How does Ruby on Rails work with databases?

Oct 22th, 2018
at Oracle Code One 2018

Yasuo Honda

Hello!

- Yasuo Honda from Tokyo, Japan
- First time to attend Oracle Code One / Open World
  - https://twitter.com/yahonda
  - https://github.com/yahonda
  - https://www.linkedin.com/in/yahonda/

Hello!

- Maintainer of Active Record Oracle enhanced adapter
  - https://rubygems.org/gems/activerecord-oracle_enhanced-adapter
- Ruby on Rails contributor
  - https://contributors.rubyonrails.org/contributors/yasuo-honda/commits
- Oracle ACE
  - https://apex.oracle.com/pls/apex/f?p=19297:4:::NO:4:P4_ID:16420
- Software engineer at freee K.K.
  - https://www.bloomberg.com/research/stocks/private/snapshot.asp?privcapId=243549317

Topics

1. Introduction of Ruby on Rails
2. Active Record and databases
3. Ruby on Rails 6 new features

1. Introduction of Ruby on Rails

Ruby on Rails

- Full stack web application framework
- Written in Ruby
- Made of components called “rubygems”
- MVC : Active Record(M), Action View(V) and Action Controller(C)
- Latest version is 5.2. , 6.0 will be the next version

Active Record

- Object relation mapping (ORM) framework
- Work with relational databases
  - SQLite3, PostgreSQL and MySQL supported by Rails bundled adapters
  - Oracle, SQL Server supported by 3rd party

Active Record examples - Migration

Active Record examples - Migration

- Generate Post and Comment “models” using rails generator
- Executing rails generator creates migration files
- Running rails migration to create database tables for two models based on migration files

$ rails generate model

```
$ rails generate model post title:string body:text
      invoke  active_record
       create    db/migrate/20181014082244_create_posts.rb
       create    app/models/post.rb
       invoke    test_unit
       create      test/models/post_test.rb
       create      test/fixtures/posts.yml
$ rails generate model comment post:references body:text
      invoke  active_record
       create    db/migrate/20181014082249_create_comments.rb
       create    app/models/comment.rb
       invoke    test_unit
       create      test/models/comment_test.rb
       create      test/fixtures/comments.yml
```

Migration file for Post

```
$ more db/migrate/20181014082244_create_posts.rb
class CreatePosts < ActiveRecord::Migration[5.2]
  def change
    create_table :posts do |t|
      t.string :title
      t.text :body

      t.timestamps
    end
  end
end
```

Migration file for Comment

```
$ more db/migrate/20181014082249_create_comments.rb
class CreateComments < ActiveRecord::Migration[5.2]
  def change
    create_table :comments do |t|
      t.references :post, foreign_key: true
      t.text :body

      t.timestamps
    end
  end
end
```

$rails db:migrate to create tables

```
$ bundle exec rails db:migrate
== 20181014082244 CreatePosts: migrating ======================================
-- create_table(:posts)
   -> 0.0812s
== 20181014082244 CreatePosts: migrated (0.0842s) =============================

== 20181014082249 CreateComments: migrating ===================================
-- create_table(:comments)
   -> 0.0947s
== 20181014082249 CreateComments: migrated (0.0963s) ==========================
```

Active Record Migration

- $ rails generate model post title:string body:text
- Use Rails data type, not database type
  - :string, :text
- Primary key and timestamps added by default
  - `id` is the default name of the primary key
  - `:timestamps` type `created_at` and `updated_at` attributes

“id” as primary_key by default

- Convention over Configuration
- One of "The Rails Doctrine”

Who cares what format your database primary keys are described by? Does it really matter whether it's “id”, “postId”, “posts_id”, or “pid”? Is this a decision that’s worthy of recurrent deliberation? No.

https://rubyonrails.org/doctrine/

Active Record Models

```
$ more app/models/post.rb
class Post < ApplicationRecord
  has_many :comments
end


$ more app/models/comment.rb
class Comment < ApplicationRecord
  belongs_to :post
end
```

Active Record Models

- Generated by Active Record generators
- Singular model name “Post”, plural table name like “POSTS”
- No metadata (like data type, attribute names) in the model files
  - Retrieved from database catalog, not from model files
- Associations between models are represented in the model code
  - Post “has_many” Comments 1:N relationship
  - Comments “belongs_to” User N:1 relationship

Behind the scene

- Rails migration code is translated into SQL statements by Active Record connection adapters
  - Abstract adapter for generic databases
  - `mysql2` adapter for MySQL database
  - `oracle_enhanced` adapter for Oracle database

[MySQL] $ rails db:migrate

```
# Post
CREATE TABLE `posts` (`id` bigint NOT NULL AUTO_INCREMENT PRIMARY KEY,
`title` varchar(255), `body` text, `created_at` datetime NOT NULL,
`updated_at` datetime NOT NULL)

# Comment
CREATE TABLE `comments` (`id` bigint NOT NULL AUTO_INCREMENT PRIMARY
KEY, `post_id` bigint, `body` text, `created_at` datetime NOT NULL,
`updated_at` datetime NOT NULL,  INDEX `index_comments_on_post_id`
(`post_id`), CONSTRAINT `fk_rails_2fd19c0db7`
FOREIGN KEY (`post_id`)
  REFERENCES `posts` (`id`)
```

[Oracle] $ rails db:migrate

```
# Post
CREATE TABLE "POSTS" ("ID" NUMBER(38) NOT NULL PRIMARY KEY, "TITLE" VARCHAR2(255),
"BODY" CLOB, "CREATED_AT" TIMESTAMP NOT NULL, "UPDATED_AT" TIMESTAMP NOT NULL)
CREATE SEQUENCE "POSTS_SEQ" START WITH 10000

# Comment
CREATE TABLE "COMMENTS" ("ID" NUMBER(38) NOT NULL PRIMARY KEY, "POST_ID"
NUMBER(19), "BODY" CLOB, "CREATED_AT" TIMESTAMP NOT NULL, "UPDATED_AT" TIMESTAMP
NOT NULL, CONSTRAI
NT "FK_RAILS_2FD19C0DB7"
FOREIGN KEY ("POST_ID")
  REFERENCES "POSTS" ("ID"))
CREATE SEQUENCE "COMMENTS_SEQ" START WITH 10000
```

Active Record examples - querying

Active Record examples - querying Post and Comment

- Query Post and Comment models using Active Record query methods
  - Method chain
- Examples:
  - Create a post
  - Create a first comment to the post
  - Find the first comment of the first post

[MySQL] Create a post

```
$ bundle exec rails c
> Post.create!(title: "Hello World", body: "This is my first
post using the new Rails application.”)

# SQL statement executed by this command
INSERT INTO `posts` (`title`, `body`, `created_at`,
`updated_at`) VALUES ('Hello World', 'This is my first post
using the new Rails application.', '2018-10-14 08:45:23',
'2018-10-14 08:45:23')
```

[MySQL] Create a comment to the post

```
> first_post = Post.first
# SQL statement executed by this command
SELECT  `posts`.* FROM `posts` ORDER BY `posts`.`id` ASC LIMIT 1


> first_post.comments.create(body: "Congratulation for your first
post")
# SQL statement executed by this command
INSERT INTO `comments` (`post_id`, `body`, `created_at`, `updated_at`)
VALUES (1, 'Congratulation for your first post', '2018-10-14 08:51:44',
'2018-10-14 08:51:44')
```

[MySQL] Find the first comment of the first post

```
> first_comment = Post.first.comments.first
# SQL statements executed by this command
SELECT  `posts`.* FROM `posts` ORDER BY `posts`.`id` ASC
LIMIT 1
SELECT  `comments`.* FROM `comments` WHERE
`comments`.`post_id` = 1 ORDER BY `comments`.`id` ASC LIMIT
1
> first_comment.body
=> "Congratulation for your first post"
```

[Oracle] Create a post

```
$ bundle exec rails c
> Post.create!(title: "Hello World", body: "This is my first
post using the new Rails application.”)
SELECT "POSTS_SEQ".NEXTVAL FROM dual
INSERT INTO "POSTS" ("TITLE", "BODY", "CREATED_AT",
"UPDATED_AT", "ID") VALUES (:a1, :a2, :a3, :a4, :a5)
[["title", "Hello World"], ["body", #<OCI8::CLOB:
0x00007f92dc19a3e0>], ["created_at", "2018-10-14
09:49:27.407027"], ["updated_at", "2018-10-14
09:49:27.407027"], ["id", 10000]]
```

[Oracle] Create a comment to the post

```
> first_post = Post.first
SELECT  "POSTS".* FROM "POSTS" ORDER BY "POSTS"."ID" ASC FETCH
FIRST :a1 ROWS ONLY  [["LIMIT", 1]]

> first_post.comments.create(body: "Congratulation for your first
post")
SELECT "COMMENTS_SEQ".NEXTVAL FROM dual
INSERT INTO "COMMENTS" ("POST_ID", "BODY", "CREATED_AT", "UPDATED_AT",
"ID") VALUES (:a1, :a2, :a3, :a4, :a5)  [["post_id", 10000], ["body",
#<OCI8::CLOB:0x0000558776b7efc8>], ["created_at", "2018-10-14
09:52:33.827675"], ["updated_at", "2018-10-14 09:52:33.827675"], ["id",
10000]]
```

[Oracle] Find the first comment of the first post

```
> first_comment = Post.first.comments.first
SELECT  "POSTS".* FROM "POSTS" ORDER BY "POSTS"."ID" ASC
FETCH FIRST :a1 ROWS ONLY  [["LIMIT", 1]]
SELECT  "COMMENTS".* FROM "COMMENTS" WHERE
"COMMENTS"."POST_ID" = :a1 ORDER BY "COMMENTS"."ID" ASC
FETCH FIRST :a2 ROWS ONLY  [["post_id", 10000], ["LIMIT",
1]]
> first_comment.body
=> "Congratulation for your first post"
```

Rails application developers write Ruby,
Rails framework translates them into SQL

“Conceptual compression means beginners don’t need to know SQL—hallelujah!”

DHH, creator of Ruby on Rails

2. Active Record and database

Active Record handles these differences

- SQL syntax : `LIMIT n` or `FETCH FIRST n ROWS ONLY`
- Quote: ` or “
- Identifier: lowercase or UPPERCASE identifiers
- `auto_increment` or “SELECT … SEQ.NEXTVAL”
- Data types:

Rails data types

Rails data types

- :string
- :integer
- :boolean
- :primary_key and more…
- : means the Ruby “symbol”

:string

- MySQL
  - VARCHAR(255)
- Oracle
  - VARCHAR2(255)

:integer

- MySQL
  - INT(4)
  - Here (4) means number of bytes used
- Oracle
  - NUMBER(38)
  - Here (38) means number of digits

:boolean

- No native boolean type in MySQL and Oracle
- MySQL
  - TINYINT(1)
  - If `emulate_booleans` is `true`
- Oracle
  - NUMBER(1) or VARCHAR2(1)
  - If `emulate_booleans_from_strings` is `true`

:primary_key

- primary key is a data type in Rails
- MySQL
  - `int auto_increment PRIMARY KEY`
  - `int` is a alias for `INT(4)`
- Oracle
  - “NUMBER(38) NOT NULL PRIMARY KEY”
  - Another "CREATE SEQUENCE” is required

INT(4) is not enough to store large data

bigint for :primary_key

- MySQL `INT(4)` store:
  - 2,147,483,647 (signed) records (about two billions)
- MySQL `BIGINT(8)` data can store:
  - 9,223,372,036,854,775,807 (unsigned)
  - Primary keys do not need signed
- Rails 5.1 creates :primary_key using `BIGINT(8)` by default

Migrating the :primary_key definition

- Easy to replace `INT(4)` with `BIGINT(8)` for new migrations
- But...

How does the old migration file, created when primary key was INT(4) work after primary_key is BIGINT(8)?

Expected behavior to support old and new migrations for :primary_key

- Old (created before Rails 5.1) migration should create:
  - :primary_key as `INT(4)`
- New migration (created after Rails 5.1) should create :
  - :primary_key as `BIGINT(8)`

Introduce ActiveRecord::Migration[VERSION] And Migration::Compatibility

ActiveRecord::Migration[VERSION]

```
$ more db/migrate/20181014082249_create_comments.rb
class CreateComments < ActiveRecord::Migration[5.2]
  def change
    create_table :comments do |t|
      t.references :post, foreign_key: true
      t.text :body

      t.timestamps
    end
  end
end
```

ActiveRecord::Migration[VERSION]

- Migration files inherits `ActiveRecord::Migration[VERSION]`
- If Migration[5.1] or higher
  - :primary_key is translated into `BIGINT(8)`
- If Migration[5.0] or lower
  - :primary_key is translated into `INT(4)`

Migration::Compatibility

- Module to support ActiveRecord::Migration[VERSION]
- Support migration version differences
  - i.e. Primary key as :bigint after Rails 5.1
- Support differences between database
  - i.e. Add options: `ENGINE=InnoDB` only for MySQL
- Available only for bundled adapters (SQLite, PostgreSQL and MySQL)

Rails supports more data types specific to each database

:json and :virtual

JSON type support

- Supported by these databases:
  - MySQL 5.7.8 or higher
  - PostgreSQL 9.2 or higher
- JSON support for Oracle is not available in Rails yet
  - No "JSON" datatype, datatype is `VARCHAR2`
  - JSON is enabled by constraint over `VARCHAR2`

Virtual type support

- Also known as generated columns
- Supported by:
  - MySQL 5.7.5 or higher
  - MariaDB 5.2.0 or higher
- Support syntax differences to store generated value to disk
  - Stored (MySQL) or Persistent (MariaDB)

Virtual type example

```
create_table :generated_columns do |t|
  t.string  :name
  t.virtual :upper_name,  type: :string,
      as: 'UPPER(name)'
  t.virtual :name_length, type: :integer,
      as: 'LENGTH(name)', stored: true
end
```

MySQL 8 support

MySQL 8 support

- Rails 5.2 supports MySQL 8.0
- Issues about supporting MySQL 8 were addressed while MySQL was in development milestone release (DMR)

MySQL 8 support

- `information_schema.key_column_usage` shows only one of two foreign keys
- Reported: “MySQL 8.0.0-dmr test_multiple_foreign_keys_can_be_added_to_the_same_table fails due to only 1 fk information shown”
  - https://github.com/rails/rails/issues/26476
- Fixed: "Information_schema Foreign key meta data differs in 8.0.0”
  - https://bugs.mysql.com/bug.php?id=82961

MySQL 8 support

- MySQL 8.0.1 deprecated `my_bool`
- Fixed : “Use `bool` instead of `my_bool` which has been removed since MySQL 8.0.1”
- https://github.com/brianmario/mysql2/pull/840

3. Ruby on Rails 6

Ruby on Rails 6

- Next major version of Ruby on Rails
- Current version is the 5.2
- Currently in alpha

Ruby on Rails 6 new features for databases

- Multiple database connections support
- `utf8mb4` is the default character set for MySQL
- [Planned] Identity datatype support for Oracle12c+
- [WIP] Migration::Compatibility per database adapter

Multiple database connections support

Multiple database connections support

- Rails 5.2: one Rails application has only one database connection
- Rails 6: one Rails application can have two or more database connections

Multiple database connectons support

```
* database.yml
production:
  primary:
    <<: *default
    database: primary_database
  replica:
    <<: *default
    database: replica_database

* Model
class AnimalsBase < ApplicationRecord
  connects_to database:
  { writing: :primary, reading: :replica }
end
```

Multiple database support

- Active Record models can have multiple connections
  - Read / Write splitting for MySQL replication
- RailsConf 2018 keynote presentation
  - https://speakerdeck.com/eileencodes/railsconf-2018-the-future-of-rails-6-scalable-by-default

`utf8mb4` is the default character set

`utf8mb4` is the default character set

- https://github.com/rails/rails/pull/33608
- Rails 5.2 default character set is `utf8`
  - 3 byte encoding
- `utf8mb4` to support emoji 😀
  - 4 byte encoding

`utf8mb4` has been supported since MySQL 5.5 in 2010.
Why will it be default in 2019?

Rails 5.2 supports MySQL 5.1

Rails 6 drops MySQL 5.1 support

The biggest reason is...

"ERROR 1071 (42000): Specified key was too long; max key length is 767 bytes”

Why ERROR 1071 occurs?

- InnoDB key length was limited to 767 bytes
- Rails `:string` datatype is mapped to MySQL `VARCHAR(255)`

Why ERROR 1071 occurs?

- Calculate index key prefix size for string:
  - `utf8` (3 byte): 255 x 3 = 765
    - Smaller than 767
  - `utf8mb4` (4 byte): 255 x 4 = 1020
    - Larger than 767, getting ERROR 1071

Creating a database in Rails 6

- MySQL 5.5 and 5.6 users will get:
  - “Configure a supported :charset and ensure innodb_large_prefix is enabled to support indexes on varchar(255) string columns.”
- `utf8mb4` is supported but not enabled by default for MySQL 5.5 and 5.6 users
  - Because it easily causes ERROR 1071, the error message asks users to:
    - Configure `innodb_large_prefix` in my.cnf if MySQL is 5.5.14 or higher
    - Configure `:charset utf8mb4` explicitly at database.yml

Creating a database in Rails 6

- MySQL 5.7.9 or higher users:
  - `utf8mb4` is the new default charset for Rails 6
  - No configuration necessary in my.conf and database.yml
  - `innodb_default_row_format = DYNAMIC` by default to support longer key prefix, 3072 byte
    - https://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html#sysvar_innodb_default_row_format

[Planned] Identity data type support with Oracle Database 12c+

Identity data type support with Oracle Database 12c+

- “IDENTITY” type : like `auto_increment` support for Oracle
- Current primary_key:
  - “NUMBER(38) NOT NULL PRIMARY KEY”
  - “CREATE SEQUENCE”
- :primary_key will be:
  - “NUMBER(38) GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY”
  - No “CREATE SEQUENCE” necessary

Identity data type support with Oracle Database 12c+

- Oracle enhanced adapter will need to support two types or `:primary_key`
- Another Migration[VERSION] is required to handle two types of :primary_key
- Migration[VERSION] is not available for 3rd party adapter in Rails 5.2

[WIP] Migration compatibility support per database adapter

- https://github.com/rails/rails/pull/33269
- Support Migration::Compatibility for 3rd party adapters
- Provide better refactored code for bundled adapters

Topics covered:

1.Introduction of Ruby on Rails

2.Active Record and databases

3.Rails 6 new features

Thank you!

@yahonda