MLDS-413 Introduction to Databases and Information Retrieval

Lecture 1
Overview; Integer Representation

Instructor: Nikos Hardavellas

Slides adapted from Steve Tarzia

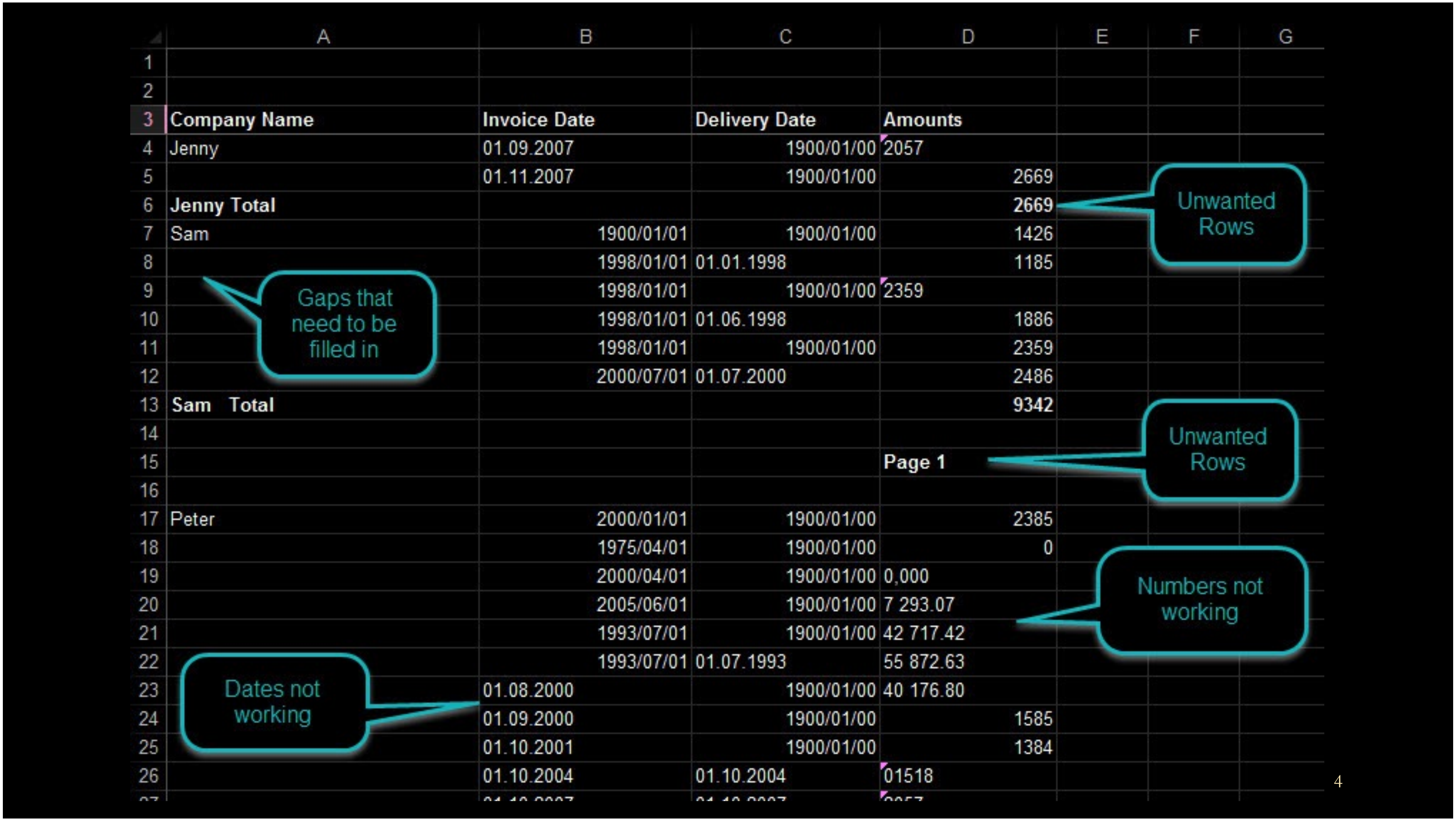
Course Summary

- Learn how to handle real-world, **complex**, **messy** data with SQL relational databases
  - A powerful foundational technology
  - Like a filesystem, but better
    - (easy queries, indexing, concurrency, crash tolerance)
- Roughly speaking "Data Science" is:
  - Data management (*this course!*)
  - Statistics (e.g., MLDS Predictive Analytics I, MLDS Data Mining)
  - Visualization (e.g., MLDS Data Visualization)

You'll learn to answer questions (about the past) using complex data sets

Things you cannot do with Excel and Matlab

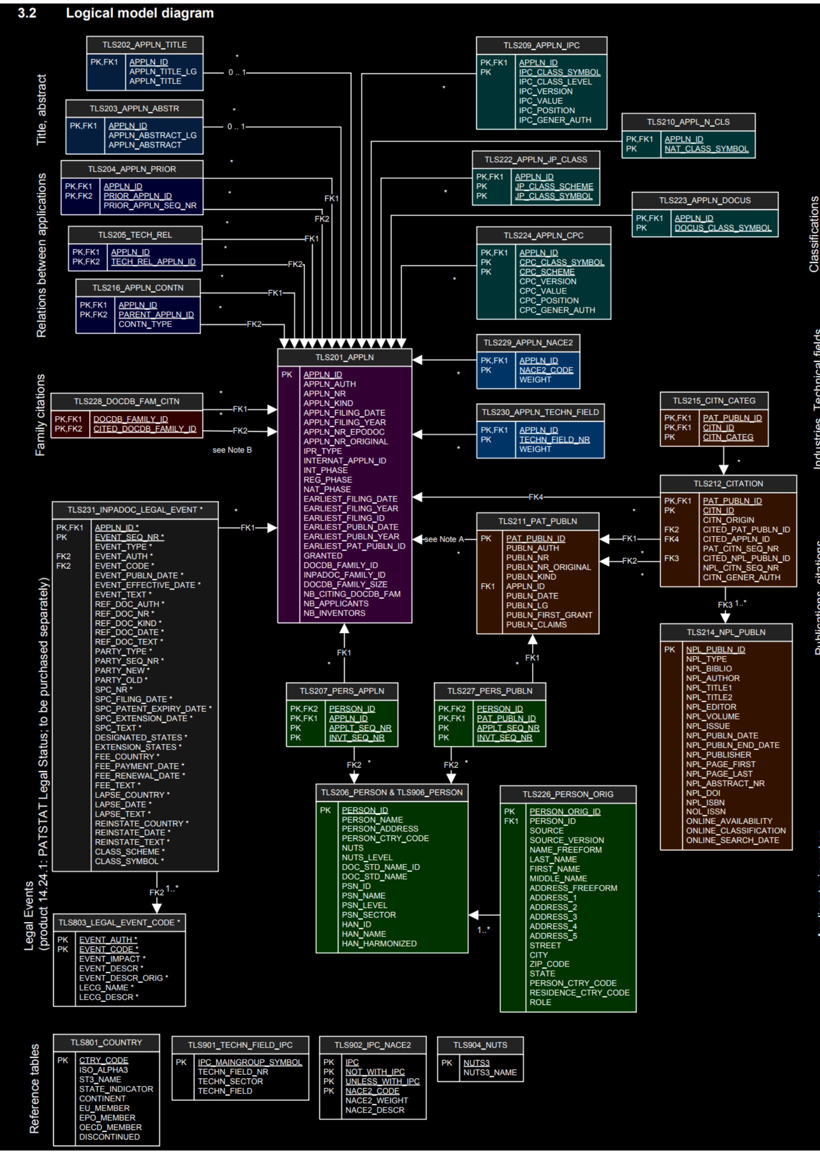
- Model complex data relationships
  - Spreadsheets and matrices are very limiting formats
  - Just have records with attributes
  - Can't model one-to-many and many-to-many relationships
  - Multiple spreadsheets / multiple matrices for different types of data are possible
    - ...but, linking them is difficult
- Enforce data integrity constraints
  - Spreadsheet cells can have all kinds of weird data
  - Matlab matrices cannot easily handle anything other than numbers
- Spreadsheets are terrible for large datasets!
  - [Excel forum on Reddit]
  - Crashing on 10K-100K rows of data
  - Freezing on changes
  - Calculations take minutes
- Keep data and analysis separate



	A	B	C	D	E	F	G
1							
2							
3	**Company Name**	**Invoice Date**	**Delivery Date**	**Amounts**			
4	Jenny	01.09.2007	1900/01/00	2057			
5		01.11.2007	1900/01/00	2669			
6	**Jenny Total**			**2669**			
7	Sam	1900/01/01	1900/01/00	1426			
8		1998/01/01	01.01.1998	1185			
9		1998/01/01	1900/01/00	2359			
10		1998/01/01	01.06.1998	1886			
11		1998/01/01	1900/01/00	2359			
12		2000/07/01	01.07.2000	2486			
13	**Sam Total**			**9342**			
14							
15				**Page 1**			
16							
17	Peter	2000/01/01	1900/01/00	2385			
18		1975/04/01	1900/01/00	0			
19		2000/04/01	1900/01/00	0,000			
20		2005/06/01	1900/01/00	7 293.07			
21		1993/07/01	1900/01/00	42 717.42			
22		1993/07/01	01.07.1993	55 872.63			
23		01.08.2000	1900/01/00	40 176.80			
24		01.09.2000	1900/01/00	1585			
25		01.10.2001	1900/01/00	1384			
26		01.10.2004	01.10.2004	01518			

PATSTAT: European Patent Office's International Patent Database

- 29 cross-referenced tables
- 6 DVDs of data
- 119GB of CSV files after unzipping



Stack Overflow Public Data Dump – 1TB

Difficulties in plain Python, R, C++, Java, etc.

- Working with data that is larger than the computer's RAM (*scalability*)
- Keeping your data around after your program finishes (*persistence*)
- Efficiently searching through lots of data (*indexing*)
- Easily filtering and summarizing data (*querying*)
- Sharing data between multiple applications (*concurrency*)

The Goal: Easy & Clean Descriptive Analytics

Answer a wide variety of complex questions using the same database:

- Where did our 10 biggest customers in 2007 live?

```
SELECT customer.name, customer.city, customer.province FROM
  customer JOIN order ON order.customer == customer.id
           JOIN order_item ON order_item.order == order.id
           JOIN product ON order_item.product = product.id
  WHERE order.placed >= “2007-01-01” AND order.placed < “2008-01-01”
  GROUP BY customer.id
  ORDER BY SUM(order_item.qty * product.price) DESC
  LIMIT 10;
```

- How many widgets are left in stock?
- What is the average price of the chairs we sell?

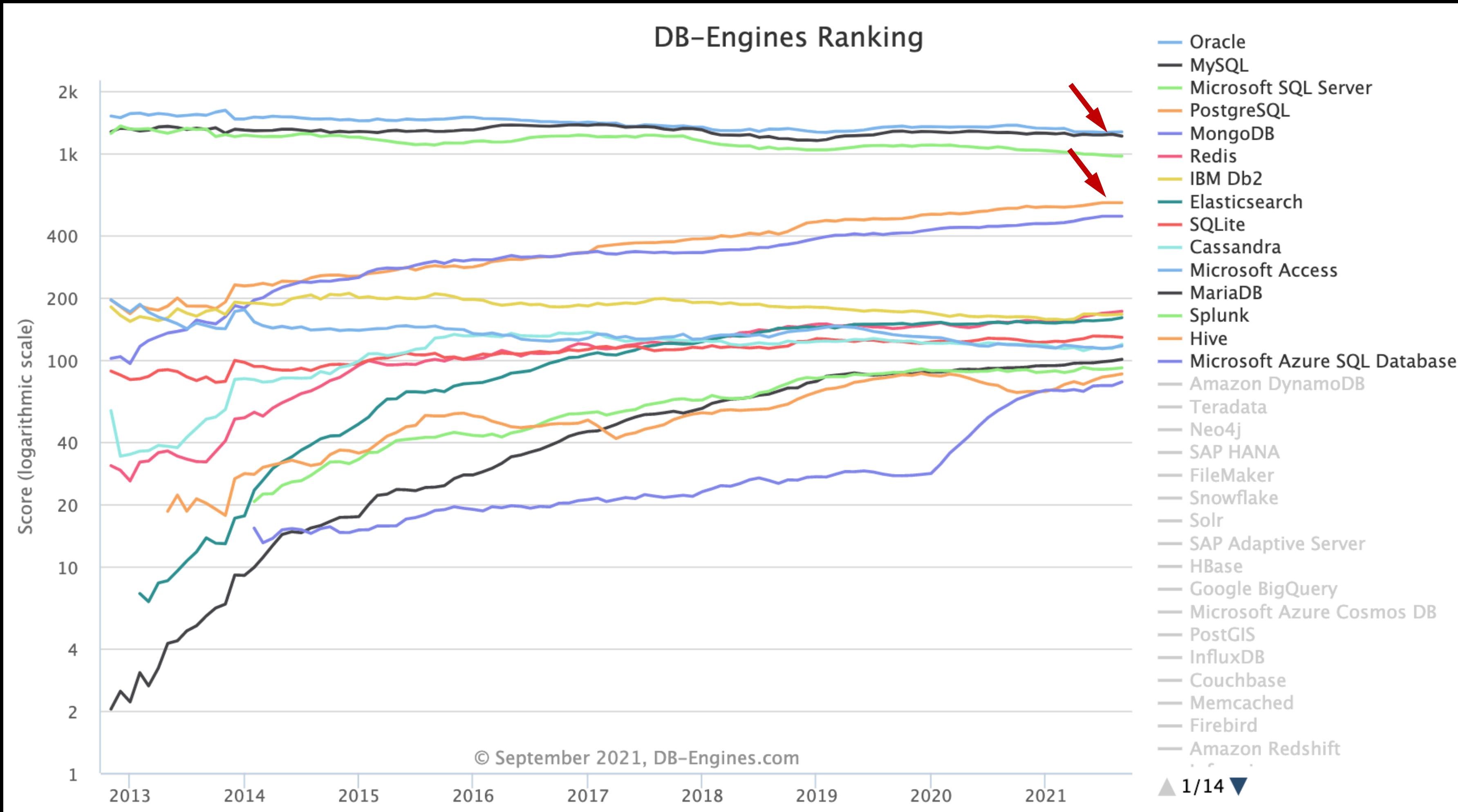
Database Management Systems

- E.g., Oracle, MS SQL Server, MySQL, PostgreSQL, (SQLite, Access)
- Often run on a remote, multi-user server
  - Typically, you need to know the hostname and have a username and password
- May be connected to one or more software applications or may be standalone
- Client libraries exist for every common programming language
  - But you usually query the database using the SQL language

Course Outline (subject to change)

- Data in detail
  - Numeric formats
    - Binary, integers, floats, precision
    - Dates and times
  - Text encodings
    - ASCII, UTF-8, special characters
  - Organizing data in files
    - CSV, XML, JSON
  - Messy data
    - Missing entries, fuzzy matching
- SQL relational databases
  - Data modelling
    - One-to-many, many-to-many relationships
    - Integrity & foreign key constraints
  - Structured Query Language (SQL)
    - Select, create table, update, delete
    - Joining tables
    - Subqueries & temporary tables
    - Indexes and execution plans
- Plus more if time permits

DB Engines for MSiA-413: **MySQL (SQLite), Postgres**



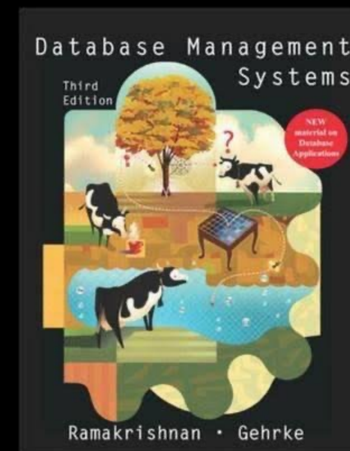
Questions about course content?

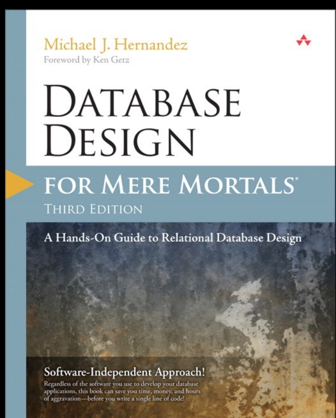
Course Logistics

- READ THE SYLLABUS !!! It is on Canvas
  - Pay particular attention to the **Academic Integrity** statements
  - Pay particular attention to the **Policy on Academic Work Missed Due to Illness**
- No exams, only homeworks (8 if time allows, roughly one a week)
  - Schedule is approximate. The number of homeworks may change
- Homeworks in groups of 2
  - All other group sizes (e.g., 1, 3) require justification and instructor approval
  - We will assign you into groups in the first week of classes
- Use slack for all offline questions regarding the course, not email
  - Please ask course questions openly if possible, so all can benefit from the answer
  - Use private messaging only for questions that reveal your answer
  - Easy to private-message the instructor, if needed
  - Please feel free to answer questions too; your classmates will be grateful!

Recommended Books (not required)

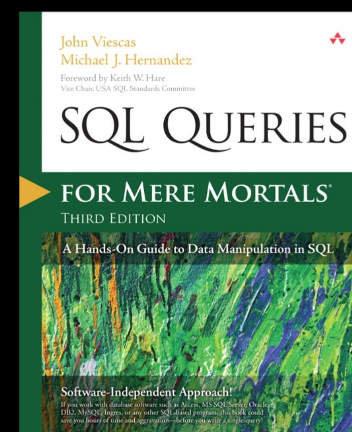
Ramakrishnan & Gehrke
"Database Management Systems"





Hernandez
"Database Design for Mere Mortals"

Viescas & Hernandez
"SQL Queries for Mere Mortals"



Questions about logistics?

Part 2:
Integer Representation

Computers store information in **binary**

- Ones and Zeros
- …000100100001001001110011011010101010111100000…
- Called "bits," meaning "**b**inary dig**its**"

- Why?
  - Simplicity
  - Noise robustness
  - By convention

- But how do we get meaning from a sequence of ones and zeros?

Data is zeros and ones plus **context**

- An **encoding** defines what the zeros and ones represent
- "01000100011000010111010001100001" can represent:
  - The number 1,147,237,473 as an **integer**
  - The number 901.8184 as a **float**
  - The four letters "Data" in the **ASCII** character encoding
  - This color (at 37% transparency) in **RGBA**
  - 32 separate True or False values
- Any crazy encoding is possible, but there are some standards

Integers

- Integers are the simplest of all data encodings
- Whole numbers only (no fractions)
- Numbers are represented directly in the "base two" positional notation
- The familiar "base ten" representation of numbers is just a convention due to the fact that humans have ten fingers
  - Still, Mayans/Aztecs used base-20 (vigesimal)
    - They had a "0", drawn as a shell
  - Babylonians used base-60 (sexagesimal)
    - Count to 12 using one hand only, with thumb pointing to each finger bone on the four fingers in turn
    - Mark that 12 by raising a finger on the other hand
    - $60 = 12 \times 5$
- What number base will octopuses evolve to use?

(drawing from http://drawingpencilarts.com/realistic-octopus-drawing/)

Integers in detail

Decimal 137_{ten}

```
   1        3      7
x10²   x10¹   x10⁰
100   + 30    + 7 = 137
```

← **powers of 10**

Binary $10001001_{two} = 137_{ten}$

```
   1     0     0     0     1     0     0     1
 x2⁷   x2⁶    x2⁵   x2⁴   x2³   x2²    x2¹   x2⁰
128   + 0   + 0   + 0   + 8 +  0   + 0   + 1 = 137
```

← **powers of 2**

Simple binary integers

Examples of 2^n

$1_{ten} = 1_{two}$

$2_{ten} = 10_{two}$

$4_{ten} = 100_{two}$

$8_{ten} = 1000_{two}$

$16_{ten} = 10000_{two}$

$32_{ten} = 100000_{two}$

$64_{ten} = 1000000_{two}$

$128_{ten} = 10000000_{two}$

Examples of 2^n-1

$3_{ten} = 11_{two}$

$7_{ten} = 111_{two}$

$15_{ten} = 1111_{two}$

$31_{ten} = 11111_{two}$

$63_{ten} = 111111_{two}$

$127_{ten} = 1111111_{two}$

$255_{ten} = 11111111_{two}$

There are only **10** types of people in this world… those who understand binary and those who don't.

Binary tricks

- Remember the first eight powers of two:
  - 2, 4, 8, 16, 32, 64, 128, 256
- Remember that $\mathbf{2^{10} = 1024} \approx 1000$
  - Lets you estimate the number of binary digits in a decimal integer: Every three decimal digits gives approximately ten binary digits
  - $2^{20} \approx$ 1 million (1,048,576)
  - $2^{30} \approx$ 1 billion (1,073,741,824)
- Remember the important large powers of two:
  - $2^8 = 256$
  - $2^{16} \approx$ 64 thousand
  - **$2^{32} \approx$ 4 billion**
  - $2^{64} \approx$ really big ($\approx$ 18 quintillion, or in CS parlance: 16 exa-...)
  - **2^{10}=kilo (Ki), 2^{20}=mega (Mi), 2^{30}=giga (Gi), 2^{40}=tera (Ti), 2^{50}=peta (Pi), 2^{60}=exa (Ei)**

Addition in binary

4 + 7 = 11

```
1 ← carry
  4
+ 7
1 1
```

100 + 111 = 1011

```
1 ← carry
  1 0 0
+ 1 1 1
1 0 1 1
```

More binary addition

63 + 98 = 161

```
1 1 ← carry
  6 3
+ 9 8
-----
1 6 1
```

11111 + 110010 = 1010001

```
1 1 1 1 1 ← carry
    1 1 1 1 1
+ 1 1 0 0 1 0
-------------
1 0 1 0 0 0 1
```

Subtraction: addition’s tricky pal

161 – 98 = 63

↝	↝	**borrow**
0~~1~~	**15**~~6~~	**1**1
-	9	8
	6	3

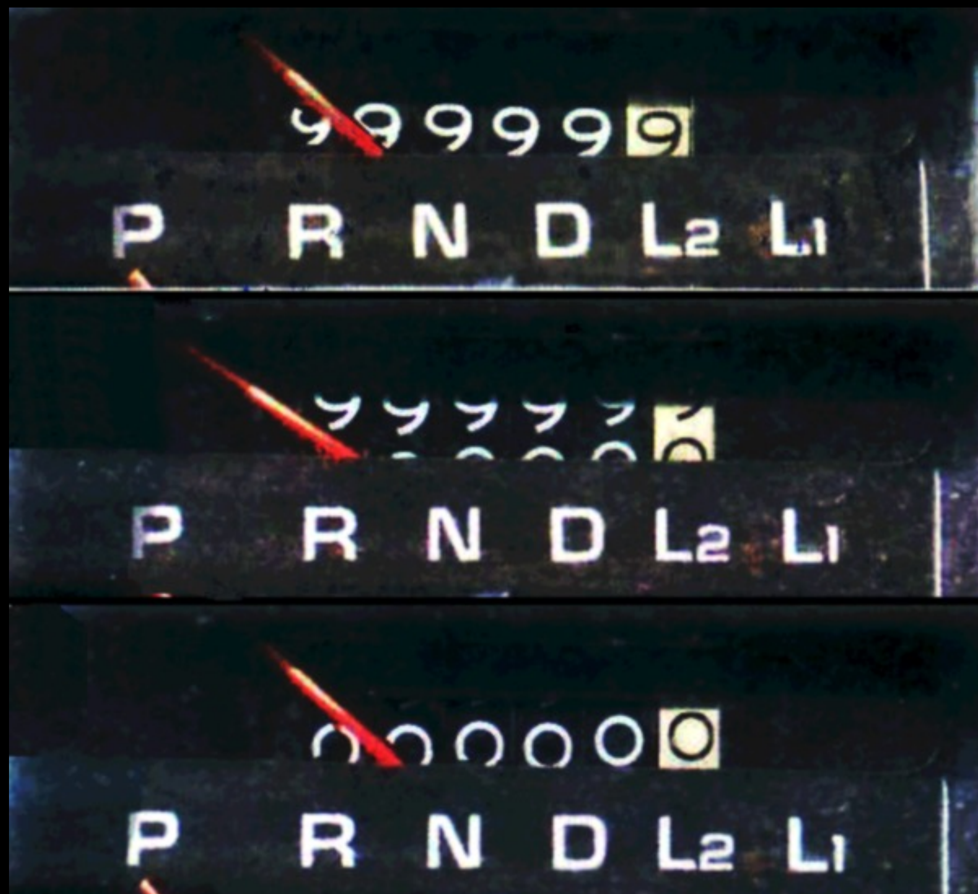
1010001 - 110010 = 11111

↝	↝		↝	↝	↝	**borrow**
0~~1~~	**1**~~0~~	**10**~~1~~	**1**~~0~~	**1**~~0~~	**1**0	1
-	1	1	0	0	1	0
		1	1	1	1	1

What about negative integers?

- **Signed integers** can represent both positive and negative integers
- We need an extra bit to represent the sign of the number
- But we don't just use a simple sign bit
- We use **two's complement** to represent negative numbers, because it
  - Simplifies the computer's addition and subtraction circuitry, and
  - It has just one representation of zero
- Negative numbers "roll over" from the top of the binary range

Works like an old-style car odometer



Two's complement for three-bit numbers

```
111
110
101
100
011
010
001
000
```

```
 3: 011
 2: 010
 1: 001
 0: 000
-1: 111
-2: 110
-3: 101
-4: 100
```

rollover

sign bit

```
1 – 2 =   1 + (-2) = -1
        001 +  110 = 111
```

Subtraction is done in the exact same way as addition!

No need to learn how to "borrow"

Subtraction works just like addition!

No need to learn how to "borrow."

Just negate the second number and add.

3 – 2 = 3 + (-2) =

```
1 1 ← carry
  0 1 1
+ 1 1 0
-------
1 0 0 1 ← our answer!
```

We ignore the final carry because it falls outside of the 3-bits we are working with. That's how we roll-over between negative and positive.

```
 3: 011
 2: 010
 1: 001
 0: 000
-1: 111
-2: 110
-3: 101
-4: 100
```

Two's complement negation

To negate a number and get its two's complement representation:

- **Flip** all the bits. Ones become zeros and zeros become ones.
- **Add one**
- **-x = ~x + 1**

For example -3

- Start with the bits for three: **`011`**
- Flip the bits: **`100`**
- Add one: **`101`**

```
 3: 011
 2: 010
 1: 001
 0: 000
-1: 111
-2: 110
-3: 101
-4: 100
```

Negating with Complement and Increment

- Claim: The following holds for 2's complement (when defined)
  - $\sim x + 1 = -x$
- Complement
  - Observation: $\sim x + x = 1111\ldots11_2 = -1$

x	1	0	0	1	1	1	0	1
+ ~x	0	1	1	0	0	0	1	0
-1	1	1	1	1	1	1	1	1

- Increment
  - $\sim x + 1 = \sim x + \boldsymbol{x} - \boldsymbol{x} + 1 = \cancel{-1} - x + \cancel{1} = -x$
  - $\sim x + 1 = -x$

Overflow: when numbers don't fit

For example, 2 + 2 = 4

4 cannot be represented in a three-bit **signed** integer.
What happens when we try this addition?

```
  1 ← carry
  0 1 0
+ 0 1 0
-------
  1 0 0 ← answer looks like -4!
```

- The computer will throw an **exception** if the signs of the operands were the same, but the sign of the result is different.
  - positive + negative cannot overflow
  - positive + positive should give a positive
  - negative + negative should give a negative
- Remember that the left-most bit indicates the sign.

```
 3: 011
 2: 010
 1: 001
 0: 000
-1: 111
-2: 110
-3: 101
-4: 100
```

Examples with 4 and 8 bits

4-bit is between -8 and 7

8-bit is between -128 and 127

(Stop and practice)

Reading Assignment and Practice

- Read "Representing Numbers in Computers" at http://www.stat.berkeley.edu/~nolan/stat133/Spr04/chapters/representations.pdf
- Practice converting numbers to and from binary
- Practice binary addition and subtraction (check with online tools)
- Browse data sets from Kaggle.com
  - Don't forget to click the "Data" tab
- Watch this video to see how addition is actually implemented in hardware https://www.youtube.com/watch?v=1I5ZMmrOfnA
  Search YouTube for "PBS ALU"