dql

Release 0.6.2

Dec 23, 2021

Contents

A simple, SQL-ish language for DynamoDB

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

CHAPTER 1

User Guide

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 throw 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);
```

(continues on next page)

(continued from previous page)

```
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 tablenames are 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>`

width	int / auto	Number of characters wide to format the display
pagesize	int / auto	Number of results to get per page for queries
display	less / stdout	The reader used to view query results
format	smart / column / expanded	Display format for query results
allow_select_scan	bool	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, clone the repo:

```
git clone https://github.com/stevearc/dql.git
```

It is recommended that you create a virtualenv to develop:

```
# python 3
python3 -m venv dql_env
# python 2
virtualenv dql_env

source ./dql_env/bin/activate
pip install -e .
```

1.5.1 Running Tests

The command to run tests is `python setup.py nosetests`, but I recommend using tox. 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.

1.5.2 Local dev Using `pyenv pyenv-virtualenv tox tox-pyenv`

Pre-requisites

- **Install pyenv**

  - Why use pyenv? Intro to pyenv
- Install pyenv-virtualenv so that you can manage virtualenvs from pyenv.
- **Install Java: I recomend using sdkman to manage your java installations.**
  - I use java version 8.0.265.j9-adpt
  - `sdk install java 8.0.265.j9-adpt`

Setting up local envs:

```
# See installed python versions
pyenv versions

# See which python you are currently using. This will also show missing versions
# required by .python-version file.
pyenv which python

# Install the required python versions using pyenv.
pyenv install <version-number>

    # Version numbers are listed in the .python-version file.
    pyenv install 3.8.2
    pyenv install 3.7.7
    pyenv install 3.6.10

# Create a virtual env named "dql-local-env" with python version 3.7.7
pyenv virtualenv 3.7.7 dql-local-env

# Look at the virtual envs. dql-local-env should have a * next to it indicating
# that it is selected.
pyenv virtualenvs

# You should be currently using "~/.pyenv/versions/dql-local-env/bin/python"
pyenv which python

# install dependencies
pip install -r requirements_dev.txt

# running tests with tox
tox

    # running specific tox env
    tox -e format
    tox -e py38-lint
    tox -e package
```

After setting up your local env, you can install the executable of dql:

```
pip install -e .

# In case you have a global dql already installed for your day to day use,
# I recommend bumping the patch number so that you know which version you
# are currently executing.
bump2version patch

# check the version
dql --version
```

1.5.3 Versioning

Use *bump2version* instead of *bumpversion* because *bump2version* is actively maintained. This advise comes from *bumpversion* project itself. See *bumpversion*'s pypi page for details.

Config based on: https://medium.com/@williamhayes/versioning-using-bumpversion-4d13c914e9b8

Usage:

```
# will update the relevant part and start a new `x.x.x-dev0` build version
$> bump2version patch
$> bump2version minor
$> bump2version major

# update the build number from `x.x.x-dev0` to `x.x.x-dev1`
$> bump2version build

# release when ready, will convert the version to `x.x.x`, commit and tag it.
$> bump2version --tag release
```

1.6 Changelog

1.6.1 0.6.2

- Fix: Issue with missing dependency (typing_extensions) made apparent by python 3.9
- Added: Run tests with python 3.9
- Chore: general cleanup & lint fixes
- Chore: removing travis-ci; adding github workflows
- Added: *clear* & *cls* commands.
- Updated: clear, cls, exit commands are no longer tracked in history.

1.6.2 0.6.1

- Feature: Retain query history across sessions. (#40)
- Fix: Cannot count(*) on an index (#37)
- Fix: Saving data to some file formats was failing
- Fix: Constraint functions accept quoted field names (#36)
- Chore: Updated config for Dynamo Local to install dependency within project root.

1.6.3 0.6.0

- Bug fix: Fixed ZeroDivisionError with ls on On-Demand tables (#32)
- Added: ls command accepts glob patterns (#30)
- Added: Better error handling and display. (#28)
- Added: Standard error handling for execution with `-c` option. (#28)

- Added: Keyboard interrupts will print spooky emojis. (#28)
- Added: `--json` argument for use with `-c` to format results as JSON
- Chore: General Dev Env & CI updates for easier development. (#27)

1.6.4 0.5.28

- Bug fix: Encoding errors for some SAVE file formats

1.6.5 0.5.27

- Bug fix: Proper throttling in Python 3

Dropping support for python 3.4

1.6.6 0.5.26

- Use python-future instead of six as compatibility library
- Now distributing a wheel package
- Bug fix: Confirmation prompts crash on Python 2

1.6.7 0.5.25

- Bug fix: Compatibility errors with Python 3

1.6.8 0.5.24

- Bug fix: Support key conditions where field has a – in the name

1.6.9 0.5.23

- Bug fix: Default encoding error on mac

Dropping support for python 2.6

1.6.10 0.5.22

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

1.6.11 0.5.21

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

1.6.12 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

1.6.13 0.5.19

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

1.6.14 0.5.18

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

1.6.15 0.5.17

- Bug fix: Can't display Binary data

1.6.16 0.5.16

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

1.6.17 0.5.15

- Gracefully handle missing imports on Windows

1.6.18 0.5.14

- Missing curses library won't cause ImportError

1.6.19 0.5.13

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

1.6.20 0.5.12

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

1.6.21 0.5.11

- SELECT <attributes> can now use full expressions

1.6.22 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

1.6.23 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

1.6.24 0.5.8

- Tab completion for Mac OS X

1.6.25 0.5.7

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

1.6.26 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

1.6.27 0.5.5

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

1.6.28 0.5.4

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

1.6.29 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

1.6.30 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

1.6.31 0.5.1

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

1.6.32 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

1.6.33 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)

1.6.34 0.4.0

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

1.6.35 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

1.6.36 0.3.1

- Bug fix: Allow '.' in table names

1.6.37 0.3.0

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

1.6.38 0.2.1

- Bug fix: Crash when printing 'COUNT' queries

1.6.39 0.2.0

- Feature: Python 3 support

1.6.40 0.1.0

- First public release

CHAPTER 2

API Reference

2.1 dql package

2.1.1 Subpackages

dql.expressions package

Submodules

dql.expressions.base module

dql.expressions.constraint module

dql.expressions.selection module

dql.expressions.update module

dql.expressions.visitor module

Module contents

dql.grammar package

Submodules

dql.grammar.common module

dql.grammar.parsed_primitives module

2.1.2 Submodules

2.1.3 Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search