Why Databases?

Mahmoud El-Haj

Why study database?

- Job market requires database admins (Oracle).
- Databases are everywhere (i.e. a lot of job opportunities)
- Who and where:
  - Google, Facebook, Amazon, …
  - Clinics, supermarkets, payroll systems
  - Universities and companies students and employees records.
  - Stock Market and financial sectors.
  - Government
  - Science (e.g. data)
- What do we store in databases?
  - Databases for search engines (google has 2.5 million servers)
  - Database of emails (gmail, outlook, ...etc)
  - Database of publications (google scholar)
- Issues:
  - Privacy and protection against cyberattacks (cybersecurity).
- Efficiency:
  - How to quickly make transactions (e.g. Bank ATMs)
  - Find what you are looking for on Amazon and Ebay

Where would a database be useful?

Example: hotels

What do I need to know?

- How to create databases
- How to efficiently query them
- How to keep them secure and up to date

EXAMPLE: HOW WOULD I CREATE A DATABASE FROM SCRATCH?

Enter...

THE EMPLOYEE RECORD SYSTEM

...because every Database course must have one

Simple Employee Record System

Name	Age	Gender
Stark	32	M

```
Employee {
    char   Name[10];
    int    Age;
    char   Gender;
}
```

Assumed layout

S	T	A	R	K	Ø	Ø	Ø	Ø	Ø
32	0	0	0						
M									

Int could be of 4 bytes
(8 bits each, simply one unsigned char of size 0-255)
256*256*256*256 = 4294967295
If signed (eg MS Access) (-127 to 127) we get -2147483647 to 2147483647

Simple Employee Record System

Name	Age	Gender
Stark	32	M

```
Employee {
    char   Name[10];
    int    Age;
    char   Gender;
}
```

Assumed layout

S	T	A	R	K	Ø	Ø	Ø	Ø	Ø
32	0	0	0						
M									

Linear file/ memory

S	T	A	R	K	Ø	Ø	Ø	Ø	Ø	32	0	0	0	M

Simple Employee Record System

Name	Age	Gender
Stark	32	M
Lannister	28	F

```
Employee {
    char   Name[10];
    int    Age;
    char   Gender;
}
```

Hint: 15 + 4 + 1 = 15 bytes

Simple Employee Record System

- We have a ***Relational*** file
  - Number of records
  - Each record has a defined set of related attributes

Employee														
Name										Age				Gender
T	Y	R	E	L	L	Ø	Ø	Ø	Ø	79	0	0	0	M
T	U	L	L	Y	Ø	Ø	Ø	Ø	Ø	24	0	0	0	F
S	T	A	R	K	Ø	Ø	Ø	Ø	Ø	32	0	0	0	M
L	A	N	N	I	S	T	E	R	Ø	28	0	0	0	F

Single record (15 bytes)

Using the Relational File

- What would happen if someone comes with:

1. A name having more than 10 characters (e.g. Hetherspoon**)**?
   - We’ll worry about this one later!

2. An idea for a data processing task…
   a) Using the data as it stands?
   b) Needing additional employee data not currently stored (e.g. marital status)?

Using the Relational File, problem 2a:

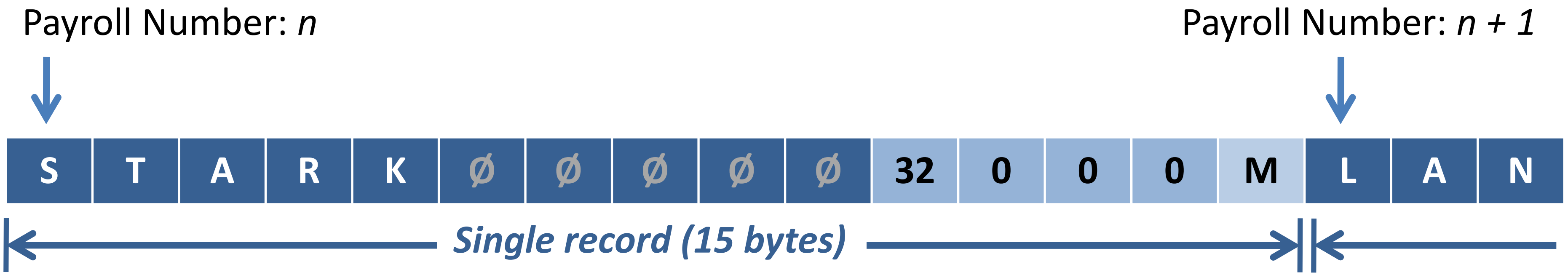
... a data processing task using the data as it stands

Accessing the Relational File

- Programmer must
  - Understand format of employee file, and…
  - Either
    a) Obtain file access routines (sequence of code) from existing program
       - Understand what code does
       - Include code in new design
    b) Write own routines that correctly handle file access

Accessing the Relational File

- Remember each record is of known length, so
  - We can easily calculate the position of each employee's record in the file



Accessing the Relational File

- Using existing code

```
Employee e = getEmployee ( payrollNumber );
```

- Writing new code

```
const int RECORD_SIZE = sizeOf ( Employee )

Employee getEmployee ( int payrollNumber ) {
   DBfile.moveto ( payrollNumber * RECORD_SIZE );
   return (Employee) DBfile.readBytes ( RECORD_SIZE );
}
```

sizeOf (Employee)

→ 15 Bytes

Pseudo code, convert to your preferred language

Accessing the Relational File

- The more application(s) that are required, the more times this will be done
- More than likely, file access routines will (eventually) be abstracted out
- Unfortunately, by this time multiple versions will likely exist in/ for each application

Using the Relational File, problem 2b:

... a data processing task needing additional employee data not currently stored

Adding Employee Marital Status

- Alter main file to include new information
  - This means the format of the file – as known to every other application – will change (records of fixed size of 15 bytes)
  - All existing applications will now fail!

Adding Employee Marital Status

Employee															
Name										Age				G	S
T	Y	R	E	L	L	ø	ø	ø	ø	79	0	0	0	M	M
T	U	L	L	Y	ø	ø	ø	ø	ø	24	0	0	0	F	S
S	T	A	R	K	ø	ø	ø	ø	ø	32	0	0	0	M	S
L	A	N	N	I	S	T	E	R	ø	28	0	0	0	F	D

Single record (16 bytes)

Adding Employee Marital Status

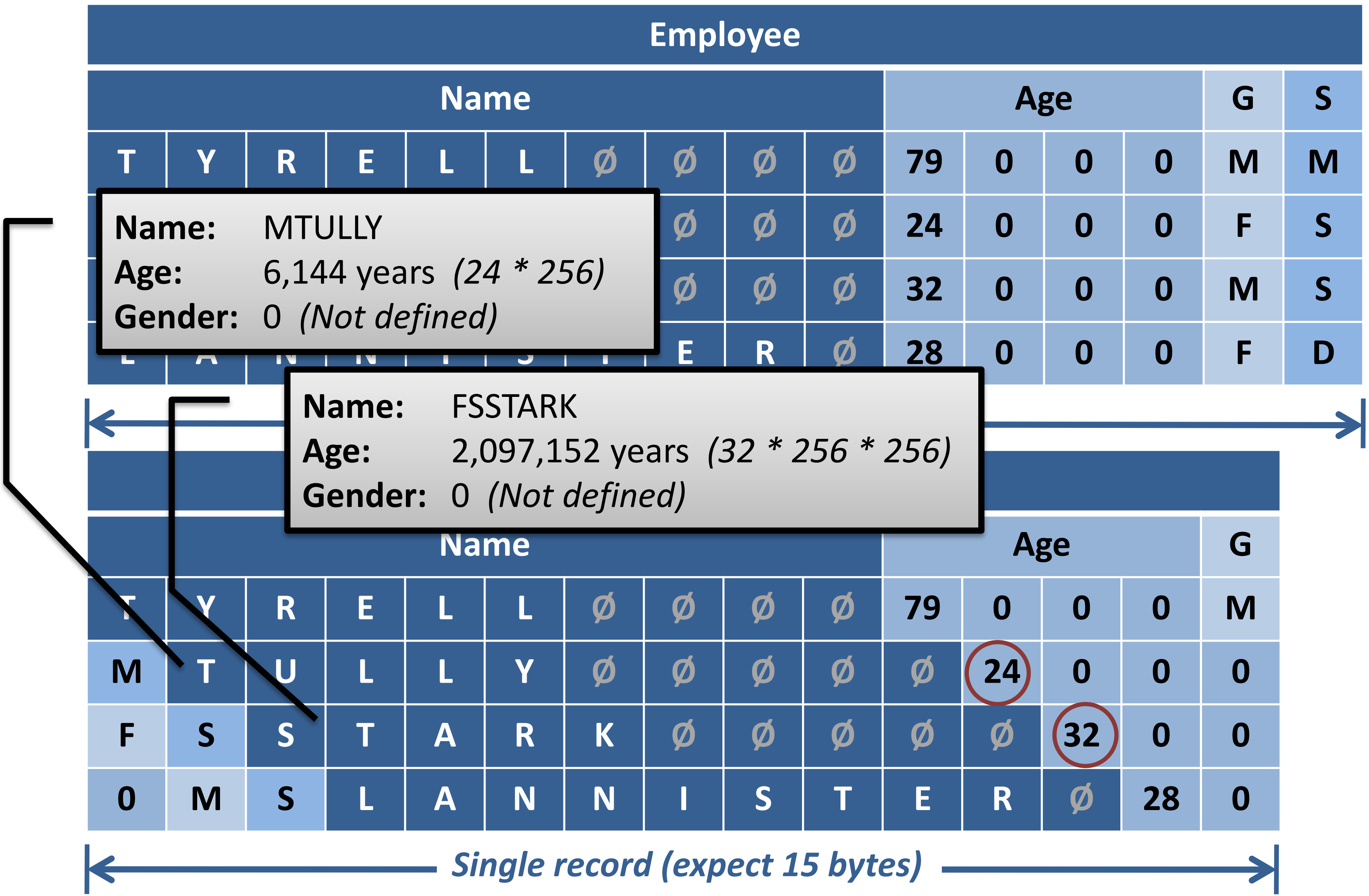
Employee															
Name										Age				G	S
T	Y	R	E	L	L	Ø	Ø	Ø	Ø	79	0	0	0	M	M
T	U	L	L	Y	Ø	Ø	Ø	Ø	Ø	24	0	0	0	F	S
S	T	A	R	K	Ø	Ø	Ø	Ø	Ø	32	0	0	0	M	S
L	A	N	N	I	S	T	E	R	Ø	28	0	0	0	F	D

Single record (16 bytes)

Employee *(as read by old code)*														
Name										Age				G
T	Y	R	E	L	L	Ø	Ø	Ø	Ø	79	0	0	0	M
M	T	U	L	L	Y	Ø	Ø	Ø	Ø	Ø	24	0	0	0
F	S	S	T	A	R	K	Ø	Ø	Ø	Ø	Ø	32	0	0
0	M	S	L	A	N	N	I	S	T	E	R	Ø	28	0

Single record (expect 15 bytes)

Adding Employee Marital Status



Test yourself

What are main problems of using a relational file as a database?

That didn’t work too well...

AN ALTERNATIVE APPROACH

Adding Marital Status

- An alternative approach would be to create a new file for the new data
  - Marital Status, Number of Dependent Children

Employee														
Name										Age				G
T	Y	R	E	L	L	Ø	Ø	Ø	Ø	79	0	0	0	M
T	U	L	L	Y	Ø	Ø	Ø	Ø	Ø	24	0	0	0	F
S	T	A	R	K	Ø	Ø	Ø	Ø	Ø	32	0	0	0	M
L	A	N	N	I	S	T	E	R	Ø	28	0	0	0	F

Status	
S	D
M	2
S	1
S	0
D	1

Problem with Multiple Files

- Two files to maintain
- Performance penalty accessing multiple files
- Old code has no knowledge of new file
  - Not a problem for reading/ simple updates
  - Danger of data duplication
    - Different programmers may independently create their own ‘marital status’ file
  - Adding/ deleting records is a problem

Major Updates Across Relational Files

- Tyrell has no further involvement with the company
- His record is deleted using the original application
  - For that read: old code
- May be ok until we compact the files...

Employee														
Name										Age				G
~~T~~	~~Y~~	~~R~~	~~E~~	~~L~~	~~L~~	Ø	Ø	Ø	Ø	~~79~~	~~0~~	~~0~~	~~0~~	~~M~~
T	U	L	L	Y	Ø	Ø	Ø	Ø	Ø	24	0	0	0	F
S	T	A	R	K	Ø	Ø	Ø	Ø	Ø	32	0	0	0	M
L	A	N	N	I	S	T	E	R	Ø	28	0	0	0	F

Status	
S	D
M	2
S	1
S	0
D	1

…there'll be trouble!

- Tully now finds herself married with an extra child
- Stark is still single but discovers he has a child
- Lannister goes from being divorced with one child to single with no children

  …and we believe someone is divorced with one child

Employee														
Name										Age				G
T	U	L	L	Y	Ø	Ø	Ø	Ø	Ø	24	0	0	0	F
S	T	A	R	K	Ø	Ø	Ø	Ø	Ø	32	0	0	0	M
L	A	N	N	I	S	T	E	R	Ø	28	0	0	0	F
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	0	0	0	0	Ø

Status	
S	D
M	2
S	1
S	0
D	1

Test yourself

So what are main problems with multiple files then?

Data Base Management System (DBMS)

instead of a

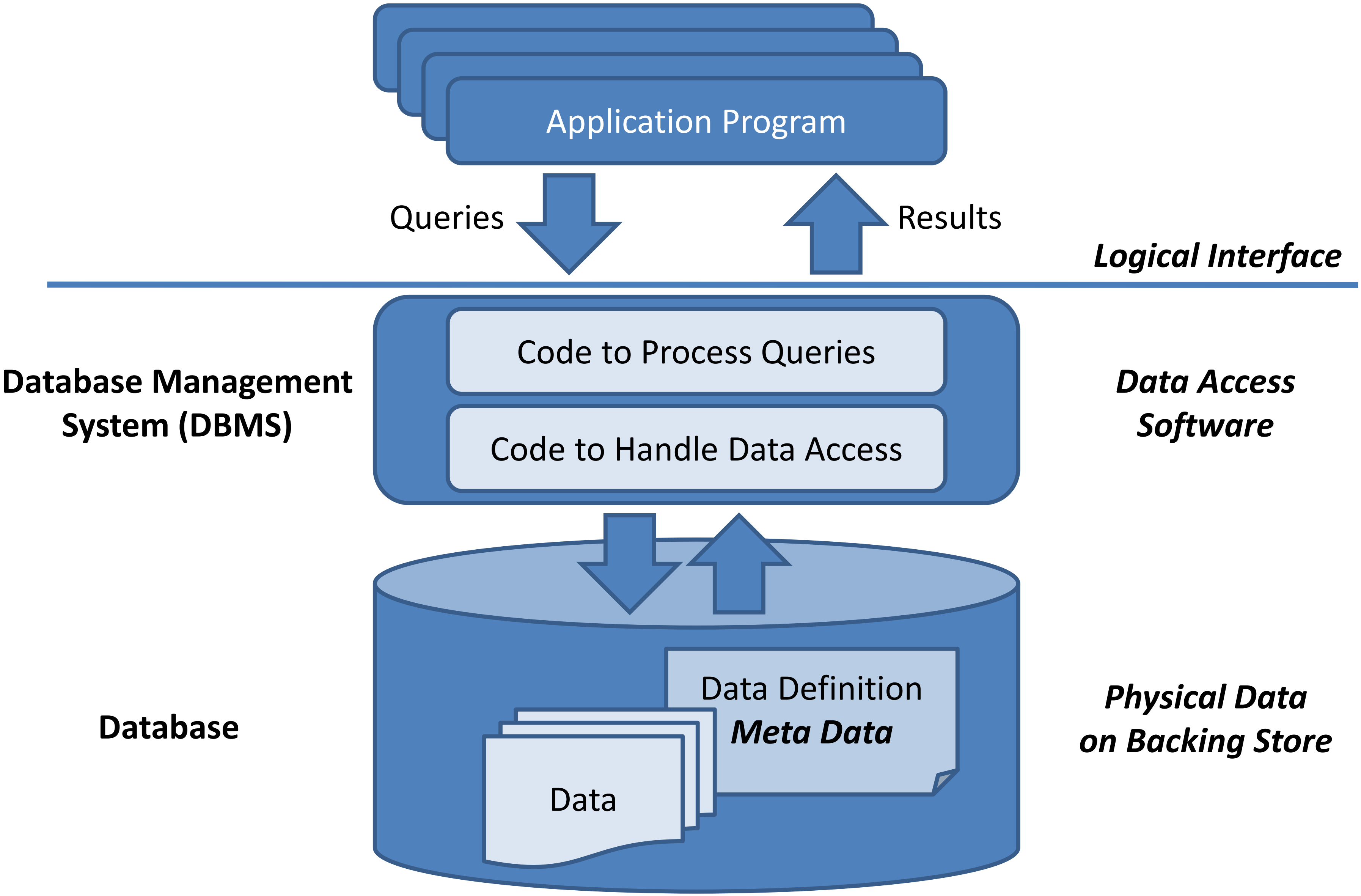
File System

Enter the Database

- We need to abstract
  - Data structure
  - Data values
  - Data access

  from the application software

Database Management System



Data Independence

- With DBMS, we have two views of data
  - Logical view, as presented to applications
  - Physical layout on backing store, as manipulated by the DBMS

- If someone wishes to add new information to a data set, this can be added to the physical layout with no ill effects to existing applications
  (as long as the logical view remains consistent)

- The logical level has to be able to access the new information without upsetting existing applications
  - This can be done by allowing applications to define the subset of the logical view they wish to see

Another question to think about...

- In what we’ve sketched out so far, using simple relational files, we can access information about an employee if we know their payroll number

- What happens if we don’t?

Test yourself

- What happens if we want to find information about Targaryen?

name	age	gender
Tyrell	26	M
Tully	32	F
Stark	43	M
Lannister	54	F
Baratheon	23	M
Targaryen	19	M

If this is a file system set-up, we only really have "brute force" option:

- Start with first record,
  check for "Targaryen", if not...
- Get second record,
  check for "Targaryen", if not...
- Get third record ...

Imagine having hundreds of thousands of records...
How long might it take?

Some organisations have gigabytes of data… 1,073,741,824 bytes, equal to 10243, or 2^30 bytes
...a thousand million bytes

There are much smarter ways
of storing/ organising these records
So we can access them much faster.
We will study these methods in the "Physical Models" part of this course.

Data Independence

- The “subtle” point of Data Independence is that we can organise and reorganise both…

  the **contents** and the **structure** of the data

  – As long as the logical view presented to the application programmer and end-user remains consistent

- This is one of our arguments as to why we **should use databases instead of files:**
  1. Data independence
  2. “Clever” access methods/physical-models that speed up access times even over very large amounts of data

Data Abstraction

- Database systems provide data independence through data abstraction.
- Data abstraction:
  - is a conceptual representation of the data
  - does not include any details of how data is stored
  - uses logical concepts such as **objects**, their **properties** and **inter-relationships**
  - hides storage and implementation details of no interest to most database users
  - offers application software a consistent (logical) interface
    - Data Access Software in the DBMS uses the ‘most efficient’ access mechanisms to manipulate the physical data

Data Abstraction 2

- Data abstraction in database systems presents a logical view of data to applications. The physical layout on the store is manipulated by the DBMS
  - If someone wishes to add new information, this can be added to the physical layout with no ill effects to existing applications (as long as the logical view remains consistent) [1]

- Database systems are self-describing
  - Unlike traditional file processing, data definition is not part of the application program
  - The data definitions, storage structure of data items, and constraints are stored in the **system catalogue**

[1] *Note this relates back to Data Independence*

Data Abstraction

- In traditional file processing users define and implement the files needed for each specific application
  - The examination office application keeps a file on **students** and their **grades**
  - The cashier's office application keeps a file on **students**, their **fees** and **payments**
- Both offices are interested in data about (the same) students
- Maintain separate files (and programs to manipulate them) as each office requires data not available from the other user's files results in:
  - Wasted storage space
  - Redundant effort to keep common data consistent

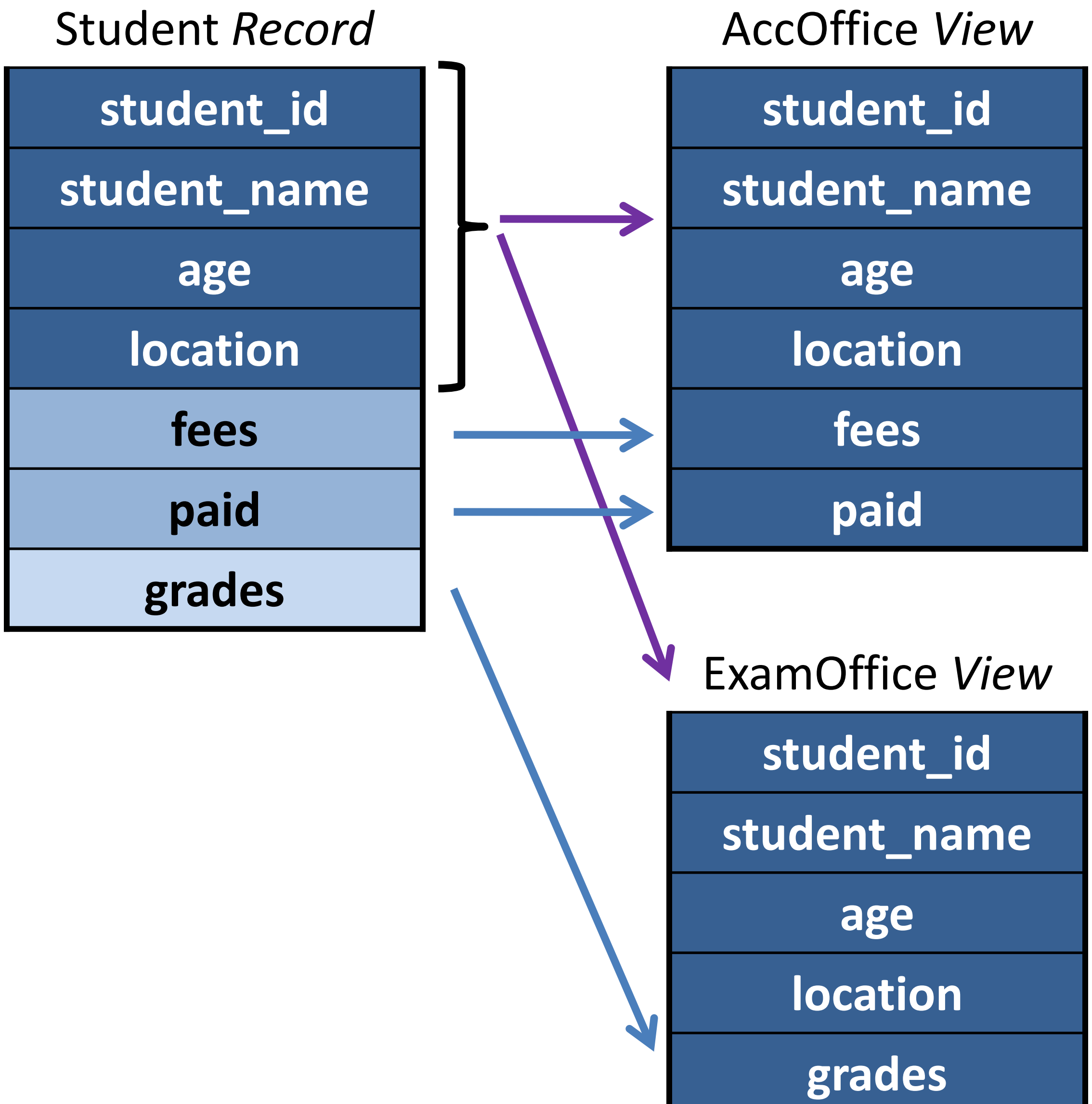
Multi-Office Example

- We have multiple copies of the same information
  - And the same update consistency problem as before

```
ExamOfficeRecord {

    int    student_id;
    String student_name;
    int    age;
    String location;

    int grades[10];

}
```

```
AccOfficeRecord {

    int    student_id;
    String student_name;
    int    age;
    String location;

    float fees[10];
    float paid[10];
}
```

Introducing Views



- **Each view contains**
  - only what each application wants to see; and
  - only what it needs to see, keeping rest hidden (security/ privacy)
- **Views may be virtual i.e. not actually stored**
  - No wasted space
  - No redundant effort maintaining common data
- ***More details on views later in the course...***

Multiple Concurrent Access

- In traditional file processing, support for multiple users accessing the same data needs to be built into the application
- A DBMS offers **concurrency control** mechanisms:
  - To ensure that several users trying to update the same data do so in a controlled fashion so that the results of the updates are correct (e.g. several reservation clerks trying to reserve the same seat)
- Systems requiring **concurrent transactions** are known as **on-line transaction processing (OLTP)** systems (e.g. Bank’s ATM)

Other Features of a DBMS

- Restricting unauthorised access (security)
- Multiple user interfaces
  - Query languages
  - Application programming interfaces (APIs)
  - GUIs
  - Natural language interfaces
  - WWW access
- Representing complex relationships among data
- Enforcing integrity constraints
  - e.g. a *uniqueness* constraint for course codes or student registration numbers
- Backup and recovery

Test yourself

Main advantages of DBMS when compared to using files?

Terminology

- Entity
  - a specific object, real or abstract, about which we need to store attributes and relationships:
    - For example, a person, car, company, etc.

- Attributes
  - Properties describing an entity
    - For example, a student entity may be described by name, age, registration number, major, etc.

- Relationships
  - Associations among entities, for example, ...
    - Person X - ***is married to*** - person Y
    - Person P - ***works for*** - company C
      etc.

Record

- Most common data aggregation mechanism

- A collection of related data values or items
  - Usually describes entities and their attributes
  - a **record type** is a collection of field names and their corresponding data types

    For example, **record Person:**
    string *name*, integer *age*, string *NI_Number*
  - a person record – an instance of person – will be:
    ("Amy Smith", 42, "AA665544X")

Schema

- A database will hold many different kinds of records, some of which will be related in some way. Most DBMSs support the definition of a schema, which allows a user to:
  - declare the **types of records** required (e.g. Person, Company, etc.)
  - declare the **types of relationships** among records (e.g. person works for a company, company employs a person, etc.)
- A schema is a "roadmap" for the data eventually stored (sometimes referred to as meta-data: information about data; although this terminology is not so common now)
- We will examine the ANSI-SPARC Three Layer Schema Architecture later in this course

Will we try this out?

- Yes of course!
- Later in the labs (starting from week 13), you'll get the chance to:
- Design a database schema and declare the type of records (i.e. students or employees)
- Declare the relationship between records
- Insert data into records (e.g. 1234, Salma, 24, computer science)
- This will help you understand the terminologies and put into practice the theory you learn in the lecture.

Overhead costs of a DBMS

- High initial investment in
  - Hardware, Software, Training

- General approach for defining and processing data

- Overhead for providing
  - Security, Concurrency control,
  - Recovery and Integrity functions

When not to use a DBMS?

- May be more desirable to use traditional files if :
  - The database and applications are simple, well-defined and not expected to change
  - There are strict real-time requirements for some programs that may not be met because of DBMS overheads
  - Multiple-user access to data is not required

So, what is a Database?

- Represents some aspect of real- or mini- world

  ...also called the *Universe of Discourse*

  – Changes to mini-world are reflected in database
    - For example: a university database,
      an airline reservation database

- Logically coherent collection of data with some inherent meaning
  – No need for storing courses taken by a student in an airline reservation system

What is a Database Management System (DBMS)?

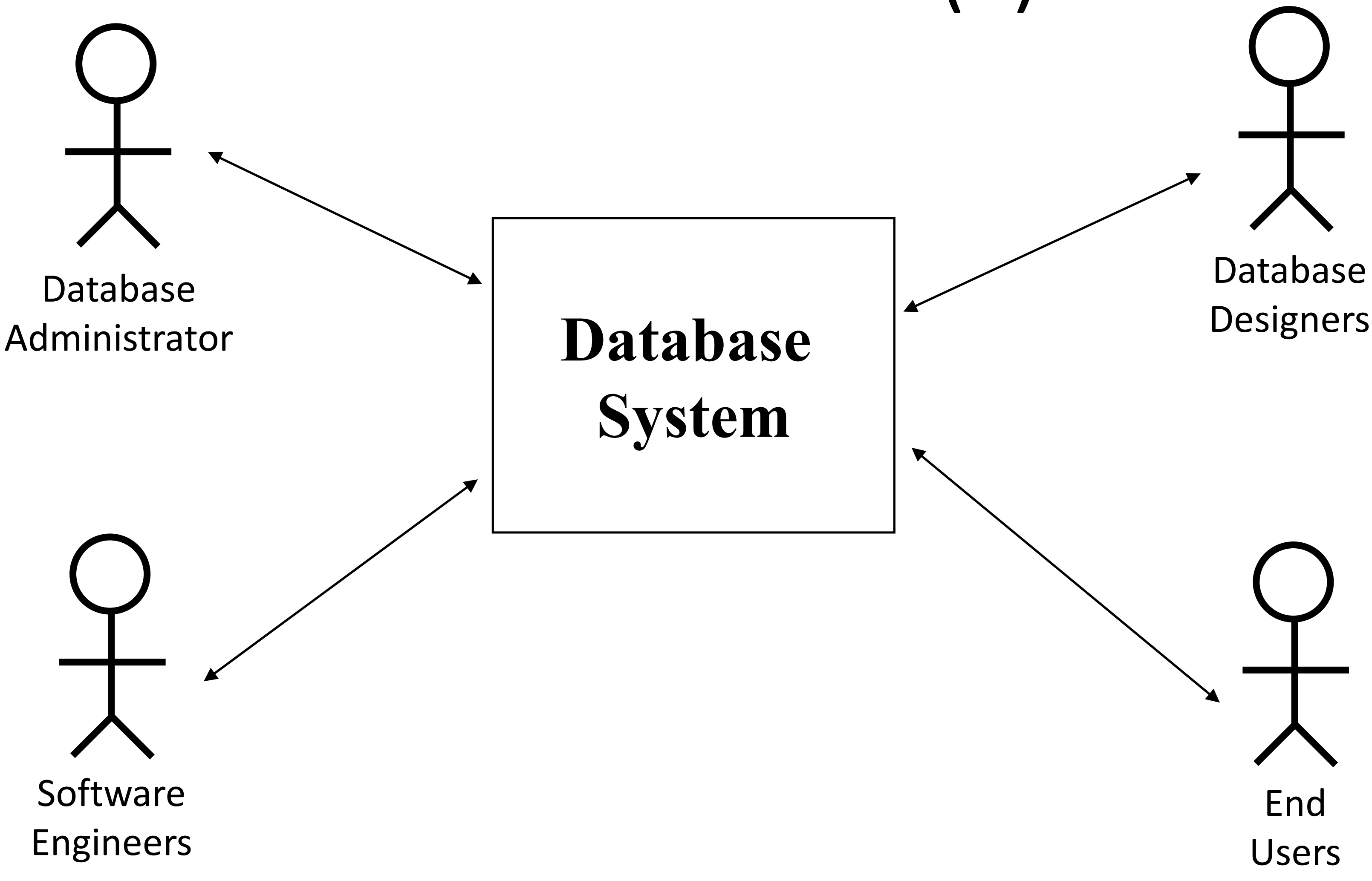
A DBMS is a collection of programs facilitating:

- Definition of a database
  - Specifying data types, structures and constraints for the data to be stored in the database
- Construction of a database
  - Storing data on storage medium controlled by the DBMS
- Manipulation of a database
  - Querying database to retrieve specific data,
    updating database to reflect changes in the mini-world,
    and generating reports from the data

Which of the following is “a specific object, real or abstract, about which we need to store attributes and relationships”?

A. Schema
B. Record
C. Entity
D. Attribute

Actors in a database system environment (1)



Actors in a database system environment (2)

- Database Administrator
  - Authorises access to the database
  - Coordinates and monitors its use
  - Acquires software and hardware resources as required

- Database Designers
  - Identify the data to be stored in the database
  - Choose appropriate structures and constraints to represent and store this data

- Software Engineers
  - Identify end user requirements (e.g. standard types of queries and updates – also called canned transactions) *pre packaged canned data which is consistent and easy in understanding.*
  - Implement, test, debug, document and maintain

Actors in a database system environment (3): *End Users*

- Casual end users
  - Occasionally access database but may need different information each time
  - Use a sophisticated database query language
- Naive or parametric end users
  - Make up a sizable portion of database end users
  - Use canned transactions
  - e.g. bank tellers, airline/ hotel reservation systems
- Sophisticated end users
  - Implement their own applications
  - e.g. engineers, scientists, business analysts
- Stand-alone end users
  - Maintain personal databases using off-the-shelf systems e.g. Microsoft Money (a personal finance management software)

You use Amazon to search for new office chair. What does that make you?

A. DB Admin
B. DB Designer
C. Casual end user
D. Sophisticated end user
E. Parametric end user
F. Stand-alone end user