Q1 from P1

employees

can belong to single dpt

employees

ID	/	name	/	dpt_id
1		Mark	/	1
2		Paulo		null

departments

ID	/	name	/
1		ADMIN	/
2		TECH	

SQL query.. if given apt name find employees in dpt

```
SELECT
    name, id
FROM
    employees
JOIN
    departments ON  dpt_id = departments.id
WHERE
    depatments.name = ?
```

find name of employees that dont belong to any dpt

```
SELECT
    name, id
FROM
    employees
WHERE
    depatments.id IS NULL
```

Q1 from P2

```
What are the disadvantages of adding an index to a table column in a database?
```

```
**Answer**: Indices do have a cost. They make writes (`INSERT`s, `DELETE`s, and
`UPDATE`s) a little more taxing because the index table must also be updated.
```

Q2 from P1

dogs

ID	/	name	/	breed
1		Fido		Lab

toys

ID	/	toy.name	/	toy.price
1		Bone		5

dog_toys

ID	/	dog_id	/	toy_id
1		1		1

toys to a dog

belongs_to :dog,
    primary_key: :id, #toys.id
    foreign_key: :dog_id => dogs.id
    class_name: :Dog

has_many: toys,
    primary_key: :id,
    foreign_key: :dog_id
    class_name:Toy

```
**Answer**: A primary key uniquely identifies a record in the
relational database table, whereas a foreign key refers to the `id`
column which is the
primary key of **another** table.
```

Q2 from P2

```
# == Schema Information
#
# Table name: users
#
#  id    :integer                not null, primary key
#  name :string                  not null


# Table name: enrollments
#
#  id    :integer                not null, primary key
#  course_id :integer            not null
#  student_id :integer           not null


# Table name: courses
#
#  id    :integer                not null, primary key
#  course_name :string           not null
#  professor_id :integer         not null
#  prereq_course_id :integer     not null
```

class Enrollment
    belongs_to :course,
        primary_key: :id,
        foreign_key: course_id,
        class_name: :Course
    belongs_to :student,
        primary_key: :id,
        foreign_key: :student_id,
        class_name: :User

class Course
    belongs_to :professor,
        primary_key: :id,
        foreign_key: :professor_id,
        class_name: :User

```
        belongs_to :prereq_course,
                primary_key: :id,
                foreign_key: :prereq_course_id,
                class_name: :Course,
                optional: true
        has_many :enrollments,
                primary_key: :id,
                foreign_key: course_id,
                class_name: :Enrollment,
                optional: true

        has_one :upper_course
                primary_key: :id,
                foreign_key: :prereq_course_id,
                class_name: :Course,
                optional: true
class User
        has_many :courses,
                primary_key: :id,
                foreign_key: :professor_id,
                class_name: :Course,
                optional: true
        has_many :enrollments
                primary_key: :id,
                foreign_key: :student_id,
                class_name: :Enrollment,
                optional: true
```

```ruby
class Enrollment < ApplicationRecord
 belongs_to :user,
   class_name: 'User',
   foreign_key: :student_id,
   primary_key: :id

 belongs_to :course,
   class_name: 'Course',
   foreign_key: :course_id,
   primary_key: :id
end

class User < ApplicationRecord
 has_many :enrollments,
   class_name: 'Enrollment',
   foreign_key: :student_id,
   primary_key: :id

 has_many :courses_taught,
   class_name: 'Course',
   foreign_key: :professor_id,
   primary_key: :id,
   optional: true
end

class Course < ApplicationRecord
 has_many :enrollments,
   class_name: 'Enrollment',
   foreign_key: :course_id,
   primary_key: :id

 belongs_to :prerequisite,  #
   class_name: 'Course',
   foreign_key: :prereq_course_id,
   primary_key: :id,
   optional: true

 belongs_to :professor,    ###
   class_name: 'User',
   foreign_key: :professor_id,
   primary_key: :id
end
```

Q3 from P1

```
# == Schema Information
#
# Table name: physicians
#
#  id   :integer          not null, primary key
#  name :string           not null


# Table name: appointments
#
#  id   :integer           not null, primary key
#  physician_id :integer   not null
#  patient_id :integer     not null


# Table name: patients
#
#  id   :integer           not null, primary key
#  name :string            not null
#  primary_physician_id :integer
```

class Physician < ApplicationRecord

```
      has_many: appointments,
            primary_key: :id,
            foreign_key: :physician_id,
            class_name: :Appointment

      has_many :primary_patients,
            primary_key: :id,
            foreign_key: :primary_physician_id,
            class_name: :Patient

      has_many: general_patients,
            through: appointments,
            source: patient

      has_many: primary_patients_appoitnments,
            through: :primary_patients,
            source: :appointments

end
```

```ruby
class Appointment < ApplicationRecord
      belongs_to: physician,
            primary_key: :id,
            foreign_key: :physician_id,
            class_name: :Physician

      belongs_to: patient,
            primary_key: :id,
            foreign_key: :patient_id,
            class_name: :Patient

end


calss Patient < ApplicationRecord

      belongs_to :primary_physician,
            primary_key: :id,
            foreign_key: :primary_physician_id,
            class_name: :Physician


      has_many: appointments,
            primary_key: :id,
            foreign_key: :patient_id,
            class_name: :Appointment

      has_many: physicians,
            through: appointments,
            source: physician


end
class Physician < ApplicationRecord
  has_many :appointments,
    class_name: 'Appointment',
    foreign_key: :physician_id,
    primary_key: :id

  has_many :primary_patients,
    class_name: 'Patient',
    foreign_key: :primary_physician_id,
    primary_key: :id

  has_many :general_patients,
```

```
    through: :appointments,
    source: :patient

  has_many :primary_patient_appointments,
    through: :primary_patients,
    source: :appointments

end

class Appointment < ApplicationRecord
  belongs_to :physician,
    class_name: 'Physician',
    foreign_key: :physician_id,
    primary_key: :id

  belongs_to :patient,
    class_name: 'Patient',
    foreign_key: :patient_id,
    primary_key: :id
end

class Patient < ApplicationRecord
  has_many :appointments
    class_name: 'Appointment',
    foreign_key: :patient_id,
    primary_key: :id

  belongs_to :primary_care_physician
    class_name: 'Physician',
    foreign_key: :primary_physician_id,
    primary_key: :id
end
```

Q3 FROM P2

```
Given all possible SQL commands order by order of query execution. (SELECT,
DISTINCT, FROM, JOIN, WHERE, GROUP BY, HAVING, LIMIT/OFFSET, ORDER).
```

FROM
JOIN
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT/OFFSET

Q4 FROM P1

The advantage of using an ORM as ActiveRecords is that we can use AR methods as well as ruby methods in Rails in order to manipulate the date from our database

It gives us freedom to call SQL methods if desired, and not the ORM is smart enough to connect the missing dots in our code.

```
**Answer**: Using an ORM (Object Relational Model) allows you to
interact with
database information in an OOP way. An ORM like ActiveRecord will
translate rows
from your SQL tables into Ruby objects on fetch, and translates your
Ruby
objects back to rows on save resulting in less overall database access
code.
```