Databases
HONP112 – Week 10
Professor Jim Williams

What Is A Database?

- A database is a structured collection of related information that can be:
  - Stored
  - Manipulated (Added, Deleted, Edited)
  - Sorted
  - Identified and Retrieved (Searching)
  - Reported On

Examples of Databases

- The Student Registration System at MSU.
- Your employment records.
- A customer information system.
- Etc.
What do all of these have in Common?

- There must be a way to break down the information into smaller, defined parts.
  - Example: A Customer may be broken down into First Name, Last Name, City, etc.
- Every similar entry contains the same parts.
  - Example: An address always consists of some elements and not others.

3 Components of a Database

- Fields – contain the smallest meaningful pieces of information.
- Records – contain separate instances of the same related fields.
- Tables – contain separate instances of the same related records.
- A DATABASE therefore is made up of tables, which store records, which composed of fields.

Designing a database

1. Understand the information we need to represent.
2. Break the information into individual fields.
3. Organize the fields into logical groups so that as a whole they describe a single "entity." This is technically called "decomposition," or in layman’s terms “breaking the fields up into RECORDS.”
4. Determine which fields, or group of fields, uniquely identifies a single instance of a record.
5. Implement the database structure as TABLES that will store the individual records.
Designing a database

- The next slides will provide a “from the ground up” way of looking at database design.
- As such we will start with defining fields and work our way up from there.
- As we go through the lesson try to think of some examples on your own, and of course ask ANY questions as you see fit.

A Field

- A field is the smallest meaningful piece of information in a database.
- Why “meaningful”?
- Because the user must clearly understand what the information represents.

Meaningful Fields

- For instance, in a customer database, the fields “First Name” and “Last Name” are clearly meaningful.
- Each piece of information can be clearly identified and understood.
- Anti-Example: A separate field for each letter in the name would be meaningless.
Field Types

- To properly and accurately define a field, we need to know what TYPE of information will be stored in it.
- When we design a database, after defining the fields that we want to use, we must also fully understand the type of information that will be "allowed" into the field.
- The following slides will explain some common field types. As we discuss these, try to think of some examples where the type will be applicable.

A note about field types...

- I will be giving you some brief information about some common field types, and how much space they typically take up on a disk.
- Keep in mind however that the list is NOT exhaustive and real-life data storage gets into some complexities that we will not cover.
- Also, I will not get into many specifics of how exactly certain data is stored behind the scenes.

Character Strings

- A very common field type is the Character String. This field can contain one or several of the common ASCII characters from our chart.
- Character string fields are also of a specified length. It does not matter how "long" a specific string really is. You set the maximum length and this same amount of space is set aside for ANY field that is stored.
- So when defining a character string field, it is important to know the length we need to account for (for instance a street address will require more characters than a 2-character state abbreviation).
- Using ASCII, character strings take up 1 byte per character.
Character Strings and Length

- Say we have a character string field called “First Name.” We set the field length to be 25 characters long.
- Now how much space do these values take up?
  - Jim
  - Sharon
  - Adrianna
  - Christopher
- The answer: ALL of them will take up 25 characters. It doesn’t matter how much space we actually “use” in each instance. The rest of the characters remain unused, but they cannot be used for anything else since they are reserved for the First Name field.
- Keep this in mind when designing fields. Don’t use more characters than you really need to.

Character Strings and Digits

- It is important to understand that the CONTEXT once again comes into play.
- For instance, consider a phone number.
- Yes, only numeric “digit” characters are used, but the data does not MEAN a “number” (i.e. a quantity of something).
- The same holds true for things like ZIP codes, course numbers, etc.
- So know that character strings CAN include digits, but only in terms of digits that need not be calculated as quantities in any way.

Numeric Fields

- Certain fields, however, DO need to be understood as numeric quantities. These are fields that must be calculated, compared against, etc. as quantities.
- For instance, “course enrollment limit,” “price”, “age”, etc.
- Different numeric fields are stored differently. Some common numeric fields include Unsigned Integer (positive whole numbers only), Signed Integer (negative or positive whole numbers), Fractional numbers (like 3.99 or 0.077), etc.
- Short integers take up 2 bytes. Long Integers take up 4 bytes. Simple fractional numbers take up 4 bytes. Very precise fractional numbers take up 8 bytes (rarely used – only for extremely critical precise scientific calculations, etc.)
Depending on the type of value we want to store, the integer field can either be 2 bytes, or 4 bytes.

Considering positive integers only, what is the largest number we can store using 2 bytes (short integer)? $2^{16} = 65536$. We have to account for the zero of course so the largest number would be 65535.

If we need to store larger numbers, we can instead use 4–bytes (long integer). What is the largest integer we can store in that case? Figure it out.

Signed integers: Long and short integers can also be stored as signed numbers – remember in this context then that about half the combinations will mean negative, and the other half positive (have to account for the zero too). Do not worry about the “behind the scenes format” for signed integers!

In our terms we call these “real” numbers or commonly “decimal” numbers, but I avoid the use “decimal” because it is confusing in our context (since the numbers are really stored in binary!).

These are used for any quantity that has the possibility of not being a whole number. For example: prices, measurements, distances, calculations, GPA, etc. Try to think of some yourself.

In computer speak these are called “floating point” numbers – in case you ever encounter that term. Don’t worry about what that means now 😊

Date fields do not represent a quantity, but they are not character strings either.


If we considered them character strings, the earlier date would appear alphabetically AFTER the later date!

So we also need a special field that defines the value as a date. This way, date comparisons, sorting, and calculations can be done correctly.

Dates are usually stored using 3 bytes.

There are other related formats like “time” and “date/time” – stored differently but you get the idea.
Boolean Fields

- Certain fields only need to be stored as simple, one-bit values of either yes or no.
- So there is another special type called a boolean (or true/false, or yes/no) type.
- Examples: “currently enrolled?”, “over 21?”, “eligible for Financial Aid?”, etc.
- Now you can theoretically use a 1-character string field and store either “Y” or “N”, but that would take up an entire byte. So, why waste space?

Long Text (AKA “Memo”) fields

- Consider character string fields. As you know, these must be defined using a set length.
- For most character data, this is sufficient. But what about things like “brief summary”, “biography”, etc.?
- In these cases we use “long text” or “Memo” fields. In layman’s terms, these allow for an “open-ended” length. These work behind the scenes by storing a “pointer” to some disk address, and maintains its own “table of contents” to tell the system where potentially paragraphs worth of data are stored on the disk. We will not get into specifics!

Other field types

- There are other field types available in some systems too, but will not be studying them here. But in case you encounter them here are some examples (and there are more):
  - BLOB (Binary Large Object) – can store entire digital “files” in binary format (like photos for ID records, etc.).
  - Time – ex: 9:00PM, 14:00.000, etc.
  - Date/Time – combines data field with a timestamp
  - Hyperlink – an Internet address
  - etc. etc. etc....
Sample Problem – Disk Space

Consider the following record structure. How much space in BITS will each **single record** take up on a disk?

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student ID</td>
<td>Short Integer</td>
<td></td>
</tr>
<tr>
<td>Date First Enrolled</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>Last Name</td>
<td>Character</td>
<td>35</td>
</tr>
<tr>
<td>First Name</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>FA Eligible?</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>Phone Number</td>
<td>Character</td>
<td>10</td>
</tr>
</tbody>
</table>

1 short integer * 2 bytes each = 2 bytes.
65 characters * 1 byte each = 65 bytes
1 date * 3 bytes each = 3 bytes
1 boolean at 1 bit each = 1 bit
Total: 70 bytes (560 bits) + 1 bit = 561 bits.

Sample Problem – Disk Space

How much space in BYTES will be taken up on the disk if the database table contains 5000 records?

5000 * 561 bits = 280500 bits = 35062.5 BYTES, or 35.0625 KB.
Fields have Properties

- There are other things that we may decide to do when defining a database field.
- For example, we may say that a field is a required entry, that is a KEY field (more on that later), that it must be no greater than or less than a certain value, that it must be a certain required length, etc.
- These are all called PROPERTIES of a field.
- There are different ways of specifying database properties. Sometimes these are defined in the database structure itself; other times it may be how we define data entry screens.

Front End / Back End

- Before we continue, I want to introduce the concept of a front end and a back end.
- The front end means any screens we use to look at or enter the data.
- The back end means the actual stored data on the disk (and how it is structured).
- Sometimes the front and back ends are part of the same system; other times (and more commonly in large systems) the front end may be separate, and needs to “communicate” with the back end.

Records

- A record is a single grouping of related fields.
- For example, a record called “Customer” will be made up of the fields First Name, Last Name, Address, etc.
- Each customer has their own customer record. The fields are the same – the contents of the fields are different for each customer.
### A Customer Record Illustrated
- **First Name:** Jim
- **Last Name:** Williams
- **Zip Code:** 07604
- **Last Purchase Date:** 12/2/2001
- **Customer Since:** 8/12/1988
- **Amount of Average Purchase:** $5.50
- **Discount Card?** : No

### Another Customer Record
- **First Name:** Sharon
- **Last Name:** Williams
- **Zip Code:** 07604
- **Last Purchase Date:** 1/2/2002
- **Customer Since:** 9/17/1992
- **Amount of Average Purchase:** $55.00
- **Discount Card?** : Yes

### An Observation
- Both records were made up of the same fields.
- The actual information in each field may be different for each customer.
- The records are constructed in a way where the group of fields makes logical sense as a whole – the record **means** something unto itself.
Meaningful Records

- Deciding which fields to group together in a single record is not always easy – sometimes it may be more of an “art” than a science.
- The bottom line is that the fields, when grouped together, clearly illustrate something understandable.
- Anti-Example: Would we make a field called “Grade Point Average” part of a customer record???

Tables

- A table is a collection of related records.
- In simple terms, we may think of a table as a list of records.
- For example, a customer table will contain a list of the information about individual customers.

The Customer Table

- Our customer table may look something like this:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim Williams</td>
<td>07/073</td>
<td>12/2/2001</td>
<td>8/12/1988</td>
</tr>
<tr>
<td>Sharon Williams</td>
<td>07/073</td>
<td>1/2/2002</td>
<td>9/17/2002</td>
</tr>
</tbody>
</table>

  (Not all fields are shown in this example.)
More Than A List

- Our simple table just looks like a list.
- But, remember that we need to be able to easily identify records in a table.
- Why? Because a database allows us to locate particular records. So, we must have a way to uniquely identify an individual record.

Uniquely Identifying a Record

- Suppose we have a customer table with thousands of records.
- Now, we want to identify an individual customer.
- How do we do it?

Some Possibilities

- Use the last name – but this won't work in our example. My wife and I have the same last name. There are probably more customers with that last name too.
- Use the last name and first name – this will work for now – but my name is very common and there may be other customers with the same name.
More Possibilities

- Use the first name, last name, and Zip code. This may work until someone else with my name and zip code becomes a customer.
- Another problem – What if both my wife and I had “Pat” as a first name?

The KEY field

- A field, or combination of fields, that allows us to uniquely identify a single record in a table is called a KEY.
- No two records in the same table can have the same KEY.
- As you can see, it is very difficult to use common fields to make a KEY.
- Usually, there would have to be a field like “Customer Number.”

Some Common Keys

- Employee ID
- Student ID
- Customer Number
- Account Number
- Etc.
Good Keys and Bad Keys

- Good keys usually involve sequential numbers. For instance, the first customer entered would have a Customer ID of 00001, the second would be 00002, etc.
- Bad keys contain meaningful data that may change. Also, some may violate privacy concerns (like the SSN).

KEYS mean nothing and everything!

- In short, a good KEY field should not contain any data that has any other meaning other than acting as a unique identifier.
- This is because data that means something else may change, and therefore we would have to change a key.
- Why don’t we want to do that?

Our Improved Customer Record

- Customer Number: 00001
- First Name: Jim
- Last Name: Williams
- Zip Code: 07604
- Last Purchase Date: 12/2/2001
- Customer Since: 8/12/1988
- Amount of Average Purchase: $5.50
- Discount Card?: No
A database can consist of many different tables.
For instance, in a sales system, we may want a separate table that contains a customer’s purchases.
The key field would “tie” these 2 tables together, so we would be able to link a customer to all her purchases.

Why 2 Tables?
Think about it. We could easily add fields to our Customer record that can store customer purchases information.
But, how many additional fields would we need?

How Many Additional Fields?
We could assume that an average customer will purchase 100 things in his/her lifetime.
So, we can just add 100 additional sets of fields to track the purchase information.
But what about a customer who purchases more?
What about a customer who only purchases 2 things?
Why We Don’t Do That

- If a customer purchases more than 100 things, we will have to change the record to allow more fields.
- Because we change the one record, all the other records must also change. This would involve too much work.
- For customers who only buy a few things, this is also a waste of space (remember we always have to allow for a specified amount of disk space depending on the field type).

Solution – A Purchases Record

- Customer Number: 00001
- Item Number: 5435
- Date Of Purchase: 3/4/2001
- Item Sold: Stereo Receiver
- Quantity Sold: 1
- Purchase Price: $55.76

The Purchases Key

- Notice the key for our Purchases record is a **COMPOUND KEY**. In other words, it consists of more than one field.
- What this means is the COMBINATION of the Customer Number, Item Number, and Date of Purchase cannot repeat in the table.
- Would this work OK?
What is the “Item Number??”

- Remember that a “purchase” record really “ties” a customer to an item that was purchased.
- So we would of course also need yet another table that contains all the Item records.

An Item Record

- Item Number: 5435
- Manufacturer: Brand X
- Model: Super Duper Model
- Description: Stereo Receiver
- Suggested Retail Price: $60.00
- Number in inventory: 7

Parents and Children

- Notice that the purchase is tied to a Customer Number.
- This way, all the records in the Purchases table with 00001 as a Customer ID can be tied to a single customer.
- Since a single customer can have many purchases, the Customer record is called the “Parent” record, and the Purchases record is the “Child” record.
Our overly simplistic simple relationship example demonstrated a common type of relationship called ONE to MANY.

This means that the field can have one unique value in a particular table, but that value can repeat many times in another table.

For instance, there can only be one customer with a given customer number. BUT, that customer can make MANY purchases.

There can also by ONE-to-ONE, and MANY-to-MANY relationships too. We won’t get into that here 🙄.

Consider this list of fields:
Business Phone Number, Last Name, Cumulative GPA, Semester Average, Semester, First Name, Street Address, Cell Phone Number, Course Number, Course Grade, City, Home State, Quality Points, Home Phone Number, Semester End Date.

Decompose these into meaningful record groupings.
Decomposition – Repeats

- **Business Phone Number**, Last Name, Cumulative GPA, Semester Average, Semester, First Name, Street Address, **Cell Phone Number**, Course Number, Course Grade, City, Home State, Quality Points, **Home Phone Number**, Semester End Date.

- Notice that the highlighted fields illustrate where several instances of the same information repeat. This tells us that they should be grouped into a separate record (because someone may have one, or several, or none, of these phone numbers!).

Decomposition – continued

- **Phone Numbers** (Business Phone Number, Cell Phone Number, Home Phone Number)

- Now let’s look at the rest:

- **Last Name**, Cumulative GPA, Semester Average, Semester, **First Name**, **Street Address**, Course Number, Course Grade, City, **State**, Quality Points, Semester End Date.

- The highlighted entries all belong to a “Person”. So we make another grouping out of those.

Decomposition – continued

- **Phone Numbers** (Business Phone Number, Cell Phone Number, Home Phone Number)

- **People** (Last Name, First Name, Street Address, City)

- Now let’s look at the rest:

- Cumulative GPA, Semester Average, Semester, Course Number, Course Grade, Quality Points, Semester End Date.

- The remainder appear to all make sense together as part of a transcript.
Decomposition – continued

- **Phone Numbers** (Business Phone Number, Cell Phone Number, Home Phone Number)
- **People** (Last Name, First Name, Street Address, City)
- **Transcript Entries** (Cumulative GPA, Semester Average, Semester, Course Number, Course Grade, Quality Points, Semester End Date)

In this very simplistic and incomplete example, we can at least illustrate how we have broken down a random group of fields into entities that make sense as a whole.

### Relationships

- **Phone Numbers** (Business Phone Number, Cell Phone Number, Home Phone Number)
- **People** (Last Name, First Name, Street Address, City)
- **Transcript Entries** (Cumulative GPA, Semester Average, Semester, Course Number, Course Grade, Quality Points, Semester End Date)

- **One** person to **MANY** phone numbers.
- **One** person to **MANY** transcript entries

### Structure

- Now since we know the records, and the relationship between them, let’s construct the records.
- When doing this, be certain to pick the appropriate Field Types and also designate which fields or groups of fields make up the KEY for each record/table.
Structure – People

- People (Last Name, First Name, Street Address, City)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person ID</td>
<td>Long Integer</td>
<td></td>
</tr>
<tr>
<td>Last Name</td>
<td>Character</td>
<td>35</td>
</tr>
<tr>
<td>First Name</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>Street Address</td>
<td>Character</td>
<td>50</td>
</tr>
<tr>
<td>City</td>
<td>Character</td>
<td>40</td>
</tr>
</tbody>
</table>

Structure – Phone Numbers

- Phone Numbers (Business Phone Number, Cell Phone Number, Home Phone Number)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person ID</td>
<td>Long Integer</td>
<td></td>
</tr>
<tr>
<td>Phone Number Type</td>
<td>Character</td>
<td>5</td>
</tr>
<tr>
<td>Phone Number</td>
<td>Character</td>
<td>20</td>
</tr>
</tbody>
</table>

Structure – Transcript Entries

- Transcript Entries (Cumulative, Semester Average, Semester, Course Number, Course Grade, Quality Points, Semester End Date)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person ID</td>
<td>Long Integer</td>
<td></td>
</tr>
<tr>
<td>Semester</td>
<td>Character</td>
<td>5</td>
</tr>
<tr>
<td>Course Number</td>
<td>Character</td>
<td>8</td>
</tr>
<tr>
<td>Semester Average</td>
<td>Decimal Number</td>
<td></td>
</tr>
<tr>
<td>Quality Points</td>
<td>Short Integer</td>
<td></td>
</tr>
<tr>
<td>Semester End Date</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>Cumulative GPA</td>
<td>Decimal Number</td>
<td></td>
</tr>
<tr>
<td>Course Grade</td>
<td>Character</td>
<td>2</td>
</tr>
</tbody>
</table>
Think about how these fields can be decomposed further... for instance, which fields are for semester summary, and which are for semester details? Any time we can have a one-to-many relationship like this, we should consider further decomposition.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person ID</td>
<td>Long Integer</td>
<td></td>
</tr>
<tr>
<td>Semester</td>
<td>Character</td>
<td>5</td>
</tr>
<tr>
<td>Course Number</td>
<td>Character</td>
<td>8</td>
</tr>
<tr>
<td>Semester Average</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>Quality Points</td>
<td>Short Integer</td>
<td></td>
</tr>
<tr>
<td>Semester End Date</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>Cumulative GPA</td>
<td>Decimal</td>
<td></td>
</tr>
<tr>
<td>Course Grade</td>
<td>Character</td>
<td>2</td>
</tr>
</tbody>
</table>

Phone Numbers
- Person ID
- Phone Number Type
- Phone Number

NOTE: Keys are bold print

A key that uniquely identifies a single record in the same table is called a PRIMARY KEY. This is how we have been using the term thus far.

However, a field in record “A” that “relates” back to a key field in record “B”, even if it is not part of the primary key in record “A,” is called a FOREIGN KEY.
The State Abbreviation field in the People record is a FOREIGN Key, because it is the PRIMARY key in the States record.
You see this very often with "look-up" tables that contain lists of valid values that can be chosen.

As you can see, designing and structuring databases is not easy.
It requires a lot of analysis and frequently a designer will need to "refine" the design several times before coming up with an optimal solution (i.e. our yet incomplete attempt at designing transcript entries…).
Hopefully in working out some example problems, and thinking of some other problems on your own, you can come to appreciate how much this work does NOT rely on technology itself, but rather in functional expertise in the type of information we wish to structure and store.

You should be asking yourself why we even bother to break down information this way. What is the purpose? After all, we can just store all sorts of information manually (on cards, paper, folders, etc.).
The whole point of this is that it makes it a lot easier for us to sort, locate and retrieve information very quickly, and then we can produce various reports on it.