



WQD7007 Big Data Management

Traditional Database

Database Management System (DBMS)

- DBMS contains information about a particular enterprise
 - Collection of interrelated data
 - Set of programs to access the data
 - An environment that is both *convenient* and *efficient* to use
- Database Applications:
 - Banking: transactions
 - Airlines: reservations, schedules
 - Universities: registration, grades
- Databases can be very large.
- Databases touch all aspects of our lives

University Database Example

- Application program examples
 - Add new students, instructors, and courses
 - Register students for courses, and generate class rosters
 - Assign grades to students, compute grade point averages (GPA) and generate transcripts
- In the early days, database applications were built directly on top of file systems

Drawbacks of using file systems to store data

- Data redundancy and inconsistency
 - Multiple file formats, duplication of information in different files
- Difficulty in accessing data
 - Need to write a new program to carry out each new task
- Data isolation
 - Multiple files and formats
- Integrity problems
 - Integrity constraints (e.g., $\text{account balance} > 0$) become “buried” in program code rather than being stated explicitly
 - Hard to add new constraints or change existing ones

Drawbacks of using file systems to store data

- Atomicity of updates
 - Failures may leave database in an inconsistent state with partial updates carried out
 - Example: Transfer of funds from one account to another should either complete or not happen at all
- Concurrent access by multiple users
 - Concurrent access needed for performance
 - Uncontrolled concurrent accesses can lead to inconsistencies
 - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
- Security problems
 - Hard to provide user access to some, but not all, data

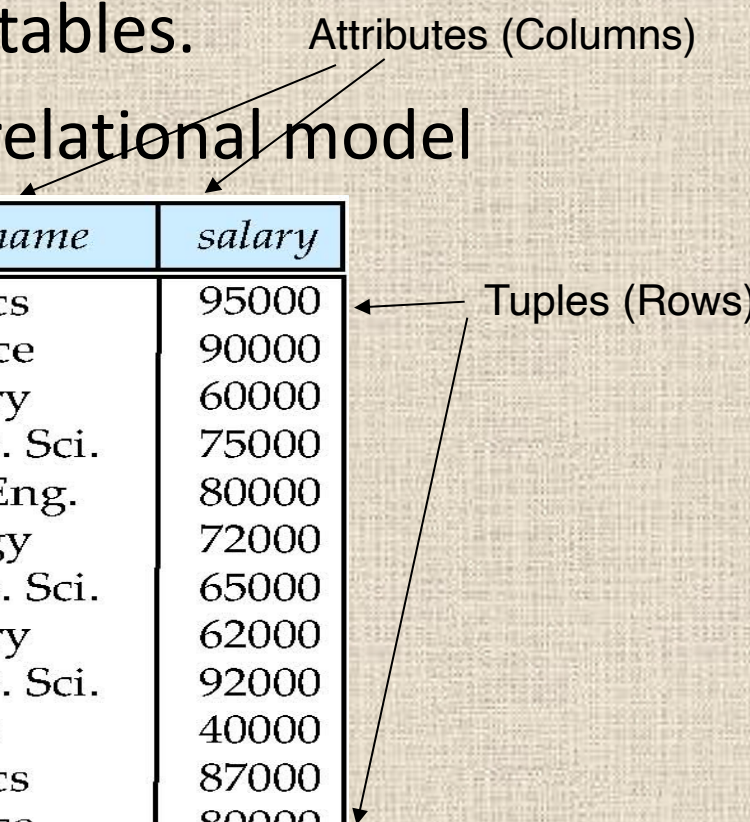
Database systems offer solutions to all the above problems

Data Models

- A collection of tools for describing
 - Data; Data relationships; Data semantics; Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semistructured data model (XML)
- Other older models:
 - Network model
 - Hierarchical model

Relational model

- All the data is stored in various tables.
- Example of tabular data in the relational model



| <i>ID</i> | <i>name</i> | <i>dept_name</i> | <i>salary</i> |
|-----------|-------------|------------------|---------------|
| 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 |

(a) The *instructor* table

A Sample Relational Database

| <i>ID</i> | <i>name</i> | <i>dept_name</i> | <i>salary</i> |
|-----------|-------------|------------------|---------------|
| 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 |

(a) The *instructor* table

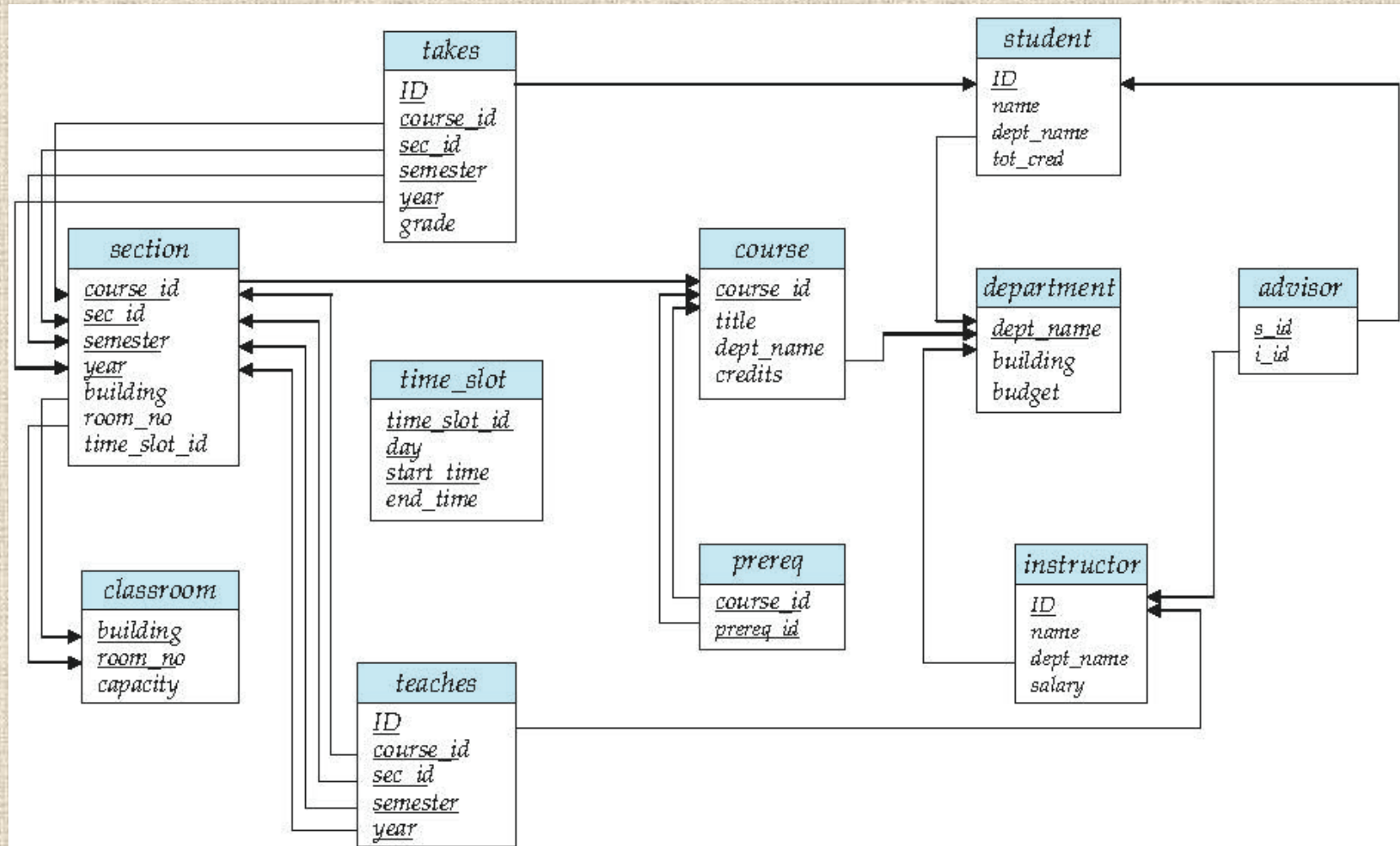
| <i>dept_name</i> | <i>building</i> | <i>budget</i> |
|------------------|-----------------|---------------|
| Comp. Sci. | Taylor | 100000 |
| Biology | Watson | 90000 |
| Elec. Eng. | Taylor | 85000 |
| Music | Packard | 80000 |
| Finance | Painter | 120000 |
| History | Painter | 50000 |
| Physics | Watson | 70000 |

(b) The *department* table

Keys

- Let $K \subseteq R$
- 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: Value in one relation must appear in another
 - Referencing relation
 - Referenced relation
 - Example – *dept_name* in *instructor* is a foreign key from *instructor* referencing *department*

Schema Diagram for University Database



SQL: Create Table Construct

- An SQL relation is defined using the create table command:

```
create table  $r$  ( $A_1 D_1, A_2 D_2, \dots, A_n D_n,$   
                (integrity-constraint1),  
                ...,  
                (integrity-constraintk))
```

- r is the name of the relation
- each A_i is an attribute name in the schema of relation r
- D_i is the data type of values in the domain of attribute A_i
- Example:

```
create table instructor (  
    ID                char(5),  
    name              varchar(20),  
    dept_name         varchar(20),  
    salary           numeric(8,2))
```


Updates to tables

- **Insert**
 - insert into *instructor* values ('10211', 'Smith', 'Biology', 66000);
- **Delete**
 - Remove all tuples from the *student* relation
 - delete from *student*
- **Drop Table**
 - drop table *r*
- **Alter**
 - alter table *r* add *A D*
 - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*.
 - All exiting tuples in the relation are assigned *null* as the value for the new attribute.
 - alter table *r* drop *A*
 - where *A* is the name of an attribute of relation *r*
 - Dropping of attributes not supported by many databases.

Basic Query Structure

- A typical SQL query has the form:

```
select  $A_1, A_2, \dots, A_n$   
from  $r_1, r_2, \dots, r_m$   
where  $P$ 
```

- A_i represents an attribute
 - R_i represents a relation
 - P is a predicate.
- The result of an SQL query is a relation.

The *select* clause

- The select clause lists the attributes desired in the result of a query
 - corresponds to the projection operation of the relational algebra

- Example: find the names of all instructors:

```
select name  
from instructor
```

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
 - E.g., *Name* \equiv *NAME* \equiv *name*
 - Some people use upper case wherever we use bold font.

The *where* clause

- The **where** clause specifies conditions that the result must satisfy
 - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

- Comparison results can be combined using the logical connectives and, or, and not
 - To find all instructors in Comp. Sci. dept with salary > 80000

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

- Comparisons can be applied to results of arithmetic expressions.

The *from* clause

- The from clause lists the relations involved in the query
 - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

```
select *  
from instructor, teaches
```

- generates every possible instructor – teaches pair, with all attributes from both relations.
 - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

Example Table: Cartesian Product

instructor

| 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 |

teaches

| ID | course_id | sec_id | semester | year |
|-------|-----------|--------|----------|------|
| 10101 | CS-101 | 1 | Fall | 2009 |
| 10101 | CS-315 | 1 | Spring | 2010 |
| 10101 | CS-347 | 1 | Fall | 2009 |
| 12121 | FIN-201 | 1 | Spring | 2010 |
| 15151 | MU-199 | 1 | Spring | 2010 |
| 22222 | PHY-101 | 1 | Fall | 2009 |

| Inst.ID | name | dept_name | salary | teaches.ID | course_id | sec_id | semester | year |
|---------|------------|------------|--------|------------|-----------|--------|----------|------|
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-101 | 1 | Fall | 2009 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-315 | 1 | Spring | 2010 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-347 | 1 | Fall | 2009 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 12121 | FIN-201 | 1 | Spring | 2010 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 15151 | MU-199 | 1 | Spring | 2010 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 22222 | PHY-101 | 1 | Fall | 2009 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 12121 | Wu | Finance | 90000 | 10101 | CS-101 | 1 | Fall | 2009 |
| 12121 | Wu | Finance | 90000 | 10101 | CS-315 | 1 | Spring | 2010 |
| 12121 | Wu | Finance | 90000 | 10101 | CS-347 | 1 | Fall | 2009 |
| 12121 | Wu | Finance | 90000 | 12121 | FIN-201 | 1 | Spring | 2010 |
| 12121 | Wu | Finance | 90000 | 15151 | MU-199 | 1 | Spring | 2010 |
| 12121 | Wu | Finance | 90000 | 22222 | PHY-101 | 1 | Fall | 2009 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |

Examples

- Find the names of all instructors who have taught some course and the course_id
 - **select** *name, course_id*
from *instructor, teaches*
where *instructor.ID = teaches.ID*
- Find the names of all instructors in the Art department who have taught some course and the course_id
 - **select** *name, course_id*
from *instructor, teaches*
where *instructor.ID = teaches.ID and instructor.dept_name = 'Art'*

Modification in the database

- **Deletion** of tuples from a given relation.
- **Insertion** of new tuples into a given relation
- **Updating** of values in some tuples in a given relation

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);
```


Update

- Increase salaries of instructors whose salary is over \$100,000 by 3%, and all others by a 5%
- Write two **update** statements:

```
update instructor
  set salary = salary * 1.03
  where salary > 100000;
update instructor
  set salary = salary * 1.05
  where salary <= 100000;
```
- The order is important

Advantages of Traditional File Access

- Data Security
 - Safely kept and **not easy to access** by unauthorized personnel.
- Complexity
 - Traditional filing systems are **less complex** than electronic systems, which can make it easier for untrained people to access and manipulate data.
 - Anyone can look through alphabetized filing cabinets to find a file. Locating and manipulating an electronic database information may require technical training, and user error can result in unintended alterations or data loss.

Disadvantages of Traditional File Access

- Access Time
 - It can **take minutes if not hours to locate a few files** in a large paper filing system.
 - Electronic databases allow for almost instantaneous access to information.
- Editing and Communication
 - Traditional file systems are cumbersome in that they **do not allow users to easily edit files** or send information to others.
 - Paper files often cannot be edited directly, forcing users to make new copies to update old files.
- Order of Data
 - Data can **get out of order** in traditional filing systems.
 - If someone **accidentally puts a file in the wrong place**, or takes a file out of a cabinet and forgets to put it back, it can lead to lost data or the creation of additional copies of files.

Characteristics of existing electronic database

- Clearly defined fields organized in records. Records are usually stored in **tables**. **Fields** have names, and relationships are defined between different fields.
 - Schema-on-write that requires data be validated against a schema before it can be written to disk.
 - A significant amount of requirements analysis, design, and effort up front can be involved in putting the data in clearly defined structured formats.
 - **This can increase the time before business value can be realized from the data.**
 - A design to get data from the disk and load the data into memory to be processed by applications.
 - This is an **extremely inefficient architecture when processing large volumes of data** this way.
 - The data is extremely large and the programs are small. The big component must move to the small component for processing.³⁵

Question:

- What is the major difference between traditional electronic database system (e.g. RDBMS) and Big Data database system (e.g. HDFS in Hadoop)?

Concluding Remarks

- This lecture is to give a feel how traditional database looks like, and how to retrieve and update information inside a traditional electronic database, using SQL language.
- Comparison between traditional and modern (electronic) data access, and between traditional electronic database and big data database is discussed.