Cheatography

ruby-plsql Cheat Sheet Cheat Sheet

by Jacek Gebal (jgebal) via cheatography.com/22528/cs/4621/

Session connect

```
plsql(:default).connect! {:username => 'hr', :password => 'hr', :database => 'xe'}
# opens a default connection to database

plsql.connect! {:username => 'hr', :password => 'hr', :database => 'xe'}
# opens a default connection to database

plsql(:another).connect! {:username => 'hr', :password => 'hr', :database => 'xe'}
# opens a second connection (referenced by Symbol :another)

plsql(:another).logoff
# disconnects connection (referenced by symbol :another)
```

Transaction

```
plsql(:another).connection.autocommit = false
#disables auto commit in :another connection

plsql.autocommit?
# returns the current status of autocommit

plsql.commit
#commits a transaction in :default connection

plsql(:another).rollback
#rollbacks changes for :another connection

plsql.savepoint "save_this_point"
#sets a transaction savepoint in :default connection

plsql.rollback_to "save_this_point"
#rollbacks to specified savepoint in :default connection
```

Insert into table

```
plsql.employees.insert {:employee_id => 1, :name => 'James bond', :hire_date => Time.local(0007,07,07)}
# inserts one row into the employees table using key-value pairs (Ruby Hash object)
```

Insert into table (cont)

```
plsql.employees.insert {:employee_id => 2, :name => 'Tony Stark'}
# inserts one row into the employees table, with partial column list

plsql.employees.insert [ {:employee_id => 3, :name => 'Darth Vader'}, {:employee_id => 4, :name => 'Luke Skywalker'}]
# inserts multiple rows into the employees table using Array of Hashes

plsql.employees.insert_values *[ [ 5, 'Batman', Time.local(1-990,01,01)], [6,'Spiderman',Time.local(1999,02,02)] ]
# inserts multiple rows, specifying Array of Array of values

plsql.employees.insert_values [:employee_id, :name], *[ [ 7, 'Super-man'], [8, 'Hulk'] ]
# inserts multiple rows, specifying columns first and subset of values

plsql.employees.insert_values [ 9, 'Thor', Time.local(1990,09,09)]
# inserts one row, specifying only Array of values

plsql.employees.insert_values [:employee_id, :name], [ 10, 'Sandman' ]
# inserts one row, specifying subset of columns (Array) and Array of values
```

Select statements

```
plsql.select(:first, "SELECT * FROM employees")
{:employee_id => 1, :name => 'James bond', :hire_date => '0007-07-07'}
# returns first row of a query as a Ruby Hash

plsql.select_one("SELECT count(*) FROM employees")
10
# returns a scalar value from a first row from single column query

plsql.select_one("SELECT employee_id FROM employees WHERE 1=2")
nil
# returns nil Object (NULL) when no data found
```



By **Jacek Gebal** (jgebal)
cheatography.com/jgebal/
www.oraclethoughts.com

Published 19th July, 2015.
Last updated 10th May, 2016.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Select statements (cont)

```
plsql.select(:all, "SELECT * FROM employees ORDER BY employee_id")
```

```
[ {:employee_id => 1, :name => 'James bond', :hire_date => '0007-07-07'}, {...}, ... ]
# returns all rows from a query as an Array of Hashes
```

Delete from table/view

```
plsql.employees.delete :employee_id => 10
plsql.employees.delete "employee_id = 10"
```

```
#delete record in table with WHERE condition
```

Table/View meta-data

```
plsql.execute "CREATE OR REPLACE VIEW employees_v AS SELECT * FROM employees"
```

```
#creates a VIEW
```

```
plsql.employees_v.class
```

```
PLSQL::View
# The employees_v Object is of PLSQL::View class
```

```
plsql.employees.class
```

```
PLSQL::Table
# The employees Object is of PLSQL::Table class
```

```
plsql.employees_synonym.class
```

```
PLSQL::Table
# The emplyees_synonym Object is also of PLSQL::Table class
```

```
plsql.employees.column_names
plsql.employees_v.column_names
```

```
[ employee_id, name, hire_date ]
# returns all column names in table
```

```
plsql.employees.columns
plsql.employees_v.columns
```

```
{ :employee_id => {
    :position=>1, :data_type=>"NUMBER", :data_length=>22, :data_precision=>15, :data_scale=>0, :char_used=>nil,
    :type_owner=>nil, :type_name=>nil, :sql_type_name=>nil, :nullable => false, :data_default => nil}
  , ...}
# returns column meta-data
```

Record and Object Types

```
#Given a FUNCTION get_full_name( p_employee employees%ROWTYPE ) RETURN VARCHAR2

plsql.get_full_name( {:p_employee => {:employee_id => 2, :first_name => 'Tony', :last_name => 'Stark', :hire_date => nil} } )
plsql.get_full_name( {:employee_id => 2, :first_name => 'Tony', :last_name => 'Stark', :hire_date => nil} )
plsql.get_full_name( {'EMPLOYEE_ID' => 2, 'first_name' => 'Tony', 'last_NaMe' => 'Stark', 'hire_date' => nil} )
```

```
'Tony Stark'
```

```
# Accepts a record as a parameter (by name or by position) and executes the function returning String (VARCHAR2)
# Record fields can be defined as a Symbol (:employee_id) or as a String ('employee_id')
# Works the same way with package level record types and Oracle object types
```

Varrays and Nested Tables

```
#Given a TYPE table_of_int IS TABLE OF INTEGER;
#Given FUNCTION sum_items(p_items TABLE_OF_INT) RETURN INTEGER

plsql.sum_items( [1,2,3,4,5] )
plsql.sum_items( :p_items => [1,2,3,4,5] )
```

```
15
```

```
# Nested tables are passed in and returned as Ruby Array Object type
# Works the same way for VARRAYS
```

Associative arrays (plsql tables, index-by tables)

```
#Given a package MY_PACKAGE
# contains TYPE index_table_of_int IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;
# contains FUNCTION sum_items(p_items INDEX_TABLE_OF_INT) RETURN INTEGER;

plsql.my_package.sum_items( { -1 => 1, 5 => 2, 3 => 3, 4 => 4} )
```

```
10
```

```
# Associative arrays are passed in and returned as a Ruby Hash containing list of key value pairs
# Where key is the element position in Array and value is the value at the position
```


Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Cheatography

ruby-plsql Cheat Sheet Cheat Sheet

by Jacek Gebal (jgebal) via cheatography.com/22528/cs/4621/

Resources

https://github.com/rsim/ruby-plsql ruby-plsql project page
https://github.com/rsim/ruby-plsql-spec ruby-plsql-spec project page
http://blog.rayapps.com/tags/ruby-plsql/ Raimonds Simanovskis blog
http://www.oraclethoughts.com/tag/ruby-plsql/My blog

Connection parameters

```
plsql.connection.prefetch_rows = 100
  # sets number of rows to be fetched at once
plsql.connection.database_version
  # returns version of database as an Array => [11, 2, 0, 2]
plsql.dbms_output_stream = STDOUT
  # redirects dbms_output to standard output (console)
plsql.dbms_output_buffer_size = 100_000
  # sets dbms_output buffer size to 100,000
```

Execute SQL statement or PLSQL block

```
plsql.execute "CREATE SYNONYM employees_synonym FOR employees"
  # executes any given string as a SQL or PLSQL statement
plsql.execute <<-SQL
  CREATE TABLE test_employees (
    employee_id NUMBER(15),
    name VARCHAR2(50),
    hire_date DATE
  )
SQL
  #executes multi-line string statements too
```

Select from a table/view

```
plsql.employees.select(:first, "ORDER BY employee_id")
plsql.employees.first("ORDER BY employee_id")
  {:employee_id => 1, :name => 'James bond', :hire_date => '0007-07-07'}
  # returns first row from a table
plsql.employees.select(:first, "WHERE employee_id = :a", 2)
plsql.employees.first("WHERE employee_id = :a", 2)
plsql.employees.first(:employee_id => 2)
  {:employee_id => 2, :name => 'Tony Stark', :hire_date => nil}
  # returns first row from a table with WHERE condition
```

Select from a table/view (cont)

```
plsql.employees.select(:all, "ORDER BY employee_id")
plsql.employees.all("ORDER BY employee_id")
plsql.employees.all(:order_by => :employee_id)
  [ {:employee_id => 1, :name => 'James bond', :hire_date => '0007-07-07'}, {...}, ... ]
  # returns all rows from a table sorted using ORDER BY
plsql.employees.all(:employee_id => 2, :order_by => :employee_id)
  [ {:employee_id => 2, :name => 'Tony Stark', :hire_date => nil} ]
  # returns all rows from a table with WHERE condition
plsql.employees.all "WHERE employee_id = 2 AND hire_date IS NULL"
plsql.employees.all( {:employee_id => 2, :hire_date => nil} )
  [ {:employee_id => 2, :name => 'Tony Stark', :hire_date => nil} ]
  # returns all rows from a table with WHERE condition on NULL value
plsql.employees.all(:hire_date => :is_not_null)
  [ {:employee_id => 1, :name => 'James bond', :hire_date => '0007-07-07'}, {...}, ... ]
  # returns all rows from a table with WHERE condition on NOT NULL value
plsql.employees.select(:count)
plsql.employees.count
  10
  # returns count of rows in the table
```

Update table/view

```
plsql.employees.update :name => 'Test'
  # updates field name in all records
plsql.employees.update :name => 'Superman II', :where => {:employee_id => 7}
plsql.employees.update :name => 'Superman II', :where => "employee_id = 7"
  # updates field in table with a where condition
plsql.employees.update :name => 'Superman II', :hire_date => Time.local(2000,01,01), :where => "employee_id = 7"
  # updates two fields in table with a where condition
```



By **Jacek Gebal** (jgebal)
cheatography.com/jgebal/
www.oraclethoughts.com

Published 19th July, 2015.
Last updated 10th May, 2016.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Cheatography

ruby-plsql Cheat Sheet Cheat Sheet

by Jacek Gebal (jgebal) via cheatography.com/22528/cs/4621/

Sequence

```
plsql.execute "CREATE SEQUENCE employees_seq"
  #executes a statement to create a sequence
plsql.employees_seq.nextval
  1
  # returns NEXTVAL for sequence
plsql.employees_seq.currval
  1
  # returns CURRVAL for sequence
```

Package

```
plsql.test_package.class
  PLSQL::Package
  # A plsql package is Object of PLSQL::Package class
plsql.test_package.test_variable = 1
  # Assigns a value to package public variable
plsql.test_package.test_variable
  1
  # Reads a value to package public variable
```

Procedure / Function

```
# given a FUNCTION uppercase( p_string VARCHAR2 ) RETURN VARCHAR2

plsql.uppercase( 'xxx' )
plsql.uppercase( :p_string => 'xxx' )
  'XXX'
  # executes the function binding parameters by position or name and returns scalar Object as a value
# given a FUNCTION copy_function( p_from VARCHAR2, p_to OUT VARCHAR2, p_to_double OUT VARCHAR2 ) RETURN NUMBER

plsql.copy_function( 'abc', nil, nil)
plsql.copy_function( :p_from => 'abc', :p_to => nil, :p_to_double => nil)
plsql.copy_function( 'abc' )
  [ 3, { :p_to => "abc", :p_to_double => "abcabc" } ]
  # executes the function and returns 2 element Array
  # with first element being function result and second element being a Hash of OUT parameters
```

Procedure / Function (cont)

```
#Given a PROCEDURE copy_proc( p_from VARCHAR2, p_to OUT VARCHAR2, p_to_double OUT VARCHAR2 )

plsql.copy_proc( 'abc', nil, nil)
plsql.copy_proc( :p_from => 'abc', :p_to => nil, :p_to_double => nil)
plsql.copy_proc( 'abc' )
  { :p_to => 'abc', :p_to_double => 'abcabc' }
  # executes the procedure and returns a Hash of OUT parameters as a :name => 'value' pairs
```

Cursors

```
#Given a FUNCTION get_empolyees RETURN SYS_REFCURSOR

plsql.get_employees do |result|
  result.fields
end
  [ :employee_id, :name, :hire_date ]
  # returns the list of columns of a cursor as an Array
plsql.get_employees do |result|
  result.fetch_hash_all
end
plsql.get_employees{ |cursor| cursor.fetch_hash_all }
plsql.get_employees{ |any_name| any_name.fetch_hash_all }
  [ {:employee_id => 1, :name => 'James bond', :hire_date => '0007-07-07'}, {...}, ... ]
  # fetches all rows from a cursor and returns them as an Array of Hashes
plsql.get_employees{ |result| result.fetch_hash }
  {:employee_id => 1, :name => 'James bond', :hire_date => '0007-07-07'}
  # fetches one row from a cursor and returns it as a Hash
plsql.get_employees{ |result| result.fetch }
  [1, 'James bond', '0007-07-07']
  # fetches one row from a cursor and returns it as a Array of values
plsql.get_employees{ |result| result.fetch_all }
  [[1, 'James bond', '0007-07-07'], [...], ... ]
  # fetches all rows from a cursor and returns them as an Array of Arrays of values
# cursor needs to be accessed inside a block ( do .. end / { .. } )
# as cursors are automatically closed after the function call ends
```



By **Jacek Gebal** (jgebal)
cheatography.com/jgebal/
www.oraclethoughts.com

Published 19th July, 2015.
Last updated 10th May, 2016.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com