Version 1.0 (Estimated Release date 02/05/2014)

Features

1. Table definition
2. Single column pkey per table
3. Secondary indexes (non-unique)
4. Typeless columns
5. Insert statement
6. Select statement
7. Update statement
8. Delete statement
9. Subscribe statement
10. Unsubscribe statement
11. Filtering (where clause) by pkey or by secondary indexes (no conditional logic yet release 1.1 :))
12. Select, Update, Subscribe and Delete can have where clause

Components

Server - cross-platform pubsubsql server (Initial release will support Linux/Windows) with CLI interface

Client

golang API
Java API
.Net API
Java GUI (Interactive Query)
.Net GUI (Interactive Query)

Interactive Query GUI will allow user to connect to pubsubsql server and execute sql statements. There will be 3 panels. Query, Results and Messages. The Results panel will have a grid to display data. In addition to request/response mechanism, user will be able to subscribe to a result set and receive updates in real time as underlying data changes (continuous querying).

Commands

Command Options

1. help
2. status
3. stop
4. start
    - start 127.0.0.1:777
    - start 777
    - start
    - response {“status”:”ok” , “ip”:”127.0.0.1”, “port} - table was successfully created
5. connect
    - connect 127.0.0.1:777
    - connect 777
    - connect

CLI output

pubsubsql>
pubsubsql 127.0.0.1:777>

CLI Commands

all of the above except for start and connect

1. exit

SQL Commands

All responses will be in the form of JSON documents. It is done in order to make it easier to implement client APIs

6. create table
    - request - create table stocks (ticker:pkey, sector**:**idx, bid, ask)
      columnName **:** pkey - defines primary key
      columnName **:** idx - defines non unique index
    - response {“status”:”ok”} - table was successfully created
    - response {“status”:”error” “msg”:”error message”}

-

7. insert
    - request - insert into stocks (ticker, sector, bid, ask) values (‘IBM’, ‘tech’, 140, 141)
    - response {“status”:”ok”, “rows”:1} - 1 row was inserted
    - response {“status”:”error” “msg”:”error message”}

8. update
    - request - update stocks set bid = 142, ask = 143 where ticker = ‘IBM’
    - response {“status”:”ok”, “rows”:1} - 1 row was updated
    - response {“status”:”ok”, “rows”:0} - 0 row was updated
    - response {“status”:”error” “msg”:”error message”}

9. delete
    - request - delete from stocks where ticker = ‘IBM’
    - response {“status”:”ok”, “rows”:1} - 1 row was deleted
    - response {“status”:”ok”, “rows”:0} - 0 row was deleted
    - response {“status”:”error” “msg”:”error message”}

10. select
    - request - select * from stocks where sector = ‘tech’
      To be decided if anything other than * is supported in release 1.0

    - response
      {“status”:”ok”, “rows”:2”, “data”:
      [
      {“ticker”:”IBM” , “sector”:”tech”, “bid”:140 , “ask”:141 } ,
      {“ticker”:”GOOG” , “sector”:”tech”, “bid”:1200 , “ask”:1220 } ,
      }
    - response {“status”:”ok”, “rows”:0} - 0 row was returned
    - response {“status”:”error” “msg”:”error message”}

11. subscribe
    - request - subscribe * from stocks where sector = ‘tech’
      To be decided if anything other than * is supported in release 1.0
    - response {“status”:”ok”, “pubsubid”:1 } pubsubid is a server generated unique identifier for a particular subscription
    - response {“status”:”error” “msg”:”error message”}

12. unsubscribe
    - request - unsubscribe from stocks - removes all of the subscriptions for a given table for a particular connection
    - request - unsubscribe from stocks where pubsubid = 1- removes subscription with subid 1 for a particular connection
    - response {“status”:”ok” }
    - response {“status”:”error” “msg”:”error message”} - in case no subscriptions were found