



Active Record & SQL

Juan “Harek” Urrios
@xharekx33

What is a “database”

A database, formally, is an organized collection of data.

Most frequently when we talk about databases really mean the software used to manage it: It's Database Management System, that is used to capture and analyze data and write it to permanent storage.



Relational databases

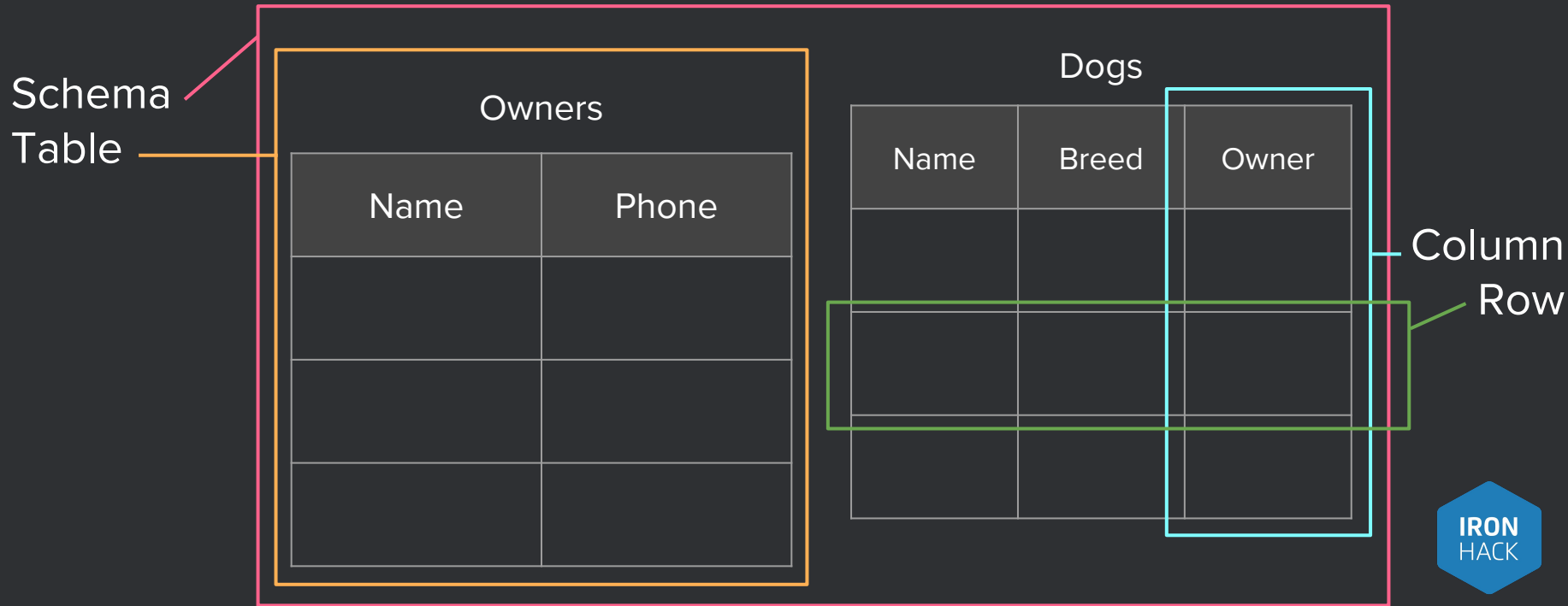
Relational databases are organized around data and its relations, providing the way to organize, persist, and query that data.

The components of a relational database are:

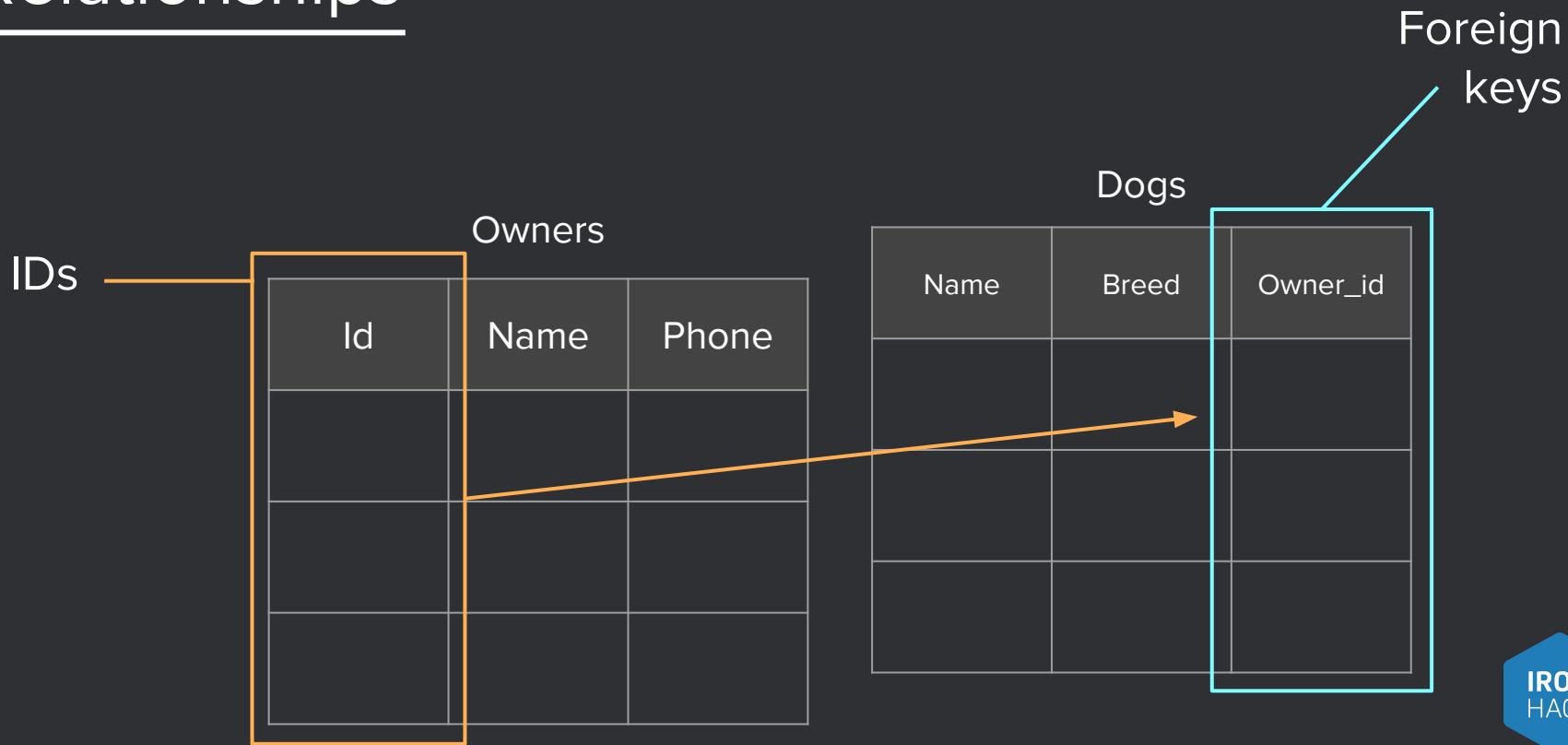
- schema
- tables
- rows
- columns
- data types



Relational database components



Relationships



What is SQL

Virtually all relational database systems use SQL (Structured Query Language) as the domain specific language for querying and maintaining the database.

It's syntax can be different depending on the database: MySQL, PostgreSQL, Oracle.

MySQL: `UPDATE a,b SET a.id=b.id WHERE a.f2 = b.f2;`

PostgreSQL: `UPDATE a SET a.id=b.id FROM b WHERE a.f2 = b.f2;`



SQL: Select

Returns a set of records from one or more tables that match a certain condition

```
SELECT
    column or columns (use * for all columns)
FROM
    table or tables (joined with JOIN)
WHERE
    condition (one or more, joined with AND/OR);

SELECT * FROM Users WHERE name LIKE 'John %';
```



SQL: Insert

Adds new records in a table

```
INSERT INTO  
  table (columns)  
VALUES  
  (values);
```

```
INSERT INTO Users(Name,Email,PhoneNumber) VALUES ('Peter  
Parker', 'pparker@dailybugle.com', '555666777');
```



@xharekx33

SQL: Update

Changes the data of one or more records in a table that match a certain condition

```
UPDATE
  table
SET
  column1 = value1, column2 = value2
WHERE
  condition (one or more, joined with AND/OR);

UPDATE Users SET Phone = '777666555' WHERE Name = 'Peter
Parker'
```



@xharekx33

SQL: Join

Combines records from two or more tables by using values common to each, without the need for a relationship

```
SELECT Players.Name, Team.Name
FROM Players
JOIN Teams
WHERE Player.city = Teams.city;
```



Basic SQL exercises

Go to <http://sqlzoo.net> and go over the first 4 tutorials:

- SELECT basics
- SELECT name
- SELECT from World
- SELECT from Nobel



Active Record Query Interface

Active Record in Rails insulates you from the need to use SQL in most cases, performing queries on the database for you.

Regardless of the database system in use, the Active Record method format will always be the same.

Sometimes it might be necessary to mix the two though.



A new app for testing

We'll be using a new app to test our queries.

Create the app, run **bundle install**, add Users and Tasks, add a default value to the boolean column for users and run **rake db:migrate**

```
$ rails new queryTester
$ cd queryTester
$ bundle install
$ rails g model User name:string email:string active:boolean
$ rails g model Task name:string status:string user_id:integer

#add "default: false" to the active column for Users

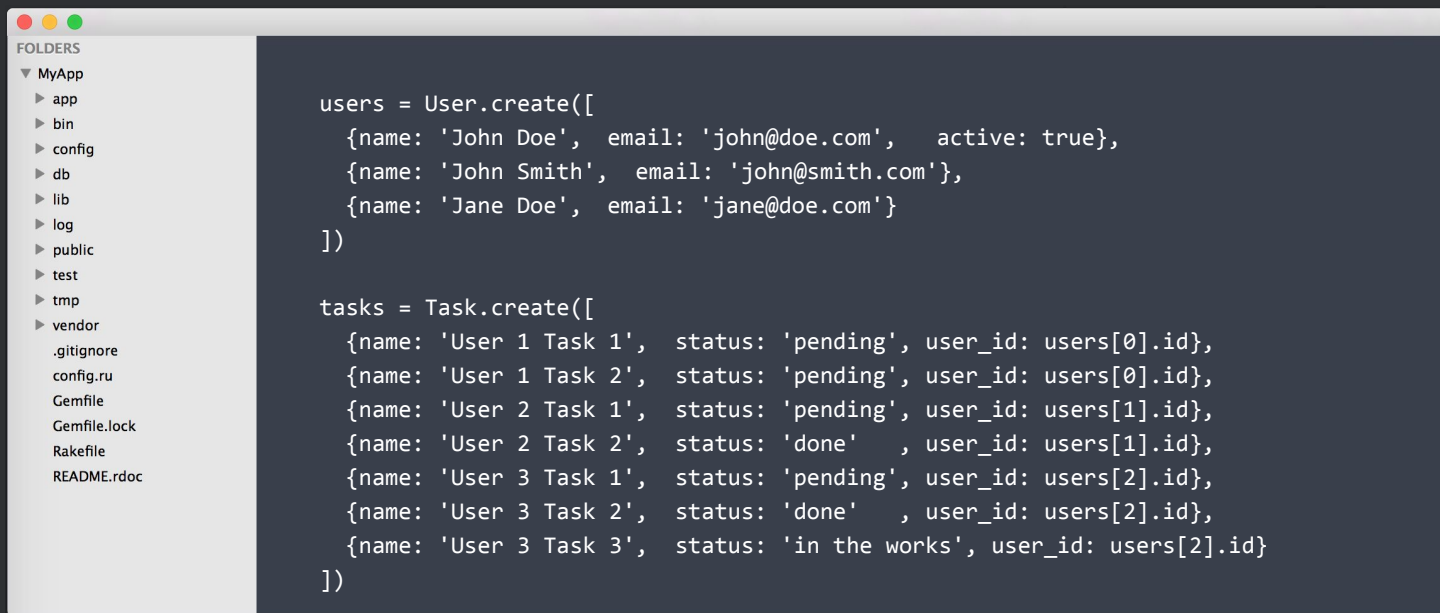
$ rake db:migrate
```



@xharekx33

Seed the database

Add users and tasks to the seed file and **run rake db:seed**



The image shows a code editor window with a sidebar on the left displaying a file tree under the name 'FOLDERS'. The tree includes 'MyApp' with subfolders 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', and 'vendor', as well as files '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The main editor area contains Ruby code for seeding a database. It defines two arrays: 'users' and 'tasks'. The 'users' array contains three user objects with names, emails, and an active status. The 'tasks' array contains nine task objects, each with a name, status, and a user_id that references a user in the 'users' array. The code uses the 'create' method to instantiate these objects.

```
users = User.create([
  {name: 'John Doe', email: 'john@doe.com', active: true},
  {name: 'John Smith', email: 'john@smith.com'},
  {name: 'Jane Doe', email: 'jane@doe.com'}
])

tasks = Task.create([
  {name: 'User 1 Task 1', status: 'pending', user_id: users[0].id},
  {name: 'User 1 Task 2', status: 'pending', user_id: users[0].id},
  {name: 'User 2 Task 1', status: 'pending', user_id: users[1].id},
  {name: 'User 2 Task 2', status: 'done', user_id: users[1].id},
  {name: 'User 3 Task 1', status: 'pending', user_id: users[2].id},
  {name: 'User 3 Task 2', status: 'done', user_id: users[2].id},
  {name: 'User 3 Task 3', status: 'in the works', user_id: users[2].id}
])
```



Comparing Active Record and SQL

Open TWO consoles. One to run `rails console` and another one for `rails dbconsole`

```
$ rails console
```

```
$ rails dbconsole
```



@xharekx33

Retrieve a single User

On the **rails console** window run:

```
User.find(2)
```

On the **rails dbconsole** window run:

```
SELECT * FROM users WHERE (id = 2) LIMIT 1;
```



List all pending Tasks for a User

On the `rails console` window run:

```
Task.where(status: 'pending', user_id: 1)
```

On the `rails dbconsole` window run:

```
SELECT * FROM tasks WHERE status = 'pending' AND  
user_id = 1;
```



List all Tasks for Users 2 and 3

On the `rails console` window run:

```
Task.where(status: ['pending', 'in the works'], user_id: [2, 3]).order(updated_at: :asc)
```

On the `rails dbconsole` window run:

```
SELECT * FROM tasks WHERE status IN ('pending', 'in the works') AND user_id IN (2, 3) ORDER BY updated_at ASC;
```



SQL in Active Record

On the **rails console** window run:

```
Task.where("user_id IN (SELECT id FROM users WHERE name LIKE  
'John %')")
```

On the **rails dbconsole** window run:

```
SELECT * FROM tasks WHERE user_id IN (SELECT id FROM  
users WHERE name LIKE 'John %');
```



Joins in Active Record (1/2)

On the `rails console` window run:

```
User.find_by(id: 1).tasks
```

On the `rails dbconsole` window run:

```
SELECT users.* FROM users JOIN tasks ON users.id = tasks.  
user_id WHERE user_id = 1;
```



Joins in Active Record (2/2)

On the `rails console` window run:

```
User.where(active: true).joins(:tasks).where(tasks: {status: ['pending', 'in the works']})
```

On the `rails dbconsole` window run:

```
SELECT users.* FROM users JOIN tasks ON users.id = tasks.user_id WHERE users.active = 't' AND tasks.status IN ('pending', 'in the works');
```



SQL Indexes

Databases use indexes to speed up the performance of SELECT queries on a table, by reducing the number of database data pages that have to be visited/scanned at the cost of extra writes and storage space.

They can cause slower INSERTS and UPDATES.

They are also used for policing database constraints: PRIMARY KEY, FOREIGN KEY, UNIQUE



Primary key index

Uniquely identifies each record in a table. A relational database must always have one and only one primary key.

Rails automatically creates a Primary Key Index on the id column.



Primary key index

Uniquely identifies each record in a table. There must always be one and only one primary key.

Rails automatically creates a Primary Key Index on the id column.

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :email
      t.boolean :active, default: false
      t.timestamps null: false
    end
  end
end
```



@xharekx33

Primary key index

Uniquely identifies each record in a table. There must always be one and only one primary key.

Rails automatically creates a Primary Key Index on the id column.

```
CREATE TABLE "users" (  
  "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,  
  "name" varchar,  
  "created_at" datetime NOT NULL,  
  "updated_at" datetime NOT NULL);
```



Foreign key index

Uniquely identifies a record in another table.

It's used to establish relationships between tables

```
def change
  create_table :tasks do |t|
    t.string :name
    t.string :status
    t.references :user, index: true, foreign_key: true
    t.timestamps null: false
  end
end
```



@xharekx33

Foreign key index

Uniquely identifies a record in another table.

It's used to establish relationships between tables

```
CREATE TABLE "tasks" (  
  "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,  
  "name" varchar,  
  "status" varchar,  
  "user_id" integer,  
  "created_at" datetime NOT NULL,  
  "updated_at" datetime NOT NULL);  
  
CREATE INDEX "index_tasks_on_user_id" ON "tasks" ("user_id");
```



Unique index

Uniquely identifies each record in a table. There can be more than one unique index.

They must be created manually.

```
def change
  create_table :users do |t|
    t.string :name
    t.string :email, index:true, unique: true
    t.timestamps null: false
  end
end
```



@xharekx33

Unique index

Uniquely identifies each record in a table. There can be more than one unique index.

They must be created manually.

```
CREATE TABLE "users" (  
  "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,  
  "name" varchar,  
  "email" varchar,  
  "created_at" datetime NOT NULL,  
  "updated_at" datetime NOT NULL);  
  
CREATE INDEX "index_users_on_email" ON "users" ("email");
```



Composite index

Uniquely identifies each record in a table across two or more columns.

The order of the columns is significant as they will be compared in turn.

```
def change
  add_index :users, [:email, :name], unique: true
end
end
```



Composite index

Uniquely identifies each record in a table across two or more columns.

The order of the columns is significant as they will be compared in turn.

```
CREATE INDEX "index_users_on_email_and_name" ON "users" ("email",  
"name");
```



When to use an index

- If you expect your app to have a lot of data.
- If you are going to access records on a table through a foreign key or through a column other than the id.
- If you are going to be sorting a column often.
- To make sure values are really unique (validation can fail)



When NOT to use an index

- On small tables.
- On tables that have frequent, large batch update or insert operations.
- On columns that contain a high number of NULL values.
- On columns that are frequently manipulated



Benchmarks

Given a User table with `name` and `email` fields (both strings), with 4 million records:

```
User.where(name: 'steven')
```

- No index: 0.618 seconds
- Composite index: 0.067 seconds
- Index on name: 0.063 seconds



Exercise

- Create a composite index for our User model on name and email
- Open the rails console and try to create two users with the same name and email.



Remember

- Databases are optimized for searching
- Whenever possible, let the database do the searching, filtering, and ordering for you.

