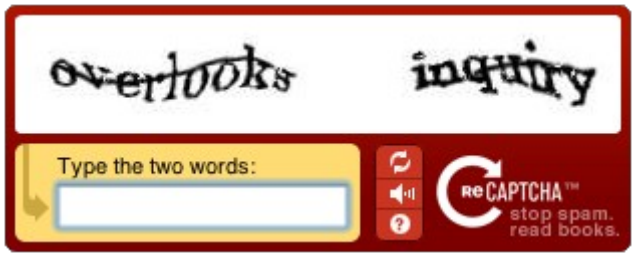
CAPTCHAs And other APIs

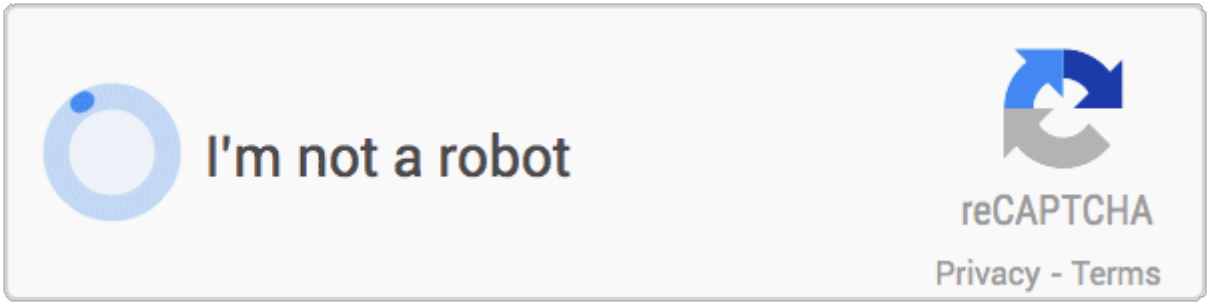
CS518

Dr. Justin F. Brunelle

CAPTCHA, reCAPTCHA

- Completely Automated Public Turing test to tell Computers and Humans Apart
- Prevents spam, blocks robots
- Verifies human vs machine





How it works

- Server-delivered challenges
- User provided answers
- Server-side validation
- Transactions via Ajax

Provenance

- Carnegie Mellon Research Project
- Two words:
  - One known (i.e., some threshold of user agreement reached)
  - One unknown (i.e., user agreement not reached)
- Known-term validation creates a “vote” for unknown term
- Known term is the only evaluation criteria

Example – Setup

- See: https://developers.google.com/recaptcha/intro
- wget [library.zip]
- unzip [library.zip]
- Register for a key https://www.google.com/recaptcha/admin
- Follow instructions

Example – Code

```
<!-- include the library -->

<script src="https://www.google.com/recaptcha/api.js" async defer></script>

<!-- form for the reCAPTCHA →
<form action="?" method="POST">
      <div class="g-recaptcha" data-sitekey="**********"></div>
      <br/>
      <input type="submit" value="Submit">
    </form>
```

http://www.cs.odu.edu/~jbrunelle/cs518/examples/captcha/

General recommendations for using APIs

- Find canonical and authoritative sources/documentation
- Include libraries
- Follow instructions for hello world
- Adapt

API Information

- Software-to-software actions
- Formal definitions, interactions
- HTTP for data
-

API Design Guidelines

- Use REST
- Scope data as necessary
  - Large amounts only when expected
  - Use pagination when available
- Use standard (and documented) formats

Keys

- Tokens for access, prevent abuse, track usage, etc.
- Sending token is like sending a session ID
  - Useful for authentication
  - Revoking for expired access
- Public-Private pairs for signing

JSON

- JavaScript Object Notation
- Emerging (emerged?) standard format
- Like XML but less verbose
- Major languages support encoding:
  - PHP's json_decode, json_encode, json_last_error

JSON example

```
{
    “opponent”: “Hampton”,
    “score”: “19”,
    “teamInfo”: {
                “mascot”: “pirate”,
                “nickname”: “pirates”,
                “division”: “FCS”
            },
    “roster”: [
            {
                “name”: “Justin”,
                “position”: “bench warmer”,
                “number”: “00”
            },
            {
                “name”: “Johnny”,
                “position”: “QB”,
                “number”: “1”
            }
        ]
}
```

GitHub APIs

- Documentation:
  - https://developer.github.com/v3/
- curl -iL "https://github.com/login/oauth/authorize?client_id=XXXX6&scope=repo&state=jbrunelle"

GitHub API: Access

- Register you app (website) at github.com
  - Gives you client id and client secret
- Authorize a user with your client_id, scope, and redirect URI
- Retrieve access token as form of authentication

Gravatar

- Avatar sharing
- No authentication
- Img src: https://www.gravatar.com/avatar/" . md5(strtolower(trim($email))) . "?&s=” . $size;
- https://en.gravatar.com/site/implement/images/php/

Searching

- Relational data base looks for specific string matching
  SELECT Date

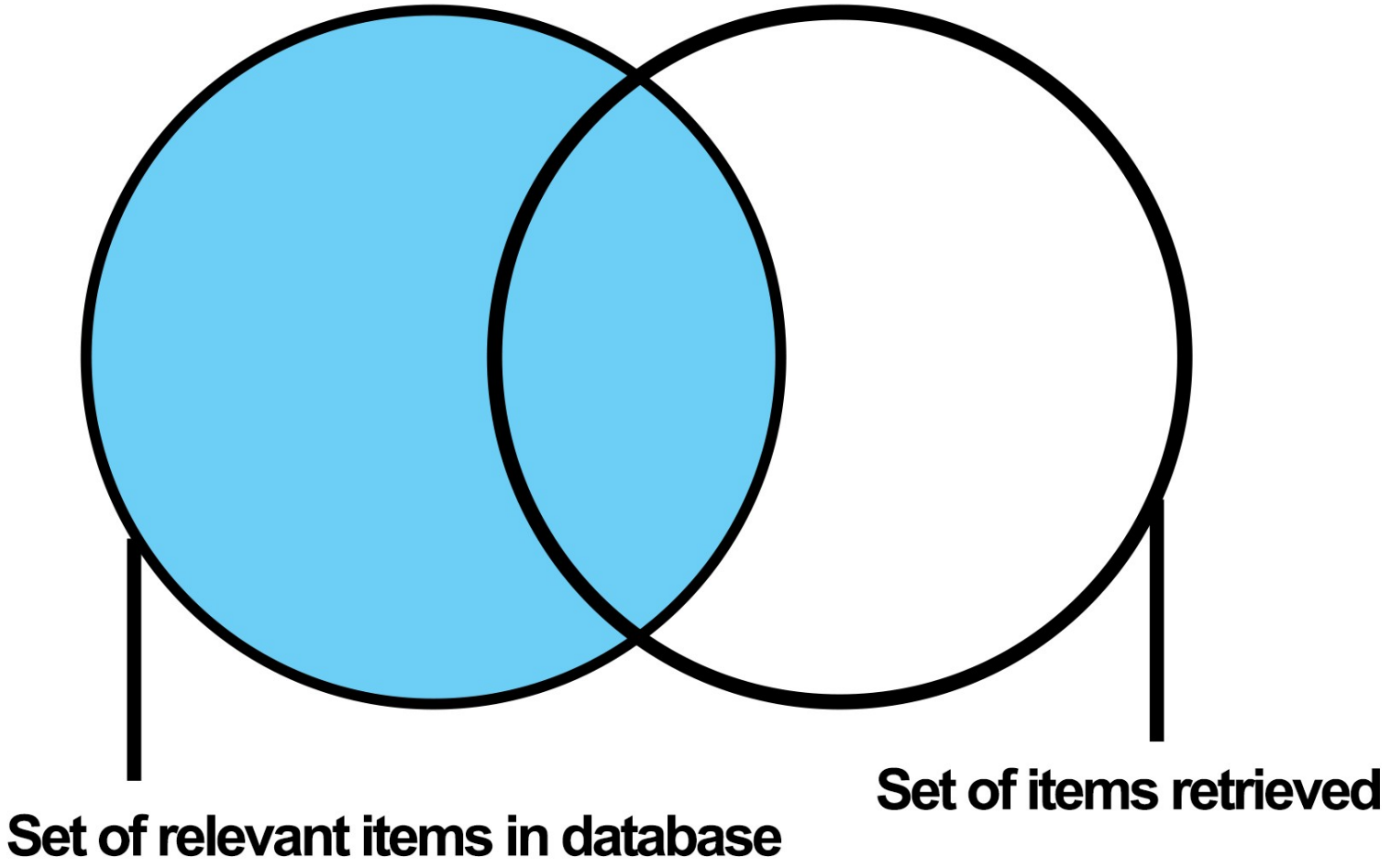
  FROM opponents

  WHERE name = "Hampton"
  AND score = “19”;

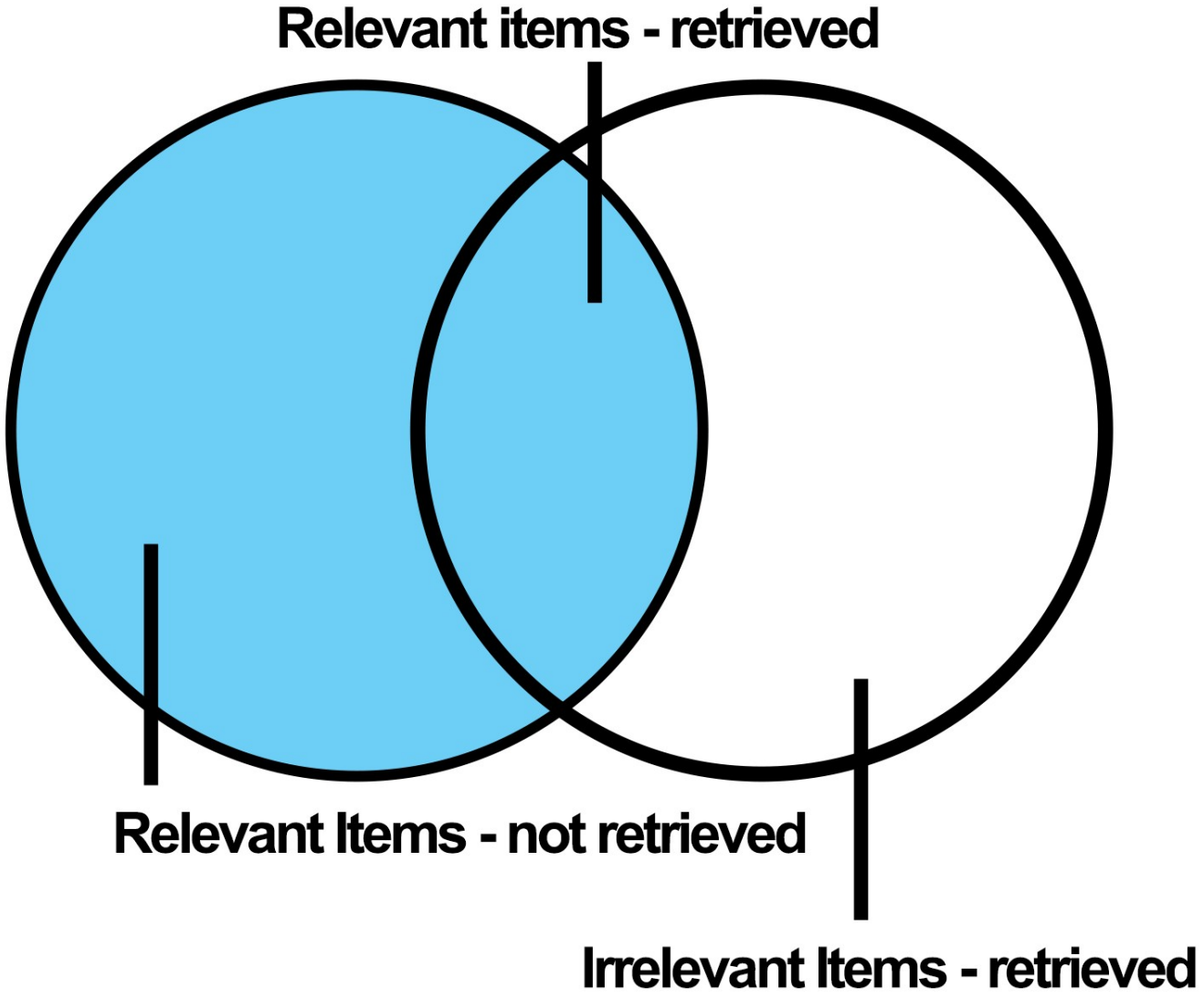
Versus...

- http://www.espn.com/college-football/game?gameId=400869346

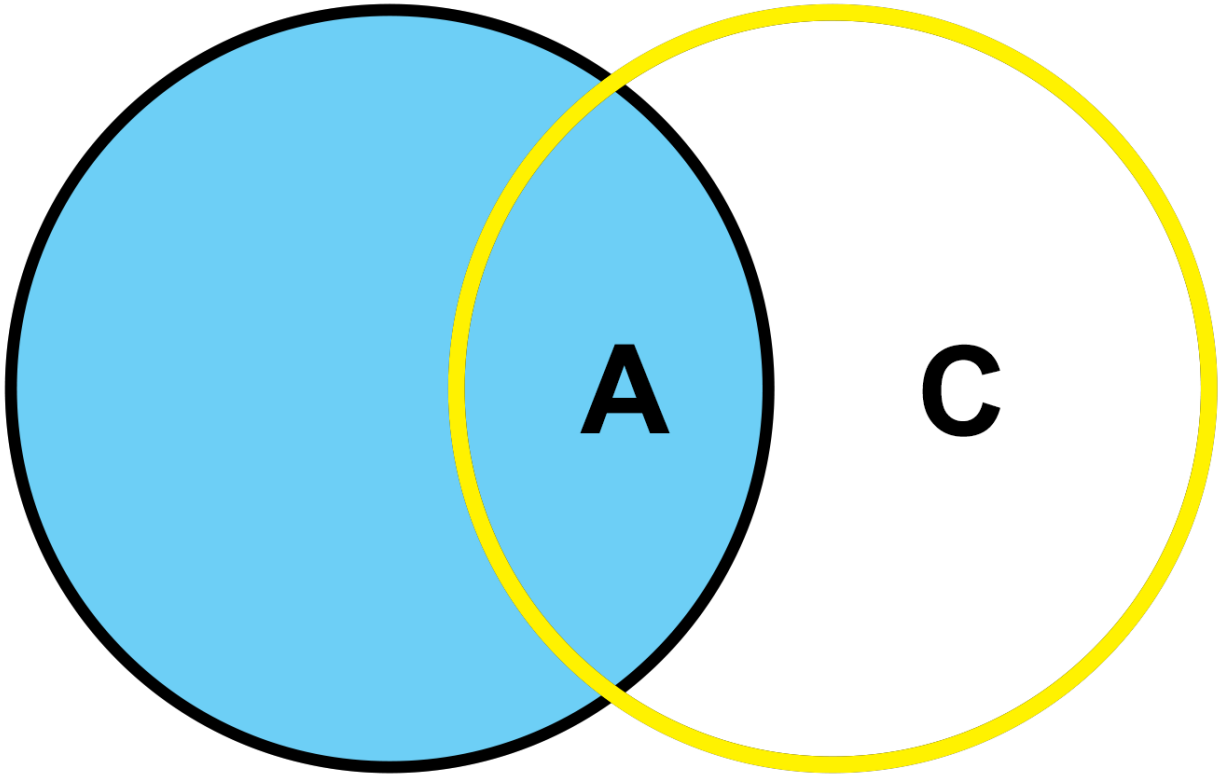
Precision vs Recall



Precision vs Recall



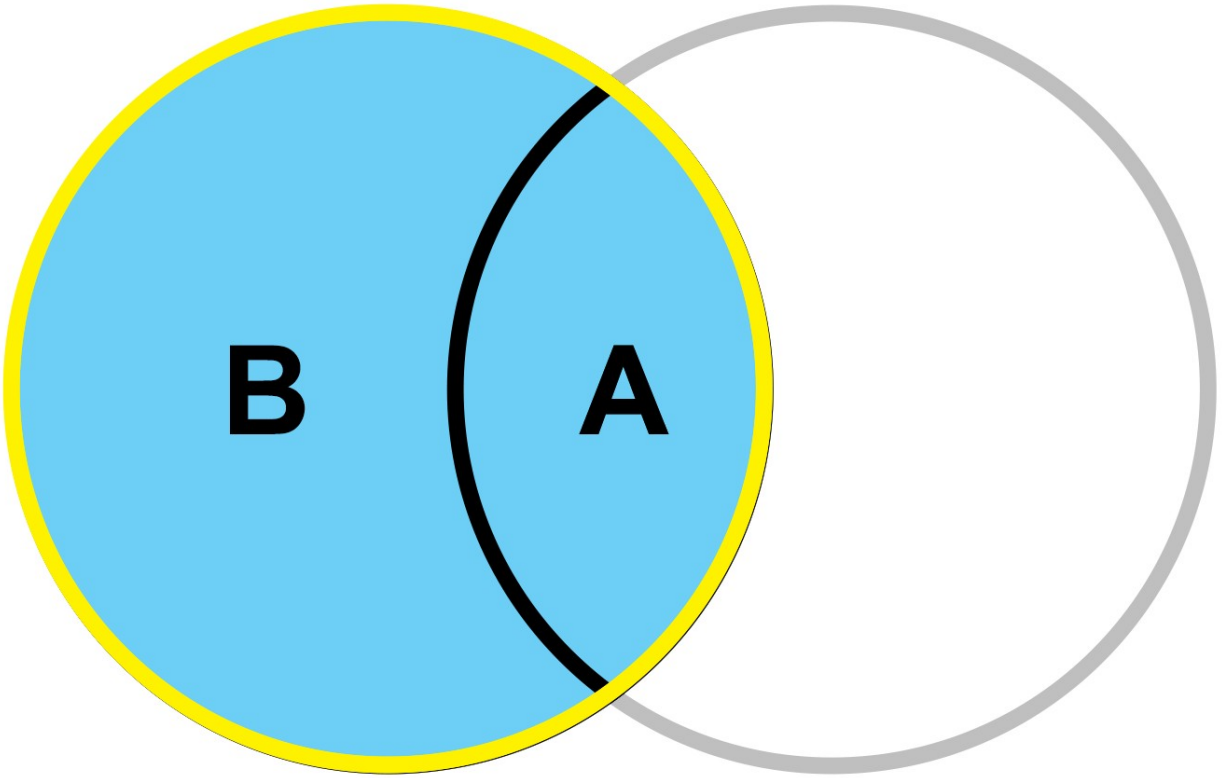
How much extra stuff did you get?



C: # of irrelevant records retrieved
A: # of relevant records retrieved

$$\text{Precision: } \frac{A}{A+C} \times 100\%$$

How much did you miss?



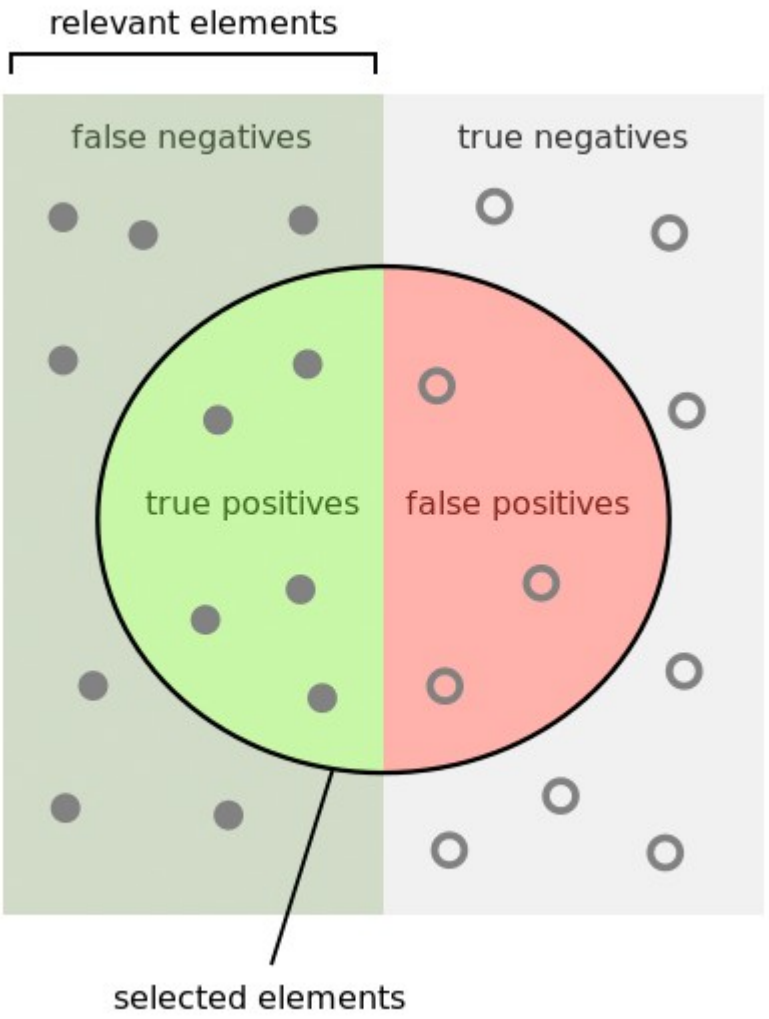
B: # of relevant records not retrieved
A: # of relevant records retrieved

$$\text{Recall: } \frac{A}{A+B} \times 100\%$$

Example

- 10 documents in index are relevant
- Search returns 20 documents, 5 are relevant
- Precision:
  - P=5/(5+15)=0.25
  - 1 out of 4 retrieved documents are relevant
- Recall:
  - R=5/(5+5)=0.5
  - Half of the relevant documents were retrieved

Another way to look at it...



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Fuzzier Matching

- LIKE and REGEXP operator in MySQL
- REGEXP
  - Pattern match succeeds if the pattern matches anywhere in the value being tested.
- LIKE
  - Pattern match succeeds only if the pattern matches the entire value

Example

pallen	m$ftw
dknuth	tek!tex
ada	wtf15b4b
cmoore	moreM00R3!
jresig	In0JS
atanen	minix!minix
linus	ilUvP3nGu1n5
aturing	1nfin1t3TAp3
lwall	oysters&camels
thewoz	4daK1d5

```
SELECT * FROM USERS WHERE username LIKE 'a%'
```

ada	wtf15b4b
atanen	minix!minix
aturing	1nfin1t3TAp3

```
SELECT * FROM USERS WHERE password REGEXP '[0-9]{2,}'
```

ada	wtf15b4b
cmoore	moreM00R3!

But it gets better...

- MATCH()...AGAINST()
  - performs a natural language search over index
- Index = set of one or more columns of the same table
  - column must have type FULLTEXT
- MATCH()
  - takes a comma-separated list that names the columns to be searched
- AGAINST()
  - takes a string to search for
- If used in WHERE clause, results returned in order of relevance score
  - relevance: similarity between search string and index row
- See http://dev.mysql.com/doc/refman/5.1/en/fulltext-natural-language.html

FULLTEXT

- Can only create FULLTEXT on CHAR, VARCHAR or TEXT columns
- "title" and "body" still available as regular columns
- If you want to search only on "title", you need to create a separate index

CREATE TABLE odu_football (id INT AUTO_INCREMENT NOT NULL PRIMARY KEY, opponent VARCHAR(200), notes TEXT, date DATE, FULLTEXT (opponent, notes))

Stopwords

- Common words that do not make good search terms
- If a word appears in more than half the rows, it's a stop word
- Stopwords vary on collections/dictionaries/corpuses