What do gambling, database design, your calculator, and human evolution have in common?

Limsoon Wong

Fun With Invariants

Limsoon Wong

Plan

- **What is an invariant?**
  - Bet on color of the bean
  - Make a list sorted
  - Take exponent faster
- **Where do Polynesians come from?**
- **Are Europeans of pure Neanderthal or pure Cro Magnon or mixed descent?**
- **What is a good database design?**

- **What will we learn?**
  - Problem solving by logical reasoning on invariants
  - Problem solving by rectifying violation of invariants
  - Solution optimization by preserving invariants

What is an invariant?

- **Suppose you have a bag of x red beans and y green beans**
- **Repeat the following:**
  - Remove 2 beans
  - If both green, discard both
  - If both red, discard one, put back one
  - If one green and one red, discard red, put back green
- **If one bean is left behind, can you predict its colour?**

Shall we bet on the color of the bean that is left behind?

Bet on the last green bean

Suppose you have a bag of x red beans and y green beans	**When the parity of green beans is odd, it remains odd…**
Repeat the following:	**Start with y=2n+1**
– Remove 2 beans	
– If both green, discard both	**y=2n+1 → y=2n-1**
– If both red, discard one, put back one	**y=2n+1 → y=2n+1**
– If one green and one red, discard red, put back green	**y=2n+1 → y=2n+1**
If one bean is left behind, can you predict its colour?	**It must be green!**

Bet on the last red bean

- **Suppose you have a bag of x red beans and y green beans**
- **Repeat the following:**
  - Remove 2 beans
  - If both green, discard both
  - If both red, discard one, put back one
  - If one green and one red, discard red, put back green
- **If one bean is left behind, can you predict its colour?**

- **When the parity of green beans is even, it remains even…**
- **Start with y=2n**
- **y=2n → y=2n-2**
- **y=2n → y=2n**
- **y=2n → y=2n**
- **It must be red!**

Bet on color of the last bean … and win!

- **Suppose you have a bag of x red beans and y green beans**
- **Repeat the following:**
  - Remove 2 beans
  - If both green, discard both
  - If both red, discard one, put back one
  - If one green and one red, discard red, put back green
- **If one bean is left behind, can you predict its colour?**

- **If you start with odd # (even #) of green beans, there will always be an odd # (even #) of green beans in the bag**

⇒ **Parity of green beans is invariant**

⇒ **Bean left behind is green iff you start with odd # of green beans**

- **What have we just seen?**

- **Problem solving by logical reasoning on invariants**

- **What is a sorted list?**

  A list L is sorted iff L[i] ≤ L[j] for all adjacent positions i, j

- **So how do you make a list M become sorted?**

  While M[i] > M[j] for some adjacent positions i, j {

  swap M[i], M[j]

  }

What makes a list a sorted list?

Sorting a list

- Invariant of sorted lists

A list L is sorted iff L[i] ≤ L[j] for all adjacent positions i, j

- Making a list M become sorted:

```
While M[i] > M[j] for some adjacent positions i, j {
    swap M[i], M[j]
}
```

- Find violation of the invariant
- Fix it
- When no more violation, the list must be sorted!

- **What have we just seen?**
- **Problem solving by rectifying violation of invariants**

- What does this program do?

  F(a, 0) = 1
  F(a, n+1) = a * F(a, n)

- We see that

  F(a, n) = a^n

Exponentiation

$$a^n = \underbrace{a \times \cdots \times a}_{n},$$

Playing the invariant…

- What does this program do?

  F(a, 0) = 1
  F(a, n+1) = a * F(a, n)

- We see that

  defn

  F(a, n) = a^n

  invariant

- Then

  by inv

- F(a, 2*n) = a^{2*n}
  = a^n * a^n
  by inv
  = y * y where y = F(a, n)

  by inv

- F(a, 2*n+1) = a^{2*n+1}
  = a * a^n * a^n
  = a * y * y where F(a, n)

  by inv

- So we get …

What's the difference?

- **Original program:**

  F(a, 0) = 1
  F(a, n+1) = a * F(a, n)

- **Cost of F(a, n) = n**

- **New program:**

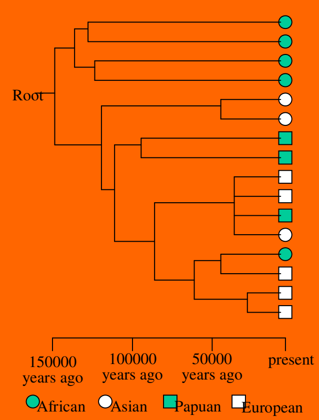
  F(a, 0) = 1
  F(a, 1) = a
  F(a, n) = if n is odd
  then a * y * y
  else y * y
  where y = F(a, n div 2)

- **Cost of F(a, n) = $\log_2$ n**

n	log n	call sequence		
8	3	4	2	1
9	3	4	2	1
10	3	5	2	1
11	3	5	2	1

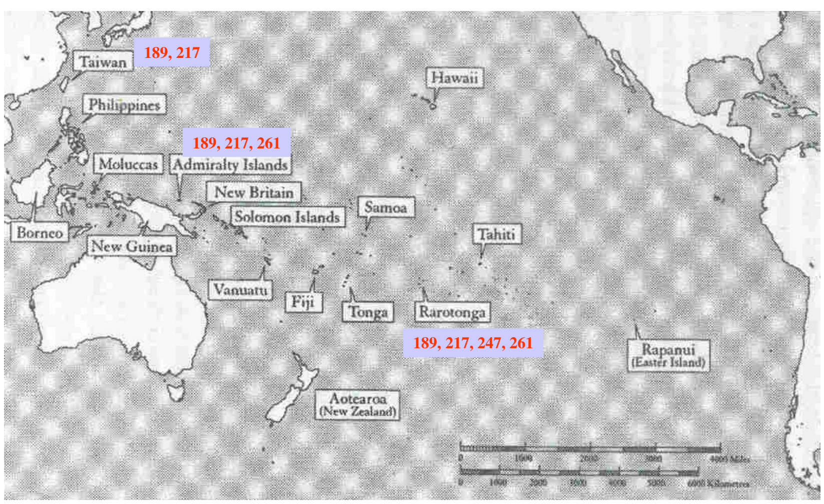
- **What have we just seen?**

- **Optimizing a solution by preserving invariant**

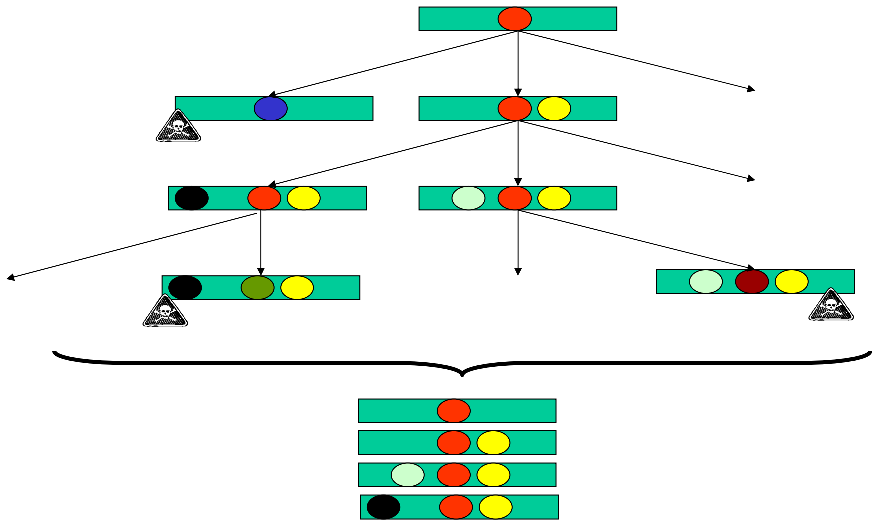


Where do Polynesians come from?

Do Polynesians come from Asia or America?



In the course of evolution…



What is the invariant?

- **Mitochrondrial DNA accumulates 1 mutation about every 10,000 years**
- **Human history is not so long relative to this**

⇒ **When a nucleotide in mitochrondrial DNA is mutated it stays mutated through future generations**

Origin of Polynesians

- Common mitochondrial control seq from Rarotonga have variants at positions 189, 217, 247, 261. Less common ones have 189, 217, 261
- Seq from Taiwan natives have variants 189, 217
- Seq from regions in betw have variants 189, 217, 261.
- More 189, 217 closer to Taiwan. More 189, 217, 261 closer to Rarotonga
- 247 not found in America

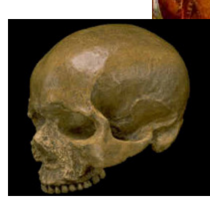
⇒ Polynesians came from Taiwan!

- Taiwan seq sometimes have extra mutations not found in other parts

⇒ These are mutations that happened since Polynesians left Taiwan!

Are Europeans descended purely from Cro Magnons? Purely from Neanderthals? Or mixed?


Neanderthal

Cro Magnon

Neanderthal vs Cro Magnon

- Based on palaeontology, Neanderthal & Cro Magnon last shared an ancestor 250,000 yrs ago
- Mitochondrial DNA accumulates 1 mutation per 10,000 yrs

⇒ If Europeans have mixed ancestry, the mitochondrial DNA betw 2 Europeans should have ~25 diff w/ high probability

- The number of diff betw Welsh is ~3, & at most 8.
- When compared w/ other Europeans, 14 diff at most

⇒ Ancestor either 100% Neanderthal or 100% Cro Magnon

- Mitochondrial DNA from Neanderthal have 26 diff from Europeans

⇒ Ancestor must be 100% Cro Magnon

The "Invariant" Perspective

- The invariant:

  When a nucleotide in mitochrondrial DNA is mutated it stays mutated through future generations

- The lesson learned:

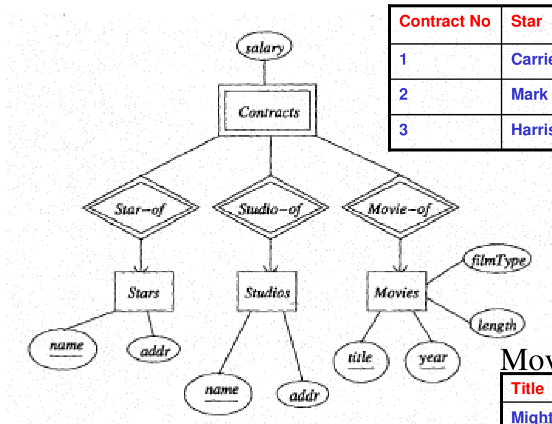
Figure out origins of Polynesians and Europeans by logical reasoning on invariant

What is a good database design?

Relational Data Model

- **Data are represented as a two-dimensional table**
- **It is a logical representation, not a physical representation**
  - Ordering of the rows is irrelevant
  - Ordering of the columns is irrelevant
  - How the rows and columns of a table are stored is irrelevant
  - …

Example



Contracts

Contract No	Star	Studio	Title	Salary
1	Carrie Fisher	Fox	Star Wars	$$$
2	Mark Hamill	Fox	Star Wars	$$$
3	Harrison Ford	Fox	Star Wars	$$$

Stars

Name	Address
Carrie Fisher	Hollywood
Mark Hamill	Brentwood
Harrison Ford	Beverly Hills

Movies

Title	Year	Length	Film Type
Mighty Ducks	1991	104	Color
Wayne's World	1992	95	Color
Star Wars	1977	124	Color

Design Issues

- **How many possible alternate ways to represent movies using tables?**
- **Why this particular set of tables to represent movies?**
- **Indeed, why not use this alternative single table below to represent movies?**

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

Anomalies

- What's wrong with the "Wrong Movies" table?

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **Redundancy**: Unnecessary repetition of info
- **Update anomalies**: If Star Wars is 125 min, we might carelessly update row 1 but not rows 2 & 3
- **Deletion anomalies**: If Emilio Estevez is deleted from stars of Mighty Ducks, we lose all info on that movie

Functional Dependency

- **Functional dependency** ($A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$)
  – If two tuples of a table R agree on attributes $A_1, \ldots, A_n$, then they must also agree on attributes $B_1, \ldots, B_m$
- Example: Title, Year → Length, Film Type, Studio
- FD ($A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$) is trivial if a B_i is an A_j

Can you identify the FD's here?

Wrong Movies

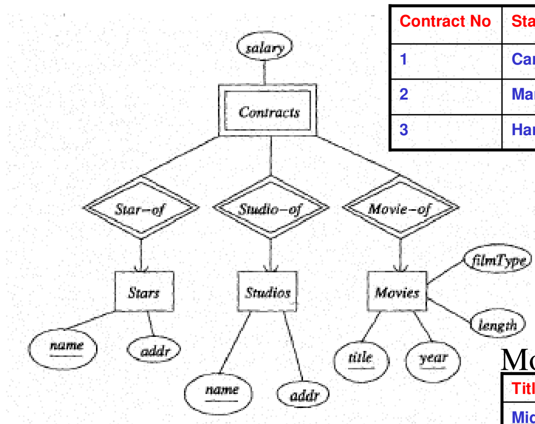
Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **Some FD's:**
  - Title, Year → Length
  - Title, Year → Film Type
  - Title, Year → Studio

Keys

- **Key**
  - A minimal set of attributes $\{A_1, \ldots, A_n\}$ that functionally determine all other attributes of a table
  - A key is trivial if it comprises the entire set of attributes of a table
- **Superkey**
  - A set of attributes that contains a key

Can you identify the keys here?



Contracts

Contract No	Star	Studio	Title	Salary
1	Carrie Fisher	Fox	Star Wars	$$$
2	Mark Hamill	Fox	Star Wars	$$$
3	Harrison Ford	Fox	Star Wars	$$$

Stars

Name	Address
Carrie Fisher	Hollywood
Mark Hamill	Brentwood
Harrison Ford	Beverly Hills

Movies

Title	Year	Length	Film Type
Mighty Ducks	1991	104	Color
Wayne's World	1992	95	Color
Star Wars	1977	124	Color

Can you identify the superkeys here?

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **Superkeys :**
  - Any set of attributes that contains {Title, Year, Star} as a subset

Boyce-Codd Normal Form

- **A relation R is in Boyce-Codd Normal Form iff whenever there is a nontrivial FD (A_1, …, A_n → B_1, …, B_m) for R, it is the case that {A_1, …, A_n} is a superkey for R**
- Theorem A1 (Codd, 1972)

  A database design has no anomalies due to FD iff all its relations are in Boyce-Codd Normal Form

How is BCNF violated here?

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **A nontrivial FD:**
  - Title, Year → Length, Film Type, Studio
  - The LHS not superset of the key {Title,Year, Star}

  ⇒ Violate BCNF!
- **Anomalies are due to FD's whose LHS is not superkey**

Towards a Better Design

- **Use an offending FD ($A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$) to decompose $R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_h)$ into 2 tables**
  - $R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$
  - $R_2(A_1, \ldots, A_n, C_1, \ldots, C_h)$

No update anomaly

No redundant info

Title	Year	Length	Film Type	Studio
Star Wars	1997	124	Color	Fox
Mighty Ducks	1991	104	Color	Disney

No deletion anomaly

	Year	Star
Star Wars	1997	Carrie Fisher
Star Wars	1997	Mark Hamill
Star Wars	1997	Harrison Ford
Mighty Ducks	1991	Emilio Estevez

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

The "Invariant" Perspective

- **The invariant:**

  FD's ($A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$) are "invariants" of the database, as $\{A_1, \ldots, A_n\}$ determines $\{B_1, \ldots, B_m\}$

- **The lesson learned:**

Deliver a better database design by fixing violated invariants

What have we learned?

What have we learned?

- **Invariant is a fundamental property of many problems**
- **Paradigms of problem solving**
  - Problem solving by logical reasoning on invariants
  - Problem solving by rectifying violation of invariants
  - Solution optimization by preserving invariants

I didn't get to telling you yet, but …

- **Every time you write a loop in a program, it involves an invariant**
- **Every time you do a recursive function call, it involves an invariant**
- **Every time you do an induction proof, it involves an invariant**
- **… Computing is about discovering, understanding, exploiting, and having fun with invariants!**