Lecture 11

IELM 230 Fall 2008

Basic Definitions

- **Database:**
  - A collection of related data.
- **Data:**
  - Known facts that can be recorded and have an implicit meaning.
- **Mini-world:**
  - Some part of the real world about which data is stored in a database. For example, student grades and transcripts at a university.
- **Database Management System (DBMS):**
  - A software package/ system to facilitate the creation and maintenance of a computerized database.
- **Database System:**
  - The DBMS software together with the data itself. Sometimes, the applications are also included.

Simplified database system environment

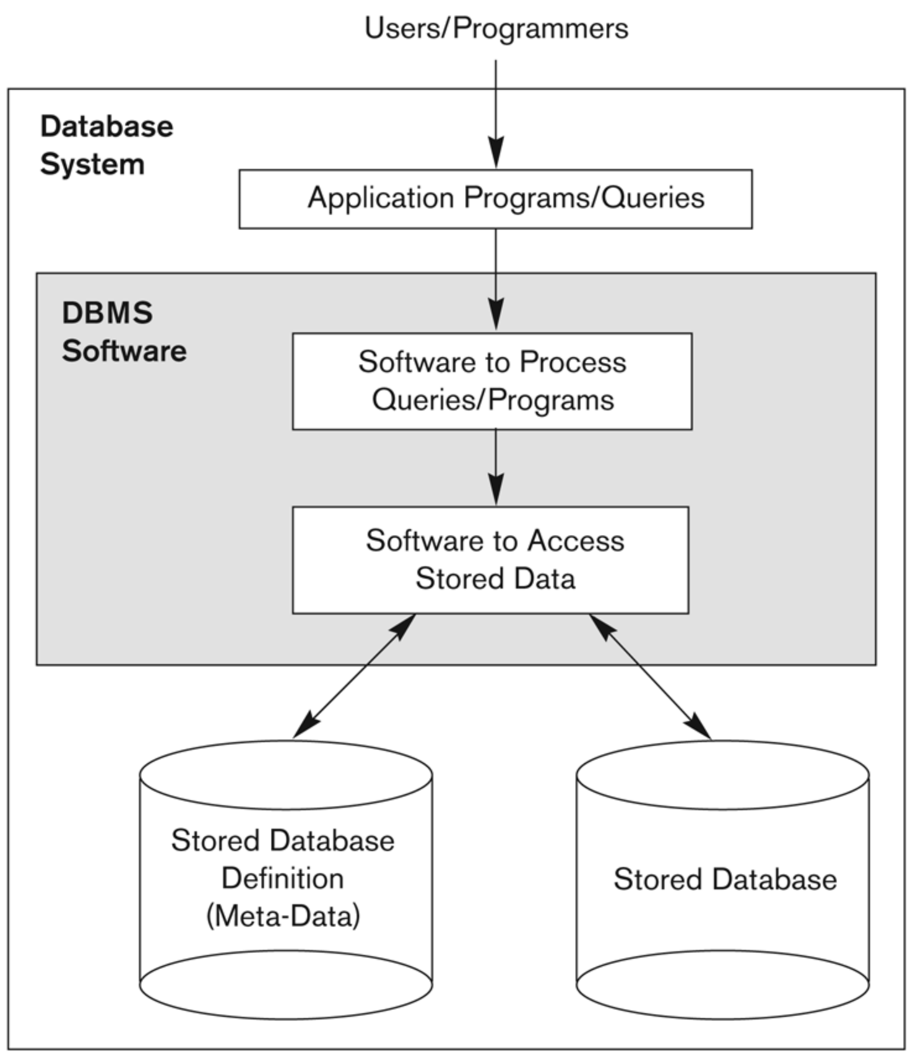

Figure 1.1
A simplified database system environment.

Typical DBMS Functionality

- *Define* a particular database in terms of its data types, structures, and constraints
- *Construct* or Load the initial database contents on a secondary storage medium
- *Manipulating* the database:
  - Retrieval: Querying, generating reports
  - Modification: Insertions, deletions and updates to its content
  - Accessing the database through Web applications
- *Processing* and *Sharing* by a set of concurrent users and application programs – yet, keeping all data valid and consistent

Typical DBMS Functionality

- Other features:
  - Protection or Security measures to prevent unauthorized access
  - Presentation and Visualization of data
  - Maintaining the database and associated programs over the lifetime of the database application

Main Characteristics of the Database Approach (continued)

- **Data Abstraction:**
  - A **data model** is used to hide storage details and present the users with a conceptual view of the database.
  - Programs refer to the data model constructs rather than data storage details
- **Support of multiple views of the data:**
  - Each user may see a different view of the database, which describes **only** the data of interest to that user.

Main Characteristics of the Database Approach (continued)

- **Sharing of data and multi-user transaction processing:**
  - Allowing a set of **concurrent users** to retrieve from and to update the database.
  - *Concurrency control* within the DBMS guarantees that each **transaction** is correctly executed or aborted
  - *Recovery* subsystem ensures each completed transaction has its effect permanently recorded in the database

Database Users

- Users may be divided into
  - Those who actually use and control the database content, and those who design, develop and maintain database applications (called "Actors on the Scene"), and
  - Those who design and develop the DBMS software and related tools, and the computer systems operators (called "Workers Behind the Scene").

Database Users

- Actors on the scene
  - **Database administrators:**
    - Responsible for authorizing access to the database, for coordinating and monitoring its use, acquiring software and hardware resources, controlling its use and monitoring efficiency of operations.
  - **Database Designers:**
    - Responsible to define the content, the structure, the constraints, and functions or transactions against the database. They must communicate with the end-users and understand their needs.

Categories of End-users

- Actors on the scene (continued)
  - **End-users:** They use the data for queries, reports and some of them update the database content. End-users can be categorized into:
    - **Casual**: access database occasionally when needed
    - **Naïve** or Parametric: they make up a large section of the end-user population.
      - They use previously well-defined functions in the form of "canned transactions" against the database.
      - Examples are bank-tellers or reservation clerks who do this activity for an entire shift of operations.

Categories of End-users (continued)

- **Sophisticated:**
  - These include business analysts, scientists, engineers, others thoroughly familiar with the system capabilities.
  - Many use tools in the form of software packages that work closely with the stored database.
- **Stand-alone:**
  - Mostly maintain personal databases using ready-to-use packaged applications.
  - An example is a tax program user that creates its own internal database.
  - Another example is a user that maintains an address book

Advantages of Using the Database Approach

- Controlling redundancy in data storage and in development and maintenance efforts.
  - Sharing of data among multiple users.
- Restricting unauthorized access to data.
- Providing persistent storage for program Objects
  - In Object-oriented DBMSs Providing Storage Structures (e.g. indexes) for efficient Query Processing

Advantages of Using the Database Approach (continued)

- Providing backup and recovery services.
- Providing multiple interfaces to different classes of users.
- Representing complex relationships among data.
- Enforcing integrity constraints on the database.
- Drawing inferences and actions from the stored data using deductive and active rules

Additional Implications of Using the Database Approach

- Potential for enforcing standards:
  - This is very crucial for the success of database applications in large organizations. **Standards** refer to data item names, display formats, screens, report structures, meta-data (description of data), Web page layouts, etc.
- Reduced application development time:
  - Incremental time to add each new application is reduced.

Additional Implications of Using the Database Approach (continued)

- Flexibility to change data structures:
  - Database structure may evolve as new requirements are defined.
- Availability of current information:
  - Extremely important for on-line transaction systems such as airline, hotel, car reservations.
- Economies of scale:
  - Wasteful overlap of resources and personnel can be avoided by consolidating data and applications across departments.

When not to use a DBMS

- Main inhibitors (costs) of using a DBMS:
  - High initial investment and possible need for additional hardware.
  - Overhead for providing generality, security, concurrency control, recovery, and integrity functions.
- When a DBMS may be unnecessary:
  - If the database and applications are simple, well defined, and not expected to change.
  - If there are stringent real-time requirements that may not be met because of DBMS overhead.
  - If access to data by multiple users is not required.

When not to use a DBMS

- When no DBMS may suffice:
  - If the database system is not able to handle the complexity of data because of modeling limitations
  - If the database users need special operations not supported by the DBMS.

Class Exercise

Student (*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(*name*: string, *meets_at*: string, *room*: string, *fid*: integer)

Enrolled(*snum*: integer, *cname*: string)

Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach

```
SELECT DISTINCT S.sname
FROM Student S, Class C, Enrolled E, Faculty F
WHERE S.snum = E.snum AND E.cname=C.name AND C.fid = F.fid AND
        F.fname = 'I. Teach' AND S.level ='JR'
```

Class Exercise

Student (*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(*name*: string, *meets_at*: string, *room*: string, *fid*: integer)

Enrolled(*snum*: integer, *cname*: string)

Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

Find the age of the oldest student who is either a History major or enrolled in a course taught by I. Teach

```
SELECT MAX(S.age)
FROM Student S
WHERE (S.major = 'History')
    OR S.snum in    (SELECT E.snum
                     FROM Class C, Enrolled E, Faculty F
                     WHERE E.cname = C.name AND C.fid = F.fid
                           AND F.fname = 'I. Teach')
```

Class Exercise

Student (*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(*name*: string, *meets_at*: string, *room*: string, *fid*: integer)

Enrolled(*snum*: integer, *cname*: string)

Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

Find the names of all classes that either meet in room R128 or have five or more students enrolled.

```
SELECT C.name
FROM Class C
WHERE  C.room= 'R128'
    OR C.name in    (SELECT E.cname
                     FROM Enrolled E
                     GROUP BY E.cname
                     HAVING COUNT(*) >=5)
```

Class Exercise

Student (*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(*name*: string, *meets_at*: string, *room*: string, *fid*: integer)

Enrolled(*snum*: integer, *cname*: string)

Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

Find the names of all students who are enrolled in two classes that meet at the same time.

```
SELECT DISTINCT S.sname
FROM Student S
WHERE  S.snum in
        (SELECT E1.snum
         FROM Enrolled E1, Enrolled E2, Class C1, Class C2
         WHERE E1.cname = C1.name AND E2.cname = C2.name AND
               E1.snum = E2.snum AND C1.name <> C2.name AND
               C1.meets_at = C2.meets_at)
```

Class Exercise

Student (<u>*snum*: integer</u>, *sname*: string, *major*: string, *level*: string, *age*: integer)
Class(<u>*name*: string</u>, *meets_at*: string, *room*: string, *fid*: integer)
Enrolled(<u>*snum*: integer, *cname*: string</u>)
Faculty(<u>*fid*: integer</u>, *fname*: string, *deptid*: integer)

Find the names of faculty members who teach in every room in which some class is taught

```
SELECT DISTINCT F.fname
FROM Faculty F
WHERE NOT EXIST (( SELECT DISTINCT C.room
                   FROM Class C

                  EXCEPT

                  SELECT DISTINCT C1.room FROM Class C1
                  WHERE C1.fid = F.fid))
```

Class Exercise

Student (*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(*name*: string, *meets_at*: string, *room*: string, *fid*: integer)

Enrolled(*snum*: integer, *cname*: string)

Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five

```
SELECT DISTINCT F.fname
FROM Faculty F
WHERE 5 > (SELECT COUNT (E.snum)
           FROM Class C, Enroll E
           WHERE C.name = E.cname AND F.fid=C.fid)
```

Class Exercise

Student (*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(*name*: string, *meets_at*: string, *room*: string, *fid*: integer)

Enrolled(*snum*: integer, *cname*: string)

Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

Print the level and the average age of students for that level, for each level

SELECT S.level, AVG (S.age)

FROM Students

GROUP BY S.level

Class Exercise

Student (*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(*name*: string, *meets_at*: string, *room*: string, *fid*: integer)

Enrolled(*snum*: integer, *cname*: string)

Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

Print the level and the average age of students for that level, for all levels except SR.

```
SELECT S.level, AVG (S.age)
FROM Students
WHERE S.level <> 'SR'
GROUP BY S.level
```

Class Exercise

Student (*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(*name*: string, *meets_at*: string, *room*: string, *fid*: integer)

Enrolled(*snum*: integer, *cname*: string)

Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught

```
SELECT    F.fname, COUNT(*) AS CourseCount
FROM      Faculty F, Class C
WHERE     F.fid = C.fid
GROUP BY  F.fid, F.fname
HAVING    EVERY ( C.room = 'R128' )
```

Class Exercise

Student (*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(*name*: string, *meets_at*: string, *room*: string, *fid*: integer)

Enrolled(*snum*: integer, *cname*: string)

Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

Find the names of students enrolled in the maximum number of

```
SELECT   DISTINCT S.sname
FROM     Student S
WHERE    S.snum IN (SELECT    E.snum
                    FROM      Enrolled E
                    GROUP BY E.snum
                   HAVING   COUNT (*) >= ALL (SELECT    COUNT (*)
                                              FROM      Enrolled E2
                                              GROUP BY E2.snum ))
```

Class Exercise

Student (<u>*snum*: integer</u>, *sname*: string, *major*: string, *level*: string, *age*: integer)

Class(<u>*name*: string</u>, *meets_at*: string, *room*: string, *fid*: integer)

Enrolled(<u>*snum*: integer, *cname*: string</u>)

Faculty(<u>*fid*: integer</u>, *fname*: string, *deptid*: integer)

Find the names of students not enrolled in any class

```
SELECT DISTINCT S.sname
FROM    Student S
WHERE   S.snum NOT IN (SELECT E.snum
                       FROM    Enrolled E )
```

Class Exercise

Suppliers(*sid*: integer, *sname*: string, *address*: string)

Parts(*pid*: integer, *pname*: string, *color*: string)

Catalog(*sid*: integer, *pid*: integer, *cost*: real)

Find the pnames of parts for which there is some supplier

```
SELECT P.pname
FROM Parts P, Catalog C
WHERE P.pid = C.pid
```

Class Exercise

Suppliers(*sid*: integer, *sname*: string, *address*: string)

Parts(*pid*: integer, *pname*: string, *color*: string)

Catalog(*sid*: integer, *pid*: integer, *cost*: real)

Find the snames of suppliers who supply every part

```
SELECT S.sname
FROM   Suppiers S
WHERE  NOT EXISTS (( SELECT *
                     FROM   Parts P )
                   EXCEPT
                   ( SELECT C.pid
                     FROM   Catalog C
                     WHERE  C.sid = S.sid ))
```

Class Exercise

Suppliers(*sid*: integer, *sname*: string, *address*: string)

Parts(*pid*: integer, *pname*: string, *color*: string)

Catalog(*sid*: integer, *pid*: integer, *cost*: real)

Find the snames of suppliers who supply every red part

```
SELECT S.sname
FROM   Suppiers S
WHERE  NOT EXISTS (( SELECT *
                     FROM   Parts P
                     WHERE  P.color = 'red' )
                    EXCEPT
                    ( SELECT C.pid
                     FROM   Catalog C, Parts P
                     WHERE  C.sid = S.sid AND
                            C.pid = P.pid AND P.color = 'red' ))
```

Class Exercise

Suppliers(<u>*sid*: integer</u>, *sname*: string, *address*: string)

Parts(<u>*pid*: integer</u>, *pname*: string, *color*: string)

Catalog(<u>*sid*: integer, *pid*: integer</u>, *cost*: real)

Find the pnames of parts supplied by Acme Widget Suppliers and no one else

```
SELECT  P.pname
FROM    Parts P, Catalog C, Suppliers S
WHERE   P.pid = C.pid AND C.sid = S.sid
AND     S1.sname = 'Acme Widget Suppliers'
AND     NOT EXISTS ( SELECT *
                     FROM   Catalog C1, Suppliers S1
                     WHERE  P.pid = C1.pid AND C1.sid = S1.sid AND
                            S1.sname <> 'Acme Widget Suppliers' )
```

Class Exercise

Suppliers(*sid*: integer, *sname*: string, *address*: string)

Parts(*pid*: integer, *pname*: string, *color*: string)

Catalog(*sid*: integer, *pid*: integer, *cost*: real)

For each part, find the sname of the supplier who charges the most for that part

```
SELECT  P.pid, S.sname
FROM    Parts P, Suppliers S, Catalog C
WHERE   C.pid = P.pid
AND     C.sid = S.sid
AND     C.cost = (SELECT MAX (C1.cost)
                  FROM   Catalog C1
                  WHERE  C1.pid = P.pid)
```

Class Exercise

Suppliers(<u>*sid*: integer</u>, *sname*: string, *address*: string)

Parts(<u>*pid*: integer</u>, *pname*: string, *color*: string)

Catalog(<u>*sid*: integer, *pid*: integer</u>, *cost*: real)

Find the sids of suppliers who supply only red parts

```
SELECT DISTINCT C.sid
FROM   Catalog C
WHERE  NOT EXISTS ( SELECT *
                    FROM   Parts P
                    WHERE  P.pid = C.pid AND P.color <> 'red' )
```

Class Exercise

Suppliers(<u>*sid*: integer</u>, *sname*: string, *address*: string)

Parts(<u>*pid*: integer</u>, *pname*: string, *color*: string)

Catalog(<u>*sid*: integer, *pid*: integer</u>, *cost*: real)

Find the sids of suppliers who supply a red part and a green part

```
SELECT DISTINCT C.sid
FROM   Catalog C, Parts P
WHERE  C.pid = P.pid AND P.color = 'red'
INTERSECT
SELECT DISTINCT C1.sid
FROM   Catalog C1, Parts P1
WHERE  C1.pid = P1.pid AND P1.color = 'green'
```

Class Exercise

Suppliers(<u>*sid*: integer</u>, *sname*: string, *address*: string)

Parts(<u>*pid*: integer</u>, *pname*: string, *color*: string)

Catalog(<u>*sid*: integer, *pid*: integer</u>, *cost*: real)

Find the sids of suppliers who supply a red part or a green part

```
SELECT DISTINCT C.sid
FROM    Catalog C, Parts P
WHERE   C.pid = P.pid AND P.color = 'red'
UNION
SELECT DISTINCT C1.sid
FROM    Catalog C1, Parts P1
WHERE   C1.pid = P1.pid AND P1.color = 'green'
```