

System.Environment

Graphics Example

Rubberband Boxes

Object

Public methods

<code>Equals</code>	Determines whether two instances are equal
<code>GetHashCode</code>	Hash function
<code>GetType</code>	Gets type of current instance
<code>ReferenceEquals</code>	Determines whether two instances are the same
<code>ToString</code>	Returns a string representation of the object

Protected methods

<code>Finalize</code>	Allows cleanup before garbage collection
<code>MemberwiseClone</code>	Makes a shallow copy

What do == and Equals do?

Value types

Reference types

Compare the values

Strings: compare values

`==` cannot be used on structs unless explicitly implemented. Many built-in types do this, e.g., `Point`.

Other: compare references unless overridden to give value semantics

Implementing Equals for your classes

For some classes, you may want `Equals` to test for equal values rather than equal references. It's up to you to decide what "equal" means, but here are some rules:

<code>x.Equals(x)</code>	returns true
<code>x.Equals(y)</code>	returns the same result as <code>y.Equals(x)</code>
<code>x.Equals(y) && y.Equals(z)</code>	returns true only if <code>x.Equals(z)</code> returns true
<code>x.Equals(y)</code>	continues to return the same value if the objects referenced by <code>x</code> and <code>y</code> are not modified
<code>x.Equals(null)</code>	returns false

Do not throw exceptions in `Equals`.

Override `GetHashCode` whenever you override `Equals`.

Guidelines for implementing Equals and ==

- Reference types for which you want "value semantics" should override `Equals`. Examples might be classes like `Rational`, or `Point3D`, and classes like these should implement `==` as well.

- `System.ValueType` provides a default implementation for `Equals` for value types, but you may be able to improve performance with a custom implementation.

- In general, override `Equals` whenever you implement `==`, and make them do the same thing. This makes code using `Dictionary`, `List`, etc. behave the same as user code written using `==`.

- If `==` is overloaded, `!=` must be overloaded also.

Databases and Data Representation

Database Management System (DBMS):

Provides **efficient, convenient, and safe multi-user storage** of and access to **massive amounts of persistent data**

Provides a **programming interface** that allows a user or program to

- Create new databases and specify their structure
- Query and modify the data

Early DBMSs were ad hoc, with no two the same

Now one approach predominates: **relational databases** and **SQL (Structured Query Language)**

Databases and Data Representation

A "relation" is a table of data

1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80

Databases and Data Representation

The columns are known as "attributes"

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80

Databases and Data Representation

The rows are called "tuples"

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80

Databases and Data Representation

It is allowable for some values to be missing

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	
4	Julie	CS106B	80

Databases and Data Representation

We can add, remove, or update tuples

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

Databases and Data Representation

Each attribute has an underlying domain, or data type

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

int string string int

SQL Database

We will generally refer to the relations, attributes, and tuples as tables, columns, and rows

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

int string string int

SQL Database

The structure of the table is referred to as its **schema**

Lecturers(LectID, Name, Course, Students)

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

int string string int

SQL Database

The structure of the table is referred to as its **schema**

Lecturers(LectID, Name, Course, Students)

SQL Database

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

SQL Database

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

Primary Key: no two tuples can have the same LectID
LectID cannot be null
In this table, lecturers can only teach one course

LectID	Name	Course	Students	Building	Room
1	Bob	CS193N	108		
2	Patrick	CS193C	60		
3	Nick	CS193I	100		
4	Julie	CS106B	80		
5	Mehran	CS106A	150		

Suppose that we want to add data about lecturers offices.

LectID	Name	Course	Students	Building	Room
1	Bob	CS193N	108		
2	Patrick	CS193C	60		
3	Nick	CS193I	100		
4	Julie	CS106B	80		
5	Mehran	CS106A	150		

Suppose that we want to add data about lecturers offices.
But suppose that lecturers can have more than one office.
We don't know how many columns to add, and we can't add additional rows for the same lecturers without violating the primary key constraint, so we use another table.

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

This table can have multiple rows per lecturer. But rather than store the names, ...

Offices

	Building	Room
	Bob	Gates 178
	Bob	Quad 1
	Mehran	Gates 189
	Nick	EE 120

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

This table can have multiple rows per lecturer. But rather than store the names, we will use LectIDs.

Offices

LectID	Building	Room
1	Gates	178
1	Quad	1
5	Gates	189
3	EE	120

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

We can use LectIDs to look up information in the Offices table. LectID establishes a relationship between the tables

Offices

LectID	Building	Room
1	Gates	178
1	Quad	1
5	Gates	189
3	EE	120

Relation

Lecturers

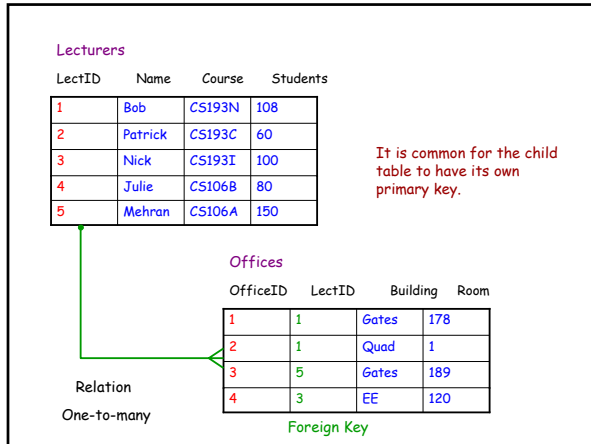
LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

We say that there is a "foreign key constraint" between the tables.
The attribute referenced in the parent table must be a primary key.
Every value in the foreign key column must actually appear in the parent table.

Offices

LectID	Building	Room
1	Gates	178
1	Quad	1
5	Gates	189
3	EE	120

Relation
One-to-many
Foreign Key



Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

SELECT returns tuples that satisfy some condition

Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

SELECT Name, Course FROM Lecturers WHERE Students > 100

Name	Course
Bob	CS193N
Mehran	CS106A

Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

SELECT Name, Course FROM Lecturers WHERE Students > 100

Name	Course
Bob	CS193N
Mehran	CS106A

Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

SELECT * FROM Lecturers WHERE Students > 100

LectID	Name	Course	Students
1	Bob	CS193N	108
5	Mehran	CS106A	150

Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

SELECT * FROM Lecturers WHERE Students > 100

- ← What attributes to output
- ← What relations are involved
- ← What tuples are of interest

Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

```
SELECT *
FROM Lecturers
WHERE Students > 100
OR Name = 'Nick'
```

LectID	Name	Course	Students
1	Bob	CS193N	108
3	Nick	CS193I	100
5	Mehran	CS106A	150

Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

```
SELECT Name, Course
FROM Lecturers
WHERE Name LIKE '%ick%'
```

Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

```
SELECT Name, Course
FROM Lecturers
WHERE Name LIKE '%ick%'
```

Single quotes

Wildcard: % matches any number of characters

Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

```
SELECT Name, Course
FROM Lecturers
WHERE Name LIKE '%ick%'
```

Name	Course
Patrick	CS193C
Nick	CS193I

Wildcard: % matches any number of characters

Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

```
SELECT Name, Course
FROM Lecturers
WHERE Name LIKE '%ick%'
```

Name	Course
Patrick	CS193C
Nick	CS193I

Wildcard: % matches any number of characters
 _ matches any single character
 [] matches any single character in set or range
 [^] matches any single character not in set or range

Simple SQL Queries

Lecturers

LectID	Name	Course	Students
1	Bob	CS193N	108
2	Patrick	CS193C	60
3	Nick	CS193I	100
4	Julie	CS106B	80
5	Mehran	CS106A	150

```
SELECT Name, Course, Students
FROM Lecturers
ORDER BY Students
```

Name	Course	Students
Patrick	CS193C	60
Julie	CS106B	80
Nick	CS193I	100
Bob	CS193N	108
Mehran	CS106A	150

