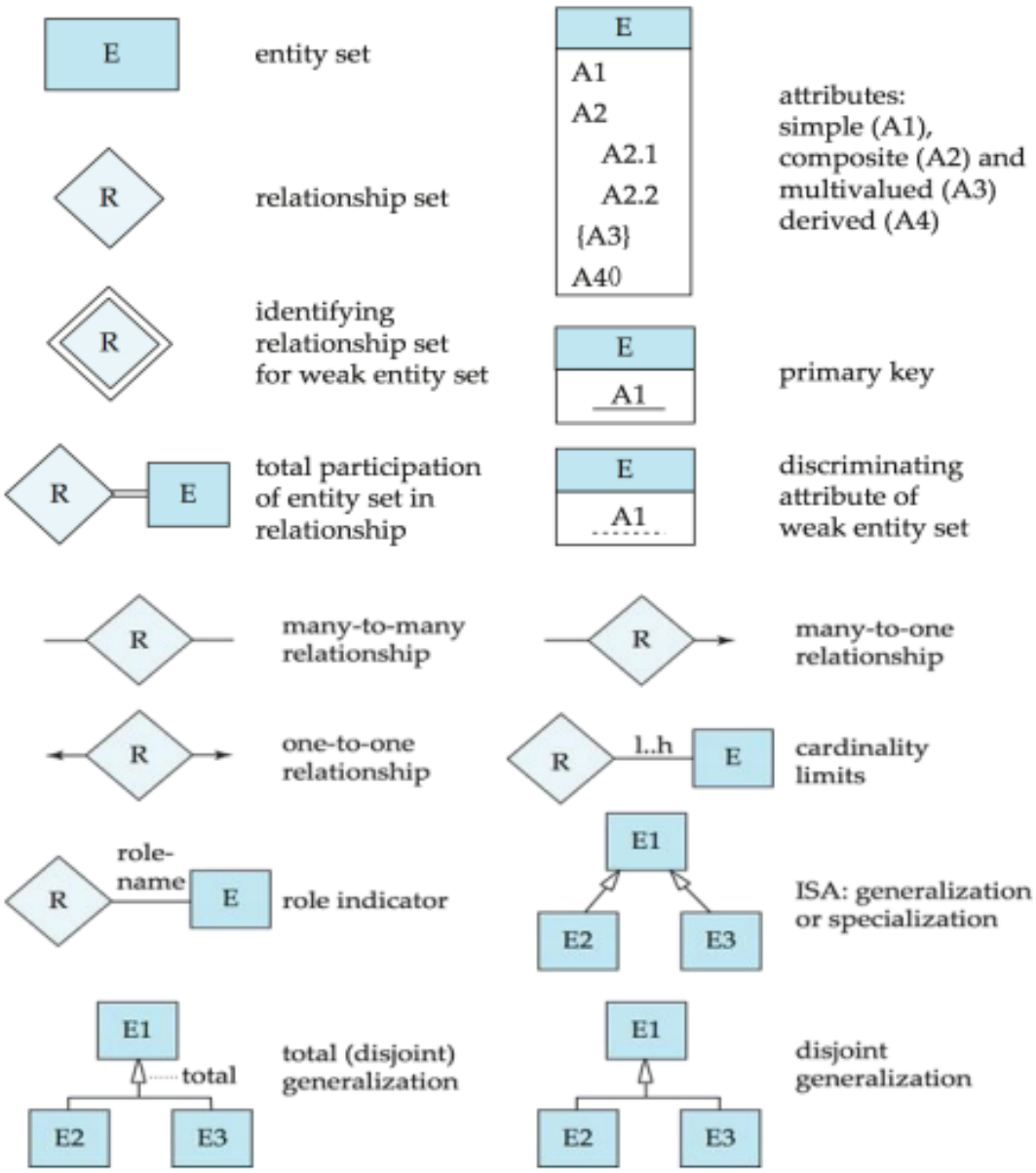
E-R diagram with composite, multivalued, and derived attributes.

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



Summary of Symbols Used in E-R Notation



Keys

➢ A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation. For example, the *ID* attribute of the relation *instructor* is sufficient to distinguish one instructor tuple from another. Thus, *ID* is a superkey.

➢ *No proper subset* is a superkey. Such *minimal* superkeys are called ***candidate keys.***

➢ **Primary key** to denote a candidate key that is chosen by the database designer as the principal means of *identifying tuples within a relation*.

➢ A **foreign key** is a set of attributes in a referencing relation, such that for each tuple in the referencing relation, the values of the foreign key attributes are guaranteed to occur *as the primary key value* of a tuple in the referenced relation.

➢ a **referential integrity constraint**; a referential integrity constraint requires that the values appearing in specified attributes of any tuple in the referencing relation also appear in specified attributes of at least one tuple in the referenced relation.

SQL – chapter 3

Types in SQL

- **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 machinedependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *n* digits to the right of decimal point.

DDL

The SQL **data-definition language (DDL)** allows the specification of information about relations, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints

  create table *instructor* (
  ID **char**(5),
  name **varchar**(20) **not null,**
  dept_name **varchar**(20),
  salary **numeric**(8,2),
  primary key (*ID*),
  foreign key *(dept_name*) **references** *department)*

- **Drop and Alter Table Constructs**

  drop table *student*
  - Deletes the table and its contents

delete from *student*

- Deletes all contents of table, but retains table

alter table

add coulmn

alter table *instructor* **add** *commision float*

drop column

alter table *instructor* **drop** *commision*

Select

To find all instructors in Comp. Sci. dept with salary > 80000

select *name*
from *instructor*
where *dept_name = '*Comp. Sci.' **and** *salary* > 80000

Joins

For all instructors who have taught some course, find their names and the course ID of the courses they taught.

select *name, course_id*
from *instructor, teaches*
where *instructor.ID = teaches.ID*

Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

select *section.course_id, semester, year, title*
from *section, course*
where *section.course_id = course.course_id* **and**
dept_name = 'Comp. Sci.'

Natural Join

Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column

select *
from *instructor* ***natural join*** *teaches;*

List the names of instructors along with the the titles of courses that they teach

select *name, title*
from *instructor* ***natural join*** *teaches, course*
where *teaches.course_id = course.course_id*;

The Rename Operation

The SQL allows renaming relations and attributes using the ***as*** clause: *old-name* **as** *new-name*

select *ID, name, salary/12* **as** *monthly_salary*
from *instructor*

Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
select distinct *T. name*
from *instructor* **as** *T, instructor* **as** *S*
where *T.salary > S.salary* **and** *S.dept_name = 'Comp. Sci.'*

Ordering the Display of Tuples

List in alphabetic order the names of all instructors

select distinct *name*
from *instructor*
order by *name asc*
We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default. Example: **order by** *name* **desc**

Where Clause Predicates

Find the names of all instructors whose name includes the substring "dar".
select *name*
from *instructor*
where *name* **like** '%dar%'

Find the names of all instructors with salary **between** $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)

select *name*
from *instructor*
where *salary* **between** 90000 **and** 100000

Set Operations

■ Find courses that ran in Fall 2009 **or** in Spring 2010?

(**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
union
(**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

■ Find courses that ran in Fall 2009 **and** in Spring 2010?

(**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
intersect
(**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

■ Find courses that ran in Fall 2009 **but not** in Spring 2010?

(**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
except
(**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

Null Values

- It is possible for tuples to have a null value for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null* as 5 + *null* returns null
- The predicate **is null** can be used to check for null values.

Find all instructors whose salary is null*?*

select *name*
from *instructor*
where *salary* **is null**

Aggregate Functions

■ Find the average salary of instructors in the Computer Science department?

select avg (*salary*)
from *instructor*
where *dept_name*= 'Comp. Sci.';

■ Find the total number of instructors who teach a course in the Spring 2010 semester?

select count (distinct ID)
from teaches
where semester = 'Spring' and year = 2010

■ Find the number of tuples in the *course* relation?

select count (*)
from *course*;

■ Find the average salary of instructors in each department?

Select *dept_name*, **avg** (*salary*)
from *instructor*
group by *dept_name*;

■ Find the names and average salaries of all departments whose average salary is greater than 42000?

select *dept_name*, **avg** (*salary*)
from *instructor*
group by *dept_name*
having avg (*salary*) > 42000

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Subquery

■ Find courses offered in Fall 2009 and in Spring 2010?

select distinct *course_id*
from *section*
where *semester* = 'Fall' **and** *year*= 2009 **and**
course_id **in** (**select** *course_id*
from *section*
where *semester* = 'Spring' **and** *year*= 2010)

■ Find courses offered in Fall 2009 but not in Spring 2010?

select distinct *course_id*
from *section*
where *semester* = 'Fall' **and** *year*= 2009
and *course_id* **not in** (**select** *course_id*
from *section*
where *semester* = 'Spring' **and** *year*= 2010);

Modification of the Database – DML

Deletion

Delete all instructors?

delete from *instructor*

Delete all instructors from the Finance department?

delete from *instructor*
where *dept_name*= 'Finance'

Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building?

```
delete from instructor
where dept_name in (select dept_name
                    from department
                    where building = 'Watson');
```

Insertion

Add a new tuple to *course?*

```
insert into course
values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

or equivalently

```
insert into course (course_id, title, dept_name, credits)
values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

Add a new tuple to *student* with *tot_creds* set to null

```
insert into student
values ('3003', 'Green', 'Finance', null);
```

Updates

Increase salaries of instructors whose salary is over $100,000 by 3%

```
update instructor
set salary = salary * 1.03
where salary > 100000;
```

Relational Algebra-Chapter 6

Select Operation

To find the instructors in Physics with a salary greater than $90,000, we write:

$$\sigma_{dept\_name = \text{“Physics”} \wedge salary > 90000}(instructor)$$

Project operator

To eliminate the *dept_name* attribute of *instructor?*

$\prod$ ***ID, name, salary (instructor)***

Set operators

To find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both?

$\prod$ *course_id* (σ *semester="Fall" ∧ year=2009* (*section*)) ∪
$\prod$ *course_id* (σ *semester="Spring" ∧ year=2010* (*section*))

To find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester?

$\prod$ *course_id* (σ *semester="Fall" ∧ year=2009* (*section*)) –
$\prod$ *course_id* (σ *semester="Spring" ∧ year=2010* (*section*))

Cartesian-Product Operation

r x s

■ Find the ids of all instructors in the Physics department, along with the *course_id* of all courses they have taught?

$\prod$ *instructor.ID,course_id* (σ *dept_name="*Physics" (σ *instructor.ID=teaches.ID* (*instructor* x *teaches*)))

Natural Join

Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach

$$\prod_{name,\ title} (\sigma_{dept\_name="Comp.\ Sci."} (instructor \bowtie teaches \bowtie course))$$

Outer join

Left Outer Join

instructor ⟕ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

instructor ⟖ ***teaches***

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

instructor* ⟗ *teaches

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*
76766	null	null	BIO-101

Aggregate Operation

- Find the average salary in each department

$\prod$ *dept_name* , **avg**(*salary*) (*instructor*)

Bank relations

- branch(**branchname**, branchcity, assets)
- customer(customername,customerstreet, customercity)
- loan(loan number, branchname, **amount**)
- borrower(customername,loan number)
- account(accountnumber,branchname, balance)
- depositor(customername,accountnumber)

- Find the names of all customers who have a loan and an account at bank.

$\prod_{customer\_name}$ (*borrower*) $\cap$ $\prod_{customer\_name}$ (*depositor*)

- Find the name of all customers who have a loan at the bank and the loan amount

$\prod_{customer\_name,\ loan\_number,\ amount}$ (*borrower* $\bowtie$ *loan*)

- Find all customers who have an account from at least the "Downtown" and the Uptown" branches.

$$\prod_{customer\_name} (\sigma_{branch\_name = \text{"Downtown"}} (depositor \bowtie account)) \cap$$
$$\prod_{customer\_name} (\sigma_{branch\_name = \text{"Uptown"}} (depositor \bowtie account))$$

- Find all customers who have an account at **all branches** located in Brooklyn city.

$$\prod_{customer\_name,\ branch\_name} (depositor \bowtie account)$$
$$\div \prod_{branch\_name} (\sigma_{branch\_city = \text{"Brooklyn"}} (branch))$$

Exercise 2.7

***employee* (*person_name*, *street*, *city*)**

***works* (*person_name*, *company name*, *salary*)**

***company* (*company name*, *city*)**

Give an expression in the relational algebra to express each of the following queries:

a. Find the names of all employees who live in city "Miami".

b. Find the names of all employees whose salary is greater than $100,000.

c. Find the names of all employees who live in "Miami" and whose salary is greater than $100,000.

Answer

a. **Π *person_name* (σ city = "Miami" (*employee*))**

b. **Π *person_name* (σ salary >= 100000 (*works*))**

c. **Π *person_name* (σ salary >= 100000 ∧ city = "Miami" (*employee*** ⋈ works**))**

Normal forms

The functional dependency

$\alpha \rightarrow \beta$

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, $t1[\alpha] = t2[\alpha] \Rightarrow t1[\beta] = t2[\beta]$

t*(*ID, name)

ID* → *building

Atomic Domains and First Normal Form

- Domain is **atomic** if its elements are considered to be *indivisible* units

Examples of non-atomic domains:

  - Set of names, composite attributes
  - Identification numbers like CS101 that can be broken up into parts.

- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic.
- *Non-atomic* values complicate storage and encourage redundant (repeated) storage of data

- *K* is a superkey for relation schema *R* if and only if $K \rightarrow R$
- *K* is a candidate key for *R* if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$

Closure of a Set of Functional Dependencies

- Given a **set *F*** of functional dependencies, there are certain other functional dependencies that are logically *implied* by *F*.
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by *F* is the **closure of *F***.
- We denote the ***closure* of *F* by F+.**
- **F+ is a superset of *F***.

Boyce–Codd Normal Form

A relation schema *R* is in **BCNF** with respect to a set *F* of functional dependencies if for all functional dependencies **in *F+*** of the form

$\alpha \rightarrow \beta$

Where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for *R*

Example1:

***Inst_dept* (*ID*, *name*, *salary*, *dept_name*, *building*, *budget*)**

- The functional dependency ***dept_name → budget*** holds on *inst_dept*, but *dept_name* is **not a superkey** (because, a department may have a number of different instructors).
- The *instructor* schema **is in BCNF**. All of the nontrivial functional dependencies that hold, such as:
  ID→name, dept name, salary

General rule for decomposing that are not in BCNF

R be a schema that is not in BCNF.

Then there is at least one nontrivial functional dependency $\alpha \rightarrow \beta$ such that α is not a superkey for *R*. We replace *R* in our design with two schemas:

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

In the case of *inst_dept* above, α = *dept_name*, β = *{building*, *budget}*, and *inst_dept* is replaced by

- $((\alpha \cup \beta) =$ (*dept_name*, *building*,*budget*)
- $(R - (\beta - \alpha)) =$ (*ID*, *name*, *dept_name*, *salary*)

Example

dept advisor (s_ID, i_ID, dept_name)
i_ID→dept_name
s_ID, dept_name→i_ID

Notice that with this design, we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship. We see that *dept_advisor* is not in BCNF because *i_ID* is **not a superkey**

BCNF decomposition, we get:
(s_ID, i_ID)
(i_ID, dept_name)

Third Normal Form

Third normal form prevents **dependency preserving**.
A relation schema *R* is in **third normal form** with respect to a set *F* of functional dependencies if, for all functional dependencies in *F*+ of the form $\alpha \rightarrow \beta$ where $\alpha \subseteq \boldsymbol{R}$ and $\beta \subseteq \boldsymbol{R}$, at least one of the following holds:-

1. $\alpha \rightarrow \beta$ **is trivial (i.e.,** $\beta \in \alpha$**)**
2. α **is a superkey for *R***
3. Each attribute *A* in $\beta - \alpha$ **is contained in a candidate key for *R*.**

- If a relation is in **BCNF it is in 3NF** (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation.

Now, let us again consider the ***dept_advisor*** relationship set, which has the following functional dependencies:

i_ID→dept_name
s_ID, dept_name→i_ID

- The functional dependency "*i_ID → dept name*" caused the *dept_advisor* schema **not** to be in BCNF. Note that here α = *i_ID*, β =*deptname*, and $\beta - \alpha$= *dept name*.

- Since the functional dependency *s_ID*, *dept name→i_ID* holds on *dept_advisor*, the attribute *dept_name* is contained in a candidate key , so *dept_advisor* **is in 3NF**.

Example of BCNF Decomposition

$R = (A, B, C)$

$F = \{A \rightarrow B$

$B \rightarrow C\}$

Key = {*A*}

R is not in BCNF ($B \rightarrow C$ but *B* is not superkey)

Decomposition

*R*1 = (*B, C)*

*R*2 = *(A,B)*

Example

class(*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)

Functional dependencies:

course_id→ title, dept_name, credits

building, room_number→capacity

course_id, sec_id, semester,year→building,room_number,time_slot_id

A candidate key {***course_id, sec_id,semester,year***}

BCNF Decomposition:

course_id→ title, *dept_name*, *credits* holds

- but *course_id* is not a superkey.

We replace *class* by:

- ***course*(*course_id*, *title*, *dept_name*, *credits*)**
- ***class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*,*room_number*, *capacity*, *time_slot_id*)**

building, *room_number→capacity* holds on *class-1*

• but {*building*, *room_number*} is not a superkey for *class-1*.

• We replace *class-1* by:

***classroom* (*building*, *room_number*, *capacity*)**

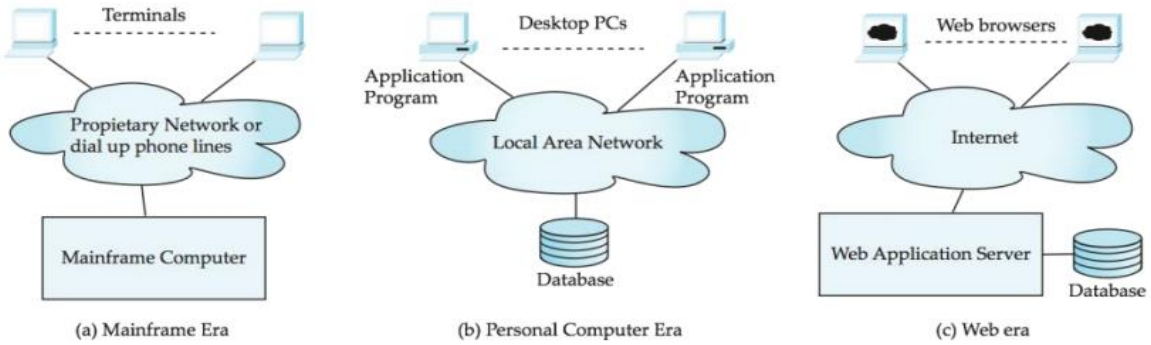
***section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*,*time_slot_id*)**

***Now classroom* and *section* are in BCNF.**

Chapter 9

Application Architecture Evolution

- Three distinct era's of application architecture
- mainframe (1960's and 70's)
- personal computer era (1980's)
- Web era (1990's onwards)



(a) Mainframe Era (b) Personal Computer Era (c) Web era

Application Architectures

Application layers

- Presentation or user interface
  - **model-view-controller (MVC)** architecture
    - **model**: business logic
    - **view**: presentation of data, depends on display device
    - **controller**: receives events, executes actions, and returns a view to the user

business-logic layer

- provides high level view of data and actions on data
  - often using an object data model
- hides details of data storage schema

data access layer

- interfaces between business logic layer and the underlying database
- provides mapping from object model of business layer to relational model of database.

Business Logic Layer

☞ Provides abstractions of entities
- e.g. students, instructors, courses, etc

☞ Enforces **business rules** for carrying out actions
- E.g. student can enroll in a class only if she has completed prerequsites, and has paid her tuition fees

☞ Supports **workflows** which define how a task involving multiple participants is to be carried out

- E.g. how to process application by a student applying to a university
- Sequence of steps to carry out task
- Error handling
  - e.g. what to do if recommendation letters not received on time

Web fundamentals

1-Uniform Resources Locators

☞ In the Web, functionality of pointers is provided by Uniform Resource Locators (URLs).

URL example:

http://www.acm.org/sigmod

- The first part indicates how the document is to be accessed
  - "http" indicates that the document is to be accessed using the Hyper Text Transfer Protocol.
- The second part gives the unique name of a machine on the Internet.
- The rest of the URL identifies the document within the machine.

☞The local identification can be:

- The path name of a file on the machine, or
- An identifier (path name) of a program, plus arguments to be passed to the program
  - E.g., http://www.google.com/search?q=silberschatz

2- HTML and HTTP

☞ HTML provides formatting, hypertext link, and image display features

- including tables, stylesheets (to alter default formatting), etc.

☞ HTML also provides input features

- Select from a set of options
  - Pop-up menus, radio buttons, check lists
- Enter values
  - Text boxes
- Filled in input sent back to the server, to be acted upon by an executable at the server

☞ HyperText Transfer Protocol (HTTP) used for communication with the Web server

3- Web Servers

☞A Web server can easily serve as a front end to a variety of information services.

☞ The document name in a URL may identify an executable program, that, when run, generates a HTML document.

- When an HTTP server receives a request for such a document, it executes the program, and sends back the HTML document that is generated.
- The Web client can pass extra arguments with the name of the document.

- To install a new service on the Web, one simply needs to create and install an executable that provides that service.
  - The Web browser provides a graphical user interface to the information service.
- Common Gateway Interface (CGI): a standard interface between web and application server

Application Security

1-SQL Injection

- Suppose query is constructed using
  - "select * from instructor where name = '" + name + "'"
- Suppose the user, instead of entering a name, enters:
  - X' or 'Y' = 'Y
- then the resulting statement becomes:
  - "select * from instructor where name = '" + "X' or 'Y' = 'Y" + "'"
  - which is:
    - select * from instructor where name = 'X' or 'Y' = 'Y'
  - User could have even used
    - X'; update instructor set salary = salary + 10000;

- Prepared statement internally uses:

  "select * from instructor where name = 'X\' or \'Y\' = \'Y'
- **Always use prepared statements, with user inputs as parameters**
- Is the following prepared statemen secure?
  - conn.prepareStatement("select * from instructor where name = '" + name + "'")

2-Cross Site Scripting

- HTML code on one page executes action on another page
  - E.g. <img src =http://mybank.com/transfermoney?amount=1000&toaccount=14523>
  - Risk: if user viewing page with above code is currently logged into mybank, the transfer may succeed

  - Above example simplistic, since GET method is normally not used for updates, but if the code were instead a script, it could execute POST methods

- Above vulnerability called **cross-site scripting (XSS)** or **cross-site request forgery (XSRF or CSRF)**

- **Prevent your web site from being used to launch XSS or XSRF attacks**
  - Disallow HTML tags in text input provided by users, using functions to detect and strip such tags
- **Protect your web site from XSS/XSRF attacks launched from other sites**

☞ **Protect your web site from XSS/XSRF attacks launched from other sites**

- Use **referer** value (URL of page from where a link was clicked) provided by the HTTP protocol, to check that the link was followed from a valid page served from same site, not another site
- Ensure IP of request is same as IP from where the user was authenticated
  - prevents hijacking of cookie by malicious user
- Never use a GET method to perform any updates, recommended by HTTP standard 21

3- Password Leakage

☞Never store passwords, such as database passwords, in clear text in scripts that may be accessible to users

- E.g. in files in a directory accessible to a web server
  - Normally, web server will execute, but not provide source of script files

☞ Restrict access to database server from IPs of machines running application servers

- Most databases allow restriction of access by source IP address

4- Application Authentication

☞Single factor authentication such as passwords too risky for critical applications

- guessing of passwords, sniffing of packets if passwords are not encrypted
- passwords reused by user across sites
- spyware which captures password

☞ Two-factor authentication

- e.g. password plus one-time password sent by SMS
- e.g. password plus one-time password devices
  - device generates a new pseudo-random number every minute, and displays to user
  - user enters the current number as password
  - application server generates same sequence of pseudorandom numbers to check that the number is correct.

☞ **Man-in-the-middle** attack

- E.g. web site that pretends to be mybank.com, and passes on requests from user to mybank.com, and passes results back to user
- Even two-factor authentication cannot prevent such attacks

☞ Solution: authenticate Web site to user, using digital certificates, along with secure http protocol

☞ **Central authentication** within an organization

- application redirects to central authentication service for authentication
- avoids multiplicity of sites having access to user's password
- LDAP or Active Directory used for authentication