Ruby master - Feature #16005

A variation of Time.iso8601 that can parse yyyy-MM-dd HH:mm:ss

07/15/2019 05:21 AM - matsuda (Akira Matsuda)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Let me propose a String to Time conversion method that can parse "yyyy-MM-dd HH:mm:ss" format, which is very much similar to Time.iso8601, but delimits the date part and the time part with a space character.

This format is defined as the "timestamp string" literal in SQL 92 standard:
http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt (see P. 90)
and so this format is very widely used as the default datetime / timestamp literal for major existing RDBMS implementations.

Oracle
https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Literals.html#GUID-8F4B3F82-8821-4071-84D6-FBBA21C05AC1

SQL Server
https://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/date-time-and-timestamp-literals?view=sql-server-2017

PostgreSQL
https://www.postgresql.org/docs/11/datatype-datetime.html#id-1.5.7.13.19.7.2

MySQL
https://dev.mysql.com/doc/refman/8.0/en/datetime.html

SQLite3
https://www.sqlite.org/lang_datefunc.html

In order to handle this conversion in Ruby on Rails framework, we define our own String => Time conversion method
https://github.com/rails/rails/blob/b4c715fe/activemodel/lib/active_model/type/helpers/time_value.rb#L62-L76
and Time => String conversion for now,
https://github.com/rails/rails/blob/b4c715fe/activesupport/lib/active_support/core_ext/time/conversions.rb#L7-L59
and I think it's nicer if we had them in the language level with a faster implementation.

As for the method name, maybe we can name it Time.sql92, Time.sql, Time.parse_sql92 or whatever, or maybe we can add an option to Time.iso8601 if it could be regarded as a variation of Time.iso8601? (https://en.wikipedia.org/wiki/ISO_8601#cite_note-30)

History

#1 - 07/15/2019 05:39 AM - akr (Akira Tanaka)

Time object needs timezone offset.

"yyyy-MM-dd HH:mm:ss" doesn't contain it.

Always UTC?

#2 - 07/15/2019 06:07 AM - matsuda (Akira Matsuda)

How about respecting local timezone with Z, and doing the UTC without Z?

#3 - 07/15/2019 06:30 AM - jeremyevans0 (Jeremy Evans)

For many databases, if the timestamp has a fractional component, the fractional component will be included, and if the type includes a timezone, the timezone offset will be included. For example, on PostgreSQL:

```
SELECT CAST(CAST('2019-07-14 23:21:36.638795-0700' AS timestamp) AS text) AS "v" LIMIT 1
-- "2019-07-14 23:21:36.638795"
SELECT CAST(CAST('2019-07-14 23:21:40.838896-0700' AS timestamptz) AS text) AS "v" LIMIT 1
-- "2019-07-14 23:21:40.838896-07"
```

I'm guessing the expectation to handle these formats as well (with a variable number of decimal points supported), as Time.iso8601 does?

I would be OK with modifying Time.iso8601 to make either T or space a valid date/time separator.

#4 - 07/15/2019 06:34 AM - duerst (Martin Dürst)

matsuda (Akira Matsuda) wrote:

> How about respecting local timezone with Z, and doing the UTC without Z?

This seems backwards. Timezone 'Z' denotes UTC, see e.g. https://www.timeanddate.com/time/zones/z.

#5 - 07/15/2019 07:45 AM - matsuda (Akira Matsuda)

> This seems backwards. Timezone 'Z' denotes UTC, see e.g. https://www.timeanddate.com/time/zones/z.

Indeed. My mistake.

#6 - 07/15/2019 07:53 AM - matsuda (Akira Matsuda)

> I would be OK with modifying Time.iso8601 to make either T or space a valid date/time separator.

In fact this was my first proposal. I firstly asked akr (Akira Tanaka) about that exact modification on Time.iso8601, but he didn't like the idea because that behavior is against ISO 8601 specification.

#7 - 07/16/2019 02:17 AM - akr (Akira Tanaka)

matsuda (Akira Matsuda) wrote:

> > I would be OK with modifying Time.iso8601 to make either T or space a valid date/time separator.
>
> In fact this was my first proposal. I firstly asked akr (Akira Tanaka) about that exact modification on Time.iso8601, but he didn't like the idea because that behavior is against ISO 8601 specification.

I said its against XML Schema.
https://www.w3.org/TR/xmlschema-2/

Time[.#]iso8601 is alias of Time[.#]xmlschema.

ISO 8601 defines various representation of date and time.
For example, 1985102T1015Z is valid (10:15 of 102th day of 1985 in UTC).
So, practically, some profile (subset) is required.

XML Schema defines a such profile and it defines that [T] is mandatory.

ISO 8601 itself describes that [T] can be omitted by mutual agreement.

But as far as considering XML Schema as a mutual agreement, we cannot omit [T].

Apart from that, I extracted timestamp related syntax from the SQL 92 standard.

```
<timestamp literal> ::=
     TIMESTAMP <timestamp string>

<timestamp string> ::=
     <quote> <date value> <space> <time value> [ <time zone interval> ] <quote>

<date value> ::=
     <years value> <minus sign> <months value> <minus sign> <days value>

<time value> ::=
     <hours value> <colon> <minutes value> <colon> <seconds value>

<time zone interval> ::=
     <sign> <hours value> <colon> <minutes value>

<years value> ::= <datetime value>
```

```
<months value> ::= <datetime value>

<days value> ::= <datetime value>

<hours value> ::= <datetime value>

<minutes value> ::= <datetime value>

<seconds value> ::=
       <seconds integer value> [ <period> [ <seconds fraction> ] ]

<seconds integer value> ::= <unsigned integer>

<seconds fraction> ::= <unsigned integer>

<datetime value> ::= <unsigned integer>

<unsigned integer> ::= <digit>...

<sign> ::= <plus sign> | <minus sign>

<digit> ::=
     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<space> ::= !! space character in character set in use

<quote> ::= '

<plus sign> ::= +

<minus sign> ::= -

<period> ::= .

<colon> ::= :
```

I feel it is difficult to consider ISO 8601 variant because
months value, days value, hours value, minutes value and
seconds integer value can be one digit (and 3 or more digits).

Also, I'm still not certain that what Time object to be generated
when time zone interval is not given.

#8 - 07/16/2019 11:14 AM - akr (Akira Tanaka)

(1) The SQL spec. defines the range of year as 0001 to 9999.
(not in the syntax but in another table.)

However Ruby Time object can represent arbitrary integer year.

There is a RDB which works with a year outside of 0001 to 9999.
It seems SQLite3 supports -4173 to 9999.
(-4713-11-24 12:00:00 is Julian day number zero in the proleptic Gregorian calendar).

```
% sqlite3
SQLite version 3.27.2 2019-02-25 16:06:06
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> SELECT datetime("-4713-11-24 11:59:59");

sqlite> SELECT datetime("-4713-11-24 12:00:00");
-4713-11-24 12:00:00
sqlite> SELECT datetime("9999-12-31 23:59:59");
9999-12-31 23:59:59
sqlite> SELECT datetime("10000-01-01 00:00:00");

sqlite>
```

I feel our method should extend the "years value" syntax with optional "minus sign" at
beginning.

(2) Another mismatch between Ruby's Time and SQL timestamp is
a resolution of timezone offset.
Ruby can represent Rational offset but SQL supports only minutes.

Olson timezone database supports seconds.
Practically, minutes is enough for current timezones.
But there is a historical timezone which use a offset not multiple of 60 seconds:
Europe/Lisbon has a timezone offset -2205.

```
% zdump -v Europe/Lisbon|head -4
Europe/Lisbon  -9223372036854775808 = NULL
Europe/Lisbon  -9223372036854689408 = NULL
Europe/Lisbon  Sun Dec 31 23:59:59 1911 UT = Sun Dec 31 23:23:14 1911 LMT isdst=0 gmtoff=-2205
Europe/Lisbon  Mon Jan  1 00:00:00 1912 UT = Mon Jan  1 00:00:00 1912 WET isdst=0 gmtoff=0
```

I feel that we can ignore such offset, though.

(3) Method name idea: sql_timestamp

#9 - 07/16/2019 02:38 PM - akr (Akira Tanaka)

akr (Akira Tanaka) wrote:

> Also, I'm still not certain that what Time object to be generated
> when time zone interval is not given.

PostgreSQL has SET TIME ZONE command to set a time zone for a session.
https://www.postgresql.org/docs/11/datatype-datetime.html#DATATYPE-TIMEZONES

"Always UTC" seems not appropriate.

However, I feel that SQL timestamp without time zone interval is dangerous for
communication between RDB and Ruby because it is ambiguous at Autumn DST change
(and other time zone offset change).

For example,
1999-10-31 01:30:00 in US/Pacific can be interpreted as
1999-10-31 08:30:00 UTC and 1999-10-31 09:30:00 UTC.

If the method is expected to be used internally (not for human interaction),
using timestamp with time zone interval is the right way.
So, I think supporting timestamp without time zone interval is not important.