Relational Databases and Web Integration

Dr. Carl Pulley
c.j.pulley@hud.ac.uk

Example: Class Diagram

Example: Rails

- Some shell commands:

```
rails example1
..
cd example1
script/generate scaffold question number:integer statement:text paper_id:integer
..
script/generate scaffold paper author:string name:string
..
..
```

- Now edit the question and paper models:

```
class Question < ActiveRecord::Base
  belongs_to :paper
end

class Paper < ActiveRecord::Base
  has_many :questions
end
```

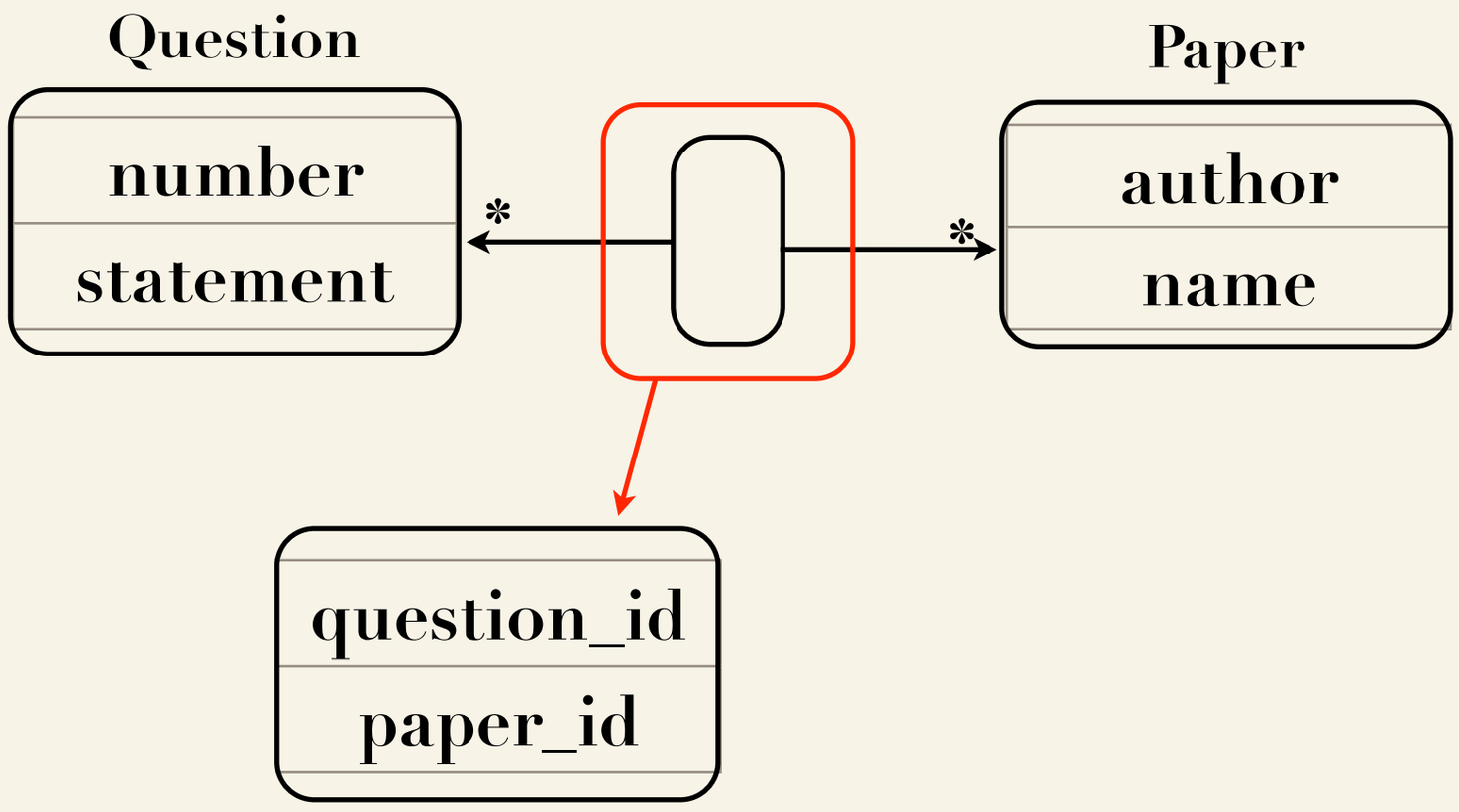
Example Refined

- **Limitation:** all papers have distinct questions!
- Surely one of the points of having questions in a database is so that we can reuse them?
  - ie. questions can belong to multiple papers!
- We need to model a many-to-many relationship

Many-to-Many Relationships

- Such relationships are collections of **pairs**
  - 1st element is question, 2nd is paper
  - we need an **extra** table to implement this!

Many-to-Many Relationships



Solution: Rails

- Some shell commands:

```
rails example2
..
cd example2
script/generate scaffold question number:integer statement:text
..
script/generate scaffold paper author:string name:string
..
script/generate migration papers_questions question_id:integer paper_id:integer
..
```

- Now edit the question and paper models:

```
class Question < ActiveRecord::Base
  has_and_belongs_to_many :papers
end

class Paper < ActiveRecord::Base
  has_and_belongs_to_many :questions
end
```


Note: the table-pair name is the alphabetic ordering of the (pluralized version) table names!

Solution: Rails

- Finally edit the newly created migration so that it builds/drops your `papers_questions` table

```
def self.up
  create_table :papers_questions do |t|
    t.integer :question_id
    t.integer :paper_id

    t.timestamps
  end
end

def self.down
  drop_table :papers_questions
end
```


Note: by using model instead of migration in the script/generate, you could have the migration correctly instantiated automatically. However, you then need to get rid of unwanted models, fixtures and unit tests!

Unit Tests

- No paper should have no questions!

```
def test_non_empty
  Paper.find(:all).each do |paper|
    assert(paper.questions.count > 0)
  end
end
```

- Papers should have author names built from characters [a-zA-Z.-]+

```
def test_author
  Paper.find(:all).each do |paper|
    # assert(paper.author =~ /^[a-zA-Z\.\- ]+$/)
    assert_match(/^[a-zA-Z\.\- ]+$/, paper.author)
  end
end
```

More Unit Tests

- Every question should belong to at least one paper

```
def test_has_paper
  Question.find(:all).each do |question|
    assert(question.papers.count > 0)
  end
end
```

- No two questions should be the same!

```
def test_uniqueness
  Question.find(:all).each do |q1|
    Question.find(:all).each do |q2|
      assert(q1 == q2 or q1.statement != q2.statement)
    end
  end
end
```

Test Fixtures

- When testing databases, you often need to populate them for testing purposes
- Fixtures are how we do this in Rails
- Fixtures are loaded at the start of a unit test
  - thus populating the test database **before** running any of your unit tests!
- Console access to *test* environment via:
  - `script/console test`

Test Fixtures: Examples

```
cis2360-main:
  author: Carl Pulley
  name: CIS2360 Summer Paper
  questions: q1, q2, q3, q4

cis2360-retake:
  author: Carl Pulley
  name: CIS2360 Autumn Paper
  questions: q2, q4, q5, q6
```

```
DEFAULTS: &DEFAULTS
  number: <%= rand(100) %>
  statement: To be completed...

q1:
  number: 20081
  statement: Super-tricky question...
q2:
  number: 20082
  <<: *DEFAULTS
q3:
  <<: *DEFAULTS
q4:
  <<: *DEFAULTS
q5:
  <<: *DEFAULTS
q6:
  <<: *DEFAULTS
```