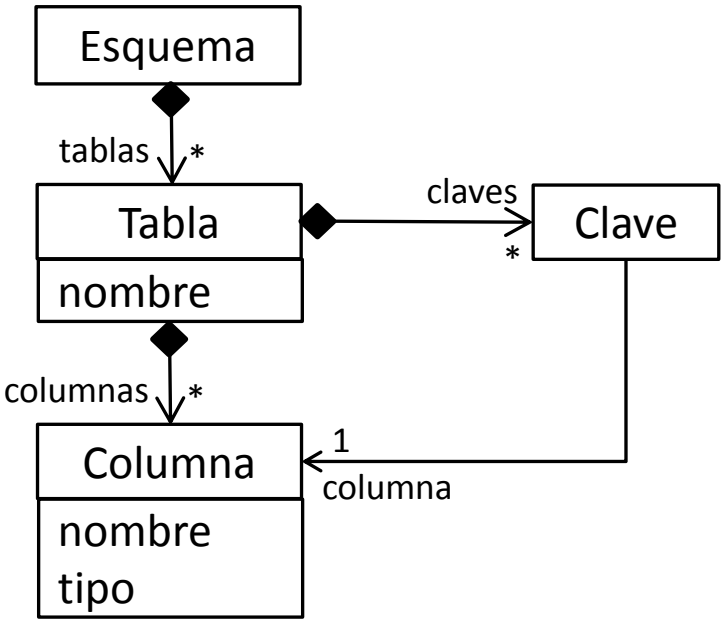
LENGUAJES DE DOMINIO ESPECÍFICO INTERNOS EN RUBY

Desarrollo Automatizado de Software

4º Ingeniería Informática

Universidad Autónoma de Madrid

Ejemplo: esquema de base de datos relacional



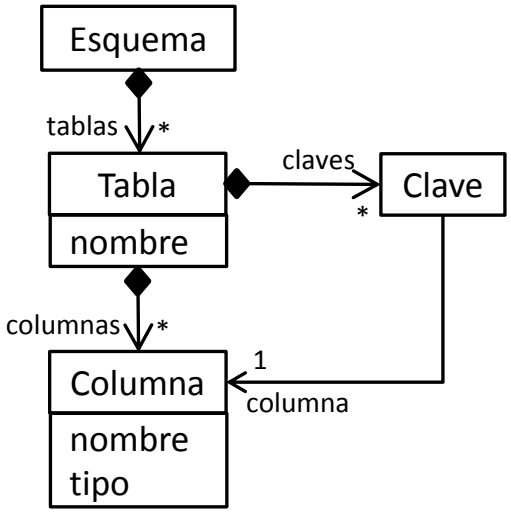
Crear LDE interno para definir esquemas de bases de datos relacionales

- Generar código para crear BBDD
- Generar modelo de datos en un lenguaje de programación, ej. Ruby o Java
- …

Cómo definir LDEs internos en Ruby

Guías generales

- Asociar a cada clase del dominio una *keyword* en el LDE
- Las *keywords* se implementan como métodos, donde:
  - El método crea un objeto de la clase asociada
  - Los atributos de la clase son parámetros del método
  - Si la clase contiene otras clases , pasar un bloque de código donde:
    - Las *keywords* que representan clases contenidas se usan en el bloque
  - Si la clase referencia otra clase, usar identificadores para hacer búsqueda




```
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

Cómo definir LDEs internos en Ruby

Dos enfoques

- Primer enfoque: métodos de clase + extensión
  - Definir una clase como contexto de ejecución (LDE)
  - Las *keywords* del LDE son métodos de clase de la clase de contexto
  - El LDE se utiliza subclasificando la clase de contexto

- Segundo enfoque: métodos de instancia + `instance_eval`
  - Definir un módulo como espacio de nombres, con punto de entrada
  - Las *keywords* del LDE son métodos de instancia de clases del dominio
  - Usar instance_eval para ejecutar bloque de código en el contexto del objeto creado en el método

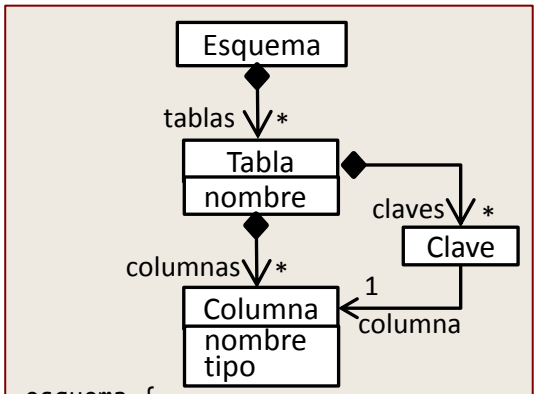
Cómo definir LDEs internos en Ruby

Primer enfoque, conceptos del dominio

```ruby
class Esquema
  def initialize
    @tablas = []
  end
  def add_tabla (tabla)
    @tablas << tabla
  end
  def num_tablas ()
    @tablas.size()
  end
  def get_tabla (posicion)
    @tablas[posicion]
  end
  # diversa funcionalidad
  def generar_codigo_SQL
    ...
  end
  def generar_codigo_ruby
    ...
  end
  def generar_codigo_java
    ...
  end
end
```

```ruby
class Tabla
  def initialize (nombre)
    @nombre   = nombre
    @columnas = []
    @claves   = []
  end
  def add_columna (columna)
    @columnas << columna
  end
  def find_columna (columna)
    @columnas.find{ |col|
      col.nombre==columna}
  end
  def add_clave (clave)
    @claves << clave
  end
end

class Columna
  def initialize (nombre, tipo)
    @nombre = nombre
    @tipo   = tipo
  end
  attr :nombre
end
```

```ruby
class Clave
  def initialize (columna)
    @columna = columna
  end
end
```




```ruby
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

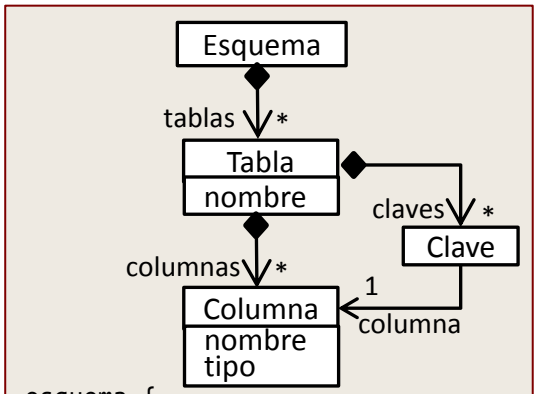
Cómo definir LDEs internos en Ruby

Primer enfoque, clase de contexto (LDE)

```
class EsquemaDSL

  def self.esquema
  ...

  end


  def self.tabla
  ...


  end



  def self.columna
  ...


  end
```

```
  def self.clave (columna)
  ...



  end



end
```

Crear clase de contexto. Definir un método de clase para cada *keyword* del lenguaje de dominio específico.



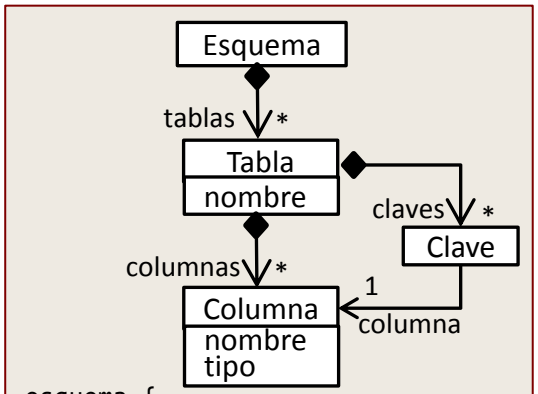

```
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

Cómo definir LDEs internos en Ruby

Primer enfoque, clase de contexto (LDE)

```
class EsquemaDSL

  def self.esquema
    @esquema = Esquema.new

  end

  # crea tabla y la añade a esquema
  def self.tabla (nombre)
    t = Tabla.new(nombre)
    @esquema.add_tabla(t)

  end

  # crea columna y la añade a la
  # última tabla creada
  def self.columna (nombre, tipo)
    tabla   = @esquema.get_tabla( @esquema.num_tablas-1 )
    columna = Columna.new(nombre,tipo)
    tabla.add_columna(columna)
  end

  # crea clave y la añade a la última tabla creada
    def self.clave (columna)
      tabla   = @esquema.get_tabla( @esquema.num_tablas-1 )
      # busca columna por nombre
      columna = tabla.find_columna(columna)
      clave = Clave.new(columna)
      tabla.add_clave(clave)
    end

end
```

Los métodos crean uno o más objetos del modelo de dominio, o realizan alguna acción. En el primer caso, reciben el valor de las variables de instancia del objeto.



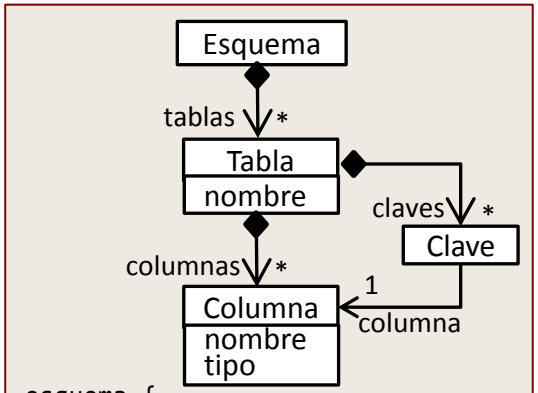

```
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

Cómo definir LDEs internos en Ruby

Primer enfoque, clase de contexto (LDE)

```
class EsquemaDSL

  def self.esquema
    @esquema = Esquema.new
    yield if block_given?
  end

  # crea tabla y la añade a esquema
  def self.tabla (nombre)
    t = Tabla.new(nombre)
    @esquema.add_tabla(t)
    yield if block_given?
  end

  # crea columna y la añade a la
  # última tabla creada
  def self.columna (nombre, tipo)
    tabla   = @esquema.get_tabla( @esquema.num_tablas-1 )
    columna = Columna.new(nombre,tipo)
    tabla.add_columna(columna)
  end

  # crea clave y la añade a la última tabla creada
  def self.clave (columna)
    tabla   = @esquema.get_tabla( @esquema.num_tablas-1 )
    # busca columna por nombre
    columna = tabla.find_columna(columna)
    clave = Clave.new(columna)
    tabla.add_clave(clave)
  end

end
```

Los métodos ejecutan el bloque de código que reciben.

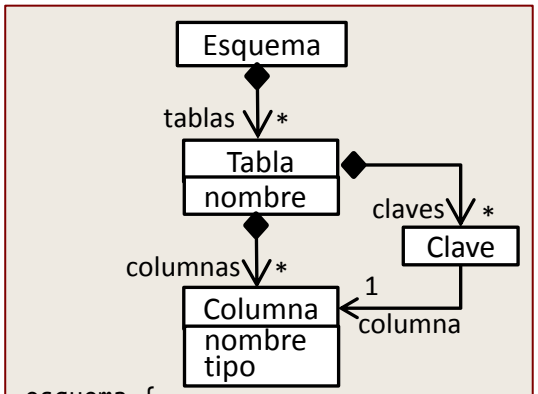



```
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

Cómo definir LDEs internos en Ruby

Primer enfoque, clase de contexto (LDE)

```
class EsquemaDSL

  def self.esquema
    @esquema = Esquema.new
    yield if block_given?
  end

  # crea tabla y la añade a esquema
  def self.tabla (nombre)
    t = Tabla.new(nombre)
    @esquema.add_tabla(t)
    yield if block_given?
  end

  # crea columna y la añade a la
  # última tabla creada
  def self.columna (nombre, tipo)
    tabla   = @esquema.get_tabla( @esquema.num_tablas-1 )
    columna = Columna.new(nombre,tipo)
    tabla.add_columna(columna)
  end

  # crea clave y la añade a la última tabla creada
  def self.clave (columna)
    tabla   = @esquema.get_tabla( @esquema.num_tablas-1 )
    # busca columna por nombre
    columna = tabla.find_columna(columna)
    clave = Clave.new(columna)
    tabla.add_clave(clave)
  end

  # devuelve el esquema
  def self.get_esquema
    @esquema
  end

end
```




```
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

Método de clase adicional para acceder a los objetos creados.

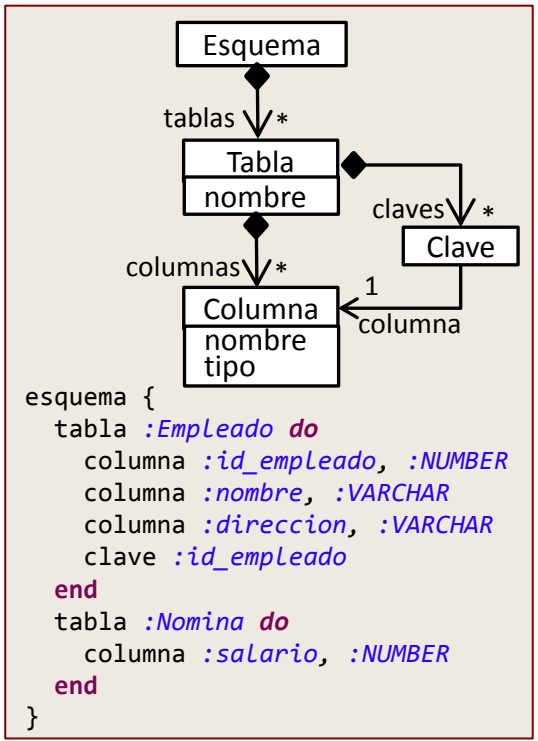
Cómo definir LDEs internos en Ruby

Primer enfoque, cómo usar el LDE

```
class Empresa < EsquemaDSL
  esquema {
    tabla :Empleado do
      columna :id_empleado, :NUMBER
      columna :nombre, :VARCHAR
      columna :direccion, :VARCHAR
      clave :id_empleado
    end
    tabla :Nomina do
      columna :salario, :NUMBER
    end
  }
end

e = Empresa.get_esquema
e.generar_codigo_SQL
e.generar_código_ruby
...
```

Subclasificamos la clase de contexto para definir nuestros “programas” como llamadas a los métodos de la superclase.

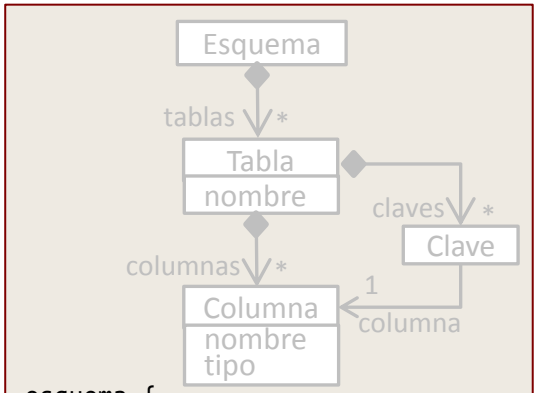
Este enfoque permite extender LDEs existentes fácilmente.



Cómo definir LDEs internos en Ruby

¿Y si no necesito crear objetos?

```ruby
class EsquemaDSL

  def self.esquema
    yield if block_given?
  end

  def self.tabla (nombre)
    @columnas = ""
    @columnas = yield if block_given?
    dbh = DBI.connect(...)
    dbh.do( "CREATE TABLE '#{nombre}' (#{columnas})" )
  end

  def self.columna (nombre, tipo, is_key=false)
    texto = 'PRIMARY KEY '
    @columnas+= "#{nombre} #{tipo} #{texto if is_key}, "
  end

end
```

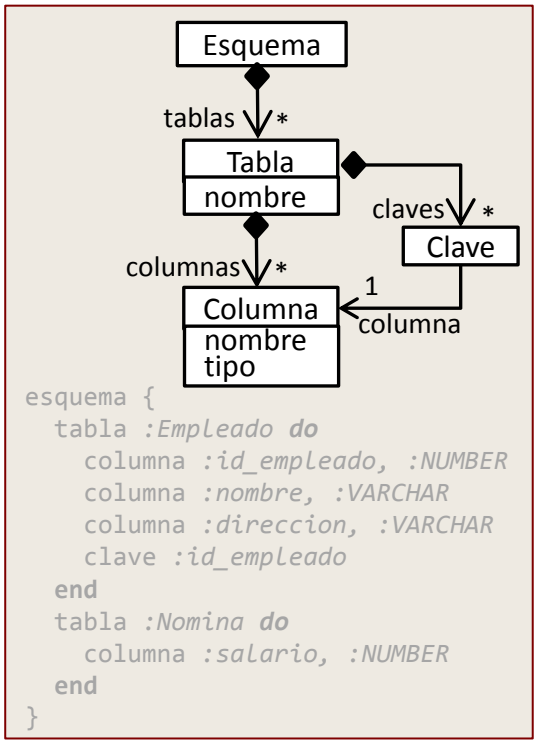



```ruby
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

Cómo definir LDEs internos en Ruby

Variaciones en la sintaxis del LDE – tablas hash

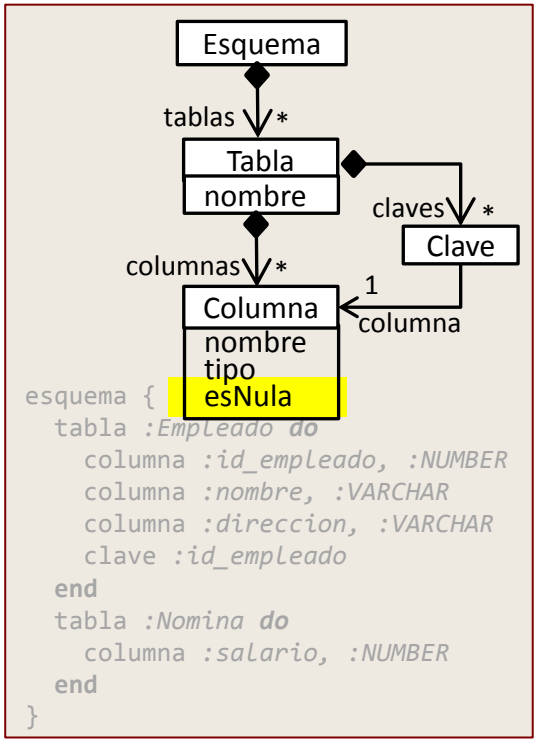
```
class EsquemaDSL
  ...
  def self.tabla (nombre, columnas={})
    t = Tabla.new(nombre)
    @esquema.add_tabla(t)
    columnas.keys.each { |nombre| t.add_columna( Columna.new(nombre, columnas[nombre]) ) }
  end
  ...
end

# ------------------------------------------------

class Empresa < EsquemaDSL
  esquema {
    tabla :Empleado,
          :id_empleado => :NUMBER,
          :nombre => :VARCHAR,
          :direccion => :VARCHAR
    tabla :Nomina,
          :salario => :NUMBER
  }
end
```



Cómo definir LDEs internos en Ruby

Variaciones en la sintaxis del LDE – *fluent interfaces*

```
class EsquemaDSL
  ...
  def self.columna (nombre, tipo)
    tabla   = @esquema.get_tabla( @esquema.num_tablas-1 )
    columna = Columna.new(nombre,tipo)
    tabla.add_columna(columna)
    columna
  end
  ...
end

class Columna
  ...
  def noNula; @esNula = false end
end
# ------------------------------------------------

class Empresa < EsquemaDSL
  esquema {
    tabla :Empleado do
      columna(:id_empleado, :NUMBER) .noNula
      columna(:nombre, :VARCHAR)
    end
  }
end
```

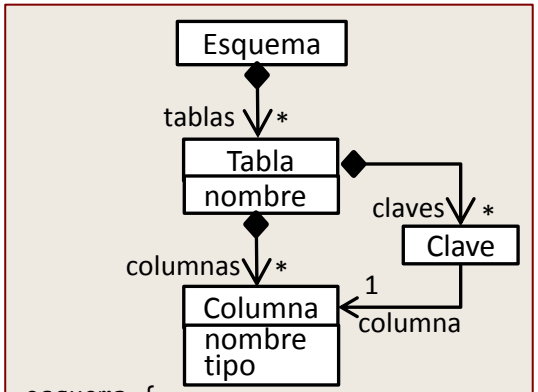


Cómo definir LDEs internos en Ruby

Segundo enfoque, conceptos del dominio

```ruby
module EsquemaDSL

  # esquema
  class Esquema
    def initialize
      @tablas = []
    end
  end

  # tabla
  class Tabla
    def initialize (nombre)
      @nombre   = nombre
      @columnas = []
      @claves   = []
    end
  end

  # clave
  class Clave
    def initialize(columna)
      @columna = columna
    end
  end
end
```

```ruby
  # columna
  class Columna
    def initialize (nombre, tipo)
      @nombre = nombre
      @tipo   = tipo
    end
    attr_reader :nombre
```




```ruby
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

Definir conceptos del lenguaje en un espacio de nombres.

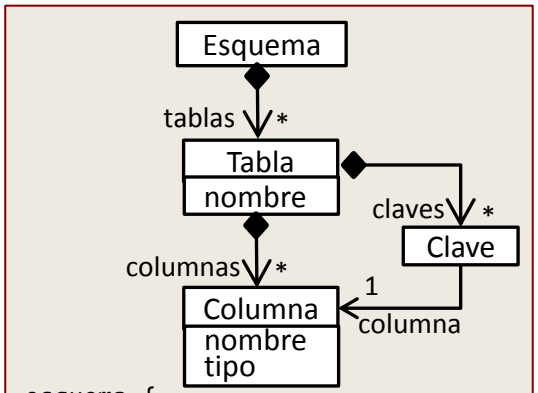
Cómo definir LDEs internos en Ruby

Segundo enfoque, *keywords* del LDE

```ruby
module EsquemaDSL

  # punto de entrada
  def self.esquema
    esquema = Esquema.new
    ...
    esquema
  end

  # esquema
  class Esquema
    def initialize
      @tablas = []
    end
    def tabla (nombre)
      tabla = Tabla.new(nombre)
      ...
      @tablas << tabla
    end
  end
```

```ruby
  # tabla
  class Tabla
    def initialize (nombre)
      @nombre   = nombre
      @columnas = []
      @claves   = []
    end
    def columna (nombre, tipo)
      col = Columna.new(nombre,tipo)
      @columnas << col
    end
    def clave (nombre)
      col = @columnas.find{ |c|
            c.nombre == nombre}
      clave = Clave.new(nombre)
      @claves << clave
    end
  end

  # columna
  class Columna
    def initialize (nombre, tipo)
```

```ruby
  # clave
  class Clave
    def initialize(columna)
      @columna = columna
    end
  end

end
```




```ruby
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

Definir un método de instancia por cada *keyword* del LDE. Los métodos crean objetos del modelo de dominio, y se definen en la clase contenedora del objeto creado.

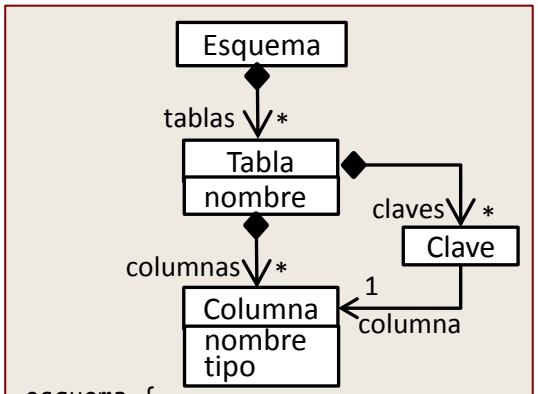
Cómo definir LDEs internos en Ruby

Segundo enfoque, `instance_eval`

```ruby
module EsquemaDSL

  # punto de entrada
  def self.create_esquema(&proc)
    esquema = Esquema.new
    esquema.instance_eval(&proc)
    esquema
  end

  # esquema
  class Esquema
    def initialize
      @tablas = []
    end
    def tabla (nombre, &proc)
      tabla = Tabla.new(nombre)
      tabla.instance_eval(&proc)
      @tablas << tabla
    end
  end
```

```ruby
  # tabla
  class Tabla
    def initialize (nombre)
      @nombre   = nombre
      @columnas = []
      @claves   = []
    end
    def columna (nombre, tipo)
      col = Columna.new(nombre,tipo)
      @columnas << col
    end
    def clave (nombre)
      col = @columnas.find{ |c|
            c.nombre == nombre}
      clave = Clave.new(nombre)
      @claves << clave
    end
  end

  # columna
  class Columna
    def initialize (nombre, tipo)
      @nombre = nombre
```

```ruby
  # clave
  class Clave
    def initialize(columna)
      @columna = columna
    end
  end

end
```



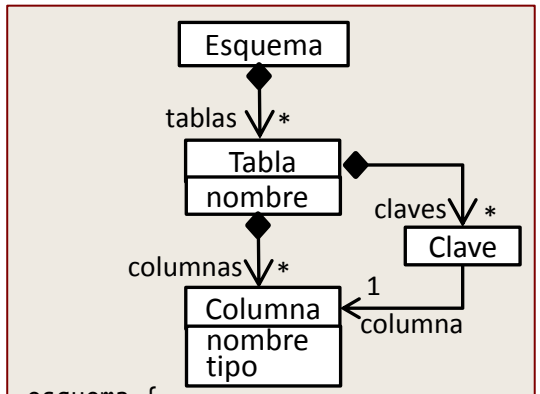

```ruby
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

Los métodos ejecutan el bloque de código que reciben en el contexto del objeto creado, usando `instance_eval`.

Cómo definir LDEs internos en Ruby

Segundo enfoque, cómo usar el LDE

```
e = EsquemaDSL.esquema {
      tabla :Empleado do
        columna :id_empleado, :NUMBER
        columna :nombre, :VARCHAR
        columna :direccion, :VARCHAR
        clave :id_empleado
      end
      tabla :Nomina do
        columna :salario, :NUMBER
      end
    }

e.generar_codigo_SQL
e.generar_código_ruby
...
```




```
esquema {
  tabla :Empleado do
    columna :id_empleado, :NUMBER
    columna :nombre, :VARCHAR
    columna :direccion, :VARCHAR
    clave :id_empleado
  end
  tabla :Nomina do
    columna :salario, :NUMBER
  end
}
```

Llamar al método de módulo definido (punto de entrada).