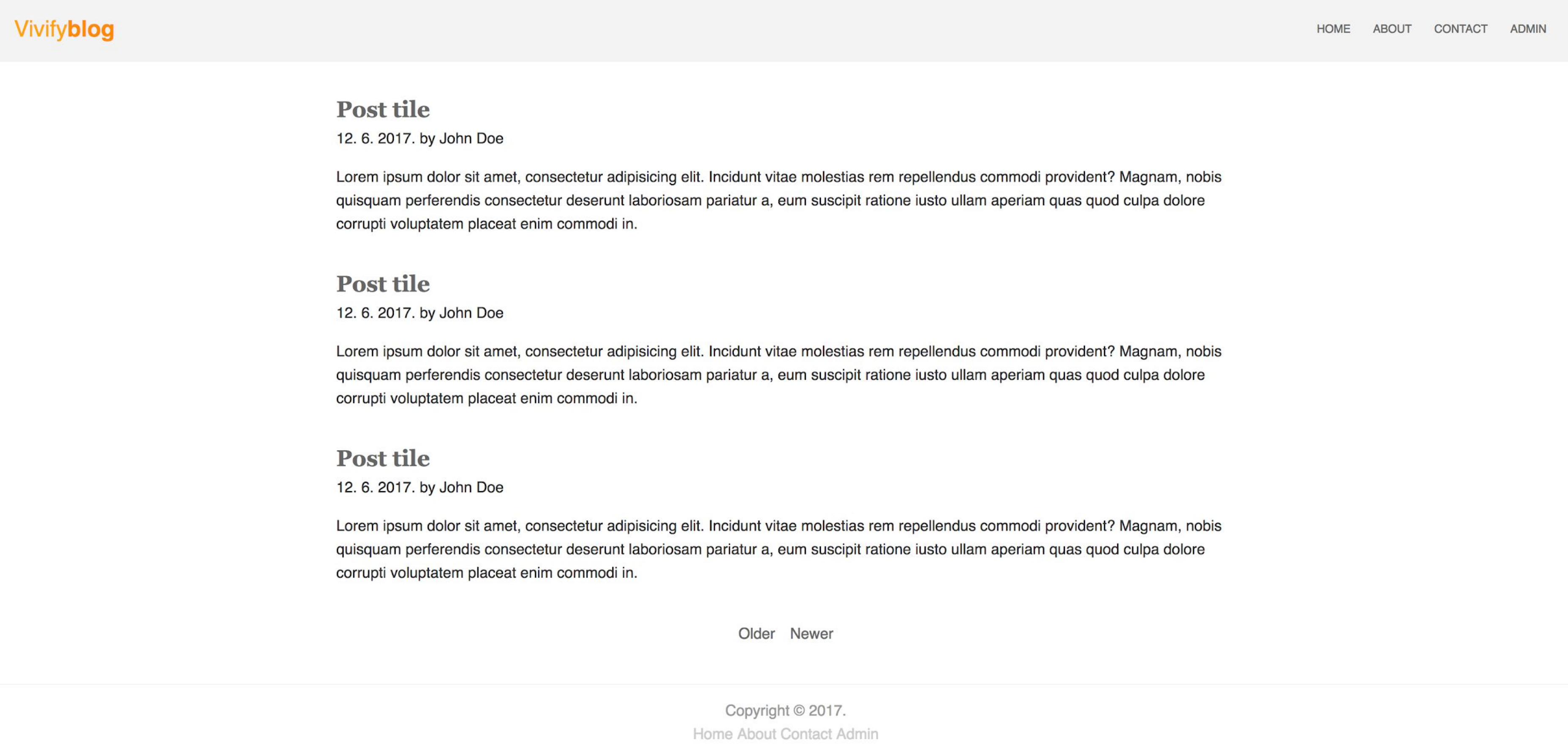
Database modelling

VIVIFY
ACADEMY

Example website - VivifyBlog

Let’s see what we have in this example website…



Post tile

12. 6. 2017. by John Doe

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Incidunt vitae molestias rem repellendus commodi provident? Magnam, nobis quisquam perferendis consectetur deserunt laboriosam pariatur a, eum suscipit ratione iusto ullam aperiam quas quod culpa dolore corrupti voluptatem placeat enim commodi in.

Post tile

12. 6. 2017. by John Doe

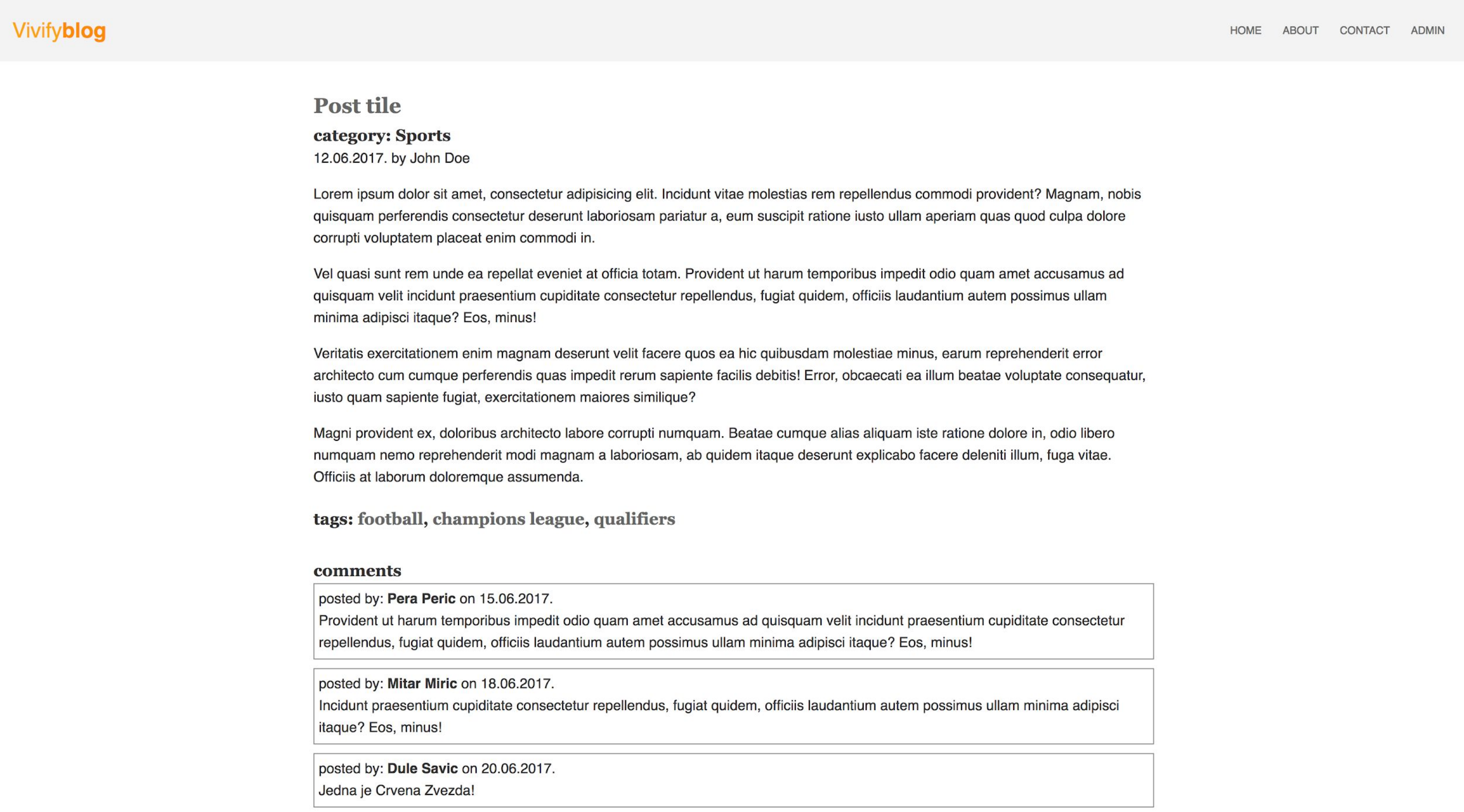
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Incidunt vitae molestias rem repellendus commodi provident? Magnam, nobis quisquam perferendis consectetur deserunt laboriosam pariatur a, eum suscipit ratione iusto ullam aperiam quas quod culpa dolore corrupti voluptatem placeat enim commodi in.

Post tile

12. 6. 2017. by John Doe

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Incidunt vitae molestias rem repellendus commodi provident? Magnam, nobis quisquam perferendis consectetur deserunt laboriosam pariatur a, eum suscipit ratione iusto ullam aperiam quas quod culpa dolore corrupti voluptatem placeat enim commodi in.

Copyright © 2017.



Post tile

category: Sports

12.06.2017. by John Doe

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Incidunt vitae molestias rem repellendus commodi provident? Magnam, nobis quisquam perferendis consectetur deserunt laboriosam pariatur a, eum suscipit ratione iusto ullam aperiam quas quod culpa dolore corrupti voluptatem placeat enim commodi in.

Vel quasi sunt rem unde ea repellat eveniet at officia totam. Provident ut harum temporibus impedit odio quam amet accusamus ad quisquam velit incidunt praesentium cupiditate consectetur repellendus, fugiat quidem, officiis laudantium autem possimus ullam minima adipisci itaque? Eos, minus!

Veritatis exercitationem enim magnam deserunt velit facere quos ea hic quibusdam molestiae minus, earum reprehenderit error architecto cum cumque perferendis quas impedit rerum sapiente facilis debitis! Error, obcaecati ea illum beatae voluptate consequatur, iusto quam sapiente fugiat, exercitationem maiores similique?

Magni provident ex, doloribus architecto labore corrupti numquam. Beatae cumque alias aliquam iste ratione dolore in, odio libero numquam nemo reprehenderit modi magnam a laboriosam, ab quidem itaque deserunt explicabo facere deleniti illum, fuga vitae. Officiis at laborum doloremque assumenda.

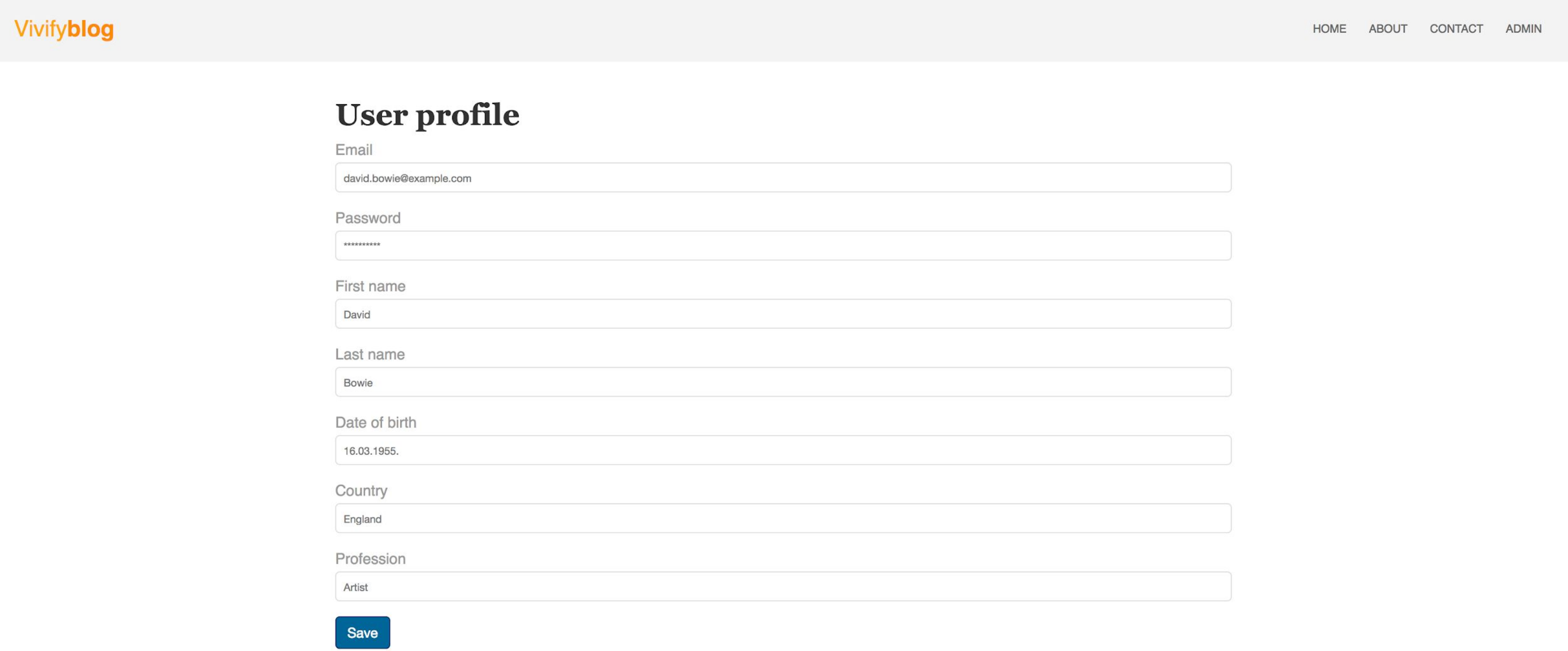
tags: football, champions league, qualifiers

comments

posted by: **Pera Peric** on 15.06.2017.
Provident ut harum temporibus impedit odio quam amet accusamus ad quisquam velit incidunt praesentium cupiditate consectetur repellendus, fugiat quidem, officiis laudantium autem possimus ullam minima adipisci itaque? Eos, minus!

posted by: **Mitar Miric** on 18.06.2017.
Incidunt praesentium cupiditate consectetur repellendus, fugiat quidem, officiis laudantium autem possimus ullam minima adipisci itaque? Eos, minus!

posted by: **Dule Savic** on 20.06.2017.
Jedna je Crvena Zvezda!



User profile

Email
david.bowie@example.com

Password
****

First name
David

Last name
Bowie

Date of birth
16.03.1955.

Country
England

Profession
Artist

Specification (users)

There are 3 roles a user can have on our blog:

1. Author (**registered** user, can create **posts** and leave **comments**)

2. Member (**registered** user, can only **comment**)

3. Guest (**not registered**, can **only browse** the site)

Specification (user profiles)

When a user registers with an **email** and a **password**, she/he can create a profile which consists of:

- first name
- last name
- date of birth
- country
- profession

Specification (posts)

- Posts can only be created by users with ‘author’ role
- Every post belongs to only one category (sports, health, fashion, food...)
- Each post has a title, a creation date and content
- Posts can receive comments
- Each post can be tagged with one or more tags

Specification (comments)

- Comments can only be created by registered users
- Each comment has its content and a creation date

Specification (tags)

- User should be able to filter posts by tags
- For example, clicking a tag on the single post page should redirect you to a page (similar to homepage) that holds only the posts tagged with that specific tag

Exercise I - defining tables

- Let’s identify all **entities (tables)** in this system
- For the start, we have **users**...
- Try to do it yourself

Exercise I - defining tables - solution

- Tables:
  - Users
  - Posts
  - Comments
  - Tags

Exercise II - defining fields in the tables

- Let’s now identify what fields should have every table.
- For example, the **users** table should have:
    - id
    - email
    - password
    - role
- Try to do it yourself

Exercise II - defining fields in the tables - solution

- users
  - Id
    email
  - password
  - Role

- profiles
  - id
    first name
  - last name
  - date of birth
  - country
  - profession

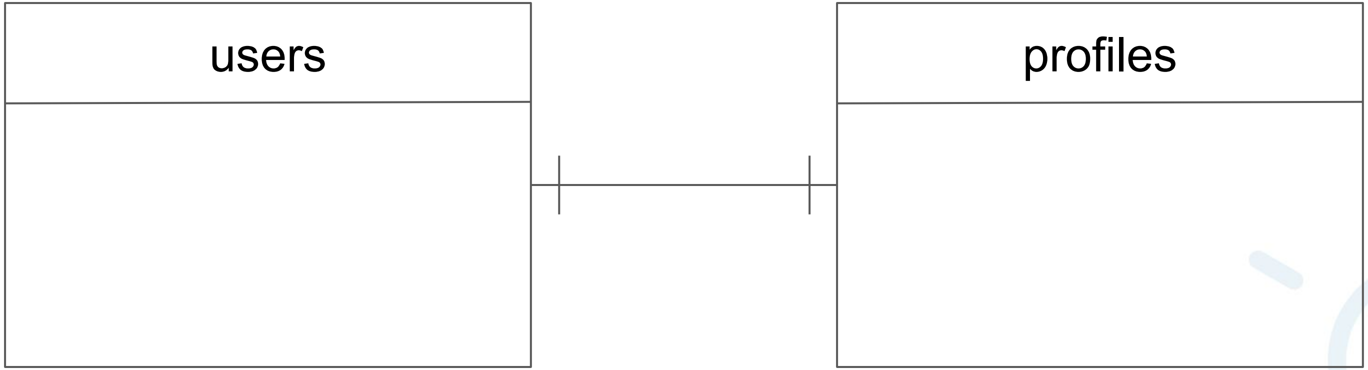
- posts
  - id
  - category
  - title
  - content
  - created_at

- comments
  - id
  - content
  - created_at

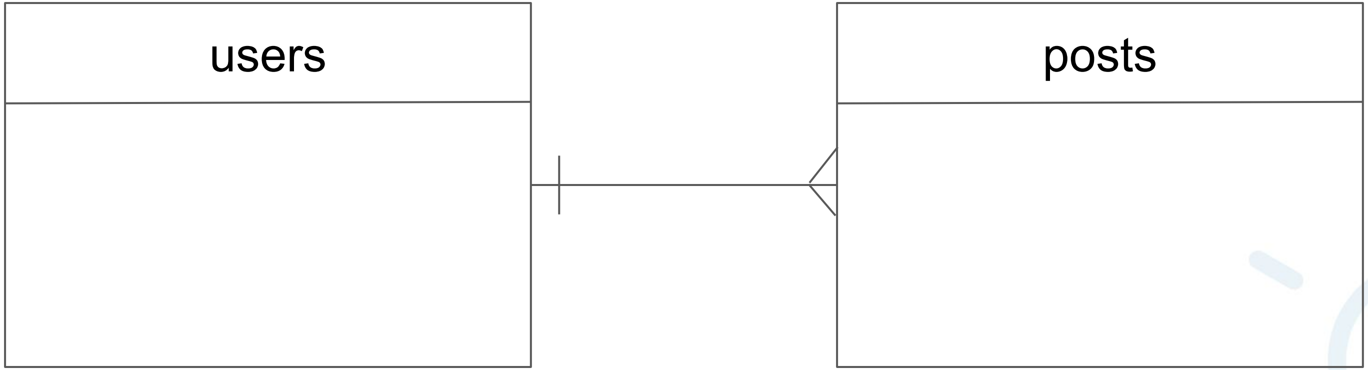
- tags
  - id
  - title

SQL Relations

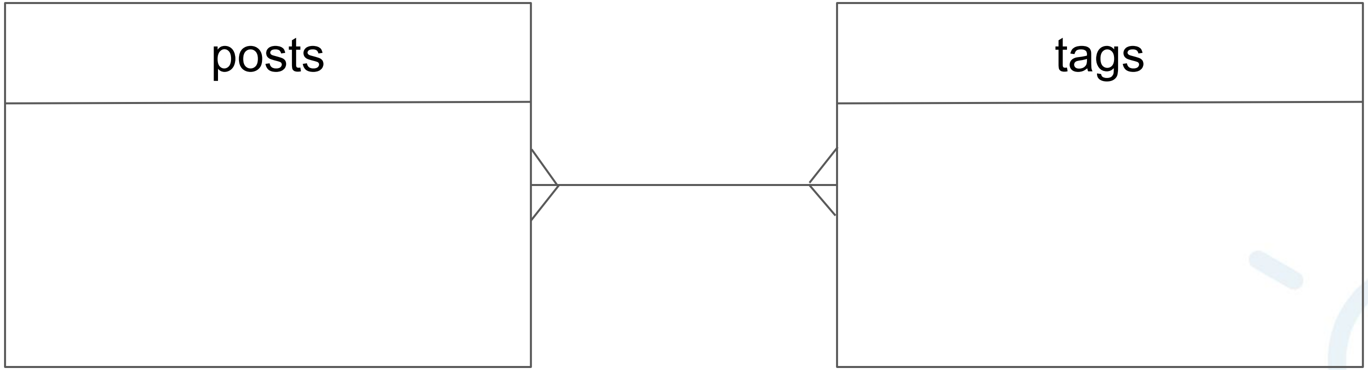
One to one relation



One to many relation



Many to many relation



Exercise III - defining relations

- Let’s now identify what relations tables have
- Rules:
  - Every user has only one profile
  - Every profile belongs to only one user
  - Every post belongs to only one author (user)
  - Every author (user) can have multiple posts
  - Every comment belongs to only one post
  - Every post can contain multiple comments
  - Every post can have multiple tags
  - Every tag can have multiple posts
- We have these relations:
  - 1 to 1
  - 1 to many
  - Many to many
- Do it yourself!

Exercise III - defining relations - solution

- Users <-> profiles - 1 to 1
- Users <-> posts - 1 to many
- Posts <-> comments - 1 to many
- Posts <-> tags - many to many

SQL constraints

not null, unique, primary key, foreign key, default, index

SQL Constraints

```
CREATE TABLE table_name ( col1 datatype constraint, col2 datatype constraint.... );
```

SQL constraints are used to specify rules for the data in a table.

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **DEFAULT** - Sets a default value for a column when no value is specified
- **INDEX** - Use to create and retrieve data from the database very quickly

NOT NULL Constraint

```
CREATE TABLE users (

    id int NOT NULL, email varchar(60) NOT NULL,

    password varchar(255) NOT NULL

);
```

- By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

UNIQUE Constraint

```
CREATE TABLE users (
    id int NOT NULL UNIQUE, email varchar(60) NOT NULL,
    password varchar(255) NOT NULL
);
```

- The UNIQUE constraint ensures that all values in a column are different.

PRIMARY KEY Constraint

```
CREATE TABLE users (

    id int NOT NULL UNIQUE, email varchar(60) NOT NULL,

    password varchar(255) NOT NULL,

    PRIMARY KEY (id)

);
```

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only one primary key, which may consist of single or multiple fields.

FOREIGN KEY Constraint

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field in one table that refers to the PRIMARY KEY in another table.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

FOREIGN KEY Constraint on CREATE TABLE

```
CREATE TABLE profiles (

    id int NOT NULL,

    name varchar(255),

    user_id int NOT NULL UNIQUE,

    PRIMARY KEY (id),

    FOREIGN KEY (user_id) REFERENCES users(id)

);
```

SQL FOREIGN KEY on ALTER TABLE

```
ALTER TABLE profiles

ADD FOREIGN KEY (user_id) REFERENCES users(id)


ALTER TABLE profiles

DROP FOREIGN KEY profiles_ibfk_1;  *foreign key name
```

SQL DEFAULT Constraint

```
CREATE TABLE profiles (
    id int NOT NULL UNIQUE,
    name varchar(255) DEFAULT ‘John’
);
```

- The DEFAULT constraint is used to provide a default value for a column.

```
ALTER TABLE profiles ALTER name DROP DEFAULT;
```

SQL CREATE INDEX Statement

```
CREATE INDEX index_name ON table_name (column1, column2, ...);
```

- Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

CREATE and DROP INDEX example

```
CREATE INDEX index_name ON table_name (column_name);

ALTER TABLE table_name DROP INDEX index_name;
```

SQL AUTO INCREMENT Field

```
CREATE TABLE users (

    id int AUTO_INCREMENT, email varchar(60) NOT NULL,

    password varchar(255) NOT NULL,

    PRIMARY KEY (id)

);
```

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
- Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

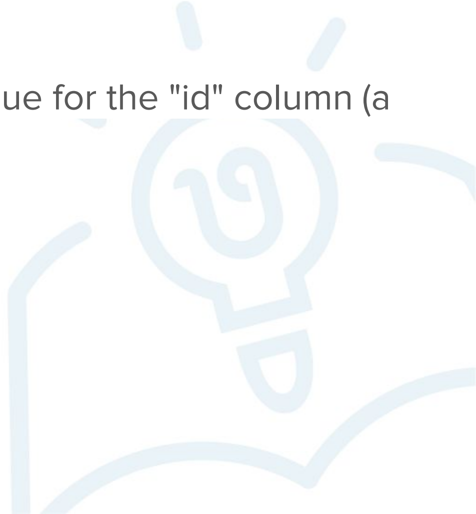
SQL AUTO INCREMENT Field

- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.
- To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

```
ALTER TABLE table_name AUTO_INCREMENT = 100;
```

- To insert a new record into the "users" table, we will NOT have to specify a value for the "id" column (a unique value will be added automatically):

```
INSERT INTO users (email, password) VALUES ('a@b.com','secret');
```



Exercise IV - creating tables in database

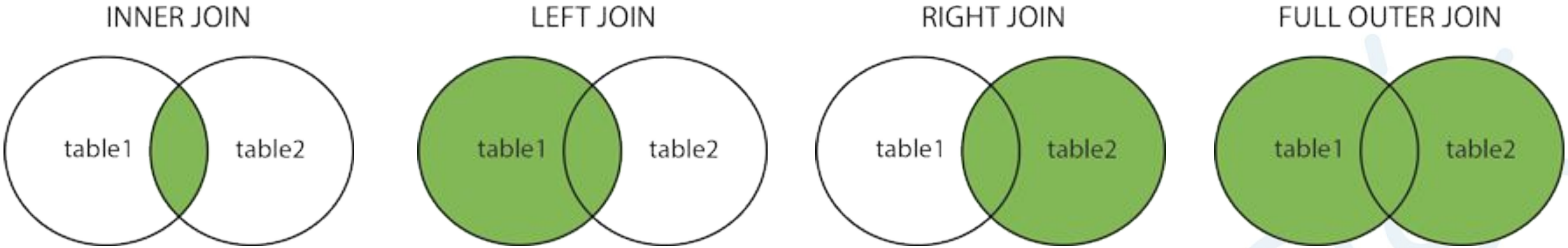
1. Sketch tables, fields(columns) and their datatypes
2. Consult instructor when done
3. Write down query for creating a table
4. Pay attention to foreign keys!
5. Execute query
6. Repeat steps 3 and 4 until all tables are created!

SQL Relation queries

inner join, left join, right join, full outer join

SQL joins

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.



The INNER JOIN Keyword

```
SELECT column_name(s) FROM table1

INNER JOIN table2 ON table1.column_name = table2.column_name;
```

- The INNER JOIN keyword selects records that have matching values in both tables.

The INNER JOIN Keyword (examples)

```
SELECT * FROM users INNER JOIN profiles ON users.id = profiles.user_id;
```

```
SELECT * FROM users AS u INNER JOIN profiles AS p ON u.id = p.user_id;
```

```
SELECT users.role, profiles.last_name, profiles.profession FROM users
INNER JOIN profiles ON users.id = profiles.user_id
WHERE profiles.profession = ‘programmer’;
```

The INNER JOIN Keyword (examples II)

```sql
SELECT u.role, p.last_name, p.profession, po.category post_category FROM users u

INNER JOIN profiles p ON u.id = p.user_id

INNER JOIN posts po ON u.id = po.created_by;
```

The LEFT JOIN Keyword

```
SELECT column_name(s) FROM table1

LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

- The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

The LEFT JOIN Keyword (examples)

```
SELECT * FROM users LEFT JOIN profiles ON users.id = profiles.user_id;
```

```
SELECT * FROM users AS u LEFT JOIN posts AS p ON u.id = p.created_by;
```

```
SELECT users.role, profiles.last_name, profiles.profession FROM users

LEFT JOIN profiles ON users.id = profiles.user_id

WHERE profiles.profession = ‘cook’;
```

The LEFT JOIN Keyword (examples II)

```sql
SELECT p.id, p.title, t.name, u.email FROM posts p

LEFT JOIN post_tags pt ON p.id = pt.post_id

LEFT JOIN tags t ON t.id = pt.tag_id

LEFT JOIN users u ON u.id = p.created_by;
```

The RIGHT JOIN Keyword

```
SELECT column_name(s) FROM table1

RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

The RIGHT JOIN Keyword (examples)

```
SELECT * FROM profiles RIGHT JOIN users ON users.id = profiles.user_id;


SELECT * FROM profiles AS p RIGHT JOIN users AS u ON u.id = p.user_id

WHERE p.profession = ‘lawyer’ OR p.profession = ‘manager’ ORDER BY p.last_name;
```

The difference between JOINS (examples)

```sql
SELECT * FROM users INNER JOIN profiles ON users.id = profiles.user_id;

SELECT * FROM users LEFT JOIN profiles ON users.id = profiles.user_id;

SELECT * FROM profiles RIGHT JOIN users ON users.id = profiles.user_id;
```