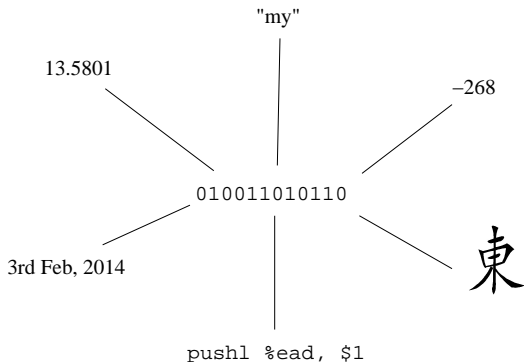


LBSC 690: Information Technology
Lecture 05
Structured data and databases

William Webber
CIS, University of Maryland

Spring semester, 2012

Interpreting bits



- ▶ At heart, all data is stored as a string (or strings) of bits.
- ▶ But interpretation of those bits depends upon type of data.
- ▶ The one and the same bit string means different things when interpreted as a different type.

Integral data types

Unsigned integer Interpret bit string as fixed-width binary number

Signed integer First bit gives sign (positive or negative). For negative numbers, remaining bits count down from largest negative value.

Both have various fixed-width bit sizes (8 bits, 16 bits, etc.)

Floating point data types

Sign	Exponent	Significand
1	10000011	010010000000000000000000

$$-(2^4) * 0.128125 = -2.05 \quad (1)$$

- ▶ Used to represent values with (binary) decimal places (also very large values with limited precision).
- ▶ The (binary) decimal point “floats” to different locations.
- ▶ Bit string has three parts:
 - ▶ Significand, gives the placeless, signless value
 - ▶ Exponent, specifies what place the (binary) decimal point is
 - ▶ Sign bit, says whether value is positive or negative

Character data types

$01100011 \rightarrow 143 \rightarrow 'c'$ (2)

- ▶ A **character** is, roughly, a letter from an alphabet or script.
- ▶ Each numerical value mapped to a different character.
- ▶ Mapping from number to character (and back) called a **character set** or **charset**.
- ▶ (A sequence of characters sometimes called a (character) **string**)

Excursus: designing a character set for English

- ▶ How many distinct values are there in the English (lower-case) alphabet?
- ▶ How many bits to represent these many distinct values?
- ▶ How many if we add in upper case? Numerical digits? Punctuation?

ASCII

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

- ▶ ASCII (1963) standardized English character set
- ▶ Uses 7 bits (leaves 8th bit of byte unused)
- ▶ Covers:
 - ▶ letters (upper and lower)
 - ▶ numerical digits
 - ▶ common punctuation and symbols
 - ▶ whitespace (space, tab, newline)
 - ▶ “system” values (delete; backspace; ring a bell)

Extending ASCII

```
.....  
.....  
!"#$%&'()*+,-./  
0123456789:;<=>?  
@ABCDEFGHIJKLMNO  
PQRSTUVWXYZ[\]^_  
`abcdefghijklmnopq  
rstuvwxyz{|}~.  
€.,f,...t+^%Š<€.Ž.  
. ' ' " • -- ~™š >œ.žÿ  
¡¢£¥¦§¨ª«¬®¯  
°±²³´µ¶·¸¹º»¼½¾¿  
ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ  
ÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß  
àáâãäåæçèéêëìíîï  
ðñòóôõö÷øùúûüýþÿ
```


































Figure: windows-1252
charset

- ▶ ASCII does not cover even all Western European languages (no ò, no ä, no ß, etc.)
- ▶ Various extensions use the eight bit, such as windows-1252
- ▶ These can turn up even under English to give (e.g.) directional quotes (“, ”, rather than "), em- and en-dashes (–, —, rather than --, ---)
- ▶ Such characters will not display on all systems! (and may stuff up your HTML code!)

A mess of character sets

- ▶ Different extensions to ASCII not compatible (Windows, DOS, and MacOS used to have quite different eighth-bit Latin character sets).
- ▶ Non-Latin alphabets needed their own characters sets.
- ▶ Chinese, Japanese character sets too large to represent in 8 bits
- ▶ No simple, standard way of representing different alphabets on the one page (e.g. a Greek–Tamil lexicon)

A general, standard solution: Unicode

	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	130A
0	 13000	 13010	 13020	 13030	 13040	 13050	 13060	 13070	 13080	 13090	 130A0
1	 13001	 13011	 13021	 13031	 13041	 13051	 13061	 13071	 13081	 13091	 130A1
2	 13002	 13012	 13022	 13032	 13042	 13052	 13062	 13072	 13082	 13092	 130A2

- ▶ Unicode provides one character set for all human languages, plus many symbol vocabularies.
- ▶ Base unicode set (“Plane 0”) provides character sets for “living” scripts, up to 2^{16} values (i.e. representable in 16 bits)
- ▶ Additional “planes” provide support for historical scripts (cuneiform, hieroglyphics), archaic forms of living scripts (ancient forms of Chinese characters)

Unicode: encodings, fonts

- ▶ Unicode specifies character set (mapping from numbers to characters), but not **encoding** (way of representing numbers).
- ▶ Three common encodings:
 - ▶ UTF-8. Single-byte for ASCII, additional bytes for trans-ASCII. Backwards compatible with ASCII for ASCII-only text.
 - ▶ UTF-16. Two-byte (16 bit) basic characters, extended to four bytes (32 bits) for additional planes.
 - ▶ UTF-32. Fixed-width, 32 bits for every character
- ▶ Just because a system recognizes the Unicode character set, doesn't mean it has a font capable of displaying all the characters in it!

Other data types

Besides numerical, character data types, various others possible:

- ▶ Date (time, date and time, timestamp)
- ▶ Boolean (true, false)
- ▶ Address in computer memory
- ▶ Machine code instruction

Compound data types can be made up from these “atomic” types (e.g. a set; an ordered list; a user; a street address).

Managing data and databases

Jane Doe is a student of LBSC 690, “Information Technology”. Her date of birth is 1st January, 1980. Her student id is 1234-5678. Her mark is 87. The lecturer for LBSC 690 subject is William Webber.

- ▶ Humans can process information about people expressed in natural language.
- ▶ Computers ~~can't~~ increasingly can, but not fluently.

Structured data

Property	Value
Given name	Jane
Family name	Doe
Date of birth	1980-01-01
Student id	1234-5678
Subject code	LBSC 690
Subject name	Information Technology
Mark	87
Instructor	William Webber

- ▶ To (readily) manipulate data via a program, we need to extract its structure.
- ▶ Simple structure is as list of “property : value” pairs.
 - ▶ We’ve seen this sort of structure with CSS.
- ▶ Subject of the properties is an **entity**.
- ▶ Not all data is structural and structure depends on application.

Data tables

Given name	Family name	Date of birth	Student id	Subject code	Subject name	Mark	Instructor
Jane	Doe	1980-01-01	1234-5678	LBSC 690	Information Technology	87	William Webber
John	Dee	1985-06-21	2233-4455	LBSC 690	Information Technology	75	William Webber
Jane	Doe	1980-01-01	1234-5678	LBSC 771	Records Management	76	Adam Adamson
John	Dee	1985-06-21	2233-4455	LBSC 601	Basket Weaving	52	William Webber

- ▶ Where structure is common to many entities, can be organized as a table.
 - ▶ Each row of table corresponds to an entity or **record**.
 - ▶ Each column corresponds to a property or **field**.
 - ▶ Each cell gives the entity's value for that field.
- ▶ These (spreadsheet-like) tables are at the heart of databases.

Schema

Property	Type
Given name	Character
Family name	Character
Date of birth	Date
Student id	Character
Subject code	Character
Subject name	Character
Mark	Integer
Instructor	Character

- ▶ The properties of a table are called the (data) **schema**
- ▶ The schema defines the names of the properties
- ▶ ... and also their types
- ▶ (and possibly also other constraints: length; can they be empty; etc.)

Indexes

Field	Type	Properties
Given name	Character	
Family name	Character	Indexed
Date of birth	Date	
Student id	Character	Indexed
Subject code	Character	
Subject name	Character	
Mark	Integer	
Instructor	Character	

- ▶ As well as storing records, we need to be able to look them up.
- ▶ For a large database with millions of records, searching through them one by one is too slow.
- ▶ Instead, we build an **index** on fields that are likely to be frequently searched (e.g. id number, name).
- ▶ Indexed fields are called **keys**

Speeding up lookup

How to look a thing up faster than going through one by one?

Consider looking up a phonebook (remember them?) for a family name.

1. Open to a page somewhere in the middle.
2. If name you want is there:
 - 2.1 STOP
3. Else, if name you want is before current page:
 - 3.1 Remember current page as new end of book.
 - 3.2 Open somewhere in middle of first half.
4. Else:
 - 4.1 Remember current page as new start of book.
 - 4.2 Open somewhere in middle of second half.
5. Go to Step 2

Congratulations – your first (possibly) algorithm!

Binary search

- ▶ Computer equivalent of telephone lookup is **binary search**.
- ▶ Always splits list strictly in half (lacks “feel”).
- ▶ Simple index:
 - ▶ order index fields (alphabetically, numerically, by date)
 - ▶ for each indexed field, keep location of record in table (e.g. row number)
 - ▶ perform binary search

Excursus: algorithmic complexity

If we have a table with 1,000 records, and we are looking up a random record, how many records do we need to look at on average to find the desired record:

- ▶ looking at them one at a time (linear search)?
- ▶ with an ordered index and performing a binary search?

Primary key

Field	Type	Properties
Id	Integer	Primary Key
Given name	Character	
Family name	Character	Indexed
Date of birth	Date	
Student id	Character	Indexed
Subject code	Character	
Subject name	Character	
Mark	Integer	
Instructor	Character	

- ▶ We also want a canonical way of referring to record.
- ▶ This indexed field is known as the **primary key**.
- ▶ Primary key must (essentially) never change ...
- ▶ ... so best to make it a field that has no application meaning (e.g. an arbitrary integer, as here)

Compound entities

Field	Type	Properties
Id	Integer	Primary Key
Given name	Character	
Family name	Character	Indexed
Date of birth	Date	
Student id	Character	Indexed
Subject code	Character	
Subject name	Character	
Mark	Integer	
Instructor	Character	

- ▶ So far, only considered single table
- ▶ But in above schema, some fields relate to subject, rather than student

Repeated information

Given name	Family name	Date of birth	Student id	Subject code	Subject name	Mark	Instructor
Jane	Doe	1980-01-01	1234-5678	LBSC 690	Information Technology	87	William Webber
John	Dee	1985-06-21	2233-4455	LBSC 690	Information Technology	75	William Webber
Jane	Doe	1980-01-01	1234-5678	LBSC 771	Records Management	76	Adam Adamson
John	Dee	1985-06-21	2233-4455	LBSC 601	Basket Weaving	52	William Webber

- ▶ Many students will be in the same subject
- ▶ ... don't want to repeat this information
- ▶ Especially since it may change

Foreign keys

Field	Type	Properties
Id	Integer	Primary Key
Given name	Character	
Family name	Character	Indexed
Date of birth	Date	
Student id	Character	Indexed
Mark	Integer	
Subject id	Integer	Foreign Key → Subject(Id)

Table: Student

Field	Type	Properties
Id	Integer	Primary Key
Code	Character	
Name	Character	
Instructor	Character	

Table: Subject

- ▶ Separate into two tables or entities
- ▶ The relationship between the two entities is captured using a **foreign key**
- ▶ One finds the subject record for a student by looking up the subject whose “Id” equals the student’s “Subject id”

Splitting into two tables

Student								
Id	Given name	Family name	Date of birth	Student id	Subject code	Subject name	Mark	Instructor
1	Jane	Doe	1980-01-01	1234-5678	LBSC 690	Info. Tech.	87	W. Webber
2	John	Dee	1985-06-21	2233-4455	LBSC 690	Info. Tech.	75	W. Webber
3	Jane	Doe	1980-01-01	1234-5678	LBSC 771	Record Mgmt	76	A. Adamson
4	John	Dee	1985-06-21	2233-4455	LBSC 601	Bskt Weaving	52	W. Webber

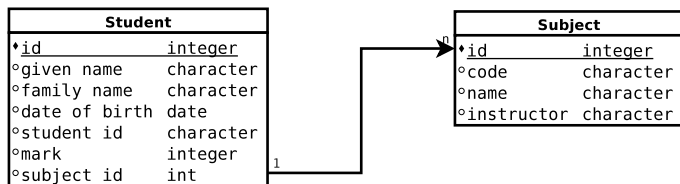
Figure: Before

Student						
Id	Given name	Family name	Date of birth	Student id	Mark	Subject id
1	Jane	Doe	1980-01-01	1234-5678	87	1
2	John	Dee	1985-06-21	2233-4455	75	1
3	Jane	Doe	1980-01-01	1234-5678	76	2
4	John	Dee	1985-06-21	2233-4455	52	3

Subject			
Id	Code	Name	Instructor
1	LBSC 690	Info. Tech.	W. Webber
2	LBSC 771	Record Mgmt	A. Adamson
3	LBSC 601	Bskt Weaving	W. Webber

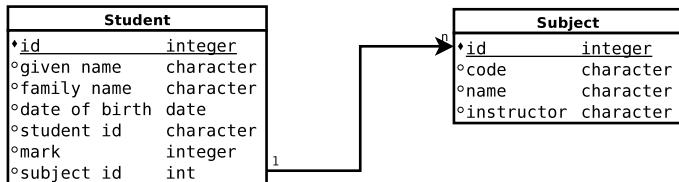
Figure: After

Entity-relation diagrams



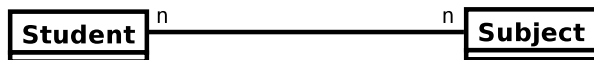
- ▶ Diagram relationship between entities during design phase.
- ▶ Several standards; we're looking at a simple one.
- ▶ Each entity represented by box, with (optionally) attributes of entity listed in box.

Relationships in ERDs



- ▶ Relationships in ERDs shown by arrow
- ▶ Arrow points from entity that has reference (here, from the foreign key attribute), to entity that is referenced
- ▶ Cardinality of membership shown at connection to entity, generally either 1 or n (for “many”).
 - ▶ Here, we are asserting that a student can have (be enrolled in) only one subject, but a subject can be had by (enrol) many students (a **one-to-many** relationship).

Further decomposition



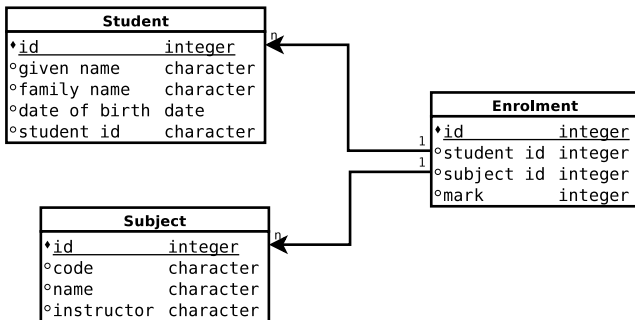
- ▶ Previous slide we said “a student can be enrolled in only one subject”; however, this is clearly wrong.
- ▶ The correct statement is:

Definition (Student-subject relationship)

A student can be enrolled in many subjects; a subject can have many students enrolled in it.

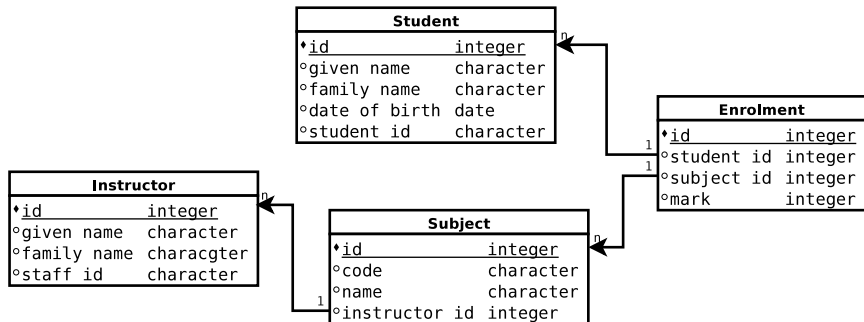
- ▶ This is a **many-to-many-relationship**.

Many-to-many relationships



- ▶ In practice, relational databases have a fixed number of fields.
- ▶ Cannot have variable number of foreign keys in one entity referencing another entity.
- ▶ Instead, for many-many relationships, need a separate entity (table) recording relation.
- ▶ This separate entity also holds ancillary data that is common in such relations (here, “mark”).

Further decompositions



- ▶ Instructor should be separated out
- ▶ Further possible:
 - ▶ Separate staff from instructor
 - ▶ Make common “person” table, as parent to staff and student
- ▶ General principle: don't repeat information!

XML for structured data

```
<student>
  <name given="Jane" family="Doe" />
  <date-of-birth>
    <date year="1980" month="01", day="01" />
  </date-of-birth>
  <student-id>1234-5678</student-id>
  <subjects>
    <subject>
      <code>LBSC 690</code>
      <!-- ... -->
    </subject>
    <subject>
      <code>LBSC 701</code>
      <!-- ... -->
    </subject>
  </subjects>
</student>
```

- ▶ An alternative representation for structured data is XML.
- ▶ XML creates tree of nested elements.
- ▶ Allows enforcement of schema by external schema.

XML: advantages

```
<student>
  <name given="Jane" family="Doe" />
  <date-of-birth>
    <date year="1980" month="01", day="01" />
  </date-of-birth>
  <student-id>1234-5678</student-id>
  <subjects>
    <subject>
      <code>LBSC 690</code>
      <!-- ... -->
    </subject>
    <subject>
      <code>LBSC 701</code>
      <!-- ... -->
    </subject>
  </subjects>
</student>
```

- ▶ Nesting of included elements (see `<subjects>` above)
- ▶ Self-documentation
- ▶ Human-editable

XML: disadvantages

```
<student>
  <name given="Jane" family="Doe" />
  <date-of-birth>
    <date year="1980" month="01", day="01" />
  </date-of-birth>
  <student-id>1234-5678</student-id>
  <subjects>
    <subject>
      <code>LBSC 690</code>
      <!-- ... -->
    </subject>
    <subject>
      <code>LBSC 701</code>
      <!-- ... -->
    </subject>
  </subjects>
</student>
```

- ▶ Shared elements (e.g. shared subjects) must be reference using `href`-like mechanism
- ▶ Verbose (tags repeated for every entity)

XML in practice

- ▶ XML mostly used in database world as a data exchange format.
- ▶ But there is ongoing work on native XML databases.

Looking ahead

- ▶ In this lecture, described relational databases conceptually, presented a graphical model for designing them (ERD).
- ▶ Next week, will examine language for defining, manipulating databases (SQL) . . .
- ▶ and will look at using an SQL-based relational database management system (RDBMS) to construct and manage a database.