
pyDAL Documentation

Release 15.5.29

web2py-developers

October 07, 2015

1	Indices and tables	3
1.1	Subpackages	3
1.2	Submodules	30
1.3	pydal.base module	30
1.4	pydal.connection module	37
1.5	pydal.objects module	37
1.6	Module contents	45
	Python Module Index	47

Contents:

Indices and tables

- *genindex*
- *modindex*
- *search*

1.1 Subpackages

1.1.1 pydal.adapters package

Submodules

pydal.adapters.base module

class `pydal.adapters.base.AdapterMeta`

Bases: `type`

Metaclass to support manipulation of adapter classes.

At the moment is used to intercept *entity_quoting* argument passed to DAL.

class `pydal.adapters.base.BaseAdapter` (*db, uri, pool_size=0, folder=None, db_codec='UTF-8', credential_decoder=<function IDENTITY at 0x7f0c9d438f50>, driver_args={}, adapter_args={}, do_connect=True, after_connection=None*)

Bases: `pydal.connection.ConnectionPool`

ADD (*first, second*)

AGGREGATE (*first, what*)

ALLOW_NULL ()

AND (*first, second*)

AS (*first, second*)

BELONGS (*first, second*)

CASE (*query, t, f*)

CAST (*first, second*)

COALESCE (*first, second*)

COALESCE_ZERO (*first*)

COMMA (*first, second*)

CONCAT (**items*)

CONTAINS (*first, second, case_sensitive=True*)

COUNT (*first, distinct=None*)

DIV (*first, second*)

ENDSWITH (*first, second*)

EPOCH (*first*)

EQ (*first, second=None*)

EXTRACT (*first, what*)

FALSE = 'F'

FALSE_exp = '0'

GE (*first, second=None*)

GT (*first, second=None*)

ILIKE (*first, second*)
Case insensitive like operator

INVERT (*first*)

JOIN ()

LE (*first, second=None*)

LEFT_JOIN ()

LENGTH (*first*)

LIKE (*first, second*)
Case sensitive like operator

LOWER (*first*)

LT (*first, second=None*)

MOD (*first, second*)

MUL (*first, second*)

NE (*first, second=None*)

NOT (*first*)

NOT_NULL (*default, field_type*)

ON (*first, second*)

OR (*first, second*)

PRIMARY_KEY (*key*)

QUOTE_TEMPLATE = "%s"

RANDOM ()

RAW (*first*)

REGEXP (*first, second*)
 Regular expression operator

REPLACE (*first, tup*)

STARTSWITH (*first, second*)

SUB (*first, second*)

SUBSTRING (*field, parameters*)

TRUE = 'T'

TRUE_exp = '1'

T_SEP = ' '

UPPER (*first*)

adapt (*obj*)

alias (*table, alias*)
 Given a table object, makes a new table object with alias name.

build_parsemap ()

bulk_insert (*table, items*)

can_join ()

can_select_for_update = True

close_connection ()

commit ()

commit_on_alter_table = False

commit_prepared (*key*)

common_filter (*query, tablename*)

concat_add (*tablename*)

connection = None

connector (**args, **kwargs*)

constraint_name (*table, fieldname*)

count (*query, distinct=None*)

create_sequence_and_triggers (*query, table, **args*)

create_table (*table, migrate=True, fake_migrate=False, polymodel=None*)

dbpath = None

delete (*tablename, query*)

distributed_transaction_begin (*key*)

driver = None

driver_auto_json = []

driver_name = None

drivers = ()

drop (*table, mode=''*)

execute (*a, **b)

expand (expression, field_type=None, colnames=False)

expand_all (fields, tablenames)

file_close (fileobj)

file_delete (filename)

file_exists (filename)

file_open (filename, mode='rb', lock=True)

find_driver (adapter_args, uri=None)

folder = None

get_table (query)

id_query (table)

insert (table, fields)

isOperationalError (exception)

isProgrammingError (exception)

is_numerical_type (ftype)

iterparse (sql, fields, colnames, blob_decode=True, cacheable=False)
Iterator to parse one row at a time. It doesn't support the old style virtual fields

iterselect (query, fields, attributes)

lastrowid (table)

log (message, table=None)
Logs migrations
It will not log changes if logfile is not specified. Defaults to sql.log

log_execute (*a, **b)

migrate_table (table, sql_fields, sql_fields_old, sql_fields_aux, logfile, fake_migrate=False)

parse (rows, fields, colnames, blob_decode=True, cacheable=False)

parse_blob (value, field_type)

parse_boolean (value, field_type)

parse_date (value, field_type)

parse_datetime (value, field_type)

parse_decimal (value, field_type)

parse_double (value, field_type)

parse_id (value, field_type)

parse_integer (value, field_type)

parse_json (value, field_type)

parse_list_integers (value, field_type)

parse_list_references (value, field_type)

parse_list_strings (value, field_type)

```

parse_reference (value, field_type)
parse_time (value, field_type)
parse_value (value, field_type, blob_decode=True)
prepare (key)
represent (obj, fieldtype)
represent_exceptions (obj, fieldtype)
rollback ()
rollback_prepared (key)
rowslice (rows, minimum=0, maximum=None)
    By default this function does nothing; overload when db does not do slicing.
save_dbt (table, sql_fields_current)
select (query, fields, attributes)
    Always returns a Rows object, possibly empty.
select_limitby (sql_s, sql_f, sql_t, sql_w, sql_o, limitby)
sequence_name (tablename)
smart_adapt (obj)
sqlsafe_field (fieldname)
sqlsafe_table (tablename, ot=None)
support_distributed_transaction = False
table_alias (tbl)
tables (*queries)
test_query = 'SELECT 1;'
trigger_name (tablename)
truncate (table, mode=' ')
types = {'reference': 'INTEGER REFERENCES %(foreign_key)s ON DELETE %(on_delete_action)s', 'text': 'TEXT'}
update (tablename, query, fields)
uploads_in_blob = False
varquote (name)

```

class `pydal.adapters.base.NoSQLAdapter` (*db, uri, pool_size=0, folder=None, db_codec='UTF-8', credential_decoder=<function IDENTITY at 0x7f0c9d438f50>, driver_args={}, adapter_args={}, do_connect=True, after_connection=None*)

Bases: `pydal.adapters.base.BaseAdapter`

```

ADD (first, second)
AGGREGATE (first, what)
AND (first, second)
AS (first, second)
DIV (first, second)

```

ENDSWITH (*first, second=None*)

EXTRACT (*first, what*)

ILIKE (*first, second*)

LEFT_JOIN ()

LENGTH (*first*)

LOWER (*first*)

MUL (*first, second*)

ON (*first, second*)

OR (*first, second*)

PRIMARY_KEY (*key*)

QUOTE_TEMPLATE = '%s'

RANDOM ()

STARTSWITH (*first, second=None*)

SUB (*first, second*)

SUBSTRING (*field, parameters*)

UPPER (*first*)

can_join ()

can_select_for_update = False

close_connection ()
remember: no transactions on many NoSQL

commit ()
remember: no transactions on many NoSQL

commit_prepared (*key*)

concat_add (*table*)

constraint_name (*table, fieldname*)

create_sequence_and_triggers (*query, table, **args*)

distributed_transaction_begin (*key*)

drop (*table, mode*)

execute (**a, **b*)

id_query (*table*)

lastrowid (*table*)

log_execute (**a, **b*)

migrate_table (**a, **b*)

parse_list_integers (*value, field_type*)

parse_list_references (*value, field_type*)

parse_list_strings (*value, field_type*)

prepare (*key*)

represent (*obj, fieldtype*)
represent_exceptions (*obj, fieldtype*)
rollback ()
 remember: no transactions on many NoSQL
rollback_prepared (*key*)
rowslice (*rows, minimum=0, maximum=None*)

pydal.adapters.couchdb module

```
class pydal.adapters.couchdb.CouchDBAdapter (db, uri='couchdb://127.0.0.1:5984',
                                             pool_size=0, folder=None, db_codec='UTF-8',
                                             credential_decoder=<function IDENTITY at 0x7f0c9d438f50>, driver_args={},
                                             adapter_args={}, do_connect=True, after_connection=None)
```

Bases: `pydal.adapters.base.NoSQLAdapter`

AND (*first, second*)

COMMA (*first, second*)

EQ (*first, second*)

NE (*first, second*)

OR (*first, second*)

count (*query, distinct=None*)

create_table (*table, migrate=True, fake_migrate=False, polymodel=None*)

delete (*tablename, query*)

drivers = ('couchdb',)

expand (*expression, field_type=None*)

file_close (*fileobj*)

file_exists (*filename*)

file_open (*filename, mode='rb', lock=True*)

insert (*table, fields*)

represent (*obj, fieldtype*)

select (*query, fields, attributes*)

types = {'string': <type 'str'>, 'reference': <type 'long'>, 'text': <type 'str'>, 'id': <type 'long'>, 'float': <type 'float'>}

update (*tablename, query, fields*)

uploads_in_blob = True

pydal.adapters.cubrid module

```
class pydal.adapters.cubrid.CubridAdapter (db, uri, pool_size=0, folder=None,
                                           db_codec='UTF-8', credential_decoder=<function IDENTITY
                                           at 0x7f0c9d438f50>, driver_args={},
                                           adapter_args={}, do_connect=True, af-
                                           ter_connection=None)
Bases: pydal.adapters.mysql.MySQLAdapter
REGEX_URI = <_sre.SRE_Pattern object at 0x29c4a30>
after_connection ()
drivers = ('cubridb',)
```

pydal.adapters.db2 module

```
class pydal.adapters.db2.DB2Adapter (db, uri, pool_size=0, folder=None, db_codec='UTF-8', cre-
                                     dential_decoder=<function IDENTITY at 0x7f0c9d438f50>,
                                     driver_args={}, adapter_args={}, do_connect=True, af-
                                     ter_connection=None)
Bases: pydal.adapters.base.BaseAdapter
LEFT_JOIN ()
RANDOM ()
drivers = ('ibm_db_dbi', 'pyodbc')
execute (command, placeholders=None)
lastrowid (table)
represent_exceptions (obj, fieldtype)
rowslice (rows, minimum=0, maximum=None)
select_limitby (sql_s, sql_f, sql_t, sql_w, sql_o, limitby)
types = {'reference': 'INT, FOREIGN KEY (%(field_name)s) REFERENCES %(foreign_key)s ON DELETE %(on_del
```

pydal.adapters.firebird module

```
class pydal.adapters.firebird.FireBirdAdapter (db, uri, pool_size=0, folder=None,
                                                db_codec='UTF-8', creden-
                                                tial_decoder=<function IDENTITY
                                                at 0x7f0c9d438f50>, driver_args={},
                                                adapter_args={}, do_connect=True, af-
                                                ter_connection=None)
Bases: pydal.adapters.base.BaseAdapter
CONTAINS (first, second, case_sensitive=False)
EPOCH (first)
LENGTH (first)
NOT_NULL (default, field_type)
RANDOM ()
```

REGEX_URI = <_sre.SRE_Pattern object at 0x29dd5e0>

SUBSTRING (*field, parameters*)

commit_on_alter_table = False

create_sequence_and_triggers (*query, table, **args*)

drivers = ('kinterbasdb', 'firebirdsql', 'fdb', 'pyodbc')

lastrowid (*table*)

select_limitby (*sql_s, sql_f, sql_t, sql_w, sql_o, limitby*)

sequence_name (*tablename*)

support_distributed_transaction = True

trigger_name (*tablename*)

types = {'reference': 'INTEGER REFERENCES %(foreign_key)s ON DELETE %(on_delete_action)s', 'text': 'BLOB'}

```
class pydal.adapters.firebird.FireBirdEmbeddedAdapter (db, uri, pool_size=0,
                                                       folder=None,
                                                       db_codec='UTF-8', credential_decoder=<function IDENTITY at 0x7f0c9d438f50>,
                                                       driver_args={},
                                                       adapter_args={},
                                                       do_connect=True, after_connection=None)
```

Bases: `pydal.adapters.firebird.FireBirdAdapter`

REGEX_URI = <_sre.SRE_Pattern object at 0x29accd0>

drivers = ('kinterbasdb', 'firebirdsql', 'fdb', 'pyodbc')

pydal.adapters.google_adapters module

Adapter for GAE

pydal.adapters.imap module

```
class pydal.adapters.imap.IMAPAdapter (db, uri, pool_size=0, folder=None, db_codec='UTF-8',
                                       credential_decoder=<function IDENTITY at 0x7f0c9d438f50>,
                                       driver_args={}, adapter_args={},
                                       do_connect=True, after_connection=None)
```

Bases: `pydal.adapters.base.NoSQLAdapter`

IMAP server adapter

This class is intended as an interface with email IMAP servers to perform simple queries in the web2py DAL query syntax, so email read, search and other related IMAP mail services (as those implemented by brands like Google(r), and Yahoo!(r)) can be managed from web2py applications.

The code uses examples by Yuji Tomita on this post: <http://yuji.wordpress.com/2011/06/22/python-imaplib-imap-example-with-gmail/#comment-1137> and is based in docs for Python imaplib, python email and email IETF's (i.e. RFC2060 and RFC3501)

This adapter was tested with a small set of operations with Gmail(r). Other services requests could raise command syntax and response data issues.

It creates its table and field names “statically”, meaning that the developer should leave the table and field definitions to the DAL instance by calling the adapter’s `.define_tables()` method. The tables are defined with the IMAP server mailbox list information.

`.define_tables()` returns a dictionary mapping dal tablenames to the server mailbox names with the following structure:

```
{<tablename>: str <server mailbox name>}
```

Here is a list of supported fields:

Field	Type	Description
uid	string	
answered	boolean	Flag
created	date	
content	list:string	A list of dict text or html parts
to	string	
cc	string	
bcc	string	
size	integer	the amount of octets of the message*
deleted	boolean	Flag
draft	boolean	Flag
flagged	boolean	Flag
sender	string	
recent	boolean	Flag
seen	boolean	Flag
subject	string	
mime	string	The mime header declaration
email	string	The complete RFC822 message (*)
attachments	list	Each non text part as dict
encoding	string	The main detected encoding

(*) At the application side it is measured as the length of the RFC822 message string

WARNING: As row id’s are mapped to email sequence numbers, make sure your imap client web2py app does not delete messages during select or update actions, to prevent updating or deleting different messages. Sequence numbers change whenever the mailbox is updated. To avoid this sequence numbers issues, it is recommended the use of uid fields in query references (although the update and delete in separate actions rule still applies).

```
# This is the code recommended to start imap support  
# at the app's model:
```

```
imapdb = DAL("imap://user:password@server:port", pool_size=1) # port 993 for ssl  
imapdb.define_tables()
```

Here is an (incomplete) list of possible imap commands:

```
# Count today's unseen messages  
# smaller than 6000 octets from the  
# inbox mailbox  
  
q = imapdb.INBOX.seen == False  
q &= imapdb.INBOX.created == datetime.date.today()  
q &= imapdb.INBOX.size < 6000  
unread = imapdb(q).count()  
  
# Fetch last query messages  
rows = imapdb(q).select()
```



```

# it is also possible to filter query select results with limitby and
# sequences of mailbox fields

set.select(<fields sequence>, limitby=(<int>, <int>))

# Mark last query messages as seen
messages = [row.uid for row in rows]
seen = imapdb(imapdb.INBOX.uid.belongs(messages)).update(seen=True)

# Delete messages in the imap database that have mails from mr. Gumby

deleted = 0
for mailbox in imapdb.tables
    deleted += imapdb(imapdb[mailbox].sender.contains("gumby")).delete()

# It is possible also to mark messages for deletion instead of erasing them
# directly with set.update(deleted=True)

# This object give access
# to the adapter auto mailbox
# mapped names (which native
# mailbox has what table name)

imapdb.mailboxes <dict> # tablename, server native name pairs

# To retrieve a table native mailbox name use:
imapdb.<table>.mailbox

### New features v2.4.1:

# Declare mailboxes statically with tablename, name pairs
# This avoids the extra server names retrieval

imapdb.define_tables({"inbox": "INBOX"})

# Selects without content/attachments/email columns will only
# fetch header and flags

imapdb(q).select(imapdb.INBOX.sender, imapdb.INBOX.subject)

AND (first, second)
BELONGS (first, second)
CONTAINS (first, second, case_sensitive=False)
EQ (first, second)
GE (first, second)
GT (first, second)
LE (first, second)
LT (first, second)
NE (first, second=None)
NOT (first)
OR (first, second)

```

REGEX_URI = <_sre.SRE_Pattern object at 0x7f0c9d37fa80>

convert_date (*date*, *add=None*, *imf=False*)

count (*query*, *distinct=None*)

create_table (**args*, ***kwargs*)

dbengine = 'imap'

define_tables (*mailbox_names=None*)

Auto create common IMAP files

This function creates fields definitions “statically” meaning that custom fields as in other adapters should not be supported and definitions handled on a service/mode basis (local syntax for Gmail(r), Ymail(r))

Returns a dictionary with tablename, server native mailbox name pairs.

delete (*tablename*, *query*)

drivers = ('imaplib',)

encode_text (*text*, *charset*, *errors='replace'*)
convert text for mail to unicode

get_charset (*message*)

get_last_message (*tablename*)

get_mailboxes ()
Query the mail database for mailbox names

get_query_mailbox (*query*)

get_uid_bounds (*tablename*)

static header_represent (*f*, *r*)

insert (*table*, *fields*)

is_flag (*flag*)

reconnect (*f=None*, *cursor=True*)
IMAP4 Pool connection method

imap connection lacks of self cursor command. A custom command should be provided as a replacement for connection pooling to prevent uncaught remote session closing

select (*query*, *fields*, *attributes*)
Searches and Fetches records and return web2py rows

types = {'boolean': <type 'bool'>, 'string': <type 'str'>, 'list:string': <type 'str'>, 'integer': <type 'int'>, 'date': <type 'date'>}

update (*tablename*, *query*, *fields*)

uri = None
MESSAGE is an identifier for sequence number

pydal.adapters.informix module

```
class pydal.adapters.informix.InformixAdapter(db, uri, pool_size=0, folder=None,
                                              db_codec='UTF-8', credential_decoder=<function IDENTITY
                                              at 0x7f0c9d438f50>, driver_args={}, adapter_args={}, do_connect=True, after_connection=None)
```

Bases: `pydal.adapters.base.BaseAdapter`

NOT_NULL (*default, field_type*)

RANDOM ()

REGEX_URI = <_sre.SRE_Pattern object at 0x7f0c9cc13030>

drivers = ('informixdb',)

execute (*command*)

lastrowid (*table*)

represent_exceptions (*obj, fieldtype*)

select_limitby (*sql_s, sql_f, sql_t, sql_w, sql_o, limitby*)

types = {'reference': 'INTEGER REFERENCES %(foreign_key)s ON DELETE %(on_delete_action)s', 'text': 'BLOB'}

```
class pydal.adapters.informix.InformixSEAdapter(db, uri, pool_size=0, folder=None,
                                              db_codec='UTF-8', credential_decoder=<function IDENTITY
                                              at 0x7f0c9d438f50>, driver_args={}, adapter_args={}, do_connect=True, after_connection=None)
```

Bases: `pydal.adapters.informix.InformixAdapter`

work in progress

rowslice (*rows, minimum=0, maximum=None*)

select_limitby (*sql_s, sql_f, sql_t, sql_w, sql_o, limitby*)

pydal.adapters.ingres module

```
class pydal.adapters.ingres.IngresAdapter(db, uri, pool_size=0, folder=None,
                                              db_codec='UTF-8', credential_decoder=<function IDENTITY
                                              at 0x7f0c9d438f50>, driver_args={}, adapter_args={}, do_connect=True, after_connection=None)
```

Bases: `pydal.adapters.base.BaseAdapter`

LEFT_JOIN ()

RANDOM ()

create_sequence_and_triggers (*query, table, **args*)

drivers = ('pyodbc',)

lastrowid (*table*)

select_limitby (*sql_s, sql_f, sql_t, sql_w, sql_o, limitby*)

```
types = {'reference': 'INT, FOREIGN KEY (%(field_name)s) REFERENCES %(foreign_key)s ON DELETE %(on_del

class pydal.adapters.ingres.IngresUnicodeAdapter(db, uri, pool_size=0, folder=None,
                                                db_codec='UTF-8', credential_decoder=<function IDENTITY
                                                at 0x7f0c9d438f50>, driver_args={}, adapter_args={}, do_connect=True,
                                                after_connection=None)

Bases: pydal.adapters.ingres.IngresAdapter

drivers = ('pyodbc',)

types = {'reference': 'INTEGER4, FOREIGN KEY (%(field_name)s) REFERENCES %(foreign_key)s ON DELETE %
```

pydal.adapters.mongo module

```
class pydal.adapters.mongo.Binary
    Bases: object

class pydal.adapters.mongo.MongoBlob
    Bases: pydal.adapters.mongo.Binary

    MONGO_BLOB_BYTES = 0

    MONGO_BLOB_NON_UTF8_STR = 1

    static decode (value)

class pydal.adapters.mongo.MongoDBAdapter(db, uri='mongodb://127.0.0.1:5984/db',
                                           pool_size=0, folder=None, db_codec='UTF-
                                           8', credential_decoder=<function IDEN-
                                           TITY at 0x7f0c9d438f50>, driver_args={},
                                           adapter_args={}, do_connect=True, af-
                                           ter_connection=None)

Bases: pydal.adapters.base.NoSQLAdapter

ADD (first, second)

AND (first, second)

AS (first, second)

BELONGS (first, second)

COMMA (first, second)

CONTAINS (first, second, case_sensitive=True)

DIV (first, second)

ENDSWITH (first, second)

EQ (first, second=None)

GE (first, second=None)

GT (first, second)

ILIKE (first, second)

INVERT (first)

LE (first, second=None)

LIKE (first, second, case_sensitive=True)
```

LT (*first, second=None*)
MOD (*first, second*)
MUL (*first, second*)
NE (*first, second=None*)
NOT (*first*)
ON (*first, second*)
OR (*first, second*)
STARTSWITH (*first, second*)
SUB (*first, second*)
bulk_insert (*table, items*)
count (*query, distinct=None, snapshot=True*)
create_table (*table, migrate=True, fake_migrate=False, polymodel=None, isCapped=False*)
delete (*tablename, query, safe=None*)
driver_auto_json = ['loads', 'dumps']
drivers = ('pymongo',)
drop (*table, mode=''*)
error_messages = {'javascript_needed': 'This must yet be replaced with javascript in order to work.}'
expand (*expression, field_type=None*)
insert (*table, fields, safe=None*)
 Safe determines whether a asynchronous request is done or a synchronous action is done For safety, we use by default synchronous requests
object_id (*arg=None*)
 Convert input to a valid MongoDB ObjectId instance
 self.object_id("<random>") -> ObjectId (not unique) instance
parse_blob (*value, field_type*)
parse_id (*value, field_type*)
parse_reference (*value, field_type*)
represent (*obj, fieldtype*)
select (*query, fields, attributes, snapshot=False*)
truncate (*table, mode, safe=None*)
types = {'string': <type 'str'>, 'reference': <type 'long'>, 'text': <type 'str'>, 'id': <type 'long'>, 'float': <type 'float'>},
update (*tablename, query, fields, safe=None*)
uploads_in_blob = False

pydal.adapters.mssql module

```
class pydal.adapters.mssql.MSSQL2Adapter (db, uri, pool_size=0, folder=None, db_codec='UTF-8',
                                          credential_decoder=<function IDENTITY at 0x7f0c9d438f50>, driver_args={},
                                          adapter_args={}, do_connect=True, srid=4326,
                                          after_connection=None)
```

Bases: `pydal.adapters.mssql.MSSQLAdapter`

ILIKE (*first, second*)

drivers = ('pyodbc')

execute (*a, **b)

represent (obj, fieldtype)

types = {'reference': 'INT, CONSTRAINT %(constraint_name)s FOREIGN KEY (%(field_name)s) REFERENCES %'

```
class pydal.adapters.mssql.MSSQL3Adapter (db, uri, pool_size=0, folder=None, db_codec='UTF-8',
                                          credential_decoder=<function IDENTITY at 0x7f0c9d438f50>, driver_args={},
                                          adapter_args={}, do_connect=True, srid=4326,
                                          after_connection=None)
```

Bases: `pydal.adapters.mssql.MSSQLAdapter`

Experimental support for pagination in MSSQL

Requires MSSQL >= 2005, uses `ROW_NUMBER()`

rowslice (*rows, minimum=0, maximum=None*)

select_limitby (*sql_s, sql_f, sql_t, sql_w, sql_o, limitby*)

types = {'reference': 'INT NULL, CONSTRAINT %(constraint_name)s FOREIGN KEY (%(field_name)s) REFEREN

```
class pydal.adapters.mssql.MSSQL3NAdapter (db, uri, pool_size=0, folder=None,
                                           db_codec='UTF-8',
                                           credential_decoder=<function IDENTITY
                                           at 0x7f0c9d438f50>, driver_args={},
                                           adapter_args={}, do_connect=True, srid=4326,
                                           after_connection=None)
```

Bases: `pydal.adapters.mssql.MSSQLNAdapter`

drivers = ('pyodbc')

Experimental support for pagination in MSSQL Experimental: see MSSQLNAdapter docstring for warnings

Requires MSSQL >= 2005, uses `ROW_NUMBER()`

rowslice (*rows, minimum=0, maximum=None*)

select_limitby (*sql_s, sql_f, sql_t, sql_w, sql_o, limitby*)

types = {'reference': 'INT NULL, CONSTRAINT %(constraint_name)s FOREIGN KEY (%(field_name)s) REFEREN

```
class pydal.adapters.mssql.MSSQL4Adapter (db, uri, pool_size=0, folder=None, db_codec='UTF-8',
                                          credential_decoder=<function IDENTITY at 0x7f0c9d438f50>,
                                          driver_args={},
                                          adapter_args={}, do_connect=True, srid=4326,
                                          after_connection=None)
```

Bases: `pydal.adapters.mssql.MSSQLAdapter`

Support for “native” pagination

Requires MSSQL >= 2012, uses *OFFSET ... ROWS ... FETCH NEXT ... ROWS ONLY*

rowslice (*rows, minimum=0, maximum=None*)

select_limitby (*sql_s, sql_f, sql_t, sql_w, sql_o, limitby*)

types = {'reference': 'INT NULL, CONSTRAINT %(constraint_name)s FOREIGN KEY (%(field_name)s) REFERENC

```
class pydal.adapters.mssql.MSSQL4NAdapter (db, uri, pool_size=0, folder=None,
                                           db_codec='UTF-8', credential_decoder=<function
                                           IDENTITY
                                           at 0x7f0c9d438f50>, driver_args={},
                                           adapter_args={}, do_connect=True, srid=4326,
                                           after_connection=None)
```

Bases: `pydal.adapters.mssql.MSSQLNAdapter`

Experimental: see MSSQLNAdapter docstring for warnings Support for “native” pagination

Unicode-compatible version Requires MSSQL >= 2012, uses *OFFSET ... ROWS ... FETCH NEXT ... ROWS ONLY* After careful testing, this should be the de-facto adapter for recent MSSQL backends

rowslice (*rows, minimum=0, maximum=None*)

select_limitby (*sql_s, sql_f, sql_t, sql_w, sql_o, limitby*)

types = {'reference': 'INT NULL, CONSTRAINT %(constraint_name)s FOREIGN KEY (%(field_name)s) REFERENC

```
class pydal.adapters.mssql.MSSQLAdapter (db, uri, pool_size=0, folder=None, db_codec='UTF-
8', credential_decoder=<function IDEN-
TITY at 0x7f0c9d438f50>, driver_args={},
adapter_args={}, do_connect=True, srid=4326,
after_connection=None)
```

Bases: `pydal.adapters.base.BaseAdapter`

AGGREGATE (*first, what*)

ALLOW_NULL ()

CAST (*first, second*)

CONCAT (**items*)

EPOCH (*first*)

EXTRACT (*field, what*)

FALSE = 0

LEFT_JOIN ()

LENGTH (*first*)

PRIMARY_KEY (*key*)

QUOTE_TEMPLATE = “”%s””

RANDOM ()

REGEX_ARGPATTERN = <_sre.SRE_Pattern object at 0x7f0c9d146440>

REGEX_DSN = <_sre.SRE_Pattern object at 0x7f0c9cc15d50>

REGEX_URI = <_sre.SRE_Pattern object at 0x29ce2d0>

ST_ASTEXT (*first*)

ST_CONTAINS (*first, second*)

ST_DISTANCE (*first, second*)

ST_EQUALS (*first, second*)

ST_INTERSECTS (*first, second*)

ST_OVERLAPS (*first, second*)

ST_TOUCHES (*first, second*)

ST_WITHIN (*first, second*)

SUBSTRING (*field, parameters*)

TRUE = 1

T_SEP = 'T'

concat_add (*tablename*)

drivers = ('pyodbc',)

lastrowid (*table*)

represent (*obj, fieldtype*)

rowslice (*rows, minimum=0, maximum=None*)

select_limitby (*sql_s, sql_f, sql_t, sql_w, sql_o, limitby*)

types = {'reference': 'INT NULL, CONSTRAINT %(constraint_name)s FOREIGN KEY (%(field_name)s) REFERENC

varquote (*name*)

```
class pydal.adapters.mssql.MSSQLAdapter (db, uri, pool_size=0, folder=None, db_codec='UTF-8', credential_decoder=<function IDENTITY at 0x7f0c9d438f50>, driver_args={}, adapter_args={}, do_connect=True, srid=4326, after_connection=None)
```

Bases: `pydal.adapters.mssql.MSSQLAdapter`

ILIKE (*first, second*)

drivers = ('pyodbc',)

Experimental – base class for handling unicode in MSSQL by default. Needs lots of testing. Try this on a fresh (or on a legacy) database. Using this in a database handled previously with non-unicode aware adapter is NOT supported

execute (**a, **b*)

represent (*obj, fieldtype*)

types = {'reference': 'INT, CONSTRAINT %(constraint_name)s FOREIGN KEY (%(field_name)s) REFERENCES %'

```
class pydal.adapters.mssql.SybaseAdapter (db, uri, pool_size=0, folder=None, db_codec='UTF-8', credential_decoder=<function IDENTITY at 0x7f0c9d438f50>, driver_args={}, adapter_args={}, do_connect=True, srid=4326, after_connection=None)
```

Bases: `pydal.adapters.mssql.MSSQLAdapter`

drivers = 'Sybase'

types = {'reference': 'INT NULL, CONSTRAINT %(constraint_name)s FOREIGN KEY (%(field_name)s) REFERENC


```

class pydal.adapters.mssql.VerticaAdapter (db, uri, pool_size=0, folder=None,
                                           db_codec='UTF-8', credential_decoder=<function
                                           IDENTITY at 0x7f0c9d438f50>, driver_args={},
                                           adapter_args={}, do_connect=True, srid=4326,
                                           after_connection=None)

Bases: pydal.adapters.mssql.MSSQLAdapter

EXTRACT (first, what)

T_SEP = ''

drivers = ('pyodbc',)

execute (a)

lastrowid (table)

select_limitby (sql_s, sql_f, sql_t, sql_w, sql_o, limitby)

types = {'big-reference': 'BIGINT REFERENCES %(foreign_key)s ON DELETE %(on_delete_action)s', 'string': 'VA

```

pydal.adapters.mysql module

```

class pydal.adapters.mysql.MySQLAdapter (db, uri, pool_size=0, folder=None, db_codec='UTF-
8', credential_decoder=<function IDENTITY at
0x7f0c9d438f50>, driver_args={}, adapter_args={},
do_connect=True, after_connection=None)

Bases: pydal.adapters.base.BaseAdapter

CAST (first, second)

CONCAT (*items)

EPOCH (first)

QUOTE_TEMPLATE = "%s"

RANDOM ()

REGEXP (first, second)

REGEX_URI = <_sre.SRE_Pattern object at 0x29c4a30>

SUBSTRING (field, parameters)

after_connection ()

commit_on_alter_table = True

commit_prepared (key)

distributed_transaction_begin (key)

drivers = ('MySQLdb', 'pymysql', 'mysqlconnector')

prepare (key)

rollback_prepared (key)

support_distributed_transaction = True

types = {'reference': 'INT, INDEX %(index_name)s (%(field_name)s), FOREIGN KEY (%(field_name)s) REFERENC

varquote (name)

```

pydal.adapters.oracle module

```
class pydal.adapters.oracle.OracleAdapter (db, uri, pool_size=0, folder=None,
                                           db_codec='UTF-8', credential_decoder=<function
                                           IDENTITY at 0x7f0c9d438f50>, driver_args={},
                                           adapter_args={}, do_connect=True, af-
                                           ter_connection=None)
```

Bases: `pydal.adapters.base.BaseAdapter`

LEFT_JOIN ()

NOT_NULL (default, field_type)

RANDOM ()

REGEXP (first, second)

after_connection ()

commit_on_alter_table = False

constraint_name (tablename, fieldname)

create_sequence_and_triggers (query, table, **args)

drivers = ('cx_Oracle',)

execute (command, args=None)

lastrowid (table)

oracle_fix = <_sre.SRE_Pattern object at 0x7f0c9d44cc70>

represent_exceptions (obj, fieldtype)

select_limitby (sql_s, sql_f, sql_t, sql_w, sql_o, limitby)

sqlsafe_table (tablename, ot=None)

trigger_name (tablename)

types = {'reference': 'NUMBER, CONSTRAINT %(constraint_name)s FOREIGN KEY (%(field_name)s) REFERENC

pydal.adapters.postgres module

```
class pydal.adapters.postgres.JDBCPostgreSQLAdapter (db, uri, pool_size=0, folder=None,
                                                       db_codec='UTF-8', credential_decoder=<function
                                                       IDENTITY at 0x7f0c9d438f50>,
                                                       driver_args={}, adapter_args={},
                                                       do_connect=True, af-
                                                       ter_connection=None)
```

Bases: `pydal.adapters.postgres.PostgreSQLAdapter`

REGEX_URI = <_sre.SRE_Pattern object at 0x7f0c9cc13030>

after_connection ()

drivers = ('zxJDBC',)

```
class pydal.adapters.postgres.NewPostgreSQLAdapter (db, uri, pool_size=0, folder=None,
                                                    db_codec='UTF-8', credential_decoder=<function IDENTITY
                                                    at 0x7f0c9d438f50>, driver_args={},
                                                    adapter_args={}, do_connect=True,
                                                    srid=4326, after_connection=None)
```

Bases: `pydal.adapters.postgres.PostgreSQLAdapter`

ANY (*first*)

CONTAINS (*first, second, case_sensitive=True*)

EQ (*first, second=None*)

ILIKE (*first, second*)

drivers = ('psycpg2', 'pg8000')

parse_list_integers (*value, field_type*)

parse_list_references (*value, field_type*)

parse_list_strings (*value, field_type*)

represent (*obj, fieldtype*)

types = {'reference': 'INTEGER REFERENCES %(foreign_key)s ON DELETE %(on_delete_action)s', 'text': 'TEXT'}

```
class pydal.adapters.postgres.PostgreSQLAdapter (db, uri, pool_size=0, folder=None,
                                                  db_codec='UTF-8', credential_decoder=<function IDENTITY
                                                  at 0x7f0c9d438f50>, driver_args={},
                                                  adapter_args={}, do_connect=True,
                                                  srid=4326, after_connection=None)
```

Bases: `pydal.adapters.base.BaseAdapter`

ADD (*first, second*)

ILIKE (*first, second*)

LIKE (*first, second*)

QUOTE_TEMPLATE = ""%s""

RANDOM ()

REGEXP (*first, second*)

REGEX_URI = <_sre.SRE_Pattern object at 0x2976000>

ST_ASJSON (*first, second*)
http://postgis.org/docs/ST_AsGeoJSON.html

ST_ASTEXT (*first*)
http://postgis.org/docs/ST_AsText.html

ST_CONTAINS (*first, second*)
http://postgis.org/docs/ST_Contains.html

ST_DISTANCE (*first, second*)
http://postgis.org/docs/ST_Distance.html

ST_DWITHIN (*first, tup*)
http://postgis.org/docs/ST_DWithin.html

ST_EQUALS (*first, second*)
http://postgis.org/docs/ST_Equals.html

ST_INTERSECTS (*first, second*)
http://postgis.org/docs/ST_Intersects.html

ST_OVERLAPS (*first, second*)
http://postgis.org/docs/ST_Overlaps.html

ST_SIMPLIFY (*first, second*)
http://postgis.org/docs/ST_Simplify.html

ST_SIMPLIFYPRESERVE TOPOLOGY (*first, second*)
http://postgis.org/docs/ST_SimplifyPreserveTopology.html

ST_TOUCHES (*first, second*)
http://postgis.org/docs/ST_Touches.html

ST_WITHIN (*first, second*)
http://postgis.org/docs/ST_Within.html

ST_X (*first*)
http://postgis.org/docs/ST_X.html

ST_Y (*first*)
http://postgis.org/docs/ST_Y.html

adapt (*obj*)

after_connection ()

commit_prepared (*key*)

create_sequence_and_triggers (*query, table, **args*)

distributed_transaction_begin (*key*)

drivers = ('psycopg2', 'pg8000')

lastrowid (*table=None*)

prepare (*key*)

represent (*obj, fieldtype*)

rollback_prepared (*key*)

sequence_name (*table*)

support_distributed_transaction = True

try_json ()

types = {'reference': 'INTEGER REFERENCES %(foreign_key)s ON DELETE %(on_delete_action)s', 'text': 'TEXT'}

varquote (*name*)

pydal.adapters.sapdb module

```
class pydal.adapters.sapdb.SAPDBAdapter (db, uri, pool_size=0, folder=None, db_codec='UTF-8', credential_decoder=<function IDENTITY at 0x7f0c9d438f50>, driver_args={}, adapter_args={}, do_connect=True, after_connection=None)
Bases: pydal.adapters.base.BaseAdapter
```

```

REGEX_URI = <_sre.SRE_Pattern object at 0x2976000>
create_sequence_and_triggers (query, table, **args)
drivers = ('sapdb',)
lastrowid (table)
select_limitby (sql_s, sql_f, sql_t, sql_w, sql_o, limitby)
sequence_name (table)
support_distributed_transaction = False
types = {'reference': 'INT, FOREIGN KEY (%(field_name)s) REFERENCES %(foreign_key)s ON DELETE %(on_del

```

pydal.adapters.sqlite module

```

class pydal.adapters.sqlite.JDBCSQLiteAdapter (db, uri, pool_size=0, folder=None,
                                               db_codec='UTF-8', credential
                                               decoder=<function IDENTITY
                                               at 0x7f0c9d438f50>, driver_args={},
                                               adapter_args={}, do_connect=True, af
                                               ter_connection=None)

```

Bases: `pydal.adapters.sqlite.SQLiteAdapter`

```

after_connection ()
drivers = ('zxJDBC_sqlite',)
execute (a)

```

```

class pydal.adapters.sqlite.SQLiteAdapter (db, uri, pool_size=0, folder=None,
                                              db_codec='UTF-8', credential
                                              decoder=<function IDENTITY
                                              at 0x7f0c9d438f50>, driver_args={},
                                              adapter_args={}, do_connect=True, af
                                              ter_connection=None)

```

Bases: `pydal.adapters.base.BaseAdapter`

```

EXTRACT (field, what)
REGEXP (first, second)
after_connection ()
can_select_for_update = None
delete (tablename, query)
drivers = ('sqlite2', 'sqlite3')
select (query, fields, attributes)
    Simulate SELECT ... FOR UPDATE with BEGIN IMMEDIATE TRANSACTION. Note that the entire
    database, rather than one record, is locked (it will be locked eventually anyway by the following UPDATE).
static web2py_extract (lookup, s)
static web2py_regexp (expression, item)

```

```
class pydal.adapters.sqlite.SpatialLiteAdapter (db, uri, pool_size=0, folder=None,
                                                db_codec='UTF-8', credential_decoder=<function IDENTITY
                                                at 0x7f0c9d438f50>, driver_args={},
                                                adapter_args={}, do_connect=True,
                                                srid=4326, after_connection=None)

Bases: pydal.adapters.sqlite.SQLiteAdapter

ST_ASJSON (first, second)
ST_ASTEXT (first)
ST_CONTAINS (first, second)
ST_DISTANCE (first, second)
ST_EQUALS (first, second)
ST_INTERSECTS (first, second)
ST_OVERLAPS (first, second)
ST_SIMPLIFY (first, second)
ST_TOUCHES (first, second)
ST_WITHIN (first, second)
after_connection ()
drivers = ('sqlite3', 'sqlite2')
represent (obj, fieldtype)
types = {'string': 'CHAR(%(length)s)', 'reference': 'INTEGER REFERENCES %(foreign_key)s ON DELETE %(on_c
```

pydal.adapters.teradata module

```
class pydal.adapters.teradata.TeradataAdapter (db, uri, pool_size=0, folder=None,
                                                db_codec='UTF-8', credential_decoder=<function IDENTITY
                                                at 0x7f0c9d438f50>, driver_args={},
                                                adapter_args={}, do_connect=True, af-
                                                ter_connection=None)

Bases: pydal.adapters.base.BaseAdapter

LEFT_JOIN ()
close (action='commit', really=True)
drivers = ('pyodbc',)
select_limitby (sql_s, sql_f, sql_t, sql_w, sql_o, limitby)
types = {'reference': 'INT', 'text': 'VARCHAR(2000)', 'float': 'REAL', 'datetime': 'TIMESTAMP', 'bigint': 'BIGINT'
```

Module contents

```
pydal.adapters.GoogleDatastoreAdapter = None
make the import available for BaseAdapter
```

1.1.2 pydal.helpers package

Submodules

pydal.helpers.classes module

```

class pydal.helpers.classes.BasicStorage (*args, **kwargs)
    Bases: object
        clear (*args, **kwargs)
        copy (*args, **kwargs)
        get (key, default=None)
        has_key (item)
        items ()
        iteritems ()
        iterkeys ()
        itervalues ()
        keys ()
        pop (*args, **kwargs)
        update (*args, **kwargs)
        values ()

class pydal.helpers.classes.DatabaseStoredFile (db, filename, mode)

    close ()
    close_connection ()
    escape (obj)
    static exists (db, filename)
    read (bytes)
    readline ()
    static try_create_web2py_filesystem (db)
    web2py_filesystems = set([])
    write (data)

class pydal.helpers.classes.MethodAdder (table)
    Bases: object
        register (method_name=None)

class pydal.helpers.classes.RecordDeleter (table, id)
    Bases: object

class pydal.helpers.classes.RecordUpdater (colset, table, id)
    Bases: object

class pydal.helpers.classes.Reference
    Bases: long

```

get (*key*, *default=None*)

`pydal.helpers.classes.Reference_pickler` (*data*)

`pydal.helpers.classes.Reference_unpickler` (*data*)

class `pydal.helpers.classes.SQLALL` (*table*)

Bases: object

Helper class providing a comma-separated string having all the field names (prefixed by table name and ‘.’)
normally only called from within gluon.dal

class `pydal.helpers.classes.SQLCallableList`

Bases: list

class `pydal.helpers.classes.SQLCustomType` (*type='string'*, *native=None*, *encoder=None*, *decoder=None*, *validator=None*, *_class=None*, *widget=None*, *represent=None*)

Bases: object

Allows defining of custom SQL types

Parameters

- **type** – the web2py type (default = ‘string’)
- **native** – the backend type
- **encoder** – how to encode the value to store it in the backend
- **decoder** – how to decode the value retrieved from the backend
- **validator** – what validators to use (default = None, will use the default validator for type)

Example:: Define as:

```
decimal = SQLCustomType( type = 'double', native = 'integer', encoder =(lambda x:
    int(float(x) * 100)), decoder = (lambda x: Decimal(“0.00”) + Decimal(str(float(x)/100))
    )
```

```
db.define_table( ‘example’, Field(‘value’, type=decimal) )
```

endswith (*text=None*)

startswith (*text=None*)

class `pydal.helpers.classes.Serializable`

Bases: object

as_dict (*flat=False*, *sanitize=True*)

as_json (*sanitize=True*)

as_xml (*sanitize=True*)

as_yaml (*sanitize=True*)

class `pydal.helpers.classes.UseDatabaseStoredFile`

file_close (*fileobj*)

file_delete (*filename*)

file_exists (*filename*)

file_open (*filename*, *mode='rb'*, *lock=True*)

`pydal.helpers.classes.pickle_basicstorage` (*s*)

pydal.helpers.methods module

`pydal.helpers.methods.archive_record` (*qset, fs, archive_table, current_record*)

`pydal.helpers.methods.auto_represent` (*field*)

`pydal.helpers.methods.auto_validators` (*field*)

`pydal.helpers.methods.bar_decode_integer` (*value*)

`pydal.helpers.methods.bar_decode_string` (*value*)

`pydal.helpers.methods.bar_encode` (*items*)

`pydal.helpers.methods.bar_escape` (*item*)

`pydal.helpers.methods.cleanup` (*text*)

Validates that the given text is clean: only contains **[0-9a-zA-Z_]**

`pydal.helpers.methods.geoLine` (**line*)

`pydal.helpers.methods.geoPoint` (*x, y*)

`pydal.helpers.methods.geoPolygon` (**line*)

`pydal.helpers.methods.hide_password` (*uri*)

`pydal.helpers.methods.int2uuid` (*n*)

`pydal.helpers.methods.list_represent` (*values, row=None*)

`pydal.helpers.methods.pluralize` (*singular, rules=[(<_sre.SRE_Pattern object at 0x7f0ca0613e00>, <_sre.SRE_Pattern object at 0x7f0ca0613e00>, 'children'), (<_sre.SRE_Pattern object at 0x7f0c9d41b450>, <_sre.SRE_Pattern object at 0x7f0c9d41b450>, 'eet'), (<_sre.SRE_Pattern object at 0x7f0ca05da570>, <_sre.SRE_Pattern object at 0x7f0ca05da570>, 'eeth'), (<_sre.SRE_Pattern object at 0x7f0ca05da630>, <_sre.SRE_Pattern object at 0x7f0ca0613ed0>, '\Naves'), (<_sre.SRE_Pattern object at 0x7f0c9d41b500>, <_sre.SRE_Pattern object at 0x7f0c9d41b500>, 'ses'), (<_sre.SRE_Pattern object at 0x7f0c9d41b5b0>, <_sre.SRE_Pattern object at 0x7f0c9d41b5b0>, 'men'), (<_sre.SRE_Pattern object at 0x7f0c9d41b660>, <_sre.SRE_Pattern object at 0x7f0c9d41b660>, 'ives'), (<_sre.SRE_Pattern object at 0x7f0c9d41b710>, <_sre.SRE_Pattern object at 0x7f0c9d41b710>, 'eaux'), (<_sre.SRE_Pattern object at 0x7f0c9d3be030>, <_sre.SRE_Pattern object at 0x7f0c9d3be030>, 'lves'), (<_sre.SRE_Pattern object at 0x7f0c9d3bf030>, <_sre.SRE_Pattern object at 0x7f0c9db10238>, 'es'), (<_sre.SRE_Pattern object at 0x7f0c9d3b42d0>, <_sre.SRE_Pattern object at 0x7f0c9db10238>, 'es'), (<_sre.SRE_Pattern object at 0x7f0ca05cc8e8>, <_sre.SRE_Pattern object at 0x7f0c9d3a5150>, 'ies'), (<_sre.SRE_Pattern object at 0x7f0c9db10238>, <_sre.SRE_Pattern object at 0x7f0c9db10238>, 's')]]*)

```
pydal.helpers.methods.smart_query (fields, text)
pydal.helpers.methods.use_common_filters (query)
pydal.helpers.methods.uuid2int (uuidv)
pydal.helpers.methods.varquote_aux (name, quotestr='%s')
pydal.helpers.methods.xorify (orderby)
```

pydal.helpers.regex module

Module contents

1.2 Submodules

1.3 pydal.base module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (<http://www.gnu.org/licenses/lgpl.html>)

This file contains the DAL support for many relational databases, including:

- SQLite & Spatialite
- MySQL
- Postgres
- Firebird
- Oracle
- MS SQL
- DB2
- Interbase
- Ingres
- Informix (9+ and SE)
- SapDB (experimental)
- Cubrid (experimental)
- CouchDB (experimental)
- MongoDB (in progress)
- Google:nosql
- Google:sql
- Teradata
- IMAP (experimental)

Example of usage:

```

>>> # from dal import DAL, Field

### create DAL connection (and create DB if it doesn't exist)
>>> db = DAL('sqlite://storage.sqlite','mysql://a:b@localhost/x'),
... folder=None)

### define a table 'person' (create/alter as necessary)
>>> person = db.define_table('person',Field('name','string'))

### insert a record
>>> id = person.insert(name='James')

### retrieve it by id
>>> james = person(id)

### retrieve it by name
>>> james = person(name='James')

### retrieve it by arbitrary query
>>> query = (person.name=='James') & (person.name.startswith('J'))
>>> james = db(query).select(person.ALL)[0]

### update one record
>>> james.update_record(name='Jim')
<Row {'id': 1, 'name': 'Jim'}>

### update multiple records by query
>>> db(person.name.like('J%')).update(name='James')
1

### delete records by query
>>> db(person.name.lower() == 'jim').delete()
0

### retrieve multiple records (rows)
>>> people = db(person).select(orderby=person.name,
... groupby=person.name, limitby=(0,100))

### further filter them
>>> james = people.find(lambda row: row.name == 'James').first()
>>> print james.id, james.name
1 James

### check aggregates
>>> counter = person.id.count()
>>> print db(person).select(counter).first()(counter)
1

### delete one record
>>> james.delete_record()
1

### delete (drop) entire database table
>>> person.drop()

```

Supported DAL URI strings:

```
'sqlite://test.db'  
'spatialite://test.db'  
'sqlite:memory'  
'spatialite:memory'  
'jdbc:sqlite://test.db'  
'mysql://root:none@localhost/test'  
'postgres://mdipierro:password@localhost/test'  
'postgres:psycopg2://mdipierro:password@localhost/test'  
'postgres:pg8000://mdipierro:password@localhost/test'  
'jdbc:postgres://mdipierro:none@localhost/test'  
'mssql://web2py:none@A64X2/web2py_test'  
'mssql2://web2py:none@A64X2/web2py_test' # alternate mappings  
'mssql3://web2py:none@A64X2/web2py_test' # better pagination (requires >= 2005)  
'mssql4://web2py:none@A64X2/web2py_test' # best pagination (requires >= 2012)  
'oracle://username:password@database'  
'firebird://user:password@server:3050/database'  
'db2:ibm_db_dbi://DSN=dsn;UID=user;PWD=pass'  
'db2:pyodbc://driver=DB2;hostname=host;database=database;uid=user;pwd=password;port=port.'  
'firebird://username:password@hostname/database'  
'firebird_embedded://username:password@c://path'  
'informix://user:password@server:3050/database'  
'informixu://user:password@server:3050/database' # unicode informix  
'ingres://database' # or use an ODBC connection string, e.g. 'ingres://dsn=dsn_name'  
'google:datastore' # for google app engine datastore (uses ndb by default)  
'google:sql' # for google app engine with sql (mysql compatible)  
'teradata://DSN=dsn;UID=user;PWD=pass; DATABASE=database' # experimental  
'imap://user:password@server:port' # experimental  
'mongodb://user:password@server:port/database' # experimental
```

For more info:

```
help(DAL)  
help(Field)
```

```
class pydal.base.DAL (uri='sqlite://dummy.db', pool_size=0, folder=None, db_codec='UTF-  
8', check_reserved=None, migrate=True, fake_migrate=False, mi-  
grate_enabled=True, fake_migrate_all=False, decode_credentials=False,  
driver_args=None, adapter_args=None, attempts=5, auto_import=False, big-  
int_id=False, debug=False, lazy_tables=False, db_uid=None, do_connect=True,  
after_connection=None, tables=None, ignore_field_case=True, en-  
tity_quoting=False, table_hash=None)
```

Bases: `pydal.helpers.classes.Serializable`, `pydal.helpers.classes.BasicStorage`

An instance of this class represents a database connection

Parameters

- **uri** (*str*) – contains information for connecting to a database. Defaults to `'sqlite://dummy.db'`

Note: experimental: you can specify a dictionary as uri parameter i.e. with:

```
db = DAL({"uri": "sqlite://storage.sqlite",  
         "tables": {...}, ...})
```

for an example of dict input you can check the output of the scaffolding db model with

```
db.as_dict()
```

Note that for compatibility with Python older than version 2.6.5 you should cast your dict input keys to str due to a syntax limitation on kwargs names. for proper DAL dictionary input

you can use one of:

```
obj = serializers.cast_keys(dict, [encoding="utf-8"])
#or else (for parsing json input)
obj = serializers.loads_json(data, unicode_keys=False)
```

-
- **pool_size** – How many open connections to make to the database object.
 - **folder** – where .table files will be created. Automatically set within web2py. Use an explicit path when using DAL outside web2py
 - **db_codec** – string encoding of the database (default: ‘UTF-8’)
 - **table_hash** – database identifier with .tables. If your connection hash change you can still using old .tables if they have db_hash as prefix
 - **check_reserved** – list of adapters to check tablename and column names against sql/nosql reserved keywords. Defaults to *None*
 - ‘common’ List of sql keywords that are common to all database types such as “SELECT, INSERT”. (recommended)
 - ‘all’ Checks against all known SQL keywords
 - ‘<adaptername>’ Checks against the specific adapters list of keywords
 - ‘<adaptername>_nonreserved’ Checks against the specific adapters list of nonreserved keywords. (if available)
 - **migrate** – sets default migrate behavior for all tables
 - **fake_migrate** – sets default fake_migrate behavior for all tables
 - **migrate_enabled** – If set to False disables ALL migrations
 - **fake_migrate_all** – If set to True fake migrates ALL tables
 - **attempts** – Number of times to attempt connecting
 - **auto_import** – If set to True, tries import automatically table definitions from the databases folder (works only for simple models)
 - **bigint_id** – If set, turn on bigint instead of int for id and reference fields
 - **lazy_tables** – delay a table definition until table access
 - **after_connection** – can a callable that will be executed after the connection

Example

Use as:

```
db = DAL('sqlite://test.db')
```

or:

```
db = DAL(**{"uri": ..., "tables": [...]...}) # experimental
```

```
db.define_table('tablename', Field('fieldname1'),
                Field('fieldname2'))
```

class Table (*db, tablename, *fields, **args*)

Bases: `pydal.helpers.classes.Serializable`, `pydal.helpers.classes.BasicStorage`

Represents a database table

Example::

You can create a table as:: `db = DAL(...)` `db.define_table('users', Field('name'))`

And then:

```
db.users.insert(name='me') # print db.users._insert(...) to see SQL
db.users.drop()
```

as_dict (*flat=False, sanitize=True*)

bulk_insert (*items*)

here items is a list of dictionaries

drop (*mode=''*)

fields

import_from_csv_file (*csvfile, id_map=None, null='<NULL>', unique='uuid', id_offset=None, *args, **kwargs*)

Import records from csv file. Column headers must have same names as table fields. Field 'id' is ignored. If column names read 'table.file' the 'table.' prefix is ignored.

- 'unique' argument is a field which must be unique (typically a uuid field)
- 'restore' argument is default False; if set True will remove old values in table first.
- 'id_map' if set to None will not map ids

The import will keep the id numbers in the restored table. This assumes that there is an field of type id that is integer and in incrementing order. Will keep the id numbers in restored table.

insert (***fields*)

on (*query*)

sqlsafe

sqlsafe_alias

truncate (*mode=None*)

update (**args, **kwargs*)

update_or_insert (*_key=<function <lambda> at 0x7f0c9d438ed8>, **values*)

validate_and_insert (***fields*)

validate_and_update (*_key=<function <lambda> at 0x7f0c9d438ed8>, **fields*)

validate_and_update_or_insert (*_key=<function <lambda> at 0x7f0c9d438ed8>, **fields*)

with_alias (*alias*)

`DAL.as_dict` (*flat=False, sanitize=True*)

`DAL.can_join` ()

`DAL.check_reserved_keyword` (*name*)

Validates *name* against SQL keywords Uses `self.check_reserve` which is a list of operators to use.

`DAL.close` ()

`DAL.commit` ()

`DAL.define_table` (*tablename*, **fields*, ***args*)

`static DAL.distributed_transaction_begin` (**instances*)

`static DAL.distributed_transaction_commit` (**instances*)

`DAL.executesql` (*query*, *placeholders=None*, *as_dict=False*, *fields=None*, *colnames=None*, *as_ordered_dict=False*)

Executes an arbitrary query

Parameters

- **query** (*str*) – the query to submit to the backend
- **placeholders** – is optional and will always be None. If using raw SQL with placeholders, placeholders may be a sequence of values to be substituted in or, (if supported by the DB driver), a dictionary with keys matching named placeholders in your SQL.
- **as_dict** – will always be None when using DAL. If using raw SQL can be set to True and the results cursor returned by the DB driver will be converted to a sequence of dictionaries keyed with the db field names. Results returned with `as_dict=True` are the same as those returned when applying `.to_list()` to a DAL query. If `“as_ordered_dict”=True` the behaviour is the same as when `“as_dict”=True` with the keys (field names) guaranteed to be in the same order as returned by the select name executed on the database.
- **fields** – list of DAL Fields that match the fields returned from the DB. The Field objects should be part of one or more Table objects defined on the DAL object. The “fields” list can include one or more DAL Table objects in addition to or instead of including Field objects, or it can be just a single table (not in a list). In that case, the Field objects will be extracted from the table(s).

Note: if either *fields* or *colnames* is provided, the results will be converted to a DAL *Rows* object using the *db_adapter.parse()* method

- **colnames** – list of field names in *tablename.fieldname* format

Note: It is also possible to specify both “fields” and the associated “colnames”. In that case, “fields” can also include DAL Expression objects in addition to Field objects. For Field objects in “fields”, the associated “colnames” must still be in *tablename.fieldname* format. For Expression objects in “fields”, the associated “colnames” can be any arbitrary labels.

DAL Table objects referred to by “fields” or “colnames” can be dummy tables and do not have to represent any real tables in the database. Also, note that the “fields” and “colnames” must be in the same order as the fields in the results cursor returned from the DB.

`DAL.export_to_csv_file` (*ofile*, **args*, ***kwargs*)

`static DAL.get_instances` ()

Returns a dictionary with uri as key with timings and defined tables:

```
{'sqlite://storage.sqlite': {
  'dbstats': [(select auth_user.email from auth_user, 0.02009)],
  'databases': {
    'defined': ['auth_cas', 'auth_event', 'auth_group',
               'auth_membership', 'auth_permission', 'auth_user'],
    'lazy': '[]'
  }
}
```

```
DAL.has_representer (name)
DAL.import_from_csv_file (ifile, id_map=None, null='<NULL>', unique='uuid',
                           map_tablenames=None, ignore_missing_tables=False, *args,
                           **kwargs)
DAL.import_table_definitions (path, migrate=False, fake_migrate=False, tables=None)
DAL.lazy_define_table (tablename, *fields, **args)
DAL.logger = <logging.Logger object at 0x7f0c9d36fe10>
DAL.parse_as_rest (patterns, args, vars, queries=None, nested_select=True)
```

Example

Use as:

```
db.define_table('person', Field('name'), Field('info'))
db.define_table('pet',
               Field('ownedby', db.person),
               Field('name'), Field('info')
               )

@request.restful()
def index():
    def GET(*args, **vars):
        patterns = [
            "/friends[person]",
           ("/{person.name}/:field",
           ("/{person.name}/pets[pet.ownedby]",
           ("/{person.name}/pets[pet.ownedby}/{pet.name}",
           ("/{person.name}/pets[pet.ownedby}/{pet.name}/:field",
            ("/dogs[pet]", db.pet.info=='dog'),
            ("/dogs[pet]/{pet.name.startswith}", db.pet.info=='dog'),
        ]
        parser = db.parse_as_rest(patterns, args, vars)
        if parser.status == 200:
            return dict(content=parser.response)
        else:
            raise HTTP(parser.status, parser.error)

    def POST(table_name, **vars):
        if table_name == 'person':
            return db.person.validate_and_insert(**vars)
        elif table_name == 'pet':
            return db.pet.validate_and_insert(**vars)
        else:
            raise HTTP(400)
    return locals()

DAL.represent (name, *args, **kwargs)
DAL.representers = {}
DAL.rollback ()
DAL.serializers = None
static DAL.set_folder (folder)
```



```

DAL.smart_query (fields, text)
DAL.tables
DAL.uuid (x)
DAL.validators = None
DAL.validators_method = None
pydal.base.DAL_pickler (db)
pydal.base.DAL_unpickler (db_uid)
class pydal.base.MetaDAL
    Bases: type

```

1.4 pydal.connection module

```

class pydal.connection.ConnectionPool
    Bases: object

    POOLS = {}

    after_connection ()

    after_connection_hook ()
        Hook for the after_connection parameter

    check_active_connection = True

    close (action='commit', really=True)

    static close_all_instances (action)
        to close cleanly databases in a multithreaded environment

    find_or_make_work_folder ()

    reconnect (f=None, cursor=True)
        Defines: self.connection and self.cursor (if cursor is True) if self.pool_size>0 it will try pull the connection
        from the pool if the connection is not active (closed by db server) it will loop if not self.pool_size or no
        active connections in pool makes a new one

    static set_folder (folder)

```

1.5 pydal.objects module

```

class pydal.objects.BasicRows
    Bases: object

    Abstract class for Rows and IterRows

    __str__ ()
        Serializes the table into a csv file

    as_csv ()
        Serializes the table into a csv file

    as_dict (key='id', compact=True, storage_to_dict=True, datetime_to_str=False, cus-
            tom_types=None)
        Returns the data as a dictionary of dictionaries (storage_to_dict=True) or records (False)

```

Parameters

- **key** – the name of the field to be used as dict key, normally the id
- **compact** – ? (default True)
- **storage_to_dict** – when True returns a dict, otherwise a list (default True)
- **datetime_to_str** – convert datetime fields as strings (default False)

as_json (*mode='object', default=None*)

Serializes the rows to a JSON list or object with objects mode='object' is not implemented (should return a nested object structure)

as_list (*compact=True, storage_to_dict=True, datetime_to_str=False, custom_types=None*)

Returns the data as a list or dictionary.

Parameters

- **storage_to_dict** – when True returns a dict, otherwise a list
- **datetime_to_str** – convert datetime fields as strings

as_trees (*parent_name='parent_id', children_name='children', render=False*)

returns the data as list of trees.

Parameters

- **parent_name** – the name of the field holding the reference to the parent (default parent_id).
- **children_name** – the name where the children of each row will be stored as a list (default children).
- **render** – whether we will render the fields using their represent (default False) can be a list of fields to render or True to render all.

as_xml (*row_name='row', rows_name='rows'*)

export_to_csv_file (*ofile, null='<NULL>', *args, **kwargs*)

Exports data to csv, the first line contains the column names

Parameters

- **ofile** – where the csv must be exported to
- **null** – how null values must be represented (default '<NULL>')
- **delimiter** – delimiter to separate values (default ';')
- **quotechar** – character to use to quote string values (default '"')
- **quoting** – quote system, use csv.QUOTE_*** (default csv.QUOTE_MINIMAL)
- **represent** – use the fields .represent value (default False)
- **colnames** – list of column names to use (default self.colnames)

This will only work when exporting rows objects!!!! DO NOT use this with db.export_to_csv()

json (*mode='object', default=None*)

Serializes the rows to a JSON list or object with objects mode='object' is not implemented (should return a nested object structure)

xml (*strict=False, row_name='row', rows_name='rows'*)

Serializes the table using sqlhtml.SQLTABLE (if present)

```

class pydal.objects.Expression (db, op, first=None, second=None, type=None, **optional_args)
    Bases: object

    abs ()

    avg ()

    belongs (*value, **kwattr)
        Accepts the following inputs:

        field.belongs (1,2)
        field.belongs ((1,2))
        field.belongs (query)

        Does NOT accept:

            field.belongs(1)

        If the set you want back includes None values, you can do:

            field.belongs ((1,None), null=True)

    coalesce (*others)

    coalesce_zero ()

    contains (value, all=False, case_sensitive=False)
        For GAE contains() is always case sensitive

    day ()

    endswith (value)

    epoch ()

    hour ()

    ilike (value)

    len ()

    like (value, case_sensitive=True)

    lower ()

    max ()

    min ()

    minutes ()

    month ()

    regexp (value)

    replace (a, b)

    seconds ()

    st_asgeojson (precision=15, options=0, version=1)

    st_astext ()

    st_contains (value)

    st_distance (other)

    st_dwithin (value, distance)

```

`st_equals` (*value*)
`st_intersects` (*value*)
`st_overlaps` (*value*)
`st_simplify` (*value*)
`st_simplifypreservetopology` (*value*)
`st_touches` (*value*)
`st_within` (*value*)
`st_x` ()
`st_y` ()
`startswith` (*value*)
`sum` ()
`upper` ()
`with_alias` (*alias*)
`year` ()

`class` `pydal.objects.Field` (*fieldname*, *type*='string', *length*=None, *default*=<function <lambda> at 0x7f0c9d438ed8>, *required*=False, *requires*=<function <lambda> at 0x7f0c9d438ed8>, *ondelete*='CASCADE', *notnull*=False, *unique*=False, *uploadfield*=True, *widget*=None, *label*=None, *comment*=None, *writable*=True, *readable*=True, *update*=None, *authorize*=None, *autodelete*=False, *represent*=None, *uploadfolder*=None, *uploadseparate*=False, *uploadfs*=None, *compute*=None, *custom_store*=None, *custom_retrieve*=None, *custom_retrieve_file_properties*=None, *custom_delete*=None, *filter_in*=None, *filter_out*=None, *custom_qualifier*=None, *map_none*=None, *rname*=None)

Bases: `pydal.objects.Expression`, `pydal.helpers.classes.Serializable`

Lazy

Represents a database field

Example

Usage:

```
a = Field(name, 'string', length=32, default=None, required=False,
         requires=IS_NOT_EMPTY(), ondelete='CASCADE',
         notnull=False, unique=False,
         uploadfield=True, widget=None, label=None, comment=None,
         uploadfield=True, # True means store on disk,
                           # 'a_field_name' means store in this field in db
                           # False means file content will be discarded.
         writable=True, readable=True, update=None, authorize=None,
         autodelete=False, represent=None, uploadfolder=None,
         uploadseparate=False # upload to separate directories by uuid_keys
                               # first 2 character and tablename.fieldname
                               # False - old behavior
                               # True - put uploaded file in
                               # <uploaddir>/<tablename>.<fieldname>/uuid_key[:2]
                               # directory)
```

```

        uploadfs=None          # a pyfilesystem where to store upload
    )

    to be used as argument of DAL.define_table

    alias of FieldMethod

Method
    alias of FieldMethod

Virtual
    alias of FieldVirtual

as_dict (flat=False, sanitize=True)

clone (point_self_references_to=False, **args)

count (distinct=None)

formatter (value)

retrieve (name, path=None, nameonly=False)
    If nameonly==True return (filename, fullfilename) instead of (filename, stream)

retrieve_file_properties (name, path=None)

set_attributes (*args, **attributes)

sqlsafe

sqlsafe_name

store (file, filename=None, path=None)

validate (value)

class pydal.objects.FieldMethod (name, f=None, handler=None)
    Bases: object

class pydal.objects.FieldVirtual (name, f=None, ftype='string', label=None, table_name=None)
    Bases: object

class pydal.objects.IterRows (db, sql, fields, colnames, blob_decode, cacheable)
    Bases: pydal.objects.BasicRows

    first ()

    next ()

class pydal.objects.LazyReferenceGetter (table, id)
    Bases: object

class pydal.objects.LazySet (field, id)
    Bases: object

    count (distinct=None, cache=None)

    delete ()

    delete_uploaded_files (upload_fields=None)

    isempty ()

    nested_select (*fields, **attributes)

    select (*fields, **attributes)

    update (**update_fields)

```

update_naive (**update_fields)

validate_and_update (**update_fields)

class pydal.objects.**Query** (db, op, first=None, second=None, ignore_common_filters=False, **optional_args)

Bases: pydal.helpers.classes.Serializable

Necessary to define a set. It can be stored or can be passed to `DAL.__call__()` to obtain a *Set*

Example

Use as:

```
query = db.users.name=='Max'  
set = db(query)  
records = set.select()
```

as_dict (flat=False, sanitize=True)

Experimental stuff

This allows to return a plain dictionary with the basic query representation. Can be used with json/xml services for client-side db I/O

Example

Usage:

```
q = db.auth_user.id != 0  
q.as_dict(flat=True)  
{  
  "op": "NE",  
  "first": {  
    "tablename": "auth_user",  
    "fieldname": "id"  
  },  
  "second": 0  
}
```

case (t=1, f=0)

class pydal.objects.**Row** (*args, **kwargs)

Bases: pydal.helpers.classes.BasicStorage

A dictionary that lets you do `d['a']` as well as `d.a` this is only used to store a *Row*

as_dict (datetime_to_str=False, custom_types=None)

as_json (mode='object', default=None, colnames=None, serialize=True, **kwargs)

serializes the row to a JSON object kwargs are passed to `.as_dict` method only “object” mode supported

`serialize = False` used by Rows.as_json

TODO: return array mode with query column order

mode and colnames are not implemented

as_xml (row_name='row', colnames=None, indent=' ')

get (key, default=None)

class `pydal.objects.Rows` (*db=None, records=[], colnames=[], compact=True, rawrows=None*)
 Bases: `pydal.objects.BasicRows`

A wrapper for the return value of a select. It basically represents a table. It has an iterator and each row is represented as a *Row* dictionary.

__iter__ ()

Iterator over records

column (*column=None*)

exclude (*f*)

Removes elements from the calling Rows object, filtered by the function *f*, and returns a new Rows object containing the removed elements

find (*f, limitby=None*)

Returns a new Rows object, a subset of the original object, filtered by the function *f*

first ()

group_by_value (**fields, **args*)

Regroups the rows, by one of the fields

last ()

render (*i=None, fields=None*)

Takes an index and returns a copy of the indexed row with values transformed via the “represent” attributes of the associated fields.

Parameters

- **i** – index. If not specified, a generator is returned for iteration over all the rows.
- **fields** – a list of fields to transform (if None, all fields with “represent” attributes will be transformed)

setvirtualfields (***keyed_virtualfields*)

For reference:

```
db.define_table('x', Field('number', 'integer'))
if db(db.x).isempty(): [db.x.insert(number=i) for i in range(10)]

from gluon.dal import lazy_virtualfield

class MyVirtualFields(object):
    # normal virtual field (backward compatible, discouraged)
    def normal_shift(self): return self.x.number+1
    # lazy virtual field (because of @staticmethod)
    @lazy_virtualfield
    def lazy_shift(instance, row, delta=4): return row.x.number+delta
db.x.virtualfields.append(MyVirtualFields())

for row in db(db.x).select():
    print row.number, row.normal_shift, row.lazy_shift(delta=7)
```

sort (*f, reverse=False*)

Returns a list of sorted elements (not sorted in place)

class `pydal.objects.Set` (*db, query, ignore_common_filters=None*)

Bases: `pydal.helpers.classes.Serializable`

Represents a set of records in the database. Records are identified by the *query=Query(...)* object. Normally the Set is generated by *DAL.__call__(Query(...))*

Given a set, for example:

```
myset = db(db.users.name=='Max')
```

you can:

```
myset.update(db.users.name='Massimo')
myset.delete() # all elements in the set
myset.select(orderby=db.users.id, groupby=db.users.name, limitby=(0,10))
```

and take subsets:

```
subset = myset(db.users.id<5)
```

as_dict (*flat=False, sanitize=True*)

build (*d*)

Experimental: see `.parse()`

count (*distinct=None, cache=None*)

delete ()

delete_uploaded_files (*upload_fields=None*)

isempty ()

iterselect (**fields, **attributes*)

nested_select (**fields, **attributes*)

parse (*dquery*)

Experimental: Turn a dictionary into a Query object

select (**fields, **attributes*)

update (***update_fields*)

update_naive (***update_fields*)

Same as update but does not call `table._before_update` and `_after_update`

validate_and_update (***update_fields*)

class `pydal.objects.Table` (*db, tablename, *fields, **args*)

Bases: `pydal.helpers.classes.Serializable`, `pydal.helpers.classes.BasicStorage`

Represents a database table

Example::

You can create a table as:: `db = DAL(...) db.define_table('users', Field('name'))`

And then:

```
db.users.insert(name='me') # print db.users._insert(...) to see SQL
db.users.drop()
```

as_dict (*flat=False, sanitize=True*)

bulk_insert (*items*)

here items is a list of dictionaries

drop (*mode=''*)

fields

import_from_csv_file (*csvfile*, *id_map=None*, *null='<NULL>'*, *unique='uuid'*, *id_offset=None*,
args*, *kwargs*)

Import records from csv file. Column headers must have same names as table fields. Field 'id' is ignored. If column names read 'table.file' the 'table.' prefix is ignored.

- 'unique' argument is a field which must be unique (typically a uuid field)
- 'restore' argument is default False; if set True will remove old values in table first.
- 'id_map' if set to None will not map ids

The import will keep the id numbers in the restored table. This assumes that there is an field of type id that is integer and in incrementing order. Will keep the id numbers in restored table.

insert (***fields*)

on (*query*)

sqlsafe

sqlsafe_alias

truncate (*mode=None*)

update (**args*, ***kwargs*)

update_or_insert (*_key=<function <lambda> at 0x7f0c9d438ed8>*, ***values*)

validate_and_insert (***fields*)

validate_and_update (*_key=<function <lambda> at 0x7f0c9d438ed8>*, ***fields*)

validate_and_update_or_insert (*_key=<function <lambda> at 0x7f0c9d438ed8>*, ***fields*)

with_alias (*alias*)

class `pydal.objects.VirtualCommand` (*method*, *row*)

Bases: `object`

`pydal.objects.pickle_row` (*s*)

1.6 Module contents

p

- pydal, 45
- pydal.adapters, 26
- pydal.adapters.base, 3
- pydal.adapters.couchdb, 9
- pydal.adapters.cubrid, 10
- pydal.adapters.db2, 10
- pydal.adapters.firebird, 10
- pydal.adapters.imap, 11
- pydal.adapters.informix, 15
- pydal.adapters.ingres, 15
- pydal.adapters.mongo, 16
- pydal.adapters.mssql, 18
- pydal.adapters.mysql, 21
- pydal.adapters.oracle, 22
- pydal.adapters.postgres, 22
- pydal.adapters.sapdb, 24
- pydal.adapters.sqlite, 25
- pydal.adapters.teradata, 26
- pydal.base, 30
- pydal.connection, 37
- pydal.helpers, 30
- pydal.helpers.classes, 27
- pydal.helpers.methods, 29
- pydal.helpers.regex, 30
- pydal.objects, 37

Symbols

`__iter__()` (pydal.objects.Rows method), 43
`__str__()` (pydal.objects.BasicRows method), 37

A

`abs()` (pydal.objects.Expression method), 39
`adapt()` (pydal.adapters.base.BaseAdapter method), 5
`adapt()` (pydal.adapters.postgres.PostgreSQLAdapter method), 24
AdapterMeta (class in pydal.adapters.base), 3
`ADD()` (pydal.adapters.base.BaseAdapter method), 3
`ADD()` (pydal.adapters.base.NoSQLAdapter method), 7
`ADD()` (pydal.adapters.mongo.MongoDBAdapter method), 16
`ADD()` (pydal.adapters.postgres.PostgreSQLAdapter method), 23
`after_connection()` (pydal.adapters.cubrid.CubridAdapter method), 10
`after_connection()` (pydal.adapters.mysql.MySQLAdapter method), 21
`after_connection()` (pydal.adapters.oracle.OracleAdapter method), 22
`after_connection()` (pydal.adapters.postgres.JDBCPostgreSQLAdapter method), 22
`after_connection()` (pydal.adapters.postgres.PostgreSQLAdapter method), 24
`after_connection()` (pydal.adapters.sqlite.JDBCSQLiteAdapter method), 25
`after_connection()` (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
`after_connection()` (pydal.adapters.sqlite.SQLiteAdapter method), 25
`after_connection()` (pydal.connection.ConnectionPool method), 37
`after_connection_hook()` (pydal.connection.ConnectionPool method), 37
`AGGREGATE()` (pydal.adapters.base.BaseAdapter method), 3
`AGGREGATE()` (pydal.adapters.base.NoSQLAdapter method), 7
`AGGREGATE()` (pydal.adapters.mssql.MSSQLAdapter method), 19
`alias()` (pydal.adapters.base.BaseAdapter method), 5
`ALLOW_NULL()` (pydal.adapters.base.BaseAdapter method), 3
`ALLOW_NULL()` (pydal.adapters.mssql.MSSQLAdapter method), 9
`AND()` (pydal.adapters.base.BaseAdapter method), 3
`AND()` (pydal.adapters.base.NoSQLAdapter method), 7
`AND()` (pydal.adapters.couchdb.CouchDBAdapter method), 9
`AND()` (pydal.adapters.imap.IMAPAdapter method), 13
`AND()` (pydal.adapters.mongo.MongoDBAdapter method), 16
`ANY()` (pydal.adapters.postgres.NewPostgreSQLAdapter method), 23
`archive_record()` (in module pydal.helpers.methods), 29
`AS()` (pydal.adapters.base.BaseAdapter method), 3
`AS()` (pydal.adapters.base.NoSQLAdapter method), 7
`AS()` (pydal.adapters.mongo.MongoDBAdapter method), 16
`as_csv()` (pydal.objects.BasicRows method), 37
`as_dict()` (pydal.base.DAL method), 34
`as_dict()` (pydal.base.DAL.Table method), 34
`as_dict()` (pydal.helpers.classes.Serializable method), 28
`as_dict()` (pydal.objects.BasicRows method), 37
`as_dict()` (pydal.objects.Field method), 41
`as_dict()` (pydal.objects.Query method), 42
`as_dict()` (pydal.objects.Row method), 42
`as_dict()` (pydal.objects.Set method), 44
`as_dict()` (pydal.objects.Table method), 44
`as_json()` (pydal.helpers.classes.Serializable method), 28
`as_json()` (pydal.objects.BasicRows method), 38
`as_json()` (pydal.objects.Row method), 42
`as_list()` (pydal.objects.BasicRows method), 38
`as_trees()` (pydal.objects.BasicRows method), 38
`as_xml()` (pydal.helpers.classes.Serializable method), 28
`as_xml()` (pydal.objects.BasicRows method), 38
`as_xml()` (pydal.objects.Row method), 42

as_yaml() (pydal.helpers.classes.Serializable method), 28
 auto_represent() (in module pydal.helpers.methods), 29
 auto_validators() (in module pydal.helpers.methods), 29
 avg() (pydal.objects.Expression method), 39

B

bar_decode_integer() (in module pydal.helpers.methods), 29
 bar_decode_string() (in module pydal.helpers.methods), 29
 bar_encode() (in module pydal.helpers.methods), 29
 bar_escape() (in module pydal.helpers.methods), 29
 BaseAdapter (class in pydal.adapters.base), 3
 BasicRows (class in pydal.objects), 37
 BasicStorage (class in pydal.helpers.classes), 27
 BELONGS() (pydal.adapters.base.BaseAdapter method), 3
 BELONGS() (pydal.adapters.imap.IMAPAdapter method), 13
 BELONGS() (pydal.adapters.mongo.MongoDBAdapter method), 16
 belongs() (pydal.objects.Expression method), 39
 Binary (class in pydal.adapters.mongo), 16
 build() (pydal.objects.Set method), 44
 build_parsemap() (pydal.adapters.base.BaseAdapter method), 5
 bulk_insert() (pydal.adapters.base.BaseAdapter method), 5
 bulk_insert() (pydal.adapters.mongo.MongoDBAdapter method), 17
 bulk_insert() (pydal.base.DAL.Table method), 34
 bulk_insert() (pydal.objects.Table method), 44

C

can_join() (pydal.adapters.base.BaseAdapter method), 5
 can_join() (pydal.adapters.base.NoSQLAdapter method), 8
 can_join() (pydal.base.DAL method), 34
 can_select_for_update (pydal.adapters.base.BaseAdapter attribute), 5
 can_select_for_update (pydal.adapters.base.NoSQLAdapter attribute), 8
 can_select_for_update (pydal.adapters.sqlite.SQLiteAdapter attribute), 25
 CASE() (pydal.adapters.base.BaseAdapter method), 3
 case() (pydal.objects.Query method), 42
 CAST() (pydal.adapters.base.BaseAdapter method), 3
 CAST() (pydal.adapters.mssql.MSSQLAdapter method), 19
 CAST() (pydal.adapters.mysql.MySQLAdapter method), 21

check_active_connection (pydal.connection.ConnectionPool attribute), 37
 check_reserved_keyword() (pydal.base.DAL method), 34
 cleanup() (in module pydal.helpers.methods), 29
 clear() (pydal.helpers.classes.BasicStorage method), 27
 clone() (pydal.objects.Field method), 41
 close() (pydal.adapters.teradata.TeradataAdapter method), 26
 close() (pydal.base.DAL method), 34
 close() (pydal.connection.ConnectionPool method), 37
 close() (pydal.helpers.classes.DatabaseStoredFile method), 27
 close_all_instances() (pydal.connection.ConnectionPool static method), 37
 close_connection() (pydal.adapters.base.BaseAdapter method), 5
 close_connection() (pydal.adapters.base.NoSQLAdapter method), 8
 close_connection() (pydal.helpers.classes.DatabaseStoredFile method), 27
 COALESCE() (pydal.adapters.base.BaseAdapter method), 3
 coalesce() (pydal.objects.Expression method), 39
 COALESCE_ZERO() (pydal.adapters.base.BaseAdapter method), 3
 coalesce_zero() (pydal.objects.Expression method), 39
 column() (pydal.objects.Rows method), 43
 COMMA() (pydal.adapters.base.BaseAdapter method), 4
 COMMA() (pydal.adapters.couchdb.CouchDBAdapter method), 9
 COMMA() (pydal.adapters.mongo.MongoDBAdapter method), 16
 commit() (pydal.adapters.base.BaseAdapter method), 5
 commit() (pydal.adapters.base.NoSQLAdapter method), 8
 commit() (pydal.base.DAL method), 34
 commit_on_alter_table (pydal.adapters.base.BaseAdapter attribute), 5
 commit_on_alter_table (pydal.adapters.firebird.FireBirdAdapter attribute), 11
 commit_on_alter_table (pydal.adapters.mysql.MySQLAdapter attribute), 21
 commit_on_alter_table (pydal.adapters.oracle.OracleAdapter attribute), 22
 commit_prepared() (pydal.adapters.base.BaseAdapter method), 5
 commit_prepared() (pydal.adapters.base.NoSQLAdapter method), 8

commit_prepared() (pydal.adapters.mysql.MySQLAdapter method), 21
 commit_prepared() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 common_filter() (pydal.adapters.base.BaseAdapter method), 5
 CONCAT() (pydal.adapters.base.BaseAdapter method), 4
 CONCAT() (pydal.adapters.mssql.MSSQLAdapter method), 19
 CONCAT() (pydal.adapters.mysql.MySQLAdapter method), 21
 concat_add() (pydal.adapters.base.BaseAdapter method), 5
 concat_add() (pydal.adapters.base.NoSQLAdapter method), 8
 concat_add() (pydal.adapters.mssql.MSSQLAdapter method), 20
 connection (pydal.adapters.base.BaseAdapter attribute), 5
 ConnectionPool (class in pydal.connection), 37
 connector() (pydal.adapters.base.BaseAdapter method), 5
 constraint_name() (pydal.adapters.base.BaseAdapter method), 5
 constraint_name() (pydal.adapters.base.NoSQLAdapter method), 8
 constraint_name() (pydal.adapters.oracle.OracleAdapter method), 22
 CONTAINS() (pydal.adapters.base.BaseAdapter method), 4
 CONTAINS() (pydal.adapters.firebird.FireBirdAdapter method), 10
 CONTAINS() (pydal.adapters.imap.IMAPAdapter method), 13
 CONTAINS() (pydal.adapters.mongo.MongoDBAdapter method), 16
 CONTAINS() (pydal.adapters.postgres.NewPostgreSQLAdapter method), 23
 contains() (pydal.objects.Expression method), 39
 convert_date() (pydal.adapters.imap.IMAPAdapter method), 14
 copy() (pydal.helpers.classes.BasicStorage method), 27
 CouchDBAdapter (class in pydal.adapters.couchdb), 9
 COUNT() (pydal.adapters.base.BaseAdapter method), 4
 count() (pydal.adapters.base.BaseAdapter method), 5
 count() (pydal.adapters.couchdb.CouchDBAdapter method), 9
 count() (pydal.adapters.imap.IMAPAdapter method), 14
 count() (pydal.adapters.mongo.MongoDBAdapter method), 17
 count() (pydal.objects.Field method), 41
 count() (pydal.objects.LazySet method), 41
 count() (pydal.objects.Set method), 44
 create_sequence_and_triggers() (pydal.adapters.base.BaseAdapter method), 5
 create_sequence_and_triggers() (pydal.adapters.base.NoSQLAdapter method), 8
 create_sequence_and_triggers() (pydal.adapters.firebird.FireBirdAdapter method), 11
 create_sequence_and_triggers() (pydal.adapters.ingres.IngresAdapter method), 15
 create_sequence_and_triggers() (pydal.adapters.oracle.OracleAdapter method), 22
 create_sequence_and_triggers() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 create_sequence_and_triggers() (pydal.adapters.sapdb.SAPDBAdapter method), 25
 create_table() (pydal.adapters.base.BaseAdapter method), 5
 create_table() (pydal.adapters.couchdb.CouchDBAdapter method), 9
 create_table() (pydal.adapters.imap.IMAPAdapter method), 14
 create_table() (pydal.adapters.mongo.MongoDBAdapter method), 17
 CubridAdapter (class in pydal.adapters.cubrid), 10

D

DAL (class in pydal.base), 32
 DAL.Table (class in pydal.base), 33
 DAL_pickler() (in module pydal.base), 37
 DAL_unpickler() (in module pydal.base), 37
 DatabaseStoredFile (class in pydal.helpers.classes), 27
 dbengine (pydal.objects.Expression method), 39
 DB2Adapter (class in pydal.adapters.db2), 10
 dbengine (pydal.adapters.imap.IMAPAdapter attribute), 14
 dbpath (pydal.adapters.base.BaseAdapter attribute), 5
 decode() (pydal.adapters.mongo.MongoBlob static method), 16
 define_table() (pydal.base.DAL method), 34
 define_tables() (pydal.adapters.imap.IMAPAdapter method), 14
 delete() (pydal.adapters.base.BaseAdapter method), 5
 delete() (pydal.adapters.couchdb.CouchDBAdapter method), 9
 delete() (pydal.adapters.imap.IMAPAdapter method), 14
 delete() (pydal.adapters.mongo.MongoDBAdapter method), 17
 delete() (pydal.adapters.sqlite.SQLiteAdapter method), 25

- delete() (pydal.objects.LazySet method), 41
 - delete() (pydal.objects.Set method), 44
 - delete_uploaded_files() (pydal.objects.LazySet method), 41
 - delete_uploaded_files() (pydal.objects.Set method), 44
 - distributed_transaction_begin() (pydal.adapters.base.BaseAdapter method), 5
 - distributed_transaction_begin() (pydal.adapters.base.NoSQLAdapter method), 8
 - distributed_transaction_begin() (pydal.adapters.mysql.MySQLAdapter method), 21
 - distributed_transaction_begin() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 - distributed_transaction_begin() (pydal.base.DAL static method), 35
 - distributed_transaction_commit() (pydal.base.DAL static method), 35
 - DIV() (pydal.adapters.base.BaseAdapter method), 4
 - DIV() (pydal.adapters.base.NoSQLAdapter method), 7
 - DIV() (pydal.adapters.mongo.MongoDBAdapter method), 16
 - driver (pydal.adapters.base.BaseAdapter attribute), 5
 - driver_auto_json (pydal.adapters.base.BaseAdapter attribute), 5
 - driver_auto_json (pydal.adapters.mongo.MongoDBAdapter attribute), 17
 - driver_name (pydal.adapters.base.BaseAdapter attribute), 5
 - drivers (pydal.adapters.base.BaseAdapter attribute), 5
 - drivers (pydal.adapters.couchdb.CouchDBAdapter attribute), 9
 - drivers (pydal.adapters.cubrid.CubridAdapter attribute), 10
 - drivers (pydal.adapters.db2.DB2Adapter attribute), 10
 - drivers (pydal.adapters.firebird.FireBirdAdapter attribute), 11
 - drivers (pydal.adapters.firebird.FireBirdEmbeddedAdapter attribute), 11
 - drivers (pydal.adapters.imap.IMAPAdapter attribute), 14
 - drivers (pydal.adapters.informix.InformixAdapter attribute), 15
 - drivers (pydal.adapters.ingres.IngresAdapter attribute), 15
 - drivers (pydal.adapters.ingres.IngresUnicodeAdapter attribute), 16
 - drivers (pydal.adapters.mongo.MongoDBAdapter attribute), 17
 - drivers (pydal.adapters.mssql.MSSQL2Adapter attribute), 18
 - drivers (pydal.adapters.mssql.MSSQL3NAdapter attribute), 18
 - drivers (pydal.adapters.mssql.MSSQLAdapter attribute), 20
 - drivers (pydal.adapters.mssql.MSSQLNAdapter attribute), 20
 - drivers (pydal.adapters.mssql.SybaseAdapter attribute), 20
 - drivers (pydal.adapters.mssql.VerticaAdapter attribute), 21
 - drivers (pydal.adapters.mysql.MySQLAdapter attribute), 21
 - drivers (pydal.adapters.oracle.OracleAdapter attribute), 22
 - drivers (pydal.adapters.postgres.JDBCPostgreSQLAdapter attribute), 22
 - drivers (pydal.adapters.postgres.NewPostgreSQLAdapter attribute), 23
 - drivers (pydal.adapters.postgres.PostgreSQLAdapter attribute), 24
 - drivers (pydal.adapters.sapdb.SAPDBAdapter attribute), 25
 - drivers (pydal.adapters.sqlite.JDBCSQLiteAdapter attribute), 25
 - drivers (pydal.adapters.sqlite.SpatialLiteAdapter attribute), 26
 - drivers (pydal.adapters.sqlite.SQLiteAdapter attribute), 25
 - drivers (pydal.adapters.teradata.TeradataAdapter attribute), 26
 - drop() (pydal.adapters.base.BaseAdapter method), 5
 - drop() (pydal.adapters.base.NoSQLAdapter method), 8
 - drop() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - drop() (pydal.base.DAL.Table method), 34
 - drop() (pydal.objects.Table method), 44
- ## E
- encode_text() (pydal.adapters.imap.IMAPAdapter method), 14
 - ENDSWITH() (pydal.adapters.base.BaseAdapter method), 4
 - ENDSWITH() (pydal.adapters.base.NoSQLAdapter method), 7
 - ENDSWITH() (pydal.adapters.mongo.MongoDBAdapter method), 16
 - endswith() (pydal.helpers.classes.SQLCustomType method), 28
 - endswith() (pydal.objects.Expression method), 39
 - EPOCH() (pydal.adapters.base.BaseAdapter method), 4
 - EPOCH() (pydal.adapters.firebird.FireBirdAdapter method), 10
 - EPOCH() (pydal.adapters.mssql.MSSQLAdapter method), 19

- EPOCH() (pydal.adapters.mysql.MySQLAdapter method), 21
- epoch() (pydal.objects.Expression method), 39
- EQ() (pydal.adapters.base.BaseAdapter method), 4
- EQ() (pydal.adapters.couchdb.CouchDBAdapter method), 9
- EQ() (pydal.adapters.imap.IMAPAdapter method), 13
- EQ() (pydal.adapters.mongo.MongoDBAdapter method), 16
- EQ() (pydal.adapters.postgres.NewPostgreSQLAdapter method), 23
- error_messages (pydal.adapters.mongo.MongoDBAdapter attribute), 17
- escape() (pydal.helpers.classes.DatabaseStoredFile method), 27
- exclude() (pydal.objects.Rows method), 43
- execute() (pydal.adapters.base.BaseAdapter method), 5
- execute() (pydal.adapters.base.NoSQLAdapter method), 8
- execute() (pydal.adapters.db2.DB2Adapter method), 10
- execute() (pydal.adapters.informix.InformixAdapter method), 15
- execute() (pydal.adapters.mssql.MSSQL2Adapter method), 18
- execute() (pydal.adapters.mssql.MSSQLNAdapter method), 20
- execute() (pydal.adapters.mssql.VerticaAdapter method), 21
- execute() (pydal.adapters.oracle.OracleAdapter method), 22
- execute() (pydal.adapters.sqlite.JDBCSQLiteAdapter method), 25
- executesql() (pydal.base.DAL method), 35
- exists() (pydal.helpers.classes.DatabaseStoredFile static method), 27
- expand() (pydal.adapters.base.BaseAdapter method), 6
- expand() (pydal.adapters.couchdb.CouchDBAdapter method), 9
- expand() (pydal.adapters.mongo.MongoDBAdapter method), 17
- expand_all() (pydal.adapters.base.BaseAdapter method), 6
- export_to_csv_file() (pydal.base.DAL method), 35
- export_to_csv_file() (pydal.objects.BasicRows method), 38
- Expression (class in pydal.objects), 38
- EXTRACT() (pydal.adapters.base.BaseAdapter method), 4
- EXTRACT() (pydal.adapters.base.NoSQLAdapter method), 8
- EXTRACT() (pydal.adapters.mssql.MSSQLAdapter method), 19
- EXTRACT() (pydal.adapters.mssql.VerticaAdapter method), 21
- EXTRACT() (pydal.adapters.sqlite.SQLiteAdapter method), 25
- ## F
- FALSE (pydal.adapters.base.BaseAdapter attribute), 4
- FALSE (pydal.adapters.mssql.MSSQLAdapter attribute), 19
- FALSE_exp (pydal.adapters.base.BaseAdapter attribute), 4
- Field (class in pydal.objects), 40
- FieldMethod (class in pydal.objects), 41
- fields (pydal.base.DAL.Table attribute), 34
- fields (pydal.objects.Table attribute), 44
- FieldVirtual (class in pydal.objects), 41
- file_close() (pydal.adapters.base.BaseAdapter method), 6
- file_close() (pydal.adapters.couchdb.CouchDBAdapter method), 9
- file_close() (pydal.helpers.classes.UseDatabaseStoredFile method), 28
- file_delete() (pydal.adapters.base.BaseAdapter method), 6
- file_delete() (pydal.helpers.classes.UseDatabaseStoredFile method), 28
- file_exists() (pydal.adapters.base.BaseAdapter method), 6
- file_exists() (pydal.adapters.couchdb.CouchDBAdapter method), 9
- file_exists() (pydal.helpers.classes.UseDatabaseStoredFile method), 28
- file_open() (pydal.adapters.base.BaseAdapter method), 6
- file_open() (pydal.adapters.couchdb.CouchDBAdapter method), 9
- file_open() (pydal.helpers.classes.UseDatabaseStoredFile method), 28
- find() (pydal.objects.Rows method), 43
- find_driver() (pydal.adapters.base.BaseAdapter method), 6
- find_or_make_work_folder() (pydal.connection.ConnectionPool method), 37
- FireBirdAdapter (class in pydal.adapters.firebird), 10
- FireBirdEmbeddedAdapter (class in pydal.adapters.firebird), 11
- first() (pydal.objects.IterRows method), 41
- first() (pydal.objects.Rows method), 43
- folder (pydal.adapters.base.BaseAdapter attribute), 6
- formatter() (pydal.objects.Field method), 41
- ## G
- GE() (pydal.adapters.base.BaseAdapter method), 4
- GE() (pydal.adapters.imap.IMAPAdapter method), 13
- GE() (pydal.adapters.mongo.MongoDBAdapter method), 16
- geoLine() (in module pydal.helpers.methods), 29
- geoPoint() (in module pydal.helpers.methods), 29

[geoPolygon\(\)](#) (in module `pydal.helpers.methods`), 29
[get\(\)](#) (`pydal.helpers.classes.BasicStorage` method), 27
[get\(\)](#) (`pydal.helpers.classes.Reference` method), 27
[get\(\)](#) (`pydal.objects.Row` method), 42
[get_charset\(\)](#) (`pydal.adapters.imap.IMAPAdapter` method), 14
[get_instances\(\)](#) (`pydal.base.DAL` static method), 35
[get_last_message\(\)](#) (`pydal.adapters.imap.IMAPAdapter` method), 14
[get_mailboxes\(\)](#) (`pydal.adapters.imap.IMAPAdapter` method), 14
[get_query_mailbox\(\)](#) (`pydal.adapters.imap.IMAPAdapter` method), 14
[get_table\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 6
[get_uid_bounds\(\)](#) (`pydal.adapters.imap.IMAPAdapter` method), 14
[GoogleDatastoreAdapter](#) (in module `pydal.adapters`), 26
[group_by_value\(\)](#) (`pydal.objects.Rows` method), 43
[GT\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 4
[GT\(\)](#) (`pydal.adapters.imap.IMAPAdapter` method), 13
[GT\(\)](#) (`pydal.adapters.mongo.MongoDBAdapter` method), 16

H

[has_key\(\)](#) (`pydal.helpers.classes.BasicStorage` method), 27
[has_representer\(\)](#) (`pydal.base.DAL` method), 35
[header_represent\(\)](#) (`pydal.adapters.imap.IMAPAdapter` static method), 14
[hide_password\(\)](#) (in module `pydal.helpers.methods`), 29
[hour\(\)](#) (`pydal.objects.Expression` method), 39

I

[id_query\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 6
[id_query\(\)](#) (`pydal.adapters.base.NoSQLAdapter` method), 8
[ILIKE\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 4
[ILIKE\(\)](#) (`pydal.adapters.base.NoSQLAdapter` method), 8
[ILIKE\(\)](#) (`pydal.adapters.mongo.MongoDBAdapter` method), 16
[ILIKE\(\)](#) (`pydal.adapters.mssql.MSSQL2Adapter` method), 18
[ILIKE\(\)](#) (`pydal.adapters.mssql.MSSQLNAdapter` method), 20
[ILIKE\(\)](#) (`pydal.adapters.postgres.NewPostgreSQLAdapter` method), 23
[ILIKE\(\)](#) (`pydal.adapters.postgres.PostgreSQLAdapter` method), 23
[ilike\(\)](#) (`pydal.objects.Expression` method), 39
[IMAPAdapter](#) (class in `pydal.adapters.imap`), 11
[import_from_csv_file\(\)](#) (`pydal.base.DAL` method), 36
[import_from_csv_file\(\)](#) (`pydal.base.DAL.Table` method), 34
[import_from_csv_file\(\)](#) (`pydal.objects.Table` method), 44

[import_table_definitions\(\)](#) (`pydal.base.DAL` method), 36
[InformixAdapter](#) (class in `pydal.adapters.informix`), 15
[InformixSEAdapter](#) (class in `pydal.adapters.informix`), 15
[IngresAdapter](#) (class in `pydal.adapters.ingres`), 15
[IngresUnicodeAdapter](#) (class in `pydal.adapters.ingres`), 16
[insert\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 6
[insert\(\)](#) (`pydal.adapters.couchdb.CouchDBAdapter` method), 9
[insert\(\)](#) (`pydal.adapters.imap.IMAPAdapter` method), 14
[insert\(\)](#) (`pydal.adapters.mongo.MongoDBAdapter` method), 17
[insert\(\)](#) (`pydal.base.DAL.Table` method), 34
[insert\(\)](#) (`pydal.objects.Table` method), 45
[int2uuid\(\)](#) (in module `pydal.helpers.methods`), 29
[INVERT\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 4
[INVERT\(\)](#) (`pydal.adapters.mongo.MongoDBAdapter` method), 16
[is_flag\(\)](#) (`pydal.adapters.imap.IMAPAdapter` method), 14
[is_numerical_type\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 6
[isempty\(\)](#) (`pydal.objects.LazySet` method), 41
[isempty\(\)](#) (`pydal.objects.Set` method), 44
[isOperationalError\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 6
[isProgrammingError\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 6
[items\(\)](#) (`pydal.helpers.classes.BasicStorage` method), 27
[iteritems\(\)](#) (`pydal.helpers.classes.BasicStorage` method), 27
[iterkeys\(\)](#) (`pydal.helpers.classes.BasicStorage` method), 27
[iterparse\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 6
[IterRows](#) (class in `pydal.objects`), 41
[iterselect\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 6
[iterselect\(\)](#) (`pydal.objects.Set` method), 44
[itervalues\(\)](#) (`pydal.helpers.classes.BasicStorage` method), 27

J

[JDBCPostgreSQLAdapter](#) (class in `pydal.adapters.postgres`), 22
[JDBCSQLiteAdapter](#) (class in `pydal.adapters.sqlite`), 25
[JOIN\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 4
[json\(\)](#) (`pydal.objects.BasicRows` method), 38

K

[keys\(\)](#) (`pydal.helpers.classes.BasicStorage` method), 27

L

[last\(\)](#) (`pydal.objects.Rows` method), 43
[lastrowid\(\)](#) (`pydal.adapters.base.BaseAdapter` method), 6
[lastrowid\(\)](#) (`pydal.adapters.base.NoSQLAdapter` method), 8

- lastrowid() (pydal.adapters.db2.DB2Adapter method), 10
- lastrowid() (pydal.adapters.firebird.FireBirdAdapter method), 11
- lastrowid() (pydal.adapters.informix.InformixAdapter method), 15
- lastrowid() (pydal.adapters.ingres.IngresAdapter method), 15
- lastrowid() (pydal.adapters.mssql.MSSQLAdapter method), 20
- lastrowid() (pydal.adapters.mssql.VerticaAdapter method), 21
- lastrowid() (pydal.adapters.oracle.OracleAdapter method), 22
- lastrowid() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
- lastrowid() (pydal.adapters.sapdb.SAPDBAdapter method), 25
- Lazy (pydal.objects.Field attribute), 40
- lazy_define_table() (pydal.base.DAL method), 36
- LazyReferenceGetter (class in pydal.objects), 41
- LazySet (class in pydal.objects), 41
- LE() (pydal.adapters.base.BaseAdapter method), 4
- LE() (pydal.adapters.imap.IMAPAdapter method), 13
- LE() (pydal.adapters.mongo.MongoDBAdapter method), 16
- LEFT_JOIN() (pydal.adapters.base.BaseAdapter method), 4
- LEFT_JOIN() (pydal.adapters.base.NoSQLAdapter method), 8
- LEFT_JOIN() (pydal.adapters.db2.DB2Adapter method), 10
- LEFT_JOIN() (pydal.adapters.ingres.IngresAdapter method), 15
- LEFT_JOIN() (pydal.adapters.mssql.MSSQLAdapter method), 19
- LEFT_JOIN() (pydal.adapters.oracle.OracleAdapter method), 22
- LEFT_JOIN() (pydal.adapters.teradata.TeradataAdapter method), 26
- len() (pydal.objects.Expression method), 39
- LENGTH() (pydal.adapters.base.BaseAdapter method), 4
- LENGTH() (pydal.adapters.base.NoSQLAdapter method), 8
- LENGTH() (pydal.adapters.firebird.FireBirdAdapter method), 10
- LENGTH() (pydal.adapters.mssql.MSSQLAdapter method), 19
- LIKE() (pydal.adapters.base.BaseAdapter method), 4
- LIKE() (pydal.adapters.mongo.MongoDBAdapter method), 16
- LIKE() (pydal.adapters.postgres.PostgreSQLAdapter method), 23
- like() (pydal.objects.Expression method), 39
- list_represent() (in module pydal.helpers.methods), 29
- log() (pydal.adapters.base.BaseAdapter method), 6
- log_execute() (pydal.adapters.base.BaseAdapter method), 6
- log_execute() (pydal.adapters.base.NoSQLAdapter method), 8
- logger (pydal.base.DAL attribute), 36
- LOWER() (pydal.adapters.base.BaseAdapter method), 4
- LOWER() (pydal.adapters.base.NoSQLAdapter method), 8
- lower() (pydal.objects.Expression method), 39
- LT() (pydal.adapters.base.BaseAdapter method), 4
- LT() (pydal.adapters.imap.IMAPAdapter method), 13
- LT() (pydal.adapters.mongo.MongoDBAdapter method), 16
- ## M
- max() (pydal.objects.Expression method), 39
- MetaDAL (class in pydal.base), 37
- Method (pydal.objects.Field attribute), 41
- MethodAdder (class in pydal.helpers.classes), 27
- migrate_table() (pydal.adapters.base.BaseAdapter method), 6
- migrate_table() (pydal.adapters.base.NoSQLAdapter method), 8
- min() (pydal.objects.Expression method), 39
- minutes() (pydal.objects.Expression method), 39
- MOD() (pydal.adapters.base.BaseAdapter method), 4
- MOD() (pydal.adapters.mongo.MongoDBAdapter method), 17
- MONGO_BLOB_BYTES (pydal.adapters.mongo.MongoBlob attribute), 16
- MONGO_BLOB_NON_UTF8_STR (pydal.adapters.mongo.MongoBlob attribute), 16
- MongoBlob (class in pydal.adapters.mongo), 16
- MongoDBAdapter (class in pydal.adapters.mongo), 16
- month() (pydal.objects.Expression method), 39
- MSSQL2Adapter (class in pydal.adapters.mssql), 18
- MSSQL3Adapter (class in pydal.adapters.mssql), 18
- MSSQL3NAdapter (class in pydal.adapters.mssql), 18
- MSSQL4Adapter (class in pydal.adapters.mssql), 18
- MSSQL4NAdapter (class in pydal.adapters.mssql), 19
- MSSQLAdapter (class in pydal.adapters.mssql), 19
- MSSQLNAdapter (class in pydal.adapters.mssql), 20
- MUL() (pydal.adapters.base.BaseAdapter method), 4
- MUL() (pydal.adapters.base.NoSQLAdapter method), 8
- MUL() (pydal.adapters.mongo.MongoDBAdapter method), 17
- MySQLAdapter (class in pydal.adapters.mysql), 21
- ## N
- NE() (pydal.adapters.base.BaseAdapter method), 4

- NE() (pydal.adapters.couchdb.CouchDBAdapter method), 9
 - NE() (pydal.adapters.imap.IMAPAdapter method), 13
 - NE() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - nested_select() (pydal.objects.LazySet method), 41
 - nested_select() (pydal.objects.Set method), 44
 - NewPostgreSQLAdapter (class in pydal.adapters.postgres), 22
 - next() (pydal.objects.IterRows method), 41
 - NoSQLAdapter (class in pydal.adapters.base), 7
 - NOT() (pydal.adapters.base.BaseAdapter method), 4
 - NOT() (pydal.adapters.imap.IMAPAdapter method), 13
 - NOT() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - NOT_NULL() (pydal.adapters.base.BaseAdapter method), 4
 - NOT_NULL() (pydal.adapters.firebird.FireBirdAdapter method), 10
 - NOT_NULL() (pydal.adapters.informix.InformixAdapter method), 15
 - NOT_NULL() (pydal.adapters.oracle.OracleAdapter method), 22
- O**
- object_id() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - ON() (pydal.adapters.base.BaseAdapter method), 4
 - ON() (pydal.adapters.base.NoSQLAdapter method), 8
 - ON() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - on() (pydal.base.DAL.Table method), 34
 - on() (pydal.objects.Table method), 45
 - OR() (pydal.adapters.base.BaseAdapter method), 4
 - OR() (pydal.adapters.base.NoSQLAdapter method), 8
 - OR() (pydal.adapters.couchdb.CouchDBAdapter method), 9
 - OR() (pydal.adapters.imap.IMAPAdapter method), 13
 - OR() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - oracle_fix (pydal.adapters.oracle.OracleAdapter attribute), 22
 - OracleAdapter (class in pydal.adapters.oracle), 22
- P**
- parse() (pydal.adapters.base.BaseAdapter method), 6
 - parse() (pydal.objects.Set method), 44
 - parse_as_rest() (pydal.base.DAL method), 36
 - parse_blob() (pydal.adapters.base.BaseAdapter method), 6
 - parse_blob() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - parse_boolean() (pydal.adapters.base.BaseAdapter method), 6
 - parse_date() (pydal.adapters.base.BaseAdapter method), 6
 - parse_datetime() (pydal.adapters.base.BaseAdapter method), 6
 - parse_decimal() (pydal.adapters.base.BaseAdapter method), 6
 - parse_double() (pydal.adapters.base.BaseAdapter method), 6
 - parse_id() (pydal.adapters.base.BaseAdapter method), 6
 - parse_id() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - parse_integer() (pydal.adapters.base.BaseAdapter method), 6
 - parse_json() (pydal.adapters.base.BaseAdapter method), 6
 - parse_list_integers() (pydal.adapters.base.BaseAdapter method), 6
 - parse_list_integers() (pydal.adapters.base.NoSQLAdapter method), 8
 - parse_list_integers() (pydal.adapters.postgres.NewPostgreSQLAdapter method), 23
 - parse_list_references() (pydal.adapters.base.BaseAdapter method), 6
 - parse_list_references() (pydal.adapters.base.NoSQLAdapter method), 8
 - parse_list_references() (pydal.adapters.postgres.NewPostgreSQLAdapter method), 23
 - parse_list_strings() (pydal.adapters.base.BaseAdapter method), 6
 - parse_list_strings() (pydal.adapters.base.NoSQLAdapter method), 8
 - parse_list_strings() (pydal.adapters.postgres.NewPostgreSQLAdapter method), 23
 - parse_reference() (pydal.adapters.base.BaseAdapter method), 6
 - parse_reference() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - parse_time() (pydal.adapters.base.BaseAdapter method), 7
 - parse_value() (pydal.adapters.base.BaseAdapter method), 7
 - pickle_basicstorage() (in module pydal.helpers.classes), 29
 - pickle_row() (in module pydal.objects), 45
 - pluralize() (in module pydal.helpers.methods), 29
 - POOLS (pydal.connection.ConnectionPool attribute), 37
 - pop() (pydal.helpers.classes.BasicStorage method), 27
 - PostgreSQLAdapter (class in pydal.adapters.postgres), 23
 - prepare() (pydal.adapters.base.BaseAdapter method), 7

prepare() (pydal.adapters.base.NoSQLAdapter method), 8

prepare() (pydal.adapters.mysql.MySQLAdapter method), 21

prepare() (pydal.adapters.postgres.PostgreSQLAdapter method), 24

PRIMARY_KEY() (pydal.adapters.base.BaseAdapter method), 4

PRIMARY_KEY() (pydal.adapters.base.NoSQLAdapter method), 8

PRIMARY_KEY() (pydal.adapters.mssql.MSSQLAdapter method), 19

pydal (module), 45

pydal.adapters (module), 26

pydal.adapters.base (module), 3

pydal.adapters.couchdb (module), 9

pydal.adapters.cubrid (module), 10

pydal.adapters.db2 (module), 10

pydal.adapters.firebird (module), 10

pydal.adapters.imap (module), 11

pydal.adapters.informix (module), 15

pydal.adapters.ingres (module), 15

pydal.adapters.mongo (module), 16

pydal.adapters.mssql (module), 18

pydal.adapters.mysql (module), 21

pydal.adapters.oracle (module), 22

pydal.adapters.postgres (module), 22

pydal.adapters.sapdb (module), 24

pydal.adapters.sqlite (module), 25

pydal.adapters.teradata (module), 26

pydal.base (module), 30

pydal.connection (module), 37

pydal.helpers (module), 30

pydal.helpers.classes (module), 27

pydal.helpers.methods (module), 29

pydal.helpers.regex (module), 30

pydal.objects (module), 37

Q

Query (class in pydal.objects), 42

QUOTE_TEMPLATE (pydal.adapters.base.BaseAdapter attribute), 4

QUOTE_TEMPLATE (pydal.adapters.base.NoSQLAdapter attribute), 8

QUOTE_TEMPLATE (pydal.adapters.mssql.MSSQLAdapter attribute), 19

QUOTE_TEMPLATE (pydal.adapters.mysql.MySQLAdapter attribute), 21

QUOTE_TEMPLATE (pydal.adapters.postgres.PostgreSQLAdapter

attribute), 23

R

RANDOM() (pydal.adapters.base.BaseAdapter method), 4

RANDOM() (pydal.adapters.base.NoSQLAdapter method), 8

RANDOM() (pydal.adapters.db2.DB2Adapter method), 10

RANDOM() (pydal.adapters.firebird.FireBirdAdapter method), 10

RANDOM() (pydal.adapters.informix.InformixAdapter method), 15

RANDOM() (pydal.adapters.ingres.IngresAdapter method), 15

RANDOM() (pydal.adapters.mssql.MSSQLAdapter method), 19

RANDOM() (pydal.adapters.mysql.MySQLAdapter method), 21

RANDOM() (pydal.adapters.oracle.OracleAdapter method), 22

RANDOM() (pydal.adapters.postgres.PostgreSQLAdapter method), 23

RAW() (pydal.adapters.base.BaseAdapter method), 4

read() (pydal.helpers.classes.DatabaseStoredFile method), 27

readline() (pydal.helpers.classes.DatabaseStoredFile method), 27

reconnect() (pydal.adapters.imap.IMAPAdapter method), 14

reconnect() (pydal.connection.ConnectionPool method), 37

RecordDeleter (class in pydal.helpers.classes), 27

RecordUpdater (class in pydal.helpers.classes), 27

Reference (class in pydal.helpers.classes), 27

Reference_pickler() (in module pydal.helpers.classes), 28

Reference_unpickler() (in module pydal.helpers.classes), 28

REGEX_ARGPATTERN (pydal.adapters.mssql.MSSQLAdapter attribute), 19

REGEX_DSN (pydal.adapters.mssql.MSSQLAdapter attribute), 19

REGEX_URI (pydal.adapters.cubrid.CubridAdapter attribute), 10

REGEX_URI (pydal.adapters.firebird.FireBirdAdapter attribute), 10

REGEX_URI (pydal.adapters.firebird.FireBirdEmbeddedAdapter attribute), 11

REGEX_URI (pydal.adapters.imap.IMAPAdapter attribute), 13

REGEX_URI (pydal.adapters.informix.InformixAdapter attribute), 15

- REGEX_URI (pydal.adapters.mssql.MSSQLAdapter attribute), 19
- REGEX_URI (pydal.adapters.mysql.MySQLAdapter attribute), 21
- REGEX_URI (pydal.adapters.postgres.JDBCPostgreSQLAdapter attribute), 22
- REGEX_URI (pydal.adapters.postgres.PostgreSQLAdapter attribute), 23
- REGEX_URI (pydal.adapters.sapdb.SAPDBAdapter attribute), 24
- REGEXP() (pydal.adapters.base.BaseAdapter method), 4
- REGEXP() (pydal.adapters.mysql.MySQLAdapter method), 21
- REGEXP() (pydal.adapters.oracle.OracleAdapter method), 22
- REGEXP() (pydal.adapters.postgres.PostgreSQLAdapter method), 23
- REGEXP() (pydal.adapters.sqlite.SQLiteAdapter method), 25
- regexp() (pydal.objects.Expression method), 39
- register() (pydal.helpers.classes.MethodAdder method), 27
- render() (pydal.objects.Rows method), 43
- REPLACE() (pydal.adapters.base.BaseAdapter method), 5
- replace() (pydal.objects.Expression method), 39
- represent() (pydal.adapters.base.BaseAdapter method), 7
- represent() (pydal.adapters.base.NoSQLAdapter method), 8
- represent() (pydal.adapters.couchdb.CouchDBAdapter method), 9
- represent() (pydal.adapters.mongo.MongoDBAdapter method), 17
- represent() (pydal.adapters.mssql.MSSQL2Adapter method), 18
- represent() (pydal.adapters.mssql.MSSQLAdapter method), 20
- represent() (pydal.adapters.mssql.MSSQLNAdapter method), 20
- represent() (pydal.adapters.postgres.NewPostgreSQLAdapter method), 23
- represent() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
- represent() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
- represent() (pydal.base.DAL method), 36
- represent_exceptions() (pydal.adapters.base.BaseAdapter method), 7
- represent_exceptions() (pydal.adapters.base.NoSQLAdapter method), 9
- represent_exceptions() (pydal.adapters.db2.DB2Adapter method), 10
- represent_exceptions() (pydal.adapters.informix.InformixAdapter method), 15
- represent_exceptions() (pydal.adapters.oracle.OracleAdapter method), 22
- representers (pydal.base.DAL attribute), 36
- retrieve() (pydal.objects.Field method), 41
- retrieve_file_properties() (pydal.objects.Field method), 41
- rollback() (pydal.adapters.base.BaseAdapter method), 7
- rollback() (pydal.adapters.base.NoSQLAdapter method), 9
- rollback() (pydal.base.DAL method), 36
- rollback_prepared() (pydal.adapters.base.BaseAdapter method), 7
- rollback_prepared() (pydal.adapters.base.NoSQLAdapter method), 9
- rollback_prepared() (pydal.adapters.mysql.MySQLAdapter method), 21
- rollback_prepared() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
- Row (class in pydal.objects), 42
- Rows (class in pydal.objects), 42
- rowslice() (pydal.adapters.base.BaseAdapter method), 7
- rowslice() (pydal.adapters.base.NoSQLAdapter method), 9
- rowslice() (pydal.adapters.db2.DB2Adapter method), 10
- rowslice() (pydal.adapters.informix.InformixSEAdapter method), 15
- rowslice() (pydal.adapters.mssql.MSSQL3Adapter method), 18
- rowslice() (pydal.adapters.mssql.MSSQL3NAdapter method), 18
- rowslice() (pydal.adapters.mssql.MSSQL4Adapter method), 19
- rowslice() (pydal.adapters.mssql.MSSQL4NAdapter method), 19
- rowslice() (pydal.adapters.mssql.MSSQLAdapter method), 20

S

- SAPDBAdapter (class in pydal.adapters.sapdb), 24
- save_dbt() (pydal.adapters.base.BaseAdapter method), 7
- seconds() (pydal.objects.Expression method), 39
- select() (pydal.adapters.base.BaseAdapter method), 7
- select() (pydal.adapters.couchdb.CouchDBAdapter method), 9
- select() (pydal.adapters.imap.IMAPAdapter method), 14
- select() (pydal.adapters.mongo.MongoDBAdapter method), 17
- select() (pydal.adapters.sqlite.SQLiteAdapter method), 25

- select() (pydal.objects.LazySet method), 41
- select() (pydal.objects.Set method), 44
- select_limitby() (pydal.adapters.base.BaseAdapter method), 7
- select_limitby() (pydal.adapters.db2.DB2Adapter method), 10
- select_limitby() (pydal.adapters.firebird.FireBirdAdapter method), 11
- select_limitby() (pydal.adapters.informix.InformixAdapter method), 15
- select_limitby() (pydal.adapters.informix.InformixSEAdapter method), 15
- select_limitby() (pydal.adapters.ingres.IngresAdapter method), 15
- select_limitby() (pydal.adapters.mssql.MSSQL3Adapter method), 18
- select_limitby() (pydal.adapters.mssql.MSSQL3NAdapter method), 18
- select_limitby() (pydal.adapters.mssql.MSSQL4Adapter method), 19
- select_limitby() (pydal.adapters.mssql.MSSQL4NAdapter method), 19
- select_limitby() (pydal.adapters.mssql.MSSQLAdapter method), 20
- select_limitby() (pydal.adapters.mssql.VerticaAdapter method), 21
- select_limitby() (pydal.adapters.oracle.OracleAdapter method), 22
- select_limitby() (pydal.adapters.sapdb.SAPDBAdapter method), 25
- select_limitby() (pydal.adapters.teradata.TeradataAdapter method), 26
- sequence_name() (pydal.adapters.base.BaseAdapter method), 7
- sequence_name() (pydal.adapters.firebird.FireBirdAdapter method), 11
- sequence_name() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
- sequence_name() (pydal.adapters.sapdb.SAPDBAdapter method), 25
- Serializable (class in pydal.helpers.classes), 28
- serializers (pydal.base.DAL attribute), 36
- Set (class in pydal.objects), 43
- set_attributes() (pydal.objects.Field method), 41
- set_folder() (pydal.base.DAL static method), 36
- set_folder() (pydal.connection.ConnectionPool static method), 37
- setvirtualfields() (pydal.objects.Rows method), 43
- smart_adapt() (pydal.adapters.base.BaseAdapter method), 7
- smart_query() (in module pydal.helpers.methods), 29
- smart_query() (pydal.base.DAL method), 36
- sort() (pydal.objects.Rows method), 43
- SpatialLiteAdapter (class in pydal.adapters.sqlite), 25
- SQLALL (class in pydal.helpers.classes), 28
- SQLCallableList (class in pydal.helpers.classes), 28
- SQLCustomType (class in pydal.helpers.classes), 28
- SQLiteAdapter (class in pydal.adapters.sqlite), 25
- sqlsafe (pydal.base.DAL.Table attribute), 34
- sqlsafe (pydal.objects.Field attribute), 41
- sqlsafe (pydal.objects.Table attribute), 45
- sqlsafe_alias (pydal.base.DAL.Table attribute), 34
- sqlsafe_alias (pydal.objects.Table attribute), 45
- sqlsafe_field() (pydal.adapters.base.BaseAdapter method), 7
- sqlsafe_name (pydal.objects.Field attribute), 41
- sqlsafe_table() (pydal.adapters.base.BaseAdapter method), 7
- sqlsafe_table() (pydal.adapters.oracle.OracleAdapter method), 22
- ST_AS GEOJSON() (pydal.adapters.postgres.PostgreSQLAdapter method), 23
- ST_AS GEOJSON() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
- st_asgeojson() (pydal.objects.Expression method), 39
- ST_ASTEXT() (pydal.adapters.mssql.MSSQLAdapter method), 19
- ST_ASTEXT() (pydal.adapters.postgres.PostgreSQLAdapter method), 23
- ST_ASTEXT() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
- st_astext() (pydal.objects.Expression method), 39
- ST_CONTAINS() (pydal.adapters.mssql.MSSQLAdapter method), 19
- ST_CONTAINS() (pydal.adapters.postgres.PostgreSQLAdapter method), 23
- ST_CONTAINS() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
- st_contains() (pydal.objects.Expression method), 39
- ST_DISTANCE() (pydal.adapters.mssql.MSSQLAdapter method), 19
- ST_DISTANCE() (pydal.adapters.postgres.PostgreSQLAdapter method), 23
- ST_DISTANCE() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
- st_distance() (pydal.objects.Expression method), 39
- ST_DWITHIN() (pydal.adapters.postgres.PostgreSQLAdapter method), 23
- st_dwithin() (pydal.objects.Expression method), 39
- ST_EQUALS() (pydal.adapters.mssql.MSSQLAdapter method), 20
- ST_EQUALS() (pydal.adapters.postgres.PostgreSQLAdapter method), 23
- ST_EQUALS() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
- st_equals() (pydal.objects.Expression method), 39

- ST_INTERSECTS() (pydal.adapters.mssql.MSSQLAdapter method), 20
 - ST_INTERSECTS() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 - ST_INTERSECTS() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
 - st_intersects() (pydal.objects.Expression method), 40
 - ST_OVERLAPS() (pydal.adapters.mssql.MSSQLAdapter method), 20
 - ST_OVERLAPS() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 - ST_OVERLAPS() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
 - st_overlaps() (pydal.objects.Expression method), 40
 - ST_SIMPLIFY() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 - ST_SIMPLIFY() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
 - st_simplify() (pydal.objects.Expression method), 40
 - ST_SIMPLIFYPRESERVETOPOLOGY() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 - st_simplifypreservetopology() (pydal.objects.Expression method), 40
 - ST_TOUCHES() (pydal.adapters.mssql.MSSQLAdapter method), 20
 - ST_TOUCHES() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 - ST_TOUCHES() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
 - st_touches() (pydal.objects.Expression method), 40
 - ST_WITHIN() (pydal.adapters.mssql.MSSQLAdapter method), 20
 - ST_WITHIN() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 - ST_WITHIN() (pydal.adapters.sqlite.SpatialLiteAdapter method), 26
 - st_within() (pydal.objects.Expression method), 40
 - ST_X() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 - st_x() (pydal.objects.Expression method), 40
 - ST_Y() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
 - st_y() (pydal.objects.Expression method), 40
 - STARTSWITH() (pydal.adapters.base.BaseAdapter method), 5
 - STARTSWITH() (pydal.adapters.base.NoSQLAdapter method), 8
 - STARTSWITH() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - startswith() (pydal.helpers.classes.SQLCustomType method), 28
 - startswith() (pydal.objects.Expression method), 40
 - store() (pydal.objects.Field method), 41
 - SUB() (pydal.adapters.base.BaseAdapter method), 5
 - SUB() (pydal.adapters.base.NoSQLAdapter method), 8
 - SUB() (pydal.adapters.mongo.MongoDBAdapter method), 17
 - SUBSTRING() (pydal.adapters.base.BaseAdapter method), 5
 - SUBSTRING() (pydal.adapters.base.NoSQLAdapter method), 8
 - SUBSTRING() (pydal.adapters.firebird.FireBirdAdapter method), 11
 - SUBSTRING() (pydal.adapters.mssql.MSSQLAdapter method), 20
 - SUBSTRING() (pydal.adapters.mysql.MySQLAdapter method), 21
 - turn() (pydal.objects.Expression method), 40
 - support_distributed_transaction (pydal.adapters.base.BaseAdapter attribute), 7
 - support_distributed_transaction (pydal.adapters.firebird.FireBirdAdapter attribute), 11
 - support_distributed_transaction (pydal.adapters.mysql.MySQLAdapter attribute), 21
 - support_distributed_transaction (pydal.adapters.postgres.PostgreSQLAdapter attribute), 24
 - support_distributed_transaction (pydal.adapters.sapdb.SAPDBAdapter attribute), 25
 - SybaseAdapter (class in pydal.adapters.mssql), 20
- ## T
- T_SEP (pydal.adapters.base.BaseAdapter attribute), 5
 - T_SEP (pydal.adapters.mssql.MSSQLAdapter attribute), 20
 - T_SEP (pydal.adapters.mssql.VerticaAdapter attribute), 21
 - Table (class in pydal.objects), 44
 - table_alias() (pydal.adapters.base.BaseAdapter method), 7
 - tables (pydal.base.DAL attribute), 37
 - tables() (pydal.adapters.base.BaseAdapter method), 7
 - TeradataAdapter (class in pydal.adapters.teradata), 26
 - test_query (pydal.adapters.base.BaseAdapter attribute), 7
 - trigger_name() (pydal.adapters.base.BaseAdapter method), 7

- trigger_name() (pydal.adapters.firebird.FireBirdAdapter method), 11
- trigger_name() (pydal.adapters.oracle.OracleAdapter method), 22
- TRUE (pydal.adapters.base.BaseAdapter attribute), 5
- TRUE (pydal.adapters.mssql.MSSQLAdapter attribute), 20
- TRUE_exp (pydal.adapters.base.BaseAdapter attribute), 5
- truncate() (pydal.adapters.base.BaseAdapter method), 7
- truncate() (pydal.adapters.mongo.MongoDBAdapter method), 17
- truncate() (pydal.base.DAL.Table method), 34
- truncate() (pydal.objects.Table method), 45
- try_create_web2py_filesystem() (pydal.helpers.classes.DatabaseStoredFile static method), 27
- try_json() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
- types (pydal.adapters.base.BaseAdapter attribute), 7
- types (pydal.adapters.couchdb.CouchDBAdapter attribute), 9
- types (pydal.adapters.db2.DB2Adapter attribute), 10
- types (pydal.adapters.firebird.FireBirdAdapter attribute), 11
- types (pydal.adapters.imap.IMAPAdapter attribute), 14
- types (pydal.adapters.informix.InformixAdapter attribute), 15
- types (pydal.adapters.ingres.IngresAdapter attribute), 15
- types (pydal.adapters.ingres.IngresUnicodeAdapter attribute), 16
- types (pydal.adapters.mongo.MongoDBAdapter attribute), 17
- types (pydal.adapters.mssql.MSSQL2Adapter attribute), 18
- types (pydal.adapters.mssql.MSSQL3Adapter attribute), 18
- types (pydal.adapters.mssql.MSSQL3NAdapter attribute), 18
- types (pydal.adapters.mssql.MSSQL4Adapter attribute), 19
- types (pydal.adapters.mssql.MSSQL4NAdapter attribute), 19
- types (pydal.adapters.mssql.MSSQLAdapter attribute), 20
- types (pydal.adapters.mssql.MSSQLNAdapter attribute), 20
- types (pydal.adapters.mssql.SybaseAdapter attribute), 20
- types (pydal.adapters.mssql.VerticaAdapter attribute), 21
- types (pydal.adapters.mysql.MySQLAdapter attribute), 21
- types (pydal.adapters.oracle.OracleAdapter attribute), 22
- types (pydal.adapters.postgres.NewPostgreSQLAdapter attribute), 23
- types (pydal.adapters.postgres.PostgreSQLAdapter attribute), 24
- types (pydal.adapters.sapdb.SAPDBAdapter attribute), 25
- types (pydal.adapters.sqlite.SpatialLiteAdapter attribute), 26
- types (pydal.adapters.teradata.TeradataAdapter attribute), 26
- ## U
- update() (pydal.adapters.base.BaseAdapter method), 7
- update() (pydal.adapters.couchdb.CouchDBAdapter method), 9
- update() (pydal.adapters.imap.IMAPAdapter method), 14
- update() (pydal.adapters.mongo.MongoDBAdapter method), 17
- update() (pydal.base.DAL.Table method), 34
- update() (pydal.helpers.classes.BasicStorage method), 27
- update() (pydal.objects.LazySet method), 41
- update() (pydal.objects.Set method), 44
- update() (pydal.objects.Table method), 45
- update_naive() (pydal.objects.LazySet method), 41
- update_naive() (pydal.objects.Set method), 44
- update_or_insert() (pydal.base.DAL.Table method), 34
- update_or_insert() (pydal.objects.Table method), 45
- uploads_in_blob (pydal.adapters.base.BaseAdapter attribute), 7
- uploads_in_blob (pydal.adapters.couchdb.CouchDBAdapter attribute), 9
- uploads_in_blob (pydal.adapters.mongo.MongoDBAdapter attribute), 17
- UPPER() (pydal.adapters.base.BaseAdapter method), 5
- UPPER() (pydal.adapters.base.NoSQLAdapter method), 8
- upper() (pydal.objects.Expression method), 40
- uri (pydal.adapters.imap.IMAPAdapter attribute), 14
- use_common_filters() (in module pydal.helpers.methods), 30
- UseDatabaseStoredFile (class in pydal.helpers.classes), 28
- uuid() (pydal.base.DAL method), 37
- uuid2int() (in module pydal.helpers.methods), 30
- ## V
- validate() (pydal.objects.Field method), 41
- validate_and_insert() (pydal.base.DAL.Table method), 34
- validate_and_insert() (pydal.objects.Table method), 45
- validate_and_update() (pydal.base.DAL.Table method), 34
- validate_and_update() (pydal.objects.LazySet method), 42
- validate_and_update() (pydal.objects.Set method), 44
- validate_and_update() (pydal.objects.Table method), 45
- validate_and_update_or_insert() (pydal.base.DAL.Table method), 34

validate_and_update_or_insert() (pydal.objects.Table method), 45
validators (pydal.base.DAL attribute), 37
validators_method (pydal.base.DAL attribute), 37
values() (pydal.helpers.classes.BasicStorage method), 27
varquote() (pydal.adapters.base.BaseAdapter method), 7
varquote() (pydal.adapters.mssql.MSSQLAdapter method), 20
varquote() (pydal.adapters.mysql.MySQLAdapter method), 21
varquote() (pydal.adapters.postgres.PostgreSQLAdapter method), 24
varquote_aux() (in module pydal.helpers.methods), 30
VerticaAdapter (class in pydal.adapters.mssql), 20
Virtual (pydal.objects.Field attribute), 41
VirtualCommand (class in pydal.objects), 45

W

web2py_extract() (pydal.adapters.sqlite.SQLiteAdapter static method), 25
web2py_filesystems (pydal.helpers.classes.DatabaseStoredFile attribute), 27
web2py_regexp() (pydal.adapters.sqlite.SQLiteAdapter static method), 25
with_alias() (pydal.base.DAL.Table method), 34
with_alias() (pydal.objects.Expression method), 40
with_alias() (pydal.objects.Table method), 45
write() (pydal.helpers.classes.DatabaseStoredFile method), 27

X

xml() (pydal.objects.BasicRows method), 38
xorify() (in module pydal.helpers.methods), 30

Y

year() (pydal.objects.Expression method), 40