Dizzy Making business beautiful.

Cheatsheet * Migrations

Column Types[1]

	db2	mysql	openbase	Oracle	postgresql	sqlite	sqlserver	Sybase
:binary	blob(32768)	blob	object	blob	bytea	blob	image	image
:boolean	decimal(1)	tinyint(1)	boolean	number(1)	boolean	boolean	bit	bit
:date	date	date	date	date	date	date	datetime	datetime
:datetime	timestamp	datetime	datetime	date	timestamp	datetime	datetime	datetime
:decimal	decimal	decimal	decimal	decimal	decimal	decimal	decimal	decimal
:float	float	float	float	number	float	float	float(8)	float(8)
:integer	int	int(11)	integer	number(38)	integer	integer	int	int
:string	varchar(255)	varchar(255)	char(4096)	varchar2(255)	*	varchar(255)	varchar(255)	varchar(255)
:text	clob(32768)	text	text	clob	text	text	text	text
:time	time	time	time	date	time	datetime	datetime	time
:timestamp	timestamp	datetime	timestamp	date	timestamp	datetime	datetime	timestamp

Shortcut methods[2]

t.column	t.change	t.rename
t.remove	t.change_default	t.references
t.remove_references	t.belongs_to	t.remove_belongs_to
t.timestamps	t.index	t.remove_index

directory structure

```
db
  schema.rb
    migrate
      20080812205401_create_products.rb
      20080812215015_add_data.rb
```

Table methods

change_table

Provides a block that enables you to alter columns on an existing table using various shortcut methods[2]

```
change_table :table_name, {options} do |t|
  t.change :column_name, :new_column_type
  t.remove :column_name
end
```

create_table

Creates a table on the database. Creates a table called `:table_name` and makes the table object available to a block that can then add columns to it by specifying column_types[1] or utilising shortcut methods[2] such as using belongs_to to specify foreign keys.

```
create_table :table_name, {table_options} do |t|
  t.string :name, {column_options}
end
```

{table_options}

:force	*true or false*	if `true`, forces drop of an existing table of the same name before creation the new one
:temporary	*true or false*	if `true`, creates a temporary table, one that goes away when the application disconnects from the database
:id	*true or false*	if `false`, defines a table with no primary key, for example when you need to define a join table
:primary_key	*:symbol*	overrides the default name of `:id` for the primary column. Use this to specify the name of the column in the database that Rails will use to store the primary key
:options	*"string"*	pass raw options to your underlying database, e.g. `auto_increment = 10000`. Note that passing options will cause you to lose the default `ENGINE=InnoDB` statement

drop_table

Destroys the specified table.

```
drop_table :table_name
```

rename_table

Renames the specified table.

```
rename_table :old_table_name, :new_table_name
```

Column methods

add_column

Creates a new column on the specified table.

```
add_column :table_name, :column_name, :column_type, {column_options}
```

{column_options}

:null	*true or false*	if `false`, the underlying column has a not null constraint added by the database engine
:limit	*integer*	set a limit on the size of the field
:default	*string*	set a default value for the column
:precision	*integer*	Specifies the precision for a `:decimal` column.
:scale	*integer*	Specifies the scale for a `:decimal` column.

change_column

Change the data type of the specified column

```
change_column :table_name, :column_name, :new_column_type
```

rename_column

Renames the specified column.

```
rename_column :table_name, :old_column_name, :new_column_name
```

remove_column

Removes the specified column.

```
remove_column :table_name, :column_name
```

script/generate

```
script/generate migration new_migration_filename field_name:column_type name:string age:integer date_of_birth:date
```

example_migration.rb

```
class CreateCustomers < ActiveRecord::Migration
  def self.up
    create_table :customers, :primary_key => :customer_id, :options =>
"auto_increment = 10000" do |t|
      t.integer :customer_id
      t.string :name,          :limit   => 30, :null => false
      t.integer :age
      t.boolean :premium,      :default => 0
      t.binary :photo,         :limit   => 2.megabytes
      t.timestamps
      t.text :notes,           :default => "No notes recorded"
    end
    add_column :customers, :surname,  :string,   :limit => 50
    add_column :orders,    :price,    :decimal,  :precision => 8,
:scale => 2
    Customer.create :name => "David", :surname => "Smith", :age =>
"32", :premium => "1", :notes => "One of our top customers!"
  end
def self.down
    drop_table :customers
  end
end
```

Fixtures

Fixtures contain data which can be loaded into your database using migrations. For example, to load data into a table named customers...

1. Create a directory, `db/migrate/data`
2. Create a file, `customers.yml`, inside `db/migrate/data`
3. Generate a new migration file: `ruby script/generate migration load_customers_data`
4. Edit it to load data from the `customers.yml` file into your customers table

customers.yml

```
melissa:
  name: Melissa
  age: 18
david:
  name: David
  age: 23
```

migration.rb

```
require 'active_record/fixtures'
class LoadCustomerData
 def self.up
  down
  directory = File.join(File.dirname(__FILE__),
"data")
  Fixtures.create_fixtures(directory, "customers")
 end
 def self.down
  Customer.delete_all
 end
end
```

Miscellaneous methods

execute

Takes a single string identifying a valid SQL command to execute directly.

```
execute "alter table line_items add constraint fk_line_item_products foreign key (product_id) references products(id)"
```

IrreversibleMigration

Use in the `down` method of a migration file to raise an exception when the `up` methods of the same migration file can not be reversed, e.g. changing a column type from `:integer` to `:string`.

```
raise ActiveRecord::IrreversibleMigration
```

Indexes

add_index

Creates an index for the specified column.

```
add_index :table_name, :column_name, :unique => true
```

remove_index

Remove an index ffrom the specified column.

```
remove_index :table_name, :column_name
```

Rake tasks

db:create db:create:all	Creates a single database specified in `config/databases.yml` for the current `RAILS_ENV` or creates all the databases
db:drop db:drop:all	Drops a single database specified in `config/databases.yml` for the current `RAILS_ENV` or drops all the databases
db:fixtures:load	Load fixtures from `test/fixtures` into the current environment's database
db:migrate	Run all unapplied migrations
db:migrate:up db:migrate:down	Move forward to the next migration, or back to the previous migration
db:migrate VERSION=18	Migrate database to specific version
db:migrate RAILS_ENV=production	Use migrations to recreate tables in the testing or production databases
db:schema:dump	Create a `db/schema.rb` file that can be portably used against any database supported by ActiveRecord
db:schema:load	Load a `schema.rb` file into the database
db:sessions:create	Create a sessions table for use with `CGI::Sessions::ActiveRecordStore`
db:sessions:clear	Clear the sessions table
db:structure:dump	Dump database structure to SQL file
db:reset	Drops the database, creates the database and then runs migrations against the database. Takes a `VERSION` argument as well as `RAILS_ENV`
db:rollback STEP=4	Takes a `STEP` argument to determine how many version to rollback, the default being one version
db:test:prepare	Clone your database structure into the test database
db:version	Tells you the current version your database is at