

---

**dql**  
*Release 0.5.22*

October 05, 2016



|          |                            |           |
|----------|----------------------------|-----------|
| <b>1</b> | <b>User Guide</b>          | <b>3</b>  |
| 1.1      | Getting Started . . . . .  | 3         |
| 1.2      | Queries . . . . .          | 4         |
| 1.3      | Data Types . . . . .       | 12        |
| 1.4      | Options . . . . .          | 13        |
| 1.5      | Developing . . . . .       | 14        |
| 1.6      | Changelog . . . . .        | 14        |
| <b>2</b> | <b>API Reference</b>       | <b>19</b> |
| 2.1      | dql package . . . . .      | 19        |
| <b>3</b> | <b>Indices and tables</b>  | <b>37</b> |
|          | <b>Python Module Index</b> | <b>39</b> |



A simple, SQL-ish language for DynamoDB

Code lives here: <https://github.com/stevearc/dql>



## 1.1 Getting Started

Install DQL with pip:

```
pip install dql
```

Since DQL uses `botocore` under the hood, the authentication mechanism is the same. You can use the `$HOME/.aws/credentials` file or set the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.

DQL uses `us-west-1` as the default region. You can change this by setting the `AWS_REGION` variable or passing it in on the command line:

```
$ dql -r us-east-1
```

You can begin using DQL immediately. Try creating a table and inserting some data

```
us-west-1> CREATE TABLE posts (username STRING HASH KEY,  
>                                postid NUMBER RANGE KEY,  
>                                ts NUMBER INDEX('ts-index'),  
>                                THROUGHPUT (5, 5));  
us-west-1> INSERT INTO posts (username, postid, ts, text)  
> VALUES ('steve', 1, 1386413481, 'Hey guys!'),  
>          ('steve', 2, 1386413516, 'Guys?'),  
>          ('drdice', 1, 1386413575, 'No one here');  
us-west-1> ls  
Name      Status  Read  Write  
posts    ACTIVE   5     5
```

You can query this data in a couple of ways. The first should look familiar

```
us-west-1> SELECT * FROM posts WHERE username = 'steve';
```

By default, `SELECT` statements are only allowed to perform index queries, not scan the table. You can enable scans by setting the `'allow_select_scan'` option (see *Options*) or replacing `SELECT` with `SCAN`:

```
us-west-1> SCAN * FROM posts WHERE postid = 2;
```

You can also perform updates to the data in a familiar way:

```
us-west-1> UPDATE posts SET text = 'Hay gusys!!!11' WHERE  
> username = 'steve' AND postid = 1;
```

The *Queries* section has detailed information about each type of query.

## 1.2 Queries

### 1.2.1 ALTER

#### Synopsis

```
ALTER TABLE tablename
  SET [INDEX index] THROUGHPUT throughput
ALTER TABLE tablename
  DROP INDEX index [IF EXISTS]
ALTER TABLE tablename
  CREATE GLOBAL [ALL|KEYS|INCLUDE] INDEX global_index [IF NOT EXISTS]
```

#### Examples

```
ALTER TABLE foobars SET THROUGHPUT (4, 8);
ALTER TABLE foobars SET THROUGHPUT (7, *);
ALTER TABLE foobars SET INDEX ts-index THROUGHPUT (5, *);
ALTER TABLE foobars SET INDEX ts-index THROUGHPUT (5, *);
ALTER TABLE foobars DROP INDEX ts-index;
ALTER TABLE foobars DROP INDEX ts-index IF EXISTS;
ALTER TABLE foobars CREATE GLOBAL INDEX ('ts-index', ts NUMBER, THROUGHPUT (5, 5));
ALTER TABLE foobars CREATE GLOBAL INDEX ('ts-index', ts NUMBER) IF NOT EXISTS;
```

#### Description

Alter changes the read/write throughput on a table. You may only decrease the throughput on a table four times per day (see the AWS docs on limits. If you wish to change one of the throughput values and not the other, pass in 0 or \* for the value you wish to remain unchanged.

#### Parameters

**tablename** The name of the table

**throughput** The read/write throughput in the form (*read\_throughput*, *write\_throughput*)

**index** The name of the global index

### 1.2.2 ANALYZE

#### Synopsis

```
ANALYZE query
```

#### Examples

```
ANALYZE SELECT * FROM foobars WHERE id = 'a';
ANALYZE INSERT INTO foobars (id, name) VALUES (1, 'dsa');
ANALYZE DELETE FROM foobars KEYS IN ('foo', 'bar'), ('baz', 'qux');
```



## Description

You can prefix any query that will read or write data with `ANALYZE` and after running the query it will print out how much capacity was consumed at every part of the query.

### 1.2.3 CREATE

#### Synopsis

```
CREATE TABLE
  [IF NOT EXISTS]
  tablename
  attributes
  [GLOBAL [ALL|KEYS|INCLUDE] INDEX global_index]
```

#### Examples

```
CREATE TABLE foobars (id STRING HASH KEY);
CREATE TABLE IF NOT EXISTS foobars (id STRING HASH KEY);
CREATE TABLE foobars (id STRING HASH KEY, foo BINARY RANGE KEY,
  THROUGHPUT (1, 1));
CREATE TABLE foobars (id STRING HASH KEY,
  foo BINARY RANGE KEY,
  ts NUMBER INDEX('ts-index'),
  views NUMBER INDEX('views-index'));
CREATE TABLE foobars (id STRING HASH KEY, bar STRING) GLOBAL INDEX
('bar-index', bar, THROUGHPUT (1, 1));
CREATE TABLE foobars (id STRING HASH KEY, baz NUMBER,
  THROUGHPUT (2, 2))
  GLOBAL INDEX ('bar-index', bar STRING, baz)
  GLOBAL INCLUDE INDEX ('baz-index', baz, ['bar'], THROUGHPUT (4, 2));
```

## Description

Create a new table. You must have exactly one hash key, zero or one range keys, and up to five local indexes and five global indexes. You must have a range key in order to have any local indexes.

#### Parameters

**IF NOT EXISTS** If present, do not through an exception if the table already exists.

**tablename** The name of the table that you want to alter

**attributes** A list of attribute declarations of the format *(name data type [key type])* The available data types are `STRING`, `NUMBER`, and `BINARY`. You will not need to specify any other type, because these fields are only used for index creation and it is (presently) impossible to index anything other than these three. The available key types are `HASH KEY`, `RANGE KEY`, and `[ALL|KEYS|INCLUDE] INDEX(name)`. At the end of the attribute list you may specify the `THROUGHPUT`, which is in the form of `(read_throughput, write_throughput)`. If throughput is not specified it will default to `(5, 5)`.

**global\_index** A global index for the table. You may provide up to 5. The format is *(name, hash key, [range key], [non-key attributes], [throughput])*. If the hash/range key is in the **attributes** declaration, you don't need to

supply a data type.. *non-key attributes* should only be provided if it is an `INCLUDE` index. If throughput is not specified it will default to `(5, 5)`.

## Schema Design at a Glance

When DynamoDB scales, it partitions based on the hash key. For this reason, all queries (not scans) *must* include the hash key in the `WHERE` clause (and optionally the range key or a local/global index). So keep that in mind as you design your schema.

The keypair formed by the hash key and range key is referred to as the ‘primary key’. If there is no range key, the primary key is just the hash key. The primary key is unique among items in the table. No two items may have the same primary key.

From a query standpoint, local indexes behave nearly the same as a range key. The main difference is that the hash key + range key pair doesn’t have to be unique.

Global indexes can be thought of as adding additional hash and range keys to the table. They allow you to query a table on a different hash key than the one defined on the table. Global indexes have throughput that is managed independently of the table they are on. Global index keys do not have a uniqueness constraint (there may be multiple items in the table that have the same hash and range key).

Read Amazon’s documentation for [Create Table](#) for more information.

## 1.2.4 DELETE

### Synopsis

```
DELETE FROM
  tablename
  [ KEYS IN primary_keys ]
  [ WHERE expression ]
  [ USING index ]
  [ THROTTLE throughput ]
```

### Examples

```
DELETE FROM foobars; -- This will delete all items in the table!
DELETE FROM foobars WHERE foo != 'bar' AND baz >= 3;
DELETE FROM foobars KEYS IN 'hkey1', 'hkey2' WHERE attribute_exists(foo);
DELETE FROM foobars KEYS IN ('hkey1', 'rkey1'), ('hkey2', 'rkey2');
DELETE FROM foobars WHERE (foo = 'bar' AND baz >= 3) USING baz-index;
```

### Description

#### Parameters

**tablename** The name of the table

**primary\_keys** List of the primary keys of the items to delete

**expression** See *SELECT* for details about the `WHERE` clause

**index** When the `WHERE` expression uses an indexed attribute, this allows you to manually specify which index name to use for the query. You will only need this if the constraints provided match more than one index.

**THROTTLE** Limit the amount of throughput this query can consume. This is a pair of values for (read\_throughput, write\_throughput). You can use a flat number or a percentage (e.g. 20 or 50%). Using \* means no limit (typically useless unless you have set a default throttle in the *Options*).

## Notes

Using the `KEYS IN` form is much more efficient because DQL will not have to perform a query first to get the primary keys.

## 1.2.5 DROP

### Synopsis

```
DROP TABLE
  [ IF EXISTS ]
  tablename
```

### Examples

```
DROP TABLE foobars;
DROP TABLE IF EXISTS foobars;
```

### Description

Deletes a table and all its items.

**Warning:** This action cannot be undone! Treat the same way you treat `rm -rf`

### Parameters

**IF EXISTS** If present, do not raise an exception if the table does not exist.

**tablename** The name of the table

## 1.2.6 DUMP

### Synopsis

```
DUMP SCHEMA [ tablename [, ...] ]
```

### Examples

```
DUMP SCHEMA;
DUMP SCHEMA foobars, widgets;
```

## Description

Print out the matching CREATE statements for your tables.

## Parameters

**tablename** The name of the table(s) whose schema you want to dump. If no tablename is present, it will dump all table schemas.

## 1.2.7 EXPLAIN

### Synopsis

```
EXPLAIN query
```

### Examples

```
EXPLAIN SELECT * FROM foobars WHERE id = 'a';
EXPLAIN INSERT INTO foobars (id, name) VALUES (1, 'dsa');
EXPLAIN DELETE FROM foobars KEYS IN ('foo', 'bar'), ('baz', 'qux');
```

## Description

This is a meta-query that will print out debug information. It will not make any actual requests except for possibly a DescribeTable if the primary key or indexes are needed to build the query. The output of the EXPLAIN will be the name of the DynamoDB Action(s) that will be called, and the parameters passed up in the request. You can use this to preview exactly what DQL will do before it happens.

## 1.2.8 INSERT

### Synopsis

```
INSERT INTO tablename
  attributes VALUES values
  [ THROTTLE throughput ]
INSERT INTO tablename
  items
  [ THROTTLE throughput ]
```

### Examples

```
INSERT INTO foobars (id) VALUES (1);
INSERT INTO foobars (id, bar) VALUES (1, 'hi'), (2, 'yo');
INSERT INTO foobars (id='foo', bar=10);
INSERT INTO foobars (id='foo'), (id='bar', baz=(1, 2, 3));
```

## Description

Insert data into a table

## Parameters

**tablename** The name of the table

**attributes** Comma-separated list of attribute names

**values** Comma-separated list of data to insert. The data is of the form *(var [, var]...)* and must contain the same number of items as the **attributes** parameter.

**items** Comma-separated key-value pairs to insert.

**THROTTLE** Limit the amount of throughput this query can consume. This is a pair of values for *(read\_throughput, write\_throughput)*. You can use a flat number or a percentage (e.g. 20 or 50%). Using *\** means no limit (typically useless unless you have set a default throttle in the *Options*).

See *Data Types* to find out how to represent the different data types of DynamoDB.

## 1.2.9 LOAD

### Synopsis

```
LOAD filename INTO tablename
[ THROTTLE throughput ]
```

### Examples

```
LOAD archive.p INTO mytable;
LOAD dump.json.gz INTO mytable;
```

## Description

Take the results of a `SELECT ... SAVE outfile` and insert all of the records into a table.

## Parameters

**filename** The file containing the records to upload

**tablename** The name of the table(s) to upload the records into

**THROTTLE** Limit the amount of throughput this query can consume. This is a pair of values for *(read\_throughput, write\_throughput)*. You can use a flat number or a percentage (e.g. 20 or 50%). Using *\** means no limit (typically useless unless you have set a default throttle in the *Options*).

## 1.2.10 SCAN

See *SELECT*. This is the exact same as a `SELECT` statement except it is always allowed to perform table scans. Note that this means a `SCAN` statement may still be doing an index query.

## 1.2.11 SELECT

### Synopsis

```

SELECT
  [ CONSISTENT ]
  attributes
FROM tablename
  [ KEYS IN primary_keys | WHERE expression ]
  [ USING index ]
  [ LIMIT limit ]
  [ SCAN LIMIT scan_limit ]
  [ ORDER BY field ]
  [ ASC | DESC ]
  [ THROTTLE throughput ]
  [ SAVE filename]

```

### Examples

```

SELECT * FROM foobars SAVE out.p;
SELECT * FROM foobars WHERE foo = 'bar';
SELECT count(*) FROM foobars WHERE foo = 'bar';
SELECT id, TIMESTAMP(updated) FROM foobars KEYS IN 'id1', 'id2';
SELECT * FROM foobars KEYS IN ('hkey', 'rkey1'), ('hkey', 'rkey2');
SELECT CONSISTENT * foobars WHERE foo = 'bar' AND baz >= 3;
SELECT * foobars WHERE foo = 'bar' AND attribute_exists(baz);
SELECT * foobars WHERE foo = 1 AND NOT (attribute_exists(bar) OR contains(baz, 'qux'));
SELECT 10 * (foo - bar) FROM foobars WHERE id = 'a' AND ts < 100 USING ts-index;
SELECT * FROM foobars WHERE foo = 'bar' LIMIT 50 DESC;
SELECT * FROM foobars THROTTLE (50%, *);

```

### Description

Query a table for items.

### Parameters

**CONSISTENT** If this is present, perform a strongly consistent read

**attributes** Comma-separated list of attributes to fetch or expressions. You can use the **TIMESTAMP** and **DATE** functions, as well as performing simple, arbitrarily nested arithmetic ( $foo + (bar - 3) / 100$ ). **SELECT** \* is a special case meaning ‘all attributes’. **SELECT** count(\*) is a special case that will return the number of results, rather than the results themselves.

**tablename** The name of the table

**index** When the **WHERE** expression uses an indexed attribute, this allows you to manually specify which index name to use for the query. You will only need this if the constraints provided match more than one index.

**limit** The maximum number of items to return.

**scan\_limit** The maximum number of items for DynamoDB to scan (not necessarily the number of matching items returned).

**ORDER BY** Sort the results by a field.

**Warning:** Using ORDER BY with LIMIT may produce unexpected results. If you use ORDER BY on the range key of the index you are querying on, it will work as expected. Otherwise, DQL will fetch the number of results specified by the LIMIT and then sort them.

**ASC | DESC** Sort the results in ASCending (the default) or DESCending order.

**THROTTLE** Limit the amount of throughput this query can consume. This is a pair of values for (read\_throughput, write\_throughput). You can use a flat number or a percentage (e.g. 20 or 50%). Using \* means no limit (typically useless unless you have set a default throttle in the *Options*).

**SAVE** Save the results to a file. By default the items will be encoded with pickle, but the '.json' and '.csv' extensions will use the proper format. You may also append a '.gz' or '.gzip' afterwards to gzip the results. Note that the JSON and CSV formats will be lossy because they cannot properly encode some data structures, such as sets.

## Where Clause

If provided, the SELECT operation will use these constraints as the `KeyConditionExpression` if possible, and if not (or if there are constraints left over), the `FilterExpression`. All query syntax is pulled directly from the AWS docs: <http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/QueryAndScan.html>

In general, you may use any syntax mentioned in the docs, but you don't need to worry about reserved words or passing in data as variables like `:var1`. DQL will handle that for you.

## Notes

When using the `KEYS IN` form, DQL will perform a batch get instead of a table query. See the [AWS docs](#) for more information on query parameters.

## 1.2.12 UPDATE

### Synopsis

```
UPDATE tablename
  update_expression
  [ KEYS IN primary_keys ]
  [ WHERE expression ]
  [ USING index ]
  [ RETURNS (NONE | ( ALL | UPDATED) (NEW | OLD)) ]
  [ THROTTLE throughput ]
```

### Examples

```
UPDATE foobars SET foo = 'a';
UPDATE foobars SET foo = 'a', bar = bar + 4 WHERE id = 1 AND foo = 'b';
UPDATE foobars SET foo = if_not_exists(foo, 'a') RETURNS ALL NEW;
UPDATE foobars SET foo = list_append(foo, 'a') WHERE size(foo) < 3;
UPDATE foobars ADD foo 1, bar 4;
UPDATE foobars ADD fooset (1, 2);
UPDATE foobars REMOVE old_attribute;
UPDATE foobars DELETE fooset (1, 2);
```

## Description

Update items in a table

## Parameters

**tablename** The name of the table

**RETURNS** Return the items that were operated on. Default is RETURNS NONE. See the Amazon docs for [UpdateItem](#) for more detail.

**THROTTLE** Limit the amount of throughput this query can consume. This is a pair of values for (read\_throughput, write\_throughput). You can use a flat number or a percentage (e.g. 20 or 50%). Using \* means no limit (typically useless unless you have set a default throttle in the *Options*).

## Update expression

All update syntax is pulled directly from the AWS docs:

<http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.Modifying.html>

In general, you may use any syntax mentioned in the docs, but you don't need to worry about reserved words or passing in data as variables like :var1. DQL will handle that for you.

## WHERE and KEYS IN

Both of these expressions are the same as in *SELECT*. Note that using KEYS IN is more efficient because DQL can perform the writes without doing a query first.

## 1.3 Data Types

Below is a list of all DynamoDB data types and examples of how to represent those types in queries.

|            |                  |
|------------|------------------|
| NUMBER     | 123              |
| STRING     | 'asdf' or "asdf" |
| BINARY     | b'datadatadata'  |
| NUMBER SET | (1, 2, 3)        |
| STRING SET | ('a', 'b', 'c')  |
| BINARY SET | (b'a', b'c')     |
| BOOL       | TRUE or FALSE    |
| LIST       | [1, 2, 3]        |
| MAP        | {'a': 1}         |

### 1.3.1 Timestamps

DQL has some limited support for timestamp types. These will all be converted to Unix timestamps under the hood.

- `TIMESTAMP ('2015-12-3 13:32:00')` or `TS ()` - parses the timestamp in your local timezone
- `UTCTIMESTAMP ('2015-12-3 13:32:00')` or `UTCTS ()` - parses the timestamp as UTC
- `NOW ()` - Returns the current timestamp



You can also add/subtract intervals from a timestamp

- `NOW() - INTERVAL("1 day")`
- `NOW() + INTERVAL "1y 2w -5 minutes"`

You can wrap any of these with `MS()` to convert the result into milliseconds instead of seconds.

`MS(NOW() + INTERVAL '2 days')`

Below is a list of all keywords that you can use for intervals

- `year, years, y`
- `month, months`
- `week, weeks, w`
- `day, days, d`
- `hour, hours, h`
- `minute, minutes, m`
- `second, seconds, s`
- `millisecond, milliseconds, ms`
- `microsecond, microseconds, us`

## 1.4 Options

The following are options you can set for DQL. Options are set with `opt <option> <value>`, and you can see the current option value with `opt <option>`

|                                |  |   |
|--------------------------------|--|---|
| <code>width</code>             | <code>int / auto</code>                | Number of characters wide to format the display   |
| <code>pagesize</code>          | <code>int / auto</code>                | Number of results to get per page for queries     |
| <code>display</code>           | <code>less / stdout</code>             | The reader used to view query results             |
| <code>format</code>            | <code>smart / column / expanded</code> | Display format for query results                  |
| <code>allow_select_scan</code> | <code>bool</code>                      | If True, SELECT statement can perform table scans |

### 1.4.1 Throttling

DQL also allows you to be careful how much throughput you consume with your queries. Use the `throttle` command to set persistent limits on all or some of your tables/indexes. Some examples:

```
# Set the total allowed throughput across all tables
> throttle 1000 100
# Set the default allowed throughput per-table/index
> throttle default 40% 20%
# Set the allowed throughput on a table
> throttle mytable 10 10
# Set the allowed throughput on a global index
> throttle mytable myindex 40 6
```

See `help throttle` for more details, and use `unthrottle` to remove a throttle. You can also set throttles on a per-query basis with the `THROTTLE` keyword.

## 1.5 Developing

To get started developing dql, run the following command:

```
wget https://raw.githubusercontent.com/stevearc/devbox/master/devbox/unbox.py && \
python unbox.py git@github.com:stevearc/dql
```

This will clone the repository and install the package into a virtualenv

### 1.5.1 Running Tests

The command to run tests is `python setup.py nosetests`. Some of these tests require [DynamoDB Local](#). There is a nose plugin that will download and run the DynamoDB Local service during the tests. It requires the java 6/7 runtime, so make sure you have that installed.

## 1.6 Changelog

### 1.6.1 0.5.22

- Bug fix: Can now run any CLI command using `-c "command"`

### 1.6.2 0.5.21

- Bug fix: Crash fix when resizing terminal with ‘watch’ command active
- ‘Watch’ columns will dynamically resize to fit terminal width

### 1.6.3 0.5.20

- Bug fix: When saving to JSON floats are no longer cast to ints
- Bug fix: Reserved words are correctly substituted when using WHERE ... IN

### 1.6.4 0.5.19

- Locked in the version of `pyarsing` after 2.1.5 broke compatibility again.

### 1.6.5 0.5.18

- Bug fix: Correct name substitution/selection logic
- Swapped out `bin/run_dql.py` for `bin/install.py`. Similar concept, better execution.

### 1.6.6 0.5.17

- Bug fix: Can't display Binary data

### 1.6.7 0.5.16

- Bug fix: Can't use boolean values in update statements

### 1.6.8 0.5.15

- Gracefully handle missing imports on Windows

### 1.6.9 0.5.14

- Missing curses library won't cause ImportError

### 1.6.10 0.5.13

- Fix bug where query would sometimes display 'No Results' even when results were found.

### 1.6.11 0.5.12

- Differentiate LIMIT and SCAN LIMIT
- Options and query syntax for `throttling` the consumed throughput
- Crash fixes and other small robustness improvements

### 1.6.12 0.5.11

- SELECT <attributes> can now use full expressions

### 1.6.13 0.5.10

- LOAD command to insert records from a file created with `SELECT ... SAVE`
- Default SAVE format is pickle
- SAVE command can gzip the file

### 1.6.14 0.5.9

- Don't print results to console when saving to a file
- 'auto' pagesize to adapt to terminal height
- When selecting specific attributes with KEYS IN only those attributes are fetched
- ORDER BY queries spanning multiple pages no longer stuck on first page
- Column formatter fits column widths more intelligently
- Smart formatter is smarter about switching to Expanded mode

### 1.6.15 0.5.8

- Tab completion for Mac OS X

### 1.6.16 0.5.7

- `run_dql.py` locks in a version
- Display output auto-detects terminal width

### 1.6.17 0.5.6

- Format option saves properly
- WHERE expressions can compare fields to fields (e.g. `WHERE foo > bar`)
- Always perform `batch_get` after querying/scanning an index that doesn't project all attributes

### 1.6.18 0.5.5

- General bug fixes
- Self contained `run_dql.py` script

### 1.6.19 0.5.4

- Fixes for `watch display`
- `SELECT` can save the results to a file

### 1.6.20 0.5.3

- `ALTER` commands can specify `IF (NOT) EXISTS`
- New `watch` command to monitor table consumed capacities
- `SELECT` can fetch attributes that aren't projected onto the queried index
- `SELECT` can `ORDER BY` non-range-key attributes

### 1.6.21 0.5.2

- `EXPLAIN <query>` will print out the DynamoDB calls that will be made when you run the query
- `ANALYZE <query>` will run the query and print out consumed capacity information

### 1.6.22 0.5.1

- Pretty-format non-item query return values (such as count)
- Disable passing AWS credentials on the command line

### 1.6.23 0.5.0

- **Breakage:** New syntax for SELECT, SCAN, UPDATE, DELETE
- **Breakage:** Removed COUNT query (now `SELECT count (*)`)
- **Breakage:** Removed the ability to embed python in queries
- New alternative syntax for INSERT
- ALTER can create and drop global indexes
- Queries and updates now use the most recent DynamoDB expressions API
- Unified options in CLI under the `opt` command

### 1.6.24 0.4.1

- Update to maintain compatibility with new versions of botocore and dynamo3
- Improving CloudWatch support (which is used to get consumed table capacity)

### 1.6.25 0.4.0

- **Breakage:** Dropping support for python 3.2 due to lack of botocore support
- Feature: Support for JSON data types

### 1.6.26 0.3.2

- Bug fix: Allow '.' in table names of DUMP SCHEMA command
- Bug fix: Passing a port argument to local connection doesn't crash
- Bug fix: Prompt says 'localhost' when connected to DynamoDB local

### 1.6.27 0.3.1

- Bug fix: Allow '.' in table names

### 1.6.28 0.3.0

- Feature: SELECT and COUNT can have FILTER clause
- Feature: FILTER clause may OR constraints together

### 1.6.29 0.2.1

- Bug fix: Crash when printing 'COUNT' queries

### 1.6.30 0.2.0

- Feature: Python 3 support

## **1.6.31 0.1.0**

- First public release

## 2.1 dql package

### 2.1.1 Subpackages

#### dql.expressions package

##### Submodules

**dql.expressions.base module** Common utilities for all expressions

**class** `dql.expressions.base.Expression`

Bases: `object`

Base class for all expressions and expression fragments

**build** (*visitor*)

Build string expression, using the visitor to encode values

**class** `dql.expressions.base.Field` (*field*)

Bases: `dql.expressions.base.Expression`

Wrapper for a field in an expression

**build** (*visitor*)

**evaluate** (*item*)

Pull the field off the item

**class** `dql.expressions.base.Value` (*val*)

Bases: `dql.expressions.base.Expression`

Wrapper for a value in an expression

**build** (*visitor*)

**evaluate** (*item*)

Values evaluate to themselves regardless of the item

**dql.expressions.constraint module** Constraint expressions for selecting

**class** `dql.expressions.constraint.BetweenConstraint` (*field, low, high*)

Bases: `dql.expressions.constraint.ConstraintExpression`

Constraint expression for BETWEEN low AND high

**build** (*visitor*)

**classmethod from\_clause** (*clause*)

Factory method

**range\_field**

**class** `dql.expressions.constraint.Conjunction` (*pieces*)

Bases: `dql.expressions.constraint.ConstraintExpression`

Use AND and OR to join 2 or more expressions

**classmethod and\_** (*constraints*)

Factory for a group AND

**build** (*visitor*)

**classmethod from\_clause** (*clause*)

Factory method

**classmethod or\_** (*constraints*)

Factory for a group OR

**possible\_hash\_fields** ()

**possible\_range\_fields** ()

**remove\_index** (*index*)

This one takes some explanation. When we do a query with a WHERE statement, it may end up being a scan and it may end up being a query. If it is a query, we need to remove the hash and range key constraints from the expression and return that as the `query_constraints`. The remaining constraints, if any, are returned as the `filter_constraints`.

**class** `dql.expressions.constraint.ConstraintExpression`

Bases: `dql.expressions.base.Expression`

Base class and entry point for constraint expressions

e.g. WHERE foo = 1

**build** (*visitor*)

**classmethod from\_clause** (*clause*)

Factory method

**classmethod from\_where** (*where*)

Factory method for creating the top-level expression

**hash\_field**

The field of the hash key this expression can select, if any

**possible\_hash\_fields** ()

Set of field names this expression could possibly be selecting for the hash key of a query.

Hash keys must be exactly specified with “hash\_key = value”

**possible\_range\_fields** ()

Set of field names this expression could possibly be selecting for the range key of a query.

Range keys can use operations such as <, >, <=, >=, begins\_with, etc.

**range\_field**

The field of the range key this expression can select, if any



**class** `dql.expressions.constraint.FunctionConstraint` (*fn\_name, field, operand=None*)  
 Bases: `dql.expressions.constraint.ConstraintExpression`

Constraint for function expressions e.g. `attribute_exists(field)`

**build** (*visitor*)

**classmethod from\_clause** (*clause*)

Factory method

**range\_field**

**class** `dql.expressions.constraint.InConstraint` (*field, values*)  
 Bases: `dql.expressions.constraint.ConstraintExpression`

Constraint expression for membership in a set

**build** (*visitor*)

**classmethod from\_clause** (*clause*)

Factory method

**class** `dql.expressions.constraint.Invert` (*constraint*)  
 Bases: `dql.expressions.constraint.ConstraintExpression`

Invert another constraint expression with NOT

**build** (*visitor*)

**class** `dql.expressions.constraint.OperatorConstraint` (*field, operator, value*)  
 Bases: `dql.expressions.constraint.ConstraintExpression`

Constraint expression for operations, e.g. `foo = 4`

**build** (*visitor*)

**classmethod from\_clause** (*clause*)

Factory method

**hash\_field**

**range\_field**

**remove\_index** (*index*)

See `remove_index()`.

This is called if the entire WHERE expression is just a “`hash_key = value`”. In this case, the `query_constraints` are just this constraint, and there are no `filter_constraints`.

**class** `dql.expressions.constraint.SizeConstraint` (*field, operator, value*)  
 Bases: `dql.expressions.constraint.ConstraintExpression`

Constraint expression for `size()` function

**build** (*visitor*)

**classmethod from\_clause** (*clause*)

Factory method

**class** `dql.expressions.constraint.TypeConstraint` (*fn\_name, field, operand=None*)  
 Bases: `dql.expressions.constraint.FunctionConstraint`

Constraint for `attribute_type()` function

**classmethod from\_clause** (*clause*)

Factory method

`dql.expressions.constraint.field_or_value` (*clause*)  
For a clause that could be a field or value, create the right one and return it

**dql.expressions.selection module** Selection expressions

**class** `dql.expressions.selection.AttributeSelection` (*expr1, op=None, expr2=None*)

Bases: `dql.expressions.base.Expression`

A tree of select expressions

**build** (*visitor*)

**evaluate** (*item*)

Evaluate this expression for a particular item

**classmethod from\_statement** (*statement*)

Factory for creating a Attribute expression

**class** `dql.expressions.selection.NamedExpression` (*expr, alias=None*)

Bases: `dql.expressions.base.Expression`

Wrapper around AttributeSelection that holds the alias (if any)

**build** (*visitor*)

**classmethod from\_statement** (*statement*)

Parse the selection expression and alias from a statement

**key**

The name that this will occupy in the final result dict

**populate** (*item, ret, sanitize*)

Evaluate the child expression and put result into return value

**class** `dql.expressions.selection.NowFunction` (*utc*)

Bases: `dql.expressions.selection.SelectFunction`

Function to grab the current time

**build** (*visitor*)

**evaluate** (*item*)

**classmethod from\_statement** (*statement*)

**class** `dql.expressions.selection.SelectFunction`

Bases: `dql.expressions.base.Expression`

Base class for special select functions

**evaluate** (*item*)

Evaluate this expression for a particular item

**classmethod from\_statement** (*statement*)

Create a selection function from a statement

**class** `dql.expressions.selection.SelectionExpression` (*expressions, is\_count=False*)

Bases: `dql.expressions.base.Expression`

Entry point for Selection expressions

**all\_fields**

A set of all fields that are required by this statement

**all\_keys**

The keys, in order, that are selected by the statement

**build** (*visitor*)

**convert** (*item*, *sanitize=False*)

Convert an item into an OrderedDict with the selected fields

**classmethod from\_selection** (*selection*)

Factory for creating a Selection expression

**class** `dql.expressions.selection.TimestampFunction` (*expr*, *utc*)

Bases: `dql.expressions.selection.SelectFunction`

Function that parses a field or literal as a datetime

**build** (*visitor*)

**evaluate** (*item*)

**classmethod from\_statement** (*statement*)

`dql.expressions.selection.add` (*a*, *b*)

Add two values, ignoring None

`dql.expressions.selection.div` (*a*, *b*)

Divide two values, ignoring None

`dql.expressions.selection.mul` (*a*, *b*)

Multiply two values, ignoring None

`dql.expressions.selection.parse_expression` (*clause*)

For a clause that could be a field, value, or expression

`dql.expressions.selection.sub` (*a*, *b*)

Subtract two values, ignoring None

**dql.expressions.update module** Update expressions

**class** `dql.expressions.update.FieldValue` (*field*, *value*)

Bases: `dql.expressions.base.Expression`

A field-value pair used in an expression

**build** (*visitor*)

**classmethod from\_clause** (*clause*)

Factory method

**class** `dql.expressions.update.SetFunction` (*fn\_name*, *value1*, *value2*)

Bases: `dql.expressions.base.Expression`

Expression fragment for a function used in a SET statement

e.g. `if_not_exists(field, value)`

**build** (*visitor*)

**classmethod from\_clause** (*clause*)

Factory method

**class** `dql.expressions.update.UpdateAdd` (*updates*)

Bases: `dql.expressions.base.Expression`

Expression fragment for an ADD statement

**build** (*visitor*)

**classmethod from\_clause** (*clause*)  
Factory method

**class** `dql.expressions.update.UpdateDelete` (*updates*)  
Bases: `dql.expressions.base.Expression`

Expression fragment for a DELETE statement

**build** (*visitor*)

**classmethod from\_clause** (*clause*)  
Factory method

**class** `dql.expressions.update.UpdateExpression` (*expressions*)  
Bases: `dql.expressions.base.Expression`

Entry point for Update expressions

**build** (*visitor*)

**classmethod from\_update** (*update*)  
Factory for creating an Update expression

**class** `dql.expressions.update.UpdateRemove` (*fields*)  
Bases: `dql.expressions.base.Expression`

Expression fragment for a REMOVE statement

**build** (*visitor*)

**classmethod from\_clause** (*clause*)  
Factory method

**class** `dql.expressions.update.UpdateSetMany` (*updates*)  
Bases: `dql.expressions.base.Expression`

Expression fragment for multiple set statements

**build** (*visitor*)

**classmethod from\_clause** (*clause*)  
Factory method

**class** `dql.expressions.update.UpdateSetOne` (*field, value1, op=None, value2=None*)  
Bases: `dql.expressions.base.Expression`

Expression fragment for a single SET statement

**build** (*visitor*)

**classmethod from\_clause** (*clause*)  
Factory method

`dql.expressions.update.field_or_value` (*clause*)  
For a clause that could be a field or value, create the right one and return it

**dql.expressions.visitor module** Visitor classes for traversing expressions

**class** `dql.expressions.visitor.DummyVisitor` (*reserved\_words=None*)  
Bases: `dql.expressions.visitor.Visitor`

No-op visitor for testing

**get\_field** (*field*)  
No-op

**get\_value** (*value*)  
No-op

**class** `dql.expressions.visitor.Visitor` (*reserved\_words=None*)

Bases: `object`

Visitor that replaces field names and values with encoded versions

**Parameters** `reserved_words` : set, optional

Set of (uppercase) words that are reserved by DynamoDB. These are used when encoding field names. If None, will default to encoding all fields.

**attribute\_names**  
Dict of encoded field names to original names

**expression\_values**  
Dict of encoded variable names to the variables

**get\_field** (*field*)  
Get the safe representation of a field name  
For example, since 'order' is reserved, it would encode it as '#f1'

**get\_value** (*value*)  
Replace variable names with placeholders (e.g. ':v1')

## Module contents

Tools for parsing and handling expressions

## dql.grammar package

### Submodules

**dql.grammar.common module** Common use grammars

`dql.grammar.common.function` (*name*, *\*args*, *\*\*kwargs*)  
Construct a parser for a standard function format

`dql.grammar.common.make_interval` (*long\_name*, *short\_name*)  
Create an interval segment

`dql.grammar.common.quoted` (*body*)  
Quote an item with ' or "

`dql.grammar.common.upkey` (*name*)  
Shortcut for creating an uppercase keyword

**dql.grammar.query module** Grammars for parsing query strings

`dql.grammar.query.create_keys_in` ()  
Create a grammar for the 'KEYS IN' clause used for queries

`dql.grammar.query.create_query_constraint` ()  
Create a constraint for a query WHERE clause

`dql.grammar.query.create_selection()`  
Create a selection expression

`dql.grammar.query.create_where()`  
Create a grammar for the 'where' clause used by 'select'

`dql.grammar.query.select_functions(expr)`  
Create the function expressions for selection

## Module contents

DQL language parser

`dql.grammar.create_alter()`  
Create the grammar for the 'alter' statement

`dql.grammar.create_create()`  
Create the grammar for the 'create' statement

`dql.grammar.create_delete()`  
Create the grammar for the 'delete' statement

`dql.grammar.create_drop()`  
Create the grammar for the 'drop' statement

`dql.grammar.create_dump()`  
Create the grammar for the 'dump' statement

`dql.grammar.create_insert()`  
Create the grammar for the 'insert' statement

`dql.grammar.create_load()`  
Create the grammar for the 'load' statement

`dql.grammar.create_parser()`  
Create the language parser

`dql.grammar.create_scan()`  
Create the grammar for the 'scan' statement

`dql.grammar.create_select()`  
Create the grammar for the 'select' statement

`dql.grammar.create_throttle()`  
Create a THROTTLE statement

`dql.grammar.create_throughput(variable=primitive)`  
Create a throughput specification

`dql.grammar.create_update()`  
Create the grammar for the 'update' statement

## 2.1.2 Submodules

### dql.cli module

Interactive DQL client

**class** `dql.cli.DQLClient` (*completekey='tab', stdin=None, stdout=None*)

Bases: `cmd.Cmd`

Interactive commandline interface.

### Attributes

|                      |   |
|----------------------|---|
| <code>running</code> | (bool) True while session is active, False after quitting |
| <code>engine</code>  | ( <code>dql.engine.FragmentEngine</code> )                |

**caution\_callback** (*action*)

Prompt user for manual continue when doing write operation on all items in a table

**complete\_file** (*text, line, \*\_*)

Autocomplete DQL file lookup

**complete\_ls** (*text, \*\_*)

Autocomplete for ls

**complete\_opt** (*text, line, begidx, endidx*)

Autocomplete for options

**complete\_opt\_allow\_select\_scan** (*text, \*\_*)

Autocomplete for allow\_select\_scan option

**complete\_opt\_display** (*text, \*\_*)

Autocomplete for display option

**complete\_opt\_format** (*text, \*\_*)

Autocomplete for format option

**complete\_opt\_pagesize** (*\*\_*)

Autocomplete for pagesize option

**complete\_opt\_width** (*\*\_*)

Autocomplete for width option

**complete\_use** (*text, \*\_*)

Autocomplete for use

**complete\_watch** (*text, \*\_*)

Autocomplete for watch

**completedefault** (*text, line, \*\_*)

Autocomplete table names in queries

**conf** = None

**default** (*command*)

**display** = None

**do\_EOF** (*arglist*)

Exit

**do\_exit** (*arglist*)

Exit

**do\_file** (*arglist*)

Read and execute a .dql file

**do\_local** (*arglist*)  
Connect to a local DynamoDB instance. Use 'local off' to disable.  
> local > local host=localhost port=8001 > local off

**do\_ls** (*arglist*)  
List all tables or print details of one table

**do\_opt** (*arglist*)  
Get and set options

**do\_shell** (*arglist*)  
Run a shell command

**do\_throttle** (*arglist*)  
Set the allowed consumed throughput for DQL.  
  
# Set the total allowed throughput across all tables > throttle 1000 100 # Set the default allowed throughput per-table/index > throttle default 40% 20% # Set the allowed throughput on a table > throttle mytable 10 10 # Set the allowed throughput on a global index > throttle mytable myindex 40 6  
  
see also: unthrottle

**do\_unthrottle** (*arglist*)  
Remove the throughput limits for DQL that were set with 'throttle'  
  
# Remove all limits > unthrottle # Remove the limit on total allowed throughput > unthrottle total # Remove the default limit > unthrottle default # Remove the limit on a table > unthrottle mytable # Remove the limit on a global index > unthrottle mytable myindex

**do\_use** (*arglist*)  
Switch the AWS region  
  
> use us-west-1 > use us-east-1

**do\_watch** (*arglist*)  
Watch Dynamo tables consumed capacity

**emptyline** ()

**engine = None**

**formatter = None**

**getopt\_default** (*option*)  
Default method to get an option

**getopt\_display** ()  
Get value for display option

**getopt\_format** ()  
Get value for format option

**help\_alter** ()  
Print the help text for ALTER

**help\_analyze** ()  
Print the help text for ALTER

**help\_create** ()  
Print the help text for CREATE

**help\_delete** ()  
Print the help text for DELETE



---

**help\_drop** ()  
Print the help text for DROP

**help\_dump** ()  
Print the help text for DUMP

**help\_explain** ()  
Print the help text for EXPLAIN

**help\_help** ()  
Print the help text for help

**help\_insert** ()  
Print the help text for INSERT

**help\_load** ()  
Print the help text for LOAD

**help\_opt** ()  
Print the help text for options

**help\_scan** ()  
Print the help text for SCAN

**help\_select** ()  
Print the help text for SELECT

**help\_update** ()  
Print the help text for UPDATE

**initialize** (*region='us-west-1', host=None, port=8000, config\_dir=None, session=None*)  
Set up the repl for execution.

**load\_config** ()  
Load your configuration settings from a file

**opt\_allow\_select\_scan** (*allow*)  
Set option allow\_select\_scan

**opt\_display** (*display*)  
Set value for display option

**opt\_format** (*format*)  
Set value for format option

**opt\_pagesize** (*pagesize*)  
Get or set the page size of the query output

**opt\_width** (*width*)  
Set width of output ('auto' will auto-detect terminal width)

**postcmd** (*stop, line*)

**run\_command** (*command*)  
Run a command passed in from the command line with -c

**running = False**

**save\_config** ()  
Save the conf file

**session = None**

**start** ()  
Start running the interactive session (blocking)

**update\_prompt** ()  
Update the prompt

`dql.cli.get_enum_key` (*key, choices*)  
Get an enum by prefix or equality

`dql.cli.indent` (*string, prefix=' '*)  
Indent a paragraph of text

`dql.cli.prompt` (*msg, default=<object object>, validate=None*)  
Prompt user for input

`dql.cli.promptyn` (*msg, default=None*)  
Display a blocking prompt until the user confirms

`dql.cli.repl_command` (*fn*)  
Decorator for cmd methods

Parses arguments from the arg string and passes them to the method as *\*args* and *\*\*kwargs*.

## dql.engine module

Execution engine

**class** `dql.engine.Engine` (*connection=None*)  
Bases: `object`

DQL execution engine

**Parameters** `connection`: `DynamoDBConnection`, optional

If not present, you will need to call `Engine.connect()`

### Attributes

|                               |   |
|-------------------------------|---|
| <code>caution_callback</code> | (callable, optional) Called to prompt user when a potentially dangerous action is about to occur. |
|-------------------------------|---|

**cloudwatch\_connection**

Lazy create a connection to cloudwatch

**connect** (*\*args, \*\*kwargs*)

Proxy to `DynamoDBConnection.connect`.

**connection**

Get the dynamo connection

**describe** (*tablename, refresh=False, metrics=False, require=False*)

Get the `TableMeta` for a table

**describe\_all** (*refresh=True*)

Describe all tables in the connected region

**execute** (*commands, pretty\_format=False*)

Parse and run a DQL string

**Parameters** `commands`: str

The DQL command string

`pretty_format`: bool

Pretty-format the return value. (e.g. 4 -> 'Updated 4 items')

**get\_capacity** (*tablename, index\_name=None*)  
Get the consumed read/write capacity

**region**  
Get the connected dynamo region or host

**exception** `dql.engine.ExplainSignal`  
Bases: `exceptions.Exception`  
Thrown to stop a query when we're doing an EXPLAIN

**class** `dql.engine.FragmentEngine` (*connection=None*)  
Bases: `dql.engine.Engine`  
A DQL execution engine that can handle query fragments

**execute** (*fragment, pretty\_format=True*)  
Run or aggregate a query fragment

Concat the fragment to any stored fragments. If they form a complete query, run it and return the result. If not, store them and return None.

**partial**  
True if there is a partial query stored

**pformat\_exc** (*exc*)  
Format an exception message for the last query's parse error

**reset** ()  
Clear any query fragments from the engine

`dql.engine.add_query_kwargs` (*kwargs, visitor, constraints, index*)  
Construct KeyConditionExpression and FilterExpression

`dql.engine.default` (*value*)  
Default encoder for JSON

`dql.engine.iter_insert_items` (*tree*)  
Iterate over the items to insert from an INSERT statement

## dql.help module

Help text for the CLI

## dql.models module

Data containers

**class** `dql.models.GlobalIndex` (*name, index\_type, status, hash\_key, range\_key, read\_throughput, write\_throughput, size, includes=None, description=None*)

Bases: `object`

Container for global index data

**classmethod** `from_description` (*description, attrs*)  
Create an object from a dynamo3 response

**pformat** (*consumed\_capacity=None*)  
Pretty format for insertion into table pformat

**schema**

The DQL fragment for constructing this index

**class** `dql.models.IndexField` (*name, data\_type, index\_type, index\_name, includes=None*)

Bases: `dql.models.TableField`

A TableField that is also part of a Local Secondary Index

**schema**

The DQL syntax for creating this item

**class** `dql.models.QueryIndex` (*name, is\_global, hash\_key, range\_key, attributes=None*)

Bases: `object`

A representation of global/local indexes that used during query building.

When building queries, we need to detect if the constraints are sufficient to perform a query or if they can only do a scan. This simple container class was specifically create to make that logic simpler.

**classmethod** `from_table_index` (*table, index*)

Factory method

**projects\_all\_attributes** (*attrs*)

Return True if the index projects all the attributes

**scannable**

Only global indexes can be scanned

**class** `dql.models.TableField` (*name, data\_type, key\_type=None*)

Bases: `object`

A DynamoDB table attribute

**Parameters** **name** : str

**data\_type** : str

The type of object (e.g. 'STRING', 'NUMBER', etc)

**key\_type** : str, optional

The type of key (e.g. 'RANGE', 'HASH', 'INDEX')

**index\_name** : str, optional

If the key\_type is 'INDEX', this will be the name of the index that uses the field as a range key.

**schema**

The DQL syntax for creating this item

**to\_index** (*index\_type, index\_name, includes=None*)

Create an index field from this field

**class** `dql.models.TableMeta` (*table, attrs, global\_indexes, read\_throughput, write\_throughput, decreases\_today, size*)

Bases: `object`

Container for table metadata

**Parameters** **name** : str

**status** : str

**attrs** : dict

Mapping of attribute name to `TableField`

**global\_indexes** : dict

Mapping of hash key to *GlobalIndex*

**read\_throughput** : int

**write\_throughput** : int

**decreases\_today** : int

**size** : int

Size of the table in bytes

**item\_count** : int

Number of items in the table

**classmethod from\_description** (*table*)

Factory method that uses the dynamo3 'describe' return value

**get\_index** (*index\_name*)

Get a specific index by name

**get\_indexes** ()

Get a dict of index names to index

**get\_matching\_indexes** (*possible\_hash, possible\_range*)

Get all indexes that could be queried on using a set of keys.

If any indexes match both hash AND range keys, indexes that only match the hash key will be excluded from the result.

**Parameters possible\_hash** : set

The names of fields that could be used as the hash key

**possible\_range** : set

The names of fields that could be used as the range key

**iter\_query\_indexes** ()

Iterator that constructs *QueryIndex* for all global and local indexes, and a special one for the default table hash & range key with the name 'TABLE'

**pformat** ()

Pretty string format

**primary\_key** (*hkey, rkey=None*)

Construct a primary key dictionary

You can either pass in a (hash\_key[, range\_key]) as the arguments, or you may pass in an Item itself

**primary\_key\_attributes**

Get the names of the primary key attributes as a tuple

**primary\_key\_tuple** (*item*)

Get the primary key tuple from an item

**schema**

The DQL query that will construct this table's schema

**total\_read\_throughput**

Combined read throughput of table and global indexes

**total\_write\_throughput**

Combined write throughput of table and global indexes

`dql.models.format_throughput` (*available, used=None*)  
Format the read/write throughput for display

### **dql.monitor module**

Utilities for monitoring the consumed capacity of tables

**class** `dql.monitor.Monitor` (*engine, tables*)  
Bases: `object`  
Tool for monitoring the consumed capacity of many tables

**refresh** (*fetch\_data*)  
Redraw the display

**run** (*stdscr*)  
Initialize curses and refresh in a loop

**start** ()  
Start the monitor

### **dql.output module**

Formatting and displaying output

**class** `dql.output.BaseFormat` (*results, ostream, width=u'auto', pagesize=u'auto'*)  
Bases: `object`  
Base class for formatters

**display** ()  
Write results to an output stream

**format\_field** (*field*)  
Format a single Dynamo value

**pagesize**  
The number of results to display at a time

**post\_write** ()  
Called once after writing all records

**pre\_write** ()  
Called once before writing the very first record

**wait** ()  
Block for user input

**width**  
The display width

**write** (*result*)  
Write a single result and stick it in an output stream

**class** `dql.output.ColumnFormat` (*\*args, \*\*kwargs*)  
Bases: `dql.output.BaseFormat`  
A layout that puts item attributes in columns

**post\_write** ()

**pre\_write** ()

**wait** ()  
Block for user input

**write** (*result*)

**class** `dql.output.ExpandedFormat` (*results, ostream, width=u'auto', pagesize=u'auto'*)  
Bases: `dql.output.BaseFormat`

A layout that puts item attributes on separate lines

**pagesize**

**write** (*result*)

**class** `dql.output.SmartBuffer` (*buf*)  
Bases: `object`

A buffer that wraps another buffer and encodes unicode strings.

**flush** ()  
flush the buffer

**write** (*arg*)  
Write a string or bytes object to the buffer

**class** `dql.output.SmartFormat` (*results, ostream, \*args, \*\*kwargs*)  
Bases: `object`

A layout that chooses column/expanded format intelligently

**display** ()  
Write results to an output stream

`dql.output.delta_to_str` (*rd*)  
Convert a relativedelta to a human-readable string

`dql.output.format_json` (*json\_object, indent*)  
Pretty-format json data

`dql.output.less_display` (*\*args, \*\*kws*)  
Use smoke and mirrors to acquire 'less' for pretty paging

`dql.output.make_list` (*obj*)  
Turn an object into a list if it isn't already

`dql.output.serialize_json_var` (*obj*)  
Serialize custom types to JSON

`dql.output.stdout_display` (*\*args, \*\*kws*)  
Print results straight to stdout

`dql.output.truncate` (*string, length, ellipsis=u'\u2026'*)  
Truncate a string to a length, ending with '...' if it overflows

`dql.output.wrap` (*string, length, indent*)  
Wrap a string at a line length

## dql.throttle module

Wrapper around the dynamo3 RateLimit class

**class** `dql.throttle.TableLimits`  
Bases: `object`

Wrapper around `dynamo3.RateLimit`

**get\_limiter** (*table\_descriptions*)  
Construct a RateLimit object from the throttle declarations

**load** (*data*)  
Load the configuration from a `save()` dict

**save** ()  
Wrapper around `__json__`

**set\_default\_limit** (*read='0', write='0'*)  
Set the default table/index limit

**set\_index\_limit** (*tablename, indexname, read='0', write='0'*)  
Set the limit on a global index

**set\_table\_limit** (*tablename, read='0', write='0'*)  
Set the limit on a table

**set\_total\_limit** (*read='0', write='0'*)  
Set the total throughput limit

## **dql.util module**

Utility methods

`dql.util.dt_to_ts` (*value*)  
If *value* is a datetime, convert to timestamp

`dql.util.eval_expression` (*value*)  
Evaluate a full time expression

`dql.util.eval_function` (*value*)  
Evaluate a timestamp function

`dql.util.eval_interval` (*interval*)  
Evaluate an interval expression

`dql.util.getmaxyx` ()  
Get the terminal height and width

`dql.util.plural` (*value, append='s'*)  
Helper function for pluralizing text

`dql.util.resolve` (*val*)  
Convert a pyparsing value to the python type

`dql.util.unwrap` (*value*)  
Unwrap a quoted string

### **2.1.3 Module contents**

Simple SQL-like query language for dynamo.

`dql.main` ()  
Start the DQL client.



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## d

- dql, 36
- dql.cli, 26
- dql.engine, 30
- dql.expressions, 25
  - dql.expressions.base, 19
  - dql.expressions.constraint, 19
  - dql.expressions.selection, 22
  - dql.expressions.update, 23
  - dql.expressions.visitor, 24
- dql.grammar, 26
  - dql.grammar.common, 25
  - dql.grammar.query, 25
- dql.help, 31
- dql.models, 31
- dql.monitor, 34
- dql.output, 34
- dql.throttle, 35
- dql.util, 36



**A**

add() (in module `dql.expressions.selection`), 23  
 add\_query\_kwargs() (in module `dql.engine`), 31  
 all\_fields (dql.expressions.selection.SelectionExpression attribute), 22  
 all\_keys (dql.expressions.selection.SelectionExpression attribute), 22  
 and\_() (dql.expressions.constraint.Conjunction class method), 20  
 attribute\_names (dql.expressions.visitor.Visitor attribute), 25  
 AttributeSelection (class in `dql.expressions.selection`), 22

**B**

BaseFormat (class in `dql.output`), 34  
 BetweenConstraint (class in `dql.expressions.constraint`), 19  
 build() (dql.expressions.base.Expression method), 19  
 build() (dql.expressions.base.Field method), 19  
 build() (dql.expressions.base.Value method), 19  
 build() (dql.expressions.constraint.BetweenConstraint method), 20  
 build() (dql.expressions.constraint.Conjunction method), 20  
 build() (dql.expressions.constraint.ConstraintExpression method), 20  
 build() (dql.expressions.constraint.FunctionConstraint method), 21  
 build() (dql.expressions.constraint.InConstraint method), 21  
 build() (dql.expressions.constraint.Invert method), 21  
 build() (dql.expressions.constraint.OperatorConstraint method), 21  
 build() (dql.expressions.constraint.SizeConstraint method), 21  
 build() (dql.expressions.selection.AttributeSelection method), 22  
 build() (dql.expressions.selection.NamedExpression method), 22

build() (dql.expressions.selection.NowFunction method), 22  
 build() (dql.expressions.selection.SelectionExpression method), 23  
 build() (dql.expressions.selection.TimestampFunction method), 23  
 build() (dql.expressions.update.FieldValue method), 23  
 build() (dql.expressions.update.SetFunction method), 23  
 build() (dql.expressions.update.UpdateAdd method), 23  
 build() (dql.expressions.update.UpdateDelete method), 24  
 build() (dql.expressions.update.UpdateExpression method), 24  
 build() (dql.expressions.update.UpdateRemove method), 24  
 build() (dql.expressions.update.UpdateSetMany method), 24  
 build() (dql.expressions.update.UpdateSetOne method), 24

**C**

caution\_callback() (dql.cli.DQLClient method), 27  
 cloudwatch\_connection (dql.engine.Engine attribute), 30  
 ColumnFormat (class in `dql.output`), 34  
 complete\_file() (dql.cli.DQLClient method), 27  
 complete\_ls() (dql.cli.DQLClient method), 27  
 complete\_opt() (dql.cli.DQLClient method), 27  
 complete\_opt\_allow\_select\_scan() (dql.cli.DQLClient method), 27  
 complete\_opt\_display() (dql.cli.DQLClient method), 27  
 complete\_opt\_format() (dql.cli.DQLClient method), 27  
 complete\_opt\_pagesize() (dql.cli.DQLClient method), 27  
 complete\_opt\_width() (dql.cli.DQLClient method), 27  
 complete\_use() (dql.cli.DQLClient method), 27  
 complete\_watch() (dql.cli.DQLClient method), 27  
 completedefault() (dql.cli.DQLClient method), 27  
 conf (dql.cli.DQLClient attribute), 27  
 Conjunction (class in `dql.expressions.constraint`), 20  
 connect() (dql.engine.Engine method), 30  
 connection (dql.engine.Engine attribute), 30

ConstraintExpression (class in dql.expressions.constraint), 20

convert() (dql.expressions.selection.SelectionExpression method), 23

create\_alter() (in module dql.grammar), 26

create\_create() (in module dql.grammar), 26

create\_delete() (in module dql.grammar), 26

create\_drop() (in module dql.grammar), 26

create\_dump() (in module dql.grammar), 26

create\_insert() (in module dql.grammar), 26

create\_keys\_in() (in module dql.grammar.query), 25

create\_load() (in module dql.grammar), 26

create\_parser() (in module dql.grammar), 26

create\_query\_constraint() (in module dql.grammar.query), 25

create\_scan() (in module dql.grammar), 26

create\_select() (in module dql.grammar), 26

create\_selection() (in module dql.grammar.query), 25

create\_throttle() (in module dql.grammar), 26

create\_throughput() (in module dql.grammar), 26

create\_update() (in module dql.grammar), 26

create\_where() (in module dql.grammar.query), 26

## D

default() (dql.cli.DQLClient method), 27

default() (in module dql.engine), 31

delta\_to\_str() (in module dql.output), 35

describe() (dql.engine.Engine method), 30

describe\_all() (dql.engine.Engine method), 30

display (dql.cli.DQLClient attribute), 27

display() (dql.output.BaseFormat method), 34

display() (dql.output.SmartFormat method), 35

div() (in module dql.expressions.selection), 23

do\_EOF() (dql.cli.DQLClient method), 27

do\_exit() (dql.cli.DQLClient method), 27

do\_file() (dql.cli.DQLClient method), 27

do\_local() (dql.cli.DQLClient method), 27

do\_ls() (dql.cli.DQLClient method), 28

do\_opt() (dql.cli.DQLClient method), 28

do\_shell() (dql.cli.DQLClient method), 28

do\_throttle() (dql.cli.DQLClient method), 28

do\_unthrottle() (dql.cli.DQLClient method), 28

do\_use() (dql.cli.DQLClient method), 28

do\_watch() (dql.cli.DQLClient method), 28

dql (module), 36

dql.cli (module), 26

dql.engine (module), 30

dql.expressions (module), 25

dql.expressions.base (module), 19

dql.expressions.constraint (module), 19

dql.expressions.selection (module), 22

dql.expressions.update (module), 23

dql.expressions.visitor (module), 24

dql.grammar (module), 26

in dql.grammar.common (module), 25

dql.grammar.query (module), 25

dql.help (module), 31

dql.models (module), 31

dql.monitor (module), 34

dql.output (module), 34

dql.throttle (module), 35

dql.util (module), 36

DQLClient (class in dql.cli), 26

dt\_to\_ts() (in module dql.util), 36

DummyVisitor (class in dql.expressions.visitor), 24

## E

emptyline() (dql.cli.DQLClient method), 28

Engine (class in dql.engine), 30

engine (dql.cli.DQLClient attribute), 28

eval\_expression() (in module dql.util), 36

eval\_function() (in module dql.util), 36

eval\_interval() (in module dql.util), 36

evaluate() (dql.expressions.base.Field method), 19

evaluate() (dql.expressions.base.Value method), 19

evaluate() (dql.expressions.selection.AttributeSelection method), 22

evaluate() (dql.expressions.selection.NowFunction method), 22

evaluate() (dql.expressions.selection.SelectFunction method), 22

evaluate() (dql.expressions.selection.TimestampFunction method), 23

execute() (dql.engine.Engine method), 30

execute() (dql.engine.FragmentEngine method), 31

ExpandedFormat (class in dql.output), 35

ExplainSignal, 31

Expression (class in dql.expressions.base), 19

expression\_values (dql.expressions.visitor.Visitor attribute), 25

## F

Field (class in dql.expressions.base), 19

field\_or\_value() (in module dql.expressions.constraint), 21

field\_or\_value() (in module dql.expressions.update), 24

FieldValue (class in dql.expressions.update), 23

flush() (dql.output.SmartBuffer method), 35

format\_field() (dql.output.BaseFormat method), 34

format\_json() (in module dql.output), 35

format\_throughput() (in module dql.models), 33

formatter (dql.cli.DQLClient attribute), 28

FragmentEngine (class in dql.engine), 31

from\_clause() (dql.expressions.constraint.BetweenConstraint class method), 20

from\_clause() (dql.expressions.constraint.Conjunction class method), 20

- from\_clause() (dql.expressions.constraint.ConstraintExpression class method), 20  
 from\_clause() (dql.expressions.constraint.FunctionConstraint class method), 21  
 from\_clause() (dql.expressions.constraint.InConstraint class method), 21  
 from\_clause() (dql.expressions.constraint.OperatorConstraint class method), 21  
 from\_clause() (dql.expressions.constraint.SizeConstraint class method), 21  
 from\_clause() (dql.expressions.constraint.TypeConstraint class method), 21  
 from\_clause() (dql.expressions.update.FieldValue class method), 23  
 from\_clause() (dql.expressions.update.SetFunction class method), 23  
 from\_clause() (dql.expressions.update.UpdateAdd class method), 24  
 from\_clause() (dql.expressions.update.UpdateDelete class method), 24  
 from\_clause() (dql.expressions.update.UpdateRemove class method), 24  
 from\_clause() (dql.expressions.update.UpdateSetMany class method), 24  
 from\_clause() (dql.expressions.update.UpdateSetOne class method), 24  
 from\_description() (dql.models.GlobalIndex class method), 31  
 from\_description() (dql.models.TableMeta class method), 33  
 from\_selection() (dql.expressions.selection.SelectionExpression class method), 23  
 from\_statement() (dql.expressions.selection.AttributeSelection class method), 22  
 from\_statement() (dql.expressions.selection.NamedExpression class method), 22  
 from\_statement() (dql.expressions.selection.NowFunction class method), 22  
 from\_statement() (dql.expressions.selection.SelectFunction class method), 22  
 from\_statement() (dql.expressions.selection.TimestampFunction class method), 23  
 from\_table\_index() (dql.models.QueryIndex class method), 32  
 from\_update() (dql.expressions.update.UpdateExpression class method), 24  
 from\_where() (dql.expressions.constraint.ConstraintExpression class method), 20  
 function() (in module dql.grammar.common), 25  
 FunctionConstraint (class in dql.expressions.constraint), 20
- G**
- get\_capacity() (dql.engine.Engine method), 31
- get\_enum\_key() (in module dql.cli), 30  
 get\_field() (dql.expressions.visitor.DummyVisitor method), 24  
 get\_field() (dql.expressions.visitor.Visitor method), 25  
 get\_index() (dql.models.TableMeta method), 33  
 get\_indexes() (dql.models.TableMeta method), 33  
 get\_limiter() (dql.throttle.TableLimits method), 36  
 get\_matching\_indexes() (dql.models.TableMeta method), 33  
 get\_value() (dql.expressions.visitor.DummyVisitor method), 25  
 get\_value() (dql.expressions.visitor.Visitor method), 25  
 getmaxyx() (in module dql.util), 36  
 getopt\_default() (dql.cli.DQLClient method), 28  
 getopt\_display() (dql.cli.DQLClient method), 28  
 getopt\_format() (dql.cli.DQLClient method), 28  
 GlobalIndex (class in dql.models), 31
- H**
- hash\_field (dql.expressions.constraint.ConstraintExpression attribute), 20  
 hash\_field (dql.expressions.constraint.OperatorConstraint attribute), 21  
 help\_alter() (dql.cli.DQLClient method), 28  
 help\_analyze() (dql.cli.DQLClient method), 28  
 help\_create() (dql.cli.DQLClient method), 28  
 help\_delete() (dql.cli.DQLClient method), 28  
 help\_drop() (dql.cli.DQLClient method), 28  
 help\_dump() (dql.cli.DQLClient method), 29  
 help\_explain() (dql.cli.DQLClient method), 29  
 help\_help() (dql.cli.DQLClient method), 29  
 help\_insert() (dql.cli.DQLClient method), 29  
 help\_load() (dql.cli.DQLClient method), 29  
 help\_opt() (dql.cli.DQLClient method), 29  
 help\_scan() (dql.cli.DQLClient method), 29  
 help\_select() (dql.cli.DQLClient method), 29  
 help\_update() (dql.cli.DQLClient method), 29
- I**
- InConstraint (class in dql.expressions.constraint), 21  
 indent() (in module dql.cli), 30  
 IndexField (class in dql.models), 32  
 initialize() (dql.cli.DQLClient method), 29  
 Invert (class in dql.expressions.constraint), 21  
 iter\_insert\_items() (in module dql.engine), 31  
 iter\_query\_indexes() (dql.models.TableMeta method), 33
- K**
- key (dql.expressions.selection.NamedExpression attribute), 22
- L**
- less\_display() (in module dql.output), 35

load() (dql.throttle.TableLimits method), 36  
 load\_config() (dql.cli.DQLClient method), 29

## M

main() (in module dql), 36  
 make\_interval() (in module dql.grammar.common), 25  
 make\_list() (in module dql.output), 35  
 Monitor (class in dql.monitor), 34  
 mul() (in module dql.expressions.selection), 23

## N

NamedExpression (class in dql.expressions.selection), 22  
 NowFunction (class in dql.expressions.selection), 22

## O

OperatorConstraint (class in dql.expressions.constraint), 21  
 opt\_allow\_select\_scan() (dql.cli.DQLClient method), 29  
 opt\_display() (dql.cli.DQLClient method), 29  
 opt\_format() (dql.cli.DQLClient method), 29  
 opt\_pagesize() (dql.cli.DQLClient method), 29  
 opt\_width() (dql.cli.DQLClient method), 29  
 or\_() (dql.expressions.constraint.Conjunction class method), 20

## P

pagesize (dql.output.BaseFormat attribute), 34  
 pagesize (dql.output.ExpandedFormat attribute), 35  
 parse\_expression() (in module dql.expressions.selection), 23  
 partial (dql.engine.FragmentEngine attribute), 31  
 pformat() (dql.models.GlobalIndex method), 31  
 pformat() (dql.models.TableMeta method), 33  
 pformat\_exc() (dql.engine.FragmentEngine method), 31  
 plural() (in module dql.util), 36  
 populate() (dql.expressions.selection.NamedExpression method), 22  
 possible\_hash\_fields() (dql.expressions.constraint.Conjunction method), 20  
 possible\_hash\_fields() (dql.expressions.constraint.ConstraintExpression method), 20  
 possible\_range\_fields() (dql.expressions.constraint.Conjunction method), 20  
 possible\_range\_fields() (dql.expressions.constraint.ConstraintExpression method), 20  
 post\_write() (dql.output.BaseFormat method), 34  
 post\_write() (dql.output.ColumnFormat method), 34  
 postcmd() (dql.cli.DQLClient method), 29  
 pre\_write() (dql.output.BaseFormat method), 34  
 pre\_write() (dql.output.ColumnFormat method), 34  
 primary\_key() (dql.models.TableMeta method), 33  
 primary\_key\_attributes (dql.models.TableMeta attribute), 33

primary\_key\_tuple() (dql.models.TableMeta method), 33  
 projects\_all\_attributes() (dql.models.QueryIndex method), 32  
 prompt() (in module dql.cli), 30  
 promptyn() (in module dql.cli), 30

## Q

QueryIndex (class in dql.models), 32  
 quoted() (in module dql.grammar.common), 25

## R

range\_field (dql.expressions.constraint.BetweenConstraint attribute), 20  
 range\_field (dql.expressions.constraint.ConstraintExpression attribute), 20  
 range\_field (dql.expressions.constraint.FunctionConstraint attribute), 21  
 range\_field (dql.expressions.constraint.OperatorConstraint attribute), 21  
 refresh() (dql.monitor.Monitor method), 34  
 region (dql.engine.Engine attribute), 31  
 remove\_index() (dql.expressions.constraint.Conjunction method), 20  
 remove\_index() (dql.expressions.constraint.OperatorConstraint method), 21  
 repl\_command() (in module dql.cli), 30  
 reset() (dql.engine.FragmentEngine method), 31  
 resolve() (in module dql.util), 36  
 run() (dql.monitor.Monitor method), 34  
 run\_command() (dql.cli.DQLClient method), 29  
 running (dql.cli.DQLClient attribute), 29

## S

save() (dql.throttle.TableLimits method), 36  
 save\_config() (dql.cli.DQLClient method), 29  
 scannable (dql.models.QueryIndex attribute), 32  
 schema (dql.models.GlobalIndex attribute), 31  
 schema (dql.models.IndexField attribute), 32  
 schema (dql.models.TableField attribute), 32  
 schema (dql.models.TableMeta attribute), 33  
 select\_functions() (in module dql.grammar.query), 26  
 SelectFunction (class in dql.expressions.selection), 22  
 SelectionExpression (class in dql.expressions.selection), 23  
 serialize\_json\_var() (in module dql.output), 35  
 session (dql.cli.DQLClient attribute), 29  
 set\_default\_limit() (dql.throttle.TableLimits method), 36  
 set\_index\_limit() (dql.throttle.TableLimits method), 36  
 set\_table\_limit() (dql.throttle.TableLimits method), 36  
 set\_total\_limit() (dql.throttle.TableLimits method), 36  
 SetFunction (class in dql.expressions.update), 23  
 SizeConstraint (class in dql.expressions.constraint), 21  
 SmartBuffer (class in dql.output), 35



SmartFormat (class in dql.output), 35  
 start() (dql.cli.DQLClient method), 29  
 start() (dql.monitor.Monitor method), 34  
 stdout\_display() (in module dql.output), 35  
 sub() (in module dql.expressions.selection), 23

## T

TableField (class in dql.models), 32  
 TableLimits (class in dql.throttle), 35  
 TableMeta (class in dql.models), 32  
 TimestampFunction (class in dql.expressions.selection),  
 23  
 to\_index() (dql.models.TableField method), 32  
 total\_read\_throughput (dql.models.TableMeta attribute),  
 33  
 total\_write\_throughput (dql.models.TableMeta attribute),  
 33  
 truncate() (in module dql.output), 35  
 TypeConstraint (class in dql.expressions.constraint), 21

## U

unwrap() (in module dql.util), 36  
 update\_prompt() (dql.cli.DQLClient method), 29  
 UpdateAdd (class in dql.expressions.update), 23  
 UpdateDelete (class in dql.expressions.update), 24  
 UpdateExpression (class in dql.expressions.update), 24  
 UpdateRemove (class in dql.expressions.update), 24  
 UpdateSetMany (class in dql.expressions.update), 24  
 UpdateSetOne (class in dql.expressions.update), 24  
 upkey() (in module dql.grammar.common), 25

## V

Value (class in dql.expressions.base), 19  
 Visitor (class in dql.expressions.visitor), 25

## W

wait() (dql.output.BaseFormat method), 34  
 wait() (dql.output.ColumnFormat method), 34  
 width (dql.output.BaseFormat attribute), 34  
 wrap() (in module dql.output), 35  
 write() (dql.output.BaseFormat method), 34  
 write() (dql.output.ColumnFormat method), 35  
 write() (dql.output.ExpandedFormat method), 35  
 write() (dql.output.SmartBuffer method), 35