Database Systems

ECE 356
University of Waterloo
Dr. Paul A.S. Ward

Acknowledgment: slides derived from Dr. Wojciech Golab

based on materials provided by
Silberschatz, Korth, and Sudarshan, copyright 2010
(source: www.db-book.com)

Learning Outcomes

- What is the problem for which DB is the solution?
- Brief history of databases.
- Advantages of modern databases over legacy file processing systems.
- Physical and logical data independence.
- How to set up your own MySQL instance.

- Textbook reading (6th ed.): 1 (skim), 2

What is the Problem?

- Data Management
  - Storage
  - Manipulation
  - Query
- Dependability
  - Against loss
  - Against corruption
- Integrity
- Transaction capacity

History of Database Systems

- 1950s and early 1960s:
  - ran on early transistor-based programmable digital computers
  - punched cards used for input (i.e., program)
  - magnetic tapes used to store data sets

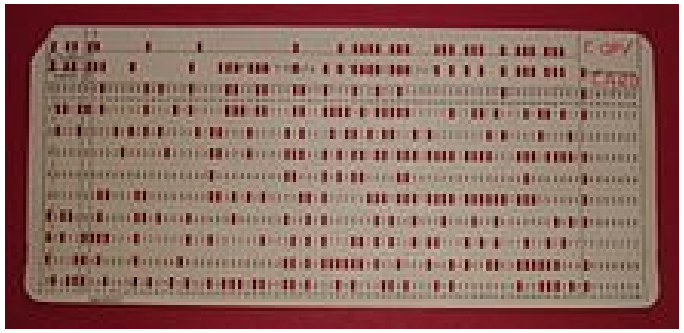


Image sources: http://en.wikipedia.org/wiki/Magnetic_tape
http://en.wikipedia.org/wiki/Punched_card

History of Database Systems

- Late 1960s and 1970s:
  - hard disks allowed more direct access to data
  - network and hierarchical data models in widespread use
  - Ted Codd (1923-2003) defines the relational data model
    - won the ACM Turing Award in 1981 for this pioneering work
    - IBM Research begins System R prototype
    - UC Berkeley begins Ingres prototype
  - high-performance (for the era) transaction processing

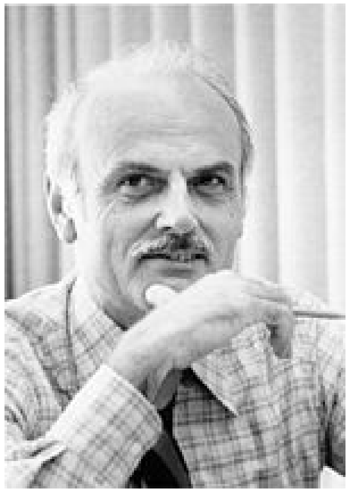


Image source:
http://en.wikipedia.org/wiki/Edgar_F._Codd

History of Database Systems

- 1980s:
  - research relational prototypes evolve into commercial systems
  - SQL becomes industry standard
  - first parallel and distributed database systems
  - object-oriented database systems
- 1990s:
  - large decision support and data-mining applications
  - large multi-terabyte data warehouses
  - emergence of Web commerce
- Early 2000s:
  - XML and XQuery standards
  - automated database administration
- Later 2000s:
  - unstructured and semi-structured data
  - scalable NoSQL systems (BigTable, PNuts, Dynamo)

Drawbacks of Using File Systems

- File systems lead to data redundancy and inconsistency.
  - multiple file formats, duplication of information in different files
- File systems cannot answer queries directly.
  - need to write a new program to carry out each new task
  - one data set may be scattered across multiple files
- File systems do not enforce integrity.
  - integrity constraints (*e.g.*, account balance $\geq$ 0) become buried in program code rather than being stated explicitly
  - adding new constraints or changing existing ones is cumbersome

Drawbacks of Using File Systems

- Updates in a file system are not always atomic.
  - failures may leave data in an inconsistent state with partial updates carried out (*e.g.*, transfer of funds from one account to another)
- File systems offer limited support for concurrent access by multiple users.
  - concurrent access needed for performance
  - uncontrolled concurrent accesses can lead to inconsistencies (*e.g.*, two people withdrawing money from the same account simultaneously)
- File systems do not provide sufficient security.
  - difficult to provide access to a specific subset of the data

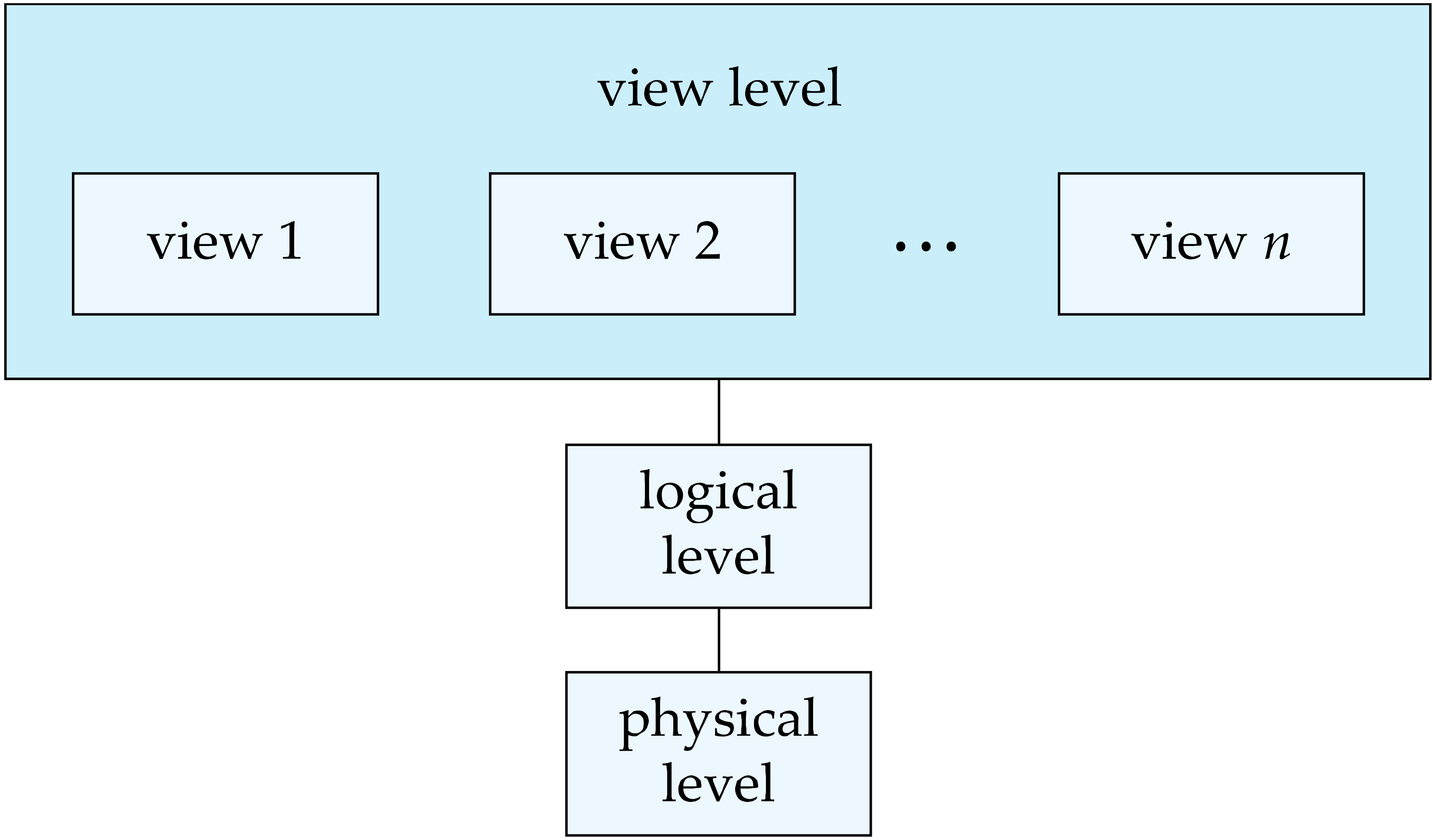
A database management system (DBMS) offers solutions to all the above problems!

(WTF? Be careful of statements like this. Why?)

Levels of Abstraction in a Database

- **Physical level:** describes how a record (*e.g.*, customer) is stored.
  - *e.g.*, customers are stored in ascending order by ID, and there is a secondary index on the name attribute
- **Logical level:** describes the structure of the data stored in a database, and the relationships among the data.
  - *e.g.* an instructor has an ID and name, and belongs to some department.
- **View level:** describes a virtual structure of the data imposed by the database designer on top of the logical level (e.g., a virtual table defined as the result of querying a base table or joining two base tables).
  - *e.g.*, students may see an instructor’s teaching evaluations, but only HR staff (or the instructor) may see the instructor’s salary

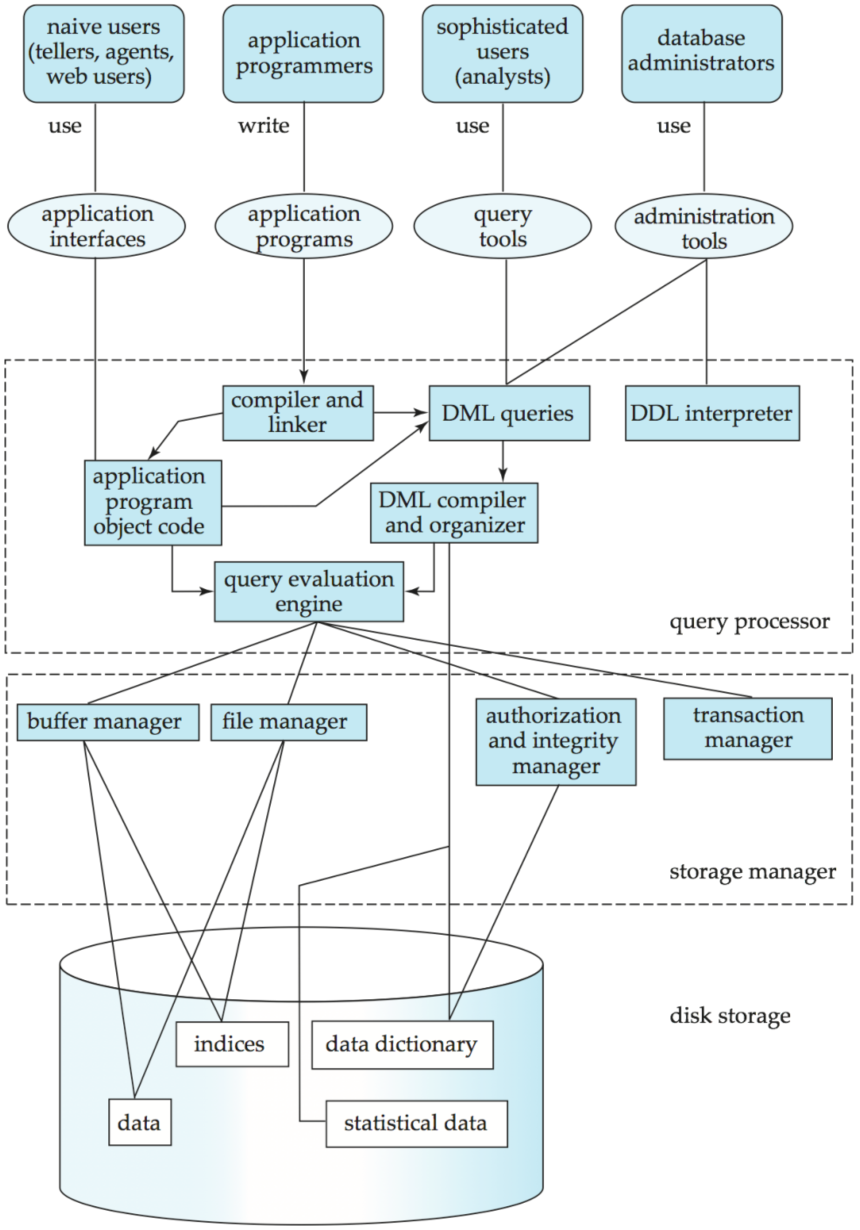
Levels of Abstraction



Physical vs. Logical Schemas

- **Schema** – the shape or structure of the database.
  - **physical schema**: design at physical level (storage, file formats, indexes)
  - **logical schema**: design at logical level (data types, relations, rows, columns)
- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema or the application program.
  - Example 1: a table that is accessed frequently can be moved to a faster disk without breaking any of the queries (and hence without breaking the applications that depend on these queries).
  - Example 2: an index can be added to speed up a specific query without breaking that query or other queries.
- **Logical Data Independence** – the ability to modify the logical schema without changing the application program.
  - Example 1: a new table can be added to the logical schema without breaking any of the queries defined over existing tables or views.
  - Example 2: a column can be added to a table without breaking any of the queries that access the table or any view defined over the table.

Structure of a Modern Database



Setting up MySQL

Preamble:

- The ECE department provides a managed instance of the MySQL database on eceweb.uwaterloo.ca to support ECE356, and therefore you are not required to set up the database yourself. That said, you are encouraged to install MySQL on your own computer for convenience and as a useful practical exercise.
- The instructions in these slides cover both Linux and Windows, and were tested against the following software versions:
  - MySQL 5.5 on Ubuntu Linux 14.04 (LTS)
  - MySQL 5.6 on Windows 8.1

Setting up MySQL

Instructions for Ubuntu Linux

- Step 1: provision a Linux box and obtain sudo access.

Using an Amazon EC2 instance:

- select Ubuntu Server 14.04 LTS (HVM) 64-bit instance
- select instance type (*e.g*., general purpose t2.medium)
- edit security group by adding a custom TCP rule to allow incoming connections on port 3306 (MySQL JDBC interface) from anywhere
- launch the instance and record its public IP address
- access the VM using ssh as user “ubuntu” (ssh ubuntu@host)

Using your personal computer:

- open TCP port 3306 in your firewall if planning remote access (*e.g*., sudo ufw allow proto tcp from any to any port 3306)

Setting up MySQL

- Step 2: install and secure MySQL
  - sudo apt-get update
  - sudo apt-get upgrade
  - sudo apt-get install mysql-server
  - mysql_secure_installation
    - enter the root password you chose earlier
    - you need not change the root password ("n" to first question)
    - answer "y" to all the other questions
      (i.e., remove anonymous users, disallow root login remotely, remove test database, reload privilege tables now)
  - If planning remote access then edit /etc/mysql/my.cnf and comment out the line with the "bind-address" property
  - sudo service mysql restart

Setting up MySQL

- Step 3: add a database, table and user
  - launch the MySQL shell: mysql -u root -p
    - enter the MySQL root password you chose earlier
  - from the mysql shell execute the following commands:
    - CREATE DATABASE ece356db;
    - CREATE USER 'user_ece356_test'@'%' IDENTIFIED BY 'user_ece356_test';
    - GRANT ALL ON ece356db.* TO 'user_ece356_test'@'%';
    - EXIT;

Setting up MySQL

- Step 4: add a table
  - launch the MySQL shell again: mysql -u user_ece356_test -p
    - enter the password “user_ece356_test”
  - from the mysql shell execute the following commands:
    - USE ece356db;
    - CREATE TABLE Persons (ID INT, FirstName VARCHAR(255), LastName VARCHAR(255));
    - EXIT;

Setting up MySQL

- Step 5: connect to the server remotely
  - launch NetBeans
  - click Window > Services to open the Services tab
  - right-click on Databases, choose “New Connection…”
  - choose “MySQL (Connector/J driver)” as the driver
  - if using EC2 then enter the public IP or DNS name of the server as the host, otherwise if using your personal computer then enter “localhost” as the host
  - enter ece356db as the database
  - enter user_ece356_test as the user and password
  - check “remember password”
  - click “Test Connection” button, on success click “Finish” otherwise go back and verify the host, database, user and password
  - right-click on new connection, click “Connect” and then “Execute Command…”

Setting up MySQL

- Step 6: execute some SQL statements use the command window in NetBeans (press Ctrl + Shift + E to run each statement):
  - SHOW TABLES;
  - SHOW GRANTS;
  - SELECT * FROM Persons;
  - INSERT INTO Persons VALUES (0, "Homer", "Simpson"), (1, "Marge", "Simpson"), (2, "Mr", "Burns");
  - SELECT * FROM Persons;
  - SELECT FirstName FROM Persons WHERE ID = 0;

Setting up MySQL

Instructions for Windows

- Step 1: provision a Windows box and obtain administrator access.
- Step 2: download and install MySQL Community Server.
  - obtain MSI installer from http://dev.mysql.com/downloads/mysql/
  - after you click the “Download” button you may be asked to log in or sign up, but note that you can bypass this step by clicking “No thanks, just start my download” near the bottom of the page
  - choose “Custom” setup and select the 32-bit or 64-bit product
  - follow the installation wizard to configure the Windows firewall if planning remote access, and enter the root password
  - after the installer exits open a Windows command prompt and locate the MySQL shell (*e.g.*, C:\Program Files\MySQL\MySQL Server 5.6\bin\mysql.exe)
- Repeat steps 3 and onward from the Linux instructions.

The Relational Model

- Tuples and relations
- Keys
- Schemas
- Schema diagrams

- A relation is a mathematical object, and a table is its physical embodiment.
  - $R \subseteq S_1 \times S_2 \times S_3 \ldots \times S_n$
- where each S_i is a set
  - at the base level, sets are the domain of attributes
- In an RDB, relations are tables with **rows** and **columns**. The rows may represent entities or relationships, and the columns represent attributes.

Attribute Types

- The set of allowed values for each attribute is called the **domain** of the attribute.
- Attribute values are (usually) required to be **atomic**, which means they are indivisible.
  - example: a database designer may insist that the first name and the last name be separated into distinct attributes
- The special value **null** is a member of every domain, and is used to represent missing or unknown data.
- Use null values judiciously because they lead to a number of complications. (More on this later on.)

Relation Schema and Instance

- Let $A_1, A_2, \ldots, A_n$ denote attributes.
- Let $D_1, D_2, \ldots. D_n$ denote their domains.
- $R = (A_1, A_2, \ldots, A_n)$ denotes a **relation schema** over these attributes

  Example:

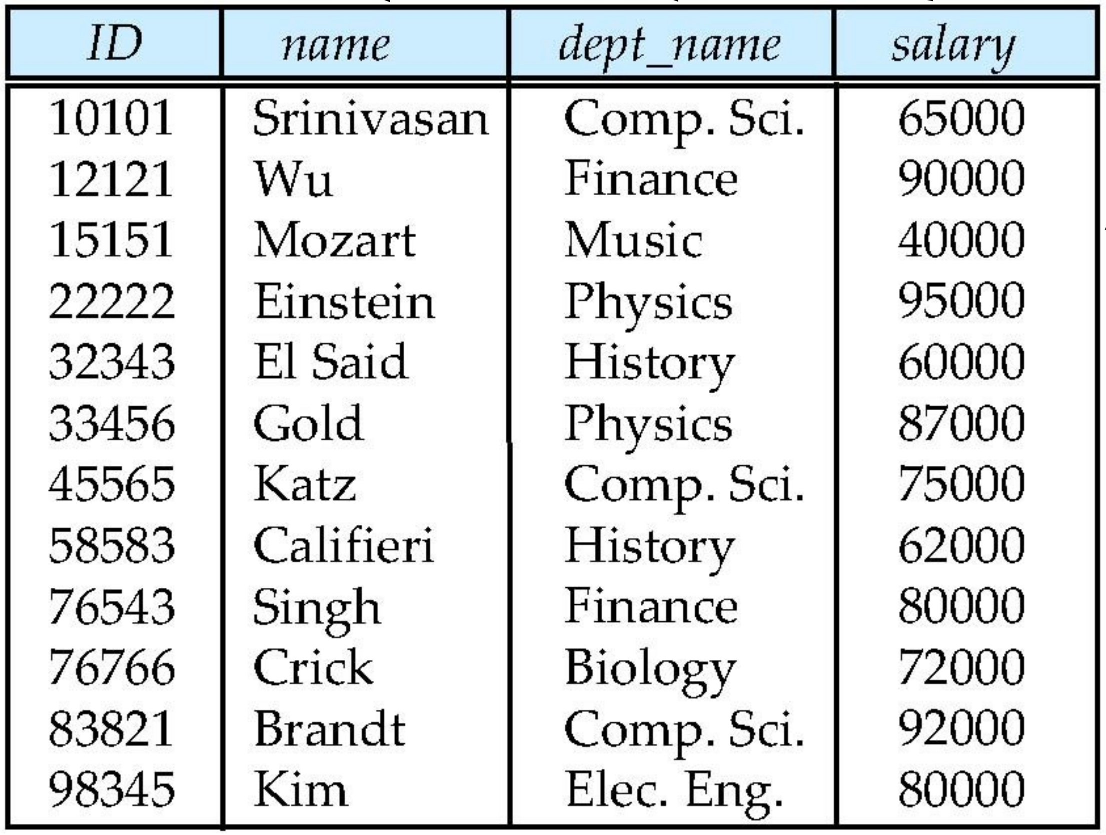
  instructor = (*ID, name, dept_name, salary*)
- A **relation *r*** conforming to schema *R*, denoted as *r*(*R*), is a subset of

  $D_1 \times D_2 \times \ldots \times D_n$

  Thus, a relation is a set of *n*-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$.
- An element ***t*** of ***r*** is a **tuple** (specifically an **n-tuple**), and corresponds to a row in a table.
- Note: the order of elements in the tuple does not matter as long as we remember the attribute corresponding to each tuple element.
- A **relation instance** refers to the concrete values of a relation. (E.g., set of Waterloo instructors as of 10am on January 9, 2014.)

Example of a Relation Instance

attributes
(or columns)



ID	*name*	*dept_name*	*salary*
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

tuples
(or rows)

Relations are Unordered

- Order of tuples is irrelevant (tuples may be listed in an arbitrary order).
- Example: *instructor* relation with tuples ordered arbitrarily.

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

Database

- A database typically comprises many relations.
- In the design process, information about an enterprise is broken up:

  instructor
  student
  advisor

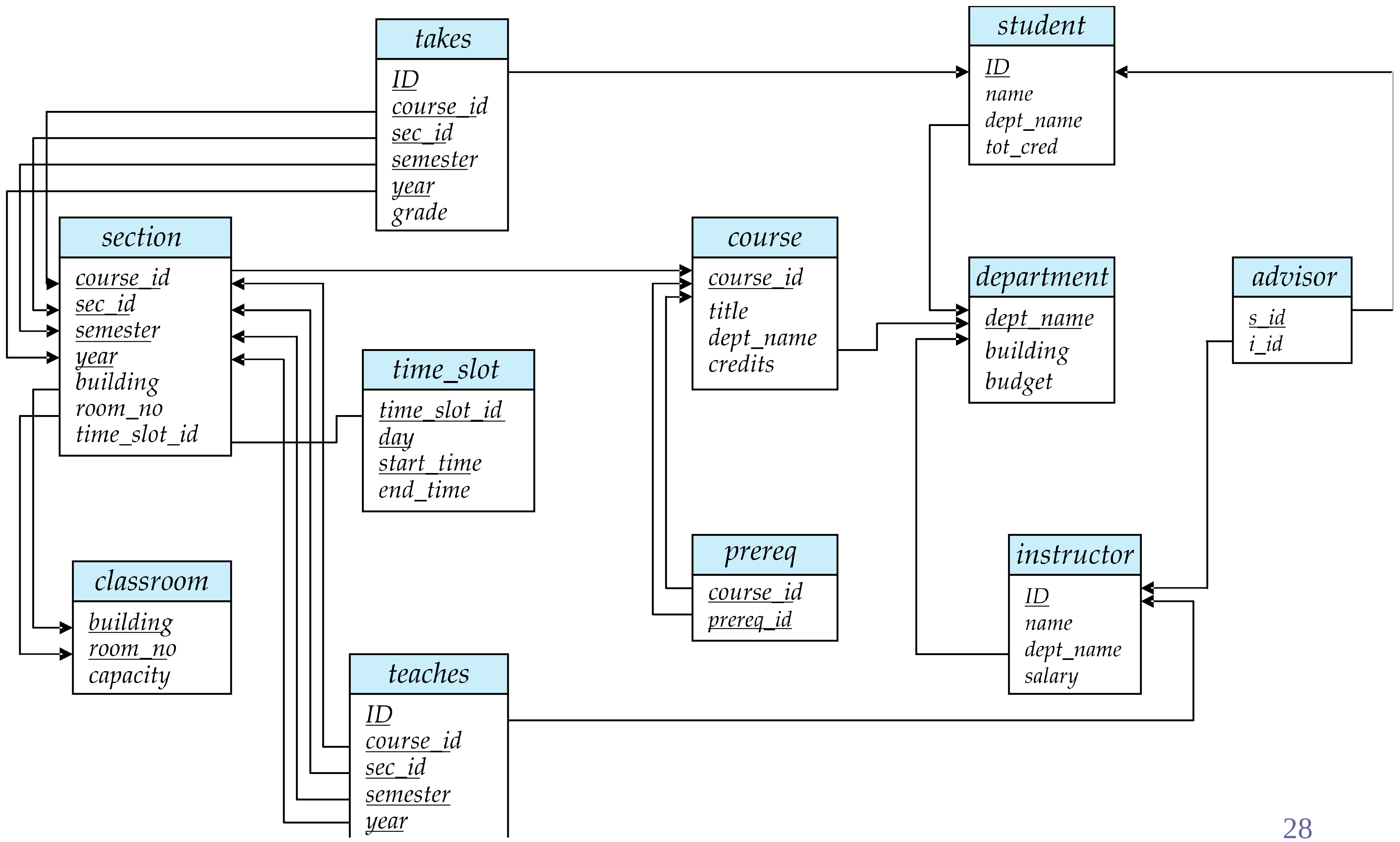
- Sometimes, database designers make questionable decisions:
  univ (*instructor_ID, name, dept_name, salary, student_ID*, ..)
  - repetition of information (e.g., two students have the same instructor)
  - the need for null values (e.g., represent a student with no advisor)
- Normalization theory (covered later in the course) deals with how to design good relational schemas that satisfy a very precise notion of "goodness".

Keys

- Let R be a relation schema and let $K \subseteq R$ (K is a subset of R's attributes).
- **Superkey** and **candidate key** are defined as in the E-R model:
  - K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation *r(R)*
    - Example: {*ID*} and {*ID*, *name*} are both superkeys of *instructor.*
  - Superkey K is a **candidate key** if K is minimal
    Example: {*ID*} is a candidate key for *instructor*
  - One of the candidate keys is selected to be the **primary key**.
    (Which one?)
- **Foreign key** constraint (new idea in the relational model):
  an attribute value in one relation that must appear in another relation.
  - **referencing relation** contains a **foreign key**
  - **referenced relation** contains a **referenced key**
    (usually the primary key)

Relational Schema Diagram for University DB

Example: Attribute *ID* in *takes* (referencing relation) is a foreign key that references *ID* in *student* (referenced relation).



takes
ID
course_id
sec_id
semester
year
grade
student
ID
name
dept_name
tot_cred
section
course_id
sec_id
semester
year
building
room_no
time_slot_id
course
course_id
title
dept_name
credits
department
dept_name
building
budget
advisor
s_id
i_id
time_slot
time_slot_id
day
start_time
end_time
classroom
building
room_no
capacity
prereq
course_id
prereq_id
instructor
ID
name
dept_name
salary
teaches
ID
course_id
sec_id
semester
year