**Advanced Web Development – “Working with Class Hierarchies in an RDBMS”**

Points: 20

Due date on blackboard

**Aim:** The aim of this lab is implementing class hierarchies in Rails ActiveRecord.

**Instructions:**

1. **Use the VM supplied in class for this. Please refrain from using your own installs for labs.**
2. Work in pairs.
3. Please answer all questions – you can be very brief.
4. You may have to refer to the book/slides to understand some things.

**Clone the following project from git:**

[**https://github.com/siddharthkaza/single\_table\_inheritance\_rails42**](https://github.com/siddharthkaza/single_table_inheritance_rails42)

**Implementing Single Table Inheritance (10)**

The aim of the project is to model a person class in a banking web application where people have many roles.

Example: roles of people

person: name, email

customer: person with a balance

employee: person with a boss and a department

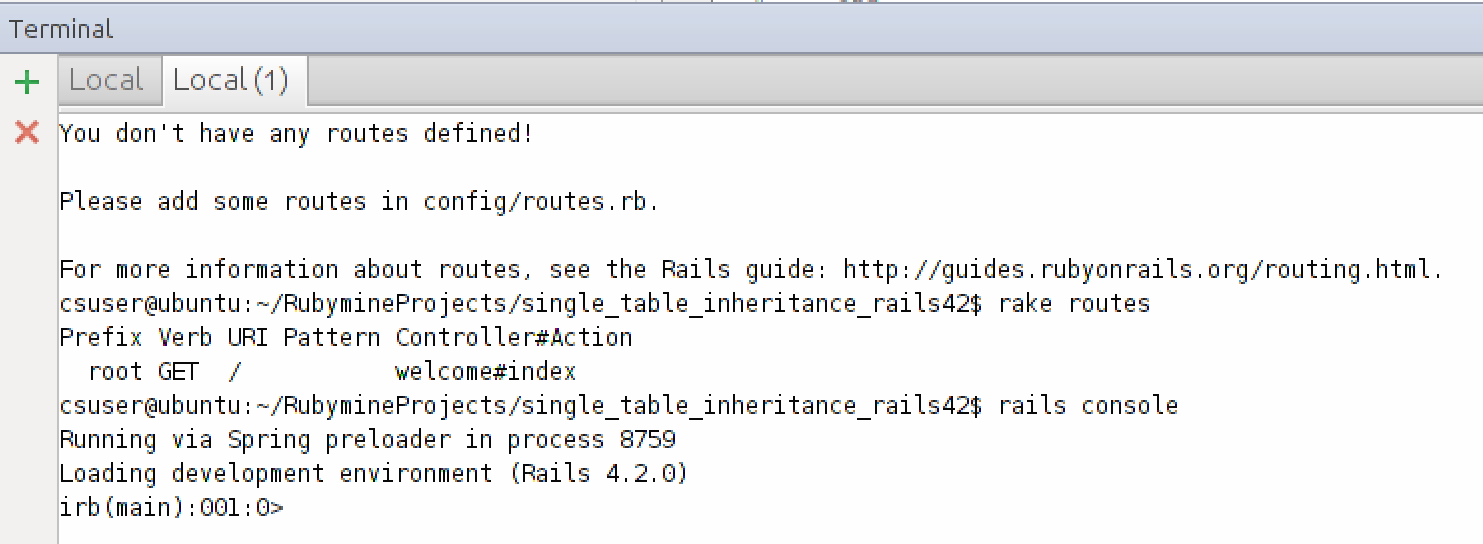
manager: employee with a group and a list of managed employees

* 1. Run the application (call the index method) to make sure its running in the browser.

*Hint:* It’s always a good idea to run ‘bundle update’ and then ‘rake db:migrate’ and ‘rake db:seed’ first - when using an application from another location. This updates any gems needed and brings the DB to its newest version.

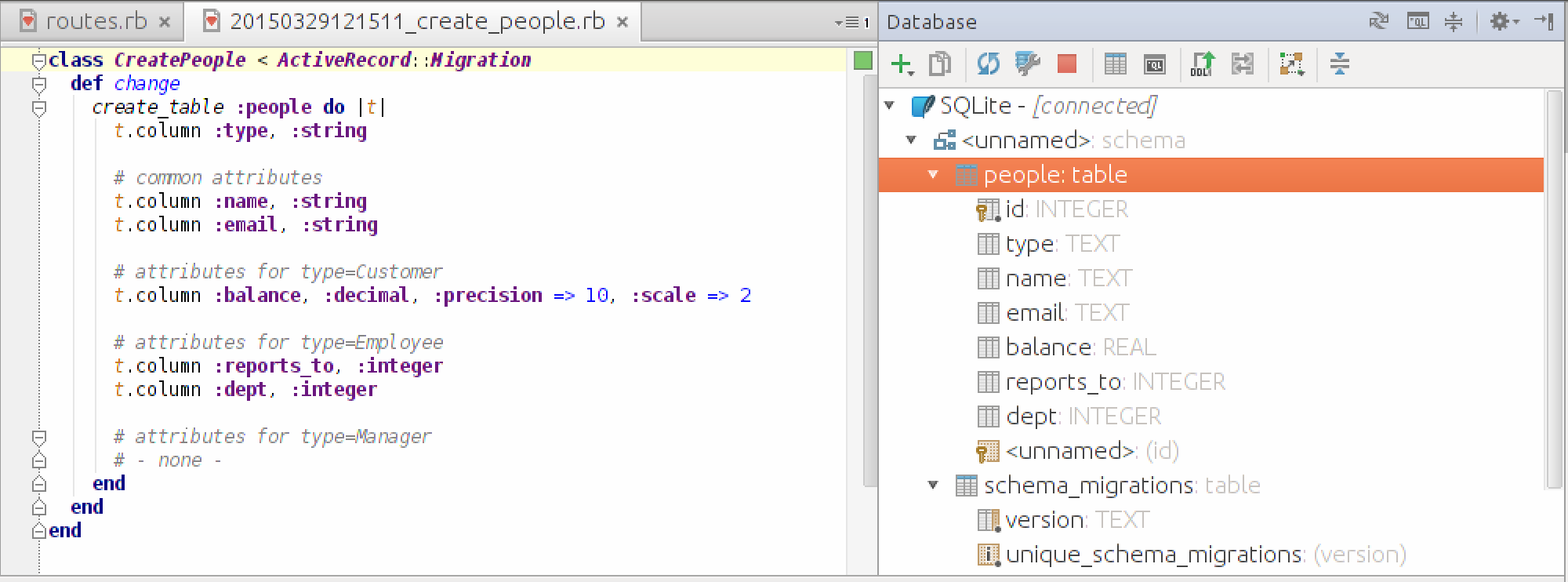
* 1. Start the ‘rails console’ within the application – Type ‘rails console’ on the terminal window for the application.

Hint: If you get the ‘cannot find readLine’ error, then you will need to install the necessary libraries for the console. See the ‘installing rails on linux’ document on blackboard for the instructions (the subsection on installing rails console).



* 1. Study the create\_table\_people migration.
     1. Open the database in your favorite GUI tool and see the result of the migration. How many tables are we using to model store data on the person class? Is this a good strategy? Would you suggest another way to store data on the Person class and all its subclasses in the DB.

**We are using only 1 table, people. It is a good strategy if it is possible that people records could move from one type to another. For example, if a person is an employee, but becomes a manager the people record would only need to be updated instead of removed from one table and added to another. Another way to store the data would be to have a separate table for each of the types of people (4): a table for employees, a table for customers, a table for managers, and a table for person.**



* 1. Study the models for the person class. Notice that there are four models (in line with the inheritance structure of our example).
     1. List the four models. List the inheritance relationship between the models using the ‘<’ symbol. (e.g., if car inherits from vehicle then car < vehicle)

**Manager < Employee < Person**

**Customer < Person**

* + 1. How many tables are the models associated with?

**One table: people**

* + 1. If you were to create a new person (Person.create(….)). Then how many attributes would the database record have?

**3 attributes: type, name, email**

* + 1. If you were to create a new Customer (Customer.create(….)). Then how many attributes would the database record have?

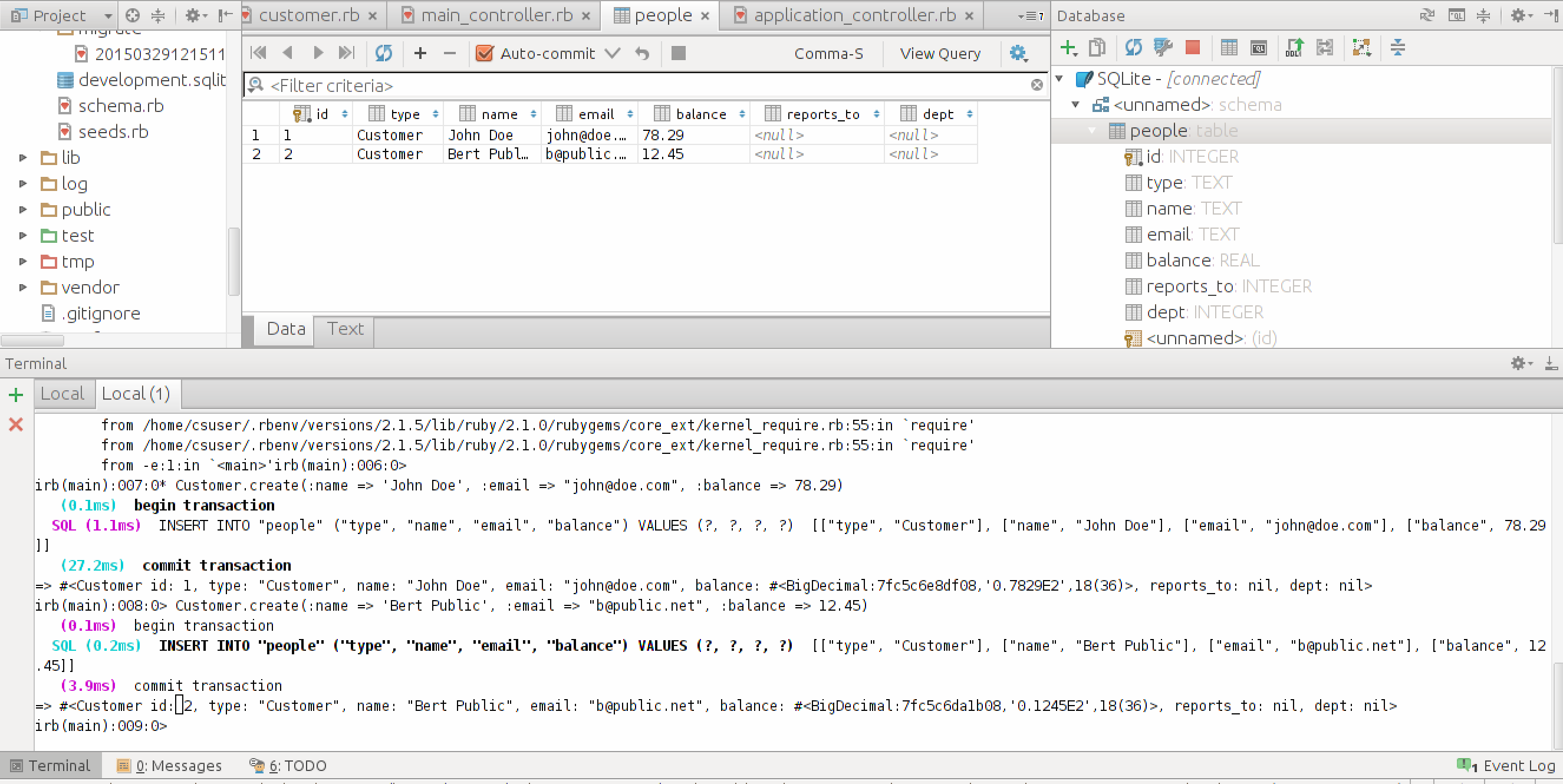
**4 attributes: type, name, email, balance**

* + 1. What do you think the ‘type’ attribute in the DB record does?

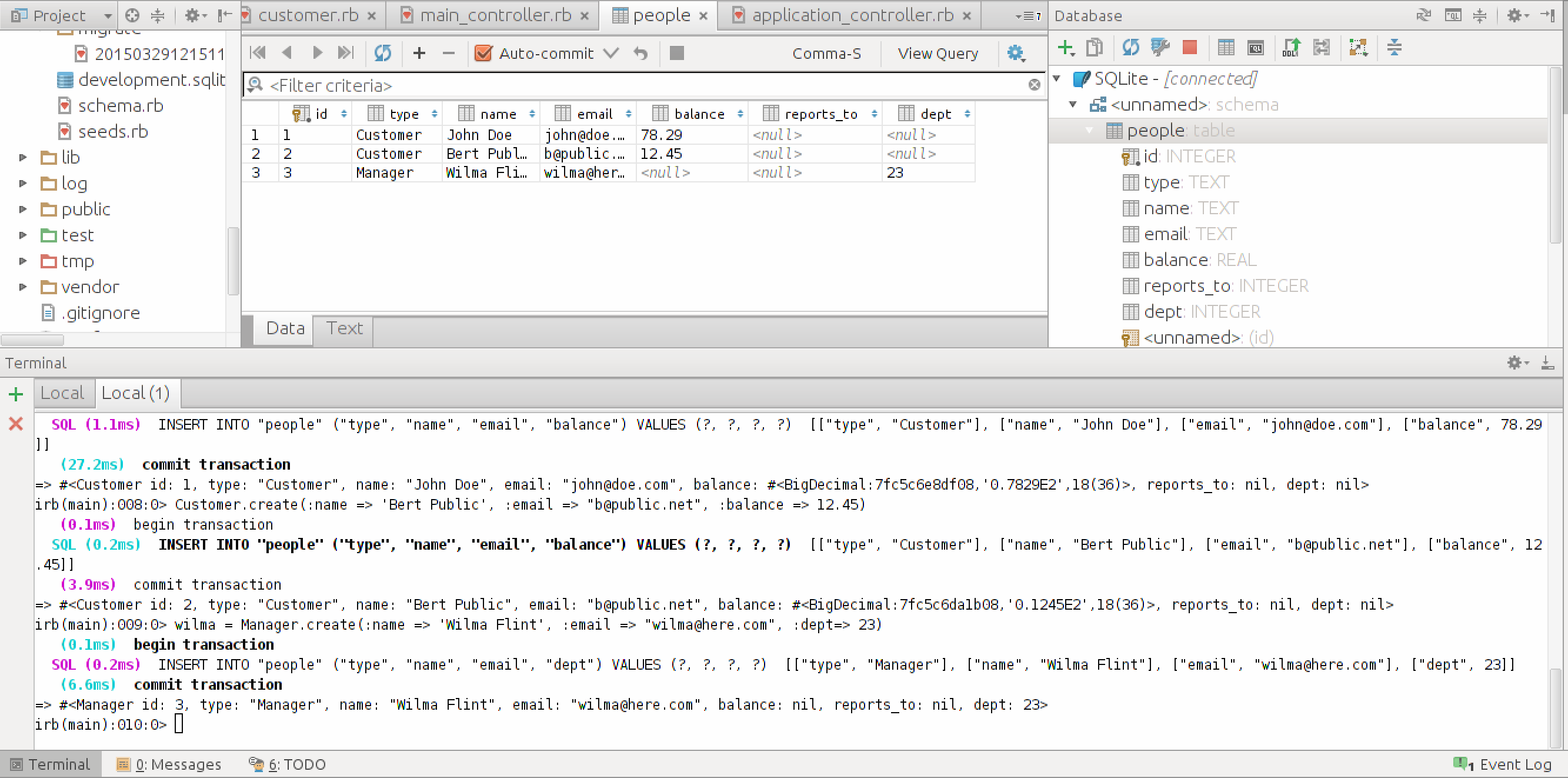
**Determines whether it is a person, customer, employee, or manager, which is used to know which fields must be present**

* 1. Enter the statements in the create\_customer, create\_manager, and create\_employee methods in the main controller in the rails console **one-by-one (in the order of the methods)**. **Please do not copy and paste.** Notice that even though we are using the rails console, the statements you are entering could have values populated from a HTML form.
     1. What is the effect of the Customer.create statements. What does the type attribute store?

**Two records are added, one for John Doe and one for Bert Public. Both have a type attribute of “Customer”**

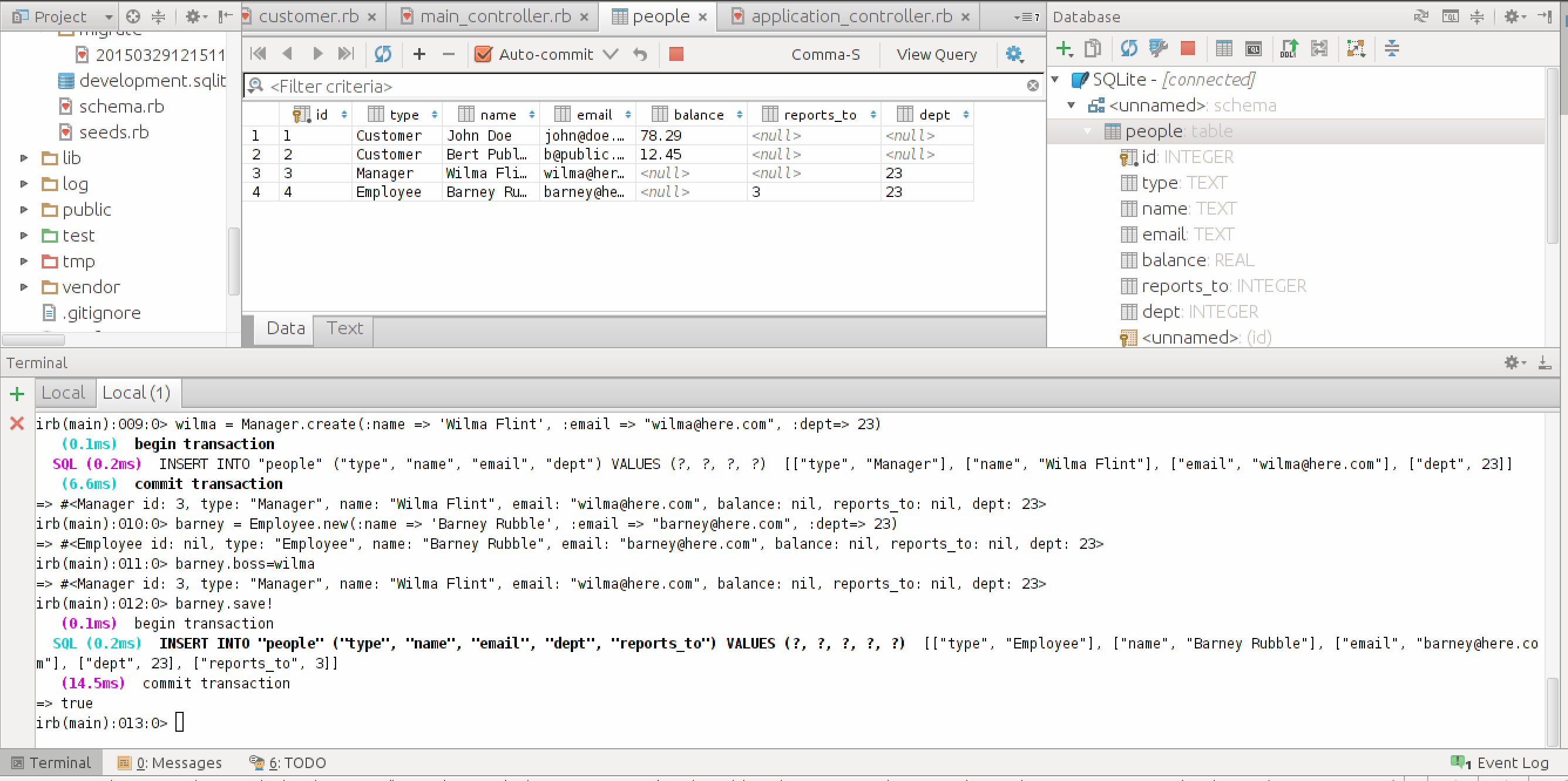
****

* + 1. Create the record for the manager ‘wilma’. Notice that the manager attributes.



* + 1. Run statements for ‘barney.’ How is the barney.boss assignment working?

**The “reports\_to” field for the ‘barney’ record is being populated with the id for Wilma (his assigned boss).**

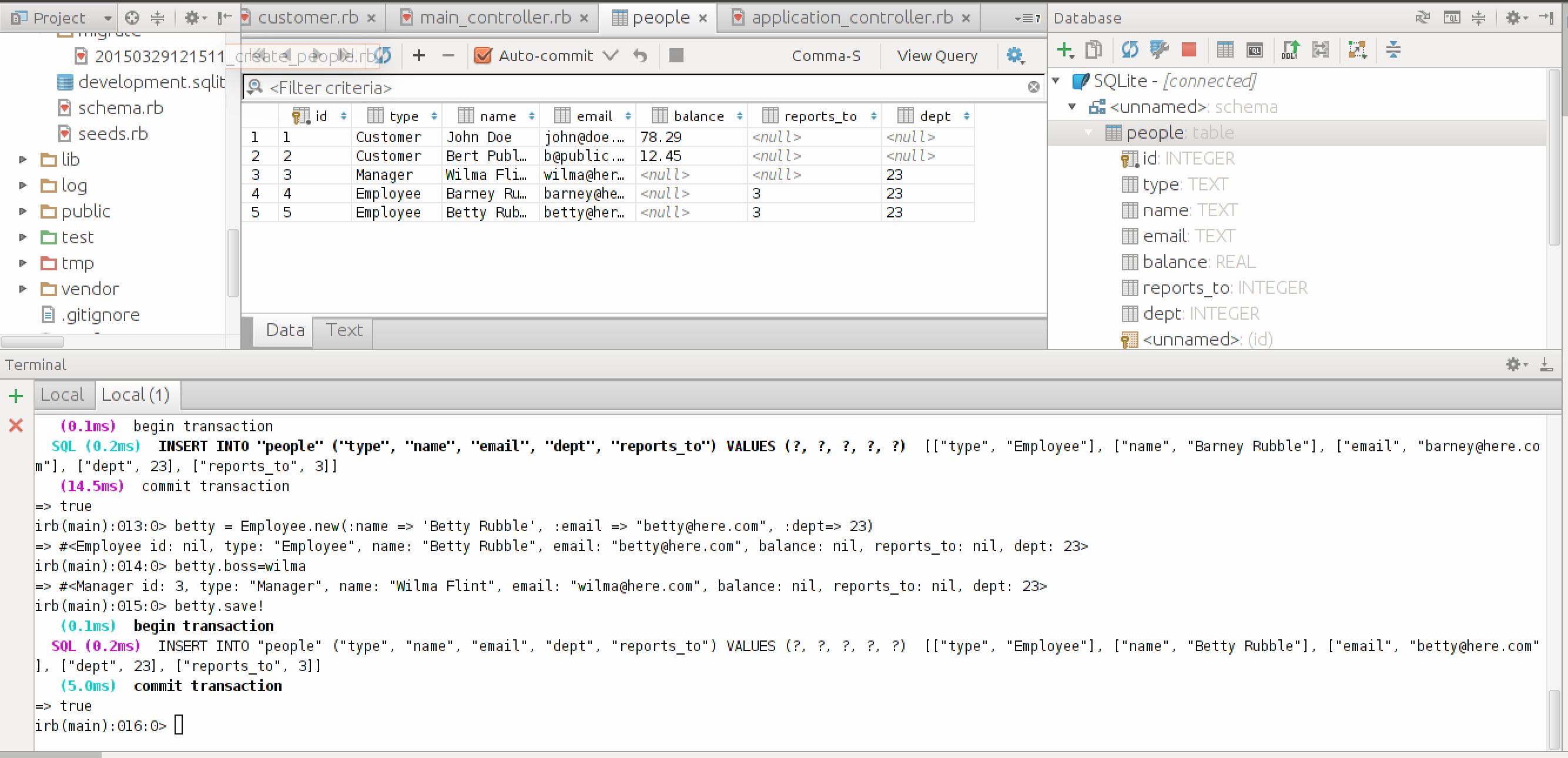
****

* + 1. Add a new Employee ‘Betty Rubble’. Wilma is Betty’s manager. Paste code here.

**betty = Employee.new(:name => “Betty Rubble”, :email => “betty@here.com”, :dept=>23)**

**betty.boss=wilma**

**betty.save!**

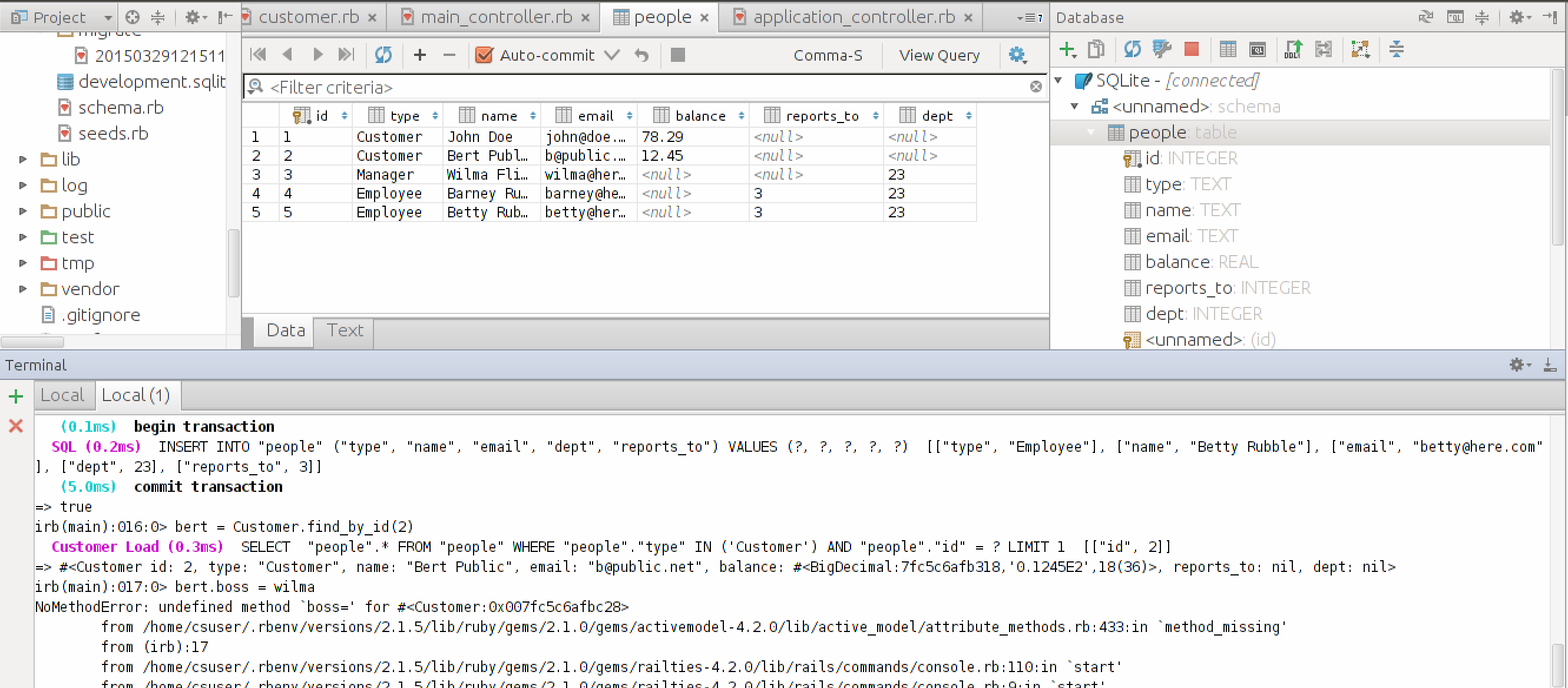
****

* + 1. Assuming you have Bert’s record in a variable called *bert.* Would *bert.boss=wilma* and a subsequent *bert.save* work?

Why/Why not?

If it does work, where would you write code to enforce that it should not.

**It will not work because bert is a customer and customers do not have a manager.**

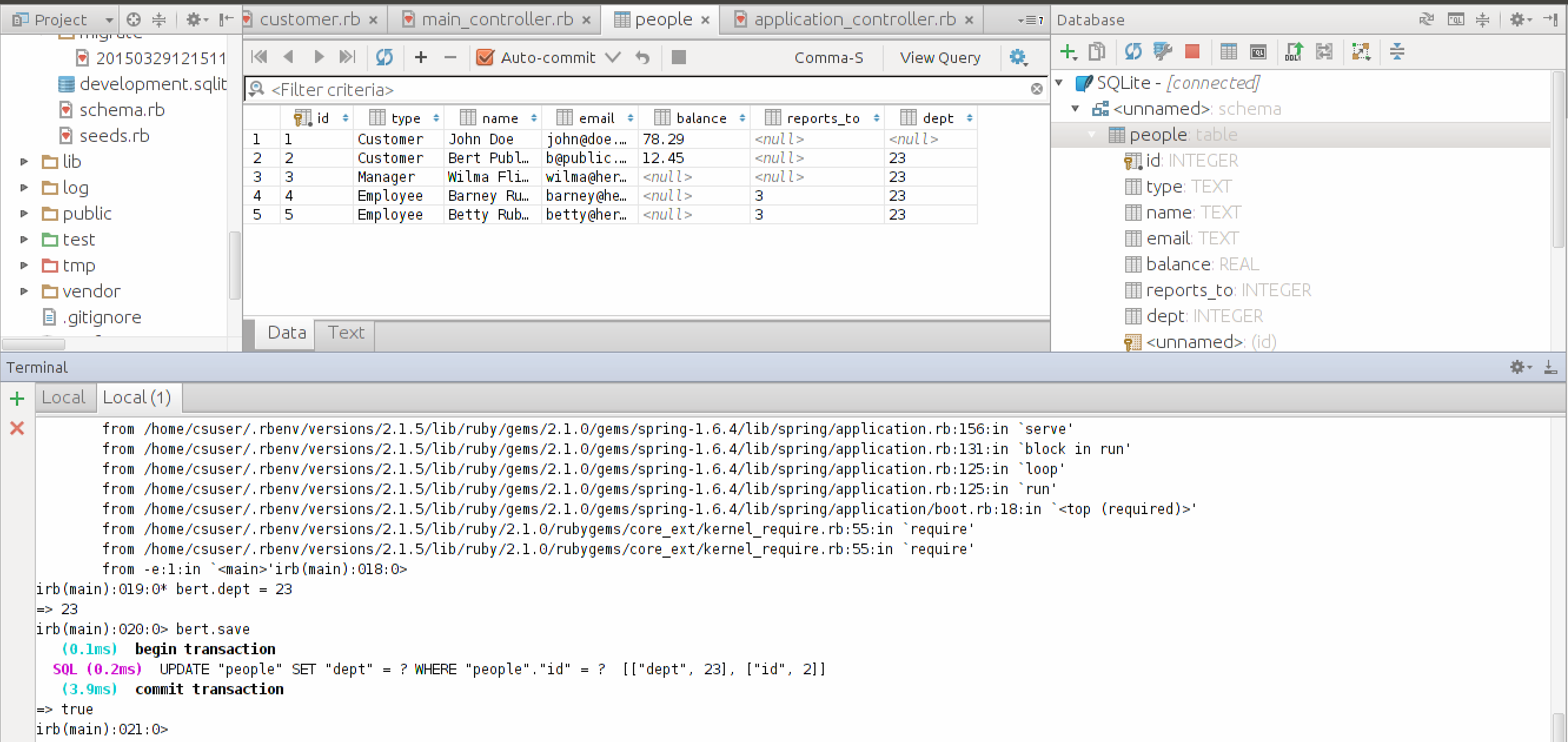
****

* + 1. Assuming you have Bert’s record in a variable called *bert.* Would *bert.dept=23* and a subsequent *bert.save* work?

Why/Why not?

If it does work, where would you write code to enforce that it should not.

**It does work, but should not because Bert is a customer and customers do not have a department. It should be enforced in the model.**

****