RDBI External API specification 1.0 draft

Erik Hollensbe <erik@hollensbe.org>

Table of Contents

1 All Classes

`Boolean reload` [Method on `All Classes`]

this method will semantically refresh items, such as Schema objects or rows, depending on the context of the object in question.

2 module DBI

`DBH connect` (*Class* `klass`, *Array* `*args`, *Proc* `&block`) [Method on `DBI`]

class is a ruby class which corresponds to the database driver. it is no longer a string.

*args is a hash with parameter `->` value associations, such as :host or :username.

Optionally yields a block for usage, yields a freshly connected DBH.

`Array of Class drivers` [Method on `DBI`]

accessor to get at known classes that can be used as drivers.

`DBH connect_cached` (*Class* `klass`, *Array* `*args`) [Method on `DBI`]

connect to a new resource if one is required (or desired, see below) with similar parameters as connect().

additional arguments :pool_name and :pool_size can be used to define a Pool (object, see below) which holds a specific subset of connected database handles. Playing with the size here introduces the ability for connect_cached to maintain a minimum number of connections which can be re-used over the lifetime of a program.

`Pool pool` (*String* `pool_name`) [Method on `DBI`]

a pool as described above is an array of database handles. this returns that data as a "Pool" object, with its own API. See later on in the document.

`Pool all_connections` [Method on `DBI`]

similar to pool(), this returns all the connections, but ignores pools.

`Integer ping` (*Class* `klass`, *Array* `*args`) [Method on `DBI`]

similar to connect(), this issues a ping to the databases. This may issue a connect() before the ping() to do it properly depending on the database implementation.

`Boolean reconnect_all` [Method on `DBI`]

reconnects all the known database handles.

`DBH last_dbh` [Method on `DBI`]

returns the last returned dbh from connect() or connect_cached()

this method, by definition, can be unpredictable in threaded environments.

3 class DBH

`NilClass transaction` (*Proc* `&block`) [Method on `DBH`]
: opens a transaction and executes the statements in the block. Yields self.

`Schema table_schema` (*Symbol* `table_name`) [Method on `DBH`]
: returns information about a specific table in a Schema object

`Array of Schema schema` (*Symbol* `schema_name`) [Method on `DBH`]
: returns information about a specific schema, the current one if none is specified.

`Boolean reconnect` [Method on `DBH`]
: reconnects to the database

`Integer ping` [Method on `DBH`]
: attempts to contact the database, measuring round-trip.

`Object driver` [Method on `DBH`]
: returns the underlying driver.

`String last_query` [Method on `DBH`]
: returns the last query executed or prepared.

`STH last_sth` [Method on `DBH`]
: returns the last statement handle prepared.

`Mutex mutex` [Method on `DBH`]
: returns the mutex for this database. thread management will be per-dbh.

`String preprocess_query` (*String* `query`) [Method on `DBH`]
: preprocesses the query and returns what it would look like right before it gets sent to the database.

`Boolean disconnect` [Method on `DBH`]
: disconnects from the database. returns success.

`Symbol bind_style` (*Symbol of [native, preprocessed]* `style`) [Method on `DBH`]
: Accessor. Native style delegates to the underlying database connector. preprocessed means we do it.

3.1 Query Methods

these methods all optionally use a block and yield a result or sth depending on context. Additionally in async environments, they return immediately, the block being transformed into a callback which will yield when the query completes.

`STH prepare` (*String* `query`) [Method on `DBH`]
: prepares a query for execution and returns a statement handle.

`Result execute` (*String* `query`, *Array* `*binds`) [Method on `DBH`]
: executes a query and returns a result. If a block is not provided, an async result will be provided which will slowly result in items being fetchable.

4 class STH

`String query` [Method on `STH`]
: accessor for the query that was used to generate this sth.

`Result execute` (*Array* ***binds***) [Method on `STH`]
: executes the prepared statement. optionally yielding a result if block given.

`Object driver` [Method on `STH`]
: if any, returns the underlying statement handle from the database object.

`Result last_result` [Method on `STH`]
: Returns the last Result this prepared statement has yielded.

`Boolean finish` [Method on `STH`]
: finishes the statement

`DBH dbh` [Method on `STH`]
: returns the dbh this statement handle was created from.

5 class Pool

`Boolean reconnect` [Method on `Pool`]

attempts to reconnect the entire pool of database connections.

`Integer ping` [Method on `Pool`]

attempts to ping and average the response time of all database connections.

`Boolean disconnect` [Method on `Pool`]

disconnects all the database connections in the pool.

6 class Result

`Boolean complete?` [Method on `Result`]

Always returns true in a sync environment. In an async environment, only returns true if all result processing has been completed.

`Boolean has_data?` [Method on `Result`]

Always returns true in a sync environment. In an async environment, only returns true if there is outstanding data to fetch.

`Boolean eof?` [Method on `Result`]

Returns true if all results have been fetched.

`NilClass rewind` [Method on `Result`]

resets the fetch iterator to the beginning. See also: #reload.

`Integer rows` [Method on `Result`]

If available, returns the number of rows in this result. Else, nil.

`Array binds` [Method on `Result`]

accessor for the binds that created this method

`NilClass as` (*Class* ***`kind`***, *Array* ***`*args`***) [Method on `Result`]

Given a Class and arguments, uses it to interpret the array. The class is constructed with the result object and the arguments provided at the end, and then a method called fetch() is attempted with the row count.

Especially for specific class designations, (XML formatting is a good example) output formats may not necessarily equate to a single row, in that case, one "unit" should be returned from #fetch, and this entailings of this unit should be specified in the driver.

If this this method is not called, fetch yields a standard array with type converted items.

`Object fetch` (*Integer* ***`row_count`***) [Method on `Result`]

fetches one item, or given an argument, *row_count* rows. If the row_count is ":all", fetches all outstanding rows. See #as for how rows may be interpreted.

`Array of Object raw_fetch` (*Integer* ***`row_count`***) [Method on `Result`]

Raw fetch performs no conversions – returns an array of objects yielding whatever the underlying driver gave us.

`Boolean finish` [Method on `Result`]

finishes the underlying statement handle and invalidates the data. reloading will no longer be possible once this is called and should raise (or maybe we should reprepare/execute?).

`STH sth` [Method on `Result`]

returns the statement handle that yielded this result.

`Schema schema` [Method on `Result`]

returns a Schema object that corresponds to the data in this result.

`NilClass each (`*`&block`*`)` [Method on `Result`]

similar to calling fetch iteratively with a callback. With proper async driver support, will register a callback from the block which will only process when there are new rows to be had.

7 class CursorResult < Result

This class is just a cursor-oriented method of transmitting results.

8 class Row

row is just an array, but this needs to be thought out a little more.

9 Schema

`Array of Column columns` [Method on `Schema`]

returns column information (see Column object below) for all elements of the Schema.

`Array of Symbol table_names` [Method on `Schema`]

returns table names (there may be more than one in the event of a query Schema) for all the objects a part of this Schema.

10 Column

`String name` [Method on `Column`]

`String type` [Method on `Column`]
this is the type the database yields

`Class ruby_type` [Method on `Column`]
Accessor. this is what ruby thinks this type should be, or you can set it directly which will be used at type conversion time.

`Integer precision` [Method on `Column`]
(alias: length) precision is the first number in a database type. it is aliased to the method 'length' because sometimes that's what precision actually is depending on the type.

`Integer scale` [Method on `Column`]
scale is the second number in a database type. this is often the right side of a decimal value or sometimes a factoring quotient.

`Boolean nullable?` [Method on `Column`]
can this column be null?

`String metadata` [Method on `Column`]
metadata is a bucket for things we don't understand; namely things like AUTOINCREMENT.

`String default` [Method on `Column`]
default is the column default – this is provided for informational aspects only and should not be used for anything sane.

Method Index