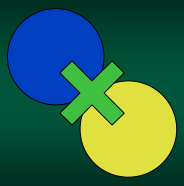
# Relations in Computer Science

Part 4B

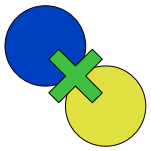


# Cross Products & Databases

SQL is set notation

## Products

- We are in the "Information Age" where knowledge is now computerized
- Information is stored in databases
- These systems are based on tuples and sets





## Fields

- *Fields* contain the smallest unit of data
  - e.g. Number, Text
  - So, each can be seen as a tuple (it can be a set, but rarely so)
- Each field has a unique field name
  - Name
  - Age
  - etc....



## Records

- A *record* is a set of data fields
  - represents a logical group of data
  - these include related numbers, text, images, etc...
- Examples
  - Course: department, number, section
  - Student: name, age, class
  - Computer: brand, speed, cost, etc...



## Database Example

| First | Last | Major | Greek |
|---|---|---|---|
| Peter | Griffen | Und | Tappa Kegga Bru |
| Joe | Gunchy | CSc | Cuppa Kappa Chino |
| Stimpy | Cat | Record | Hu Delta Phart |
| Eric | Cartman | Bus | Eta Lotta Pi |

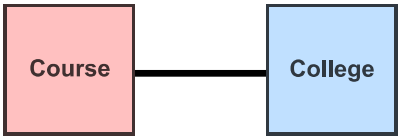
## Relationships & Cardinality

- Relationship
  - how tables are associated with each other
  - e.g. student records and class records
- Related tables are *joined* which performs a cross product on two tables
- Restrictions are used to eliminate unneeded records



## One-to-One Relationship

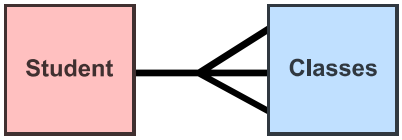
- One record is related to another single record
- e.g. Course and its Department






## One-to-Many Relationship

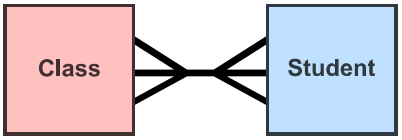
- One record is related to many records
- e.g. Students and their classes






## Many-to-Many Relationship

- One record is related to many records & vice-versa
- e.g. Class and students






## Locating Specific Data

- A query language is used to:
  - Locate information
  - Sort records
  - Change data in records
- Examples:
  - SQL (Structured Query Language)
  - Natural language queries – not used that much



## SQL Inner Join

```
SELECT A.x, B.y
FROM A
INNER JOIN B ON A.j = B.j
WHERE A.n = "Moe"
```

## SQL Inner Set Notation (simplified)

```
{ (x, y) |
    x ∈ A and y ∈ B and
    j ∈ A and j ∈ B and
    n ∈ A and n = "Moe"   }
```



## Abstract Data Types

What *int* really means

## Application of Sets

- An *abstract data type* is a set of values and operations (functions) on those values
- This is the basis for all objects, class, structures, etc….



## Integer Example

- In the code below, *int* is an ADT found in most programming languages
- It declares a variable *n* of type *int*
- n represents a value from int's set of values

```
int n;
```



## The Domain

- int is defined (normally) as 32-bit
- Set is { $-2^{31}$, …, $(2^{31} - 1)$ }
- So int ⊂ Z

```
int n;
```



## Operations

- It also defines that n can be manipulated by via functions +, -, ×, ÷
- Sometimes languages are different (division for example)

```
÷ : Z, Z → Z   in Java, C++, C#
÷ : Z, Z → R   in Visual Basic
```

# Functions

Notation varies, but logic the same

## Functions

- Many programming languages support customized functions
- The format often mirrors the discrete notation



## C Family

- C programming language includes C++, Java, and C#
- Notation is very terse

```
name : int → int

int name(int n)
```



## Visual Basic

- Visual Basic evolved from the original BASIC programming language
- Notation is far more verbose

```
name : Integer → Integer

Function name(n As Integer) As Integer
```



## Pascal

- Pascal was very popular in the 1980's and 90's
- Created many concepts that were integrated into other languages

```
name : integer → integer

function name(n : integer) : integer
```



## Swift

- Swift was created by Apple to replace older Objective-C
- Influenced by multiple languages

```
name : Int → Int

func name(n : Int) -> Int
```