CSQL User Manual

www.databasecache.com

Contents

1 Introduction

CSQL is a fast, multi-threaded SQL main memory database engine. It is a free software, licensed with the GNU GENERAL PUBLIC LICENSE http://www.gnu.org/ It aids in the development of high performance, fault-resilent applications requiring concurrent access to the shared data.

This database is built keeping ultra fast performance in mind. This database suits well as a front end database for other commercial database to increase the throughput. Throughput of queries involving single table may improve by 20x. This loads all the data into memory and avoids on demand fetches from disk and avoids the overhead of buffer manager found in commercial disk based database systems. It provides fast access through shared memory.

This main memory database shall be used as a cache for disk based database to improve the throughput of your exising application by 20x times without any code change.

1.1 Platforms Supported

Linux
Solaris

1.2 Compilers Supported

g++ in Linux
CC in Solaris

1.3 Key Features

Single row lookup takes less than 10 microsecs (100K selects/second)

Single row insert, update, delete takes less than 20 microsecs (50K writes/second, 180 Million inserts/hour)

Memory resident architecture to give ultra fast performance

Embedded Architecture - database is co-located with application

Small Footprint

Multi Layered: ODBC, JDBC, SQL API, DB API

Supports ACI of Transaction Properties

Isolation Level support:

READ UNCOMMITED, READ COMMITTED, READ REPEATABLE

Protection from process failures

Faster lookups through Hash Indexes

Row Level locking for high concurrency

Hand written user level mutexes which avoids major recovery

Multi user support with authentication

Primitive SQL support

Primitive JDBC Support

1.4 Architecture

Main memory databases are times faster than disk based database systems as, all the data is available in physical memory and avoids the buffer manager overhead which is found in disk based database systems. Moreover data access algorithms can work efficiently as compared to traditional disk based algorithms.

CSQL is compact main memory database SQL Engine which supports limited set of features and gives ultra fast response to database queries. It supports only limited features which is used by most of the real time applications which includes INSERT, UPDATE, DELETE, SELECT on single table with local predicates. The primary difference between traditional database system and main memory database system is that in the latter the whole data is stored in main memory.It differs in

Indexing - A conventional index stores a data value followed by the address at which the record can be located. When all the data is in memory anyway, there is no need to store the data value redundantly, only the address is required.

Caching/buffer management - no longer exists because all the data is in memory already. All of the processing as well as all of the disk latency associated with moving data from disk to memory (and vice versa) are removed.

Optimization - is still required but becomes much simpler and is more likely to choose the fastest path, since the number of alternatives is much less than with a disk-based database

CSQL is developed keeping one thing in mind - performance. It uses shared memory architecure where all applications access the database though direct pointer referencing. Data access

is synchronized using mutexes for internal structures and through row level locks for rows. The is absolutely no disk I/O for any DML operation, only for writing trace log messages disk is used. Rest all are handled and processed in-memory.

1.5 Compiling the source:

Go to the root directory and enter the following commands.

```
$./configure --prefix=‘pwd‘/install
$make
$make install
```

This will create "install" directory under the current directory and places all the executables created in bin directory and libraries in lib directory.

1.6 Generating API Reference

Go to the root directory and enter

```
$doxygen
```

This will create "docs/html" directory under which API Reference html files are stored. Refer index.html in that directory.

2 Getting Started

2.1 Starting the Server

You should find csqlserver executable under the bin directory of the installation. Change csql.conf file under the installation root directory if necessary.

Set the

```
CSQL_CONFIG_FILE
```

environment variable to absolute path of csql.conf file.

```
$export CSQL_CONFIG_FILE=/tmp/csql.conf
```

You should find csqlserver executable under the bin directory of the installation.

```
$./csqlserver
```

This starts the server and creates the database file.

2.2 Shutting down the Server

Pressing Ctrl-C on the terminal where csqlserver is running, is safe and will stop the server gracefully by removing the database file.

2.3 Creating tables

Tables and Indexes can be created though interactive sql tool named csql. To create table t1 with two fields integer and char

```
$csql
CSQL>create table t1 (f1 int, f2 char (20), primary key (f1));
```

If you see "Statement Executed" message after you enter the above command, it means the table is created successfully.

2.4 Inserting data into tables

Rows shall be inserted using INSERT SQL statement through csql tool

```
$csql
CSQL>insert into t1 values (10, 'Value');
```

If you see "Statement Executed: Rows Affected = 1 " message after you enter the above command, it means the row is inserted successfully.

2.5 Selecting data from tables

```
$csql
CSQL>select * from t1;
```

It will display all the rows in the table t1

2.6 Deleting data from tables

```
$csql
CSQL>select * from t1;
```

It will delete all the rows in the table t1

3 API Interfaces

3.1 SQL Support

```
CREATE TABLE {tablename} ({fielddefinitionlist}[, primary ({fieldname})])
     fielddefinitionlist shall be fieldname type [(size)] [NOT NULL]

CREATE INDEX {idxname} ON {tablename}({fieldname}) [HASH] [UNIQUE] [PRIMARY]

DROP TABLE {tablename}

DROP INDEX {indexname}

INSERT INTO {tablename} [fieldNameList] VALUES ({valuelist})

UPDATE {tablename} SET {fldname=value ,...} [WHERE {condition}]

DELETE FROM {tablename} [WHERE {condition}]

SELECT {* | fieldNameList} FROM {tablename} [WHERE {condition}]
     condition shall be [NOT] {predicate} {AND | OR } {predicate}
     predicate shall be {fieldName} {"=, !=, >,<, >=, <="} [fieldName | value]
                          LIKE {format}, BETWEEN {value} AND {value} , IN {list}
```

DataType support: INT, LONG , BIGINT, SHORT, FLOAT, DOUBLE, DATE, TIME, TIMESTAMP, CHAR

3.2 JDBC Interface

3.2.1 Supported Interfaces

```
Connection:: connect, close
Statement:: execute
```

```
PreparedStatement:: execute, params
ResultSetMetaData::
ResultSet:: Forward only
```

DataType Support: All Primitive types Date, Time, TimeStamp

3.2.2 Connecting from jdbc

The default username is root and password is manager for the csql database system. Below code snippet shows how to connect to csql database from jdbc applications

```
Class.forName("csql.jdbc.JdbcSqlDriver");
Connection con = DriverManager.getConnection("jdbc:csql", "root", "manager");
```

3.2.3 Inserting data through jdbc

The below code snippet inserts into table t1 with two fields, f1 integer and f2 char

```
PreparedStatement stmt = null;
stmt = con.prepareStatement("INSERT INTO T1 (f1, f2) VALUES (?, ?);");
int ret =0;
for (int i =0 ; i< 10 ; i++) {
   stmt.setInt(1, i);
   stmt.setString(2, String.valueOf(i+100));
   ret = stmt.executeUpdate();
   if (ret != 1) break; //error
 }
 stmt.close();
 con.commit();
```

3.3 SQLAPI

```
SqlConnection :: connect, disconnect, commit, rollback

SqlStatement:: prepare, execute,  bindField, bindParam,
               fetch, set[Type]Param, getProjFldInfo, getParamFldInfo

FieldInfo :: Meta data information for params and projection fields.
```

3.4 DB API

This allows user process and threads to access or manipulate the database. Main interfaces are Connection, DatabaseManager, Table, etc. Connection provides interface to connect and disconnect to the database file DatabaseManager provides interface to create and drop database objects including tables and indexes. Table provides interface to insert, update, delete and fetch tuples.

Refer API reference under the directory docs/html. (If there is no html, you shall generate it by yourself using doxygen tools. Refer the previous section for this.)

3.4.1 Getting a Connection

Connection interface is the heart of all the interfaces as it is the entry point for database access and it provides interface for transaction commit/rollback.

You can obtain a connection to database using the following code snippet:

```
Connection conn;
DbRetVal rv = OK;
rv = conn.open("root", ",manager");
if (rv!= OK) return -1;
...
```

3.4.2 Creating Tables

Database Manager provides interface for table creation and deletion. We shall obtain the DatabaseManager object from the connection object. The table or schema definition is encapsulated in TableDef interface. It provides methods to specify the field definition of the table. For example to create table with two fields,

```
DatabaseManager *dbMgr = conn.getDatabaseManager();
if (dbMgr == NULL) { printf("Bad connection \n"); return -1;}
TableDef tabDef;
tabDef.addField("f1", typeInt, 0, NULL, true);
tabDef.addField("f2", typeString, 196);
rv = dbMgr->createTable("t1", tabDef);
if (rv != OK) { printf("Table creation failed\n"); return -1; }
```

First argument of addField method is field name, second is the type of the field, third argument is length, fourth is default value, fifth is not null flag and last is primary key flag. In our example, field "f1" is not null and it is the primary key for the table. Call addField for

all the field definition in the table and call createTable passing table name as first argument and TableDef as second argument.

3.4.3 Inserting into the tables

Any DML operation requires a transaction to be started. All the operations happen within the context of this transaction. Application buffer should first be binded to the respective fields using the bindFld method as mentioned in the below example; insertTuple method will pick values from the binded application buffer and creates a new row or tuple in the table.

```
Table *table = dbMgr->openTable("t1");
if (table == NULL) { printf("Unable to open table\n"); return -1; }
int id = 100;
char name[196] = "Tuticorin";
table->bindFld("f1", &id);
table->bindFld("f2", name);
conn.startTransaction();
ret = table->insertTuple();
if (ret != OK) { printf("Unable to insert tuple\n"); return -1; }
conn.commit();
```

3.4.4 Selecting tuples from the tables

Application buffer should first be binded to the respective fields using the bindFld method as we did for insert; fetch method will copy the values from the row to the binded application buffer. Predicate if required shall be created using the condition interface and set in the table object before the execution. Calling execute method is must before calling the fetch method, as it will create the execution plan for the scan. It selects appropriate index based on the predicate set.

```
Condition p1;
int val1 = 100;
p1.setTerm("f1", OpEquals, &val1);
table->setCondition(p1);
int id = 100;
char name[196] = "Bangalore";
table->bindFld("f1", &id);
table->bindFld("f2", name);
table->execute();
void *tuple = (char*)table->fetch() ;
if (tuple == NULL) {printf(" No tuple found \n" ); table->close();return -1;}
```

```
printf("tuple value is %d %s \n",id ,name);
table->close();
```

3.4.5 Updating or Deleting tuples

This operation is allowed always on existing scan. When this method is called the current tuple in the scan in either updated with application buffer values which are binded, or gets deleted based on the method called. For example:

```
tuple = (char*)table->fetch() ;
if (tuple == NULL)  {printf(" No tuple found \n" ); table->close();return -1;}
strcpy(name, "50576543210"); //update the value
table->updateTuple();
```

4 Configuration

4.1 csql.conf

Configuration is read from environment variable

`CSQL_CONFIG_FILE`

. A default configuration is available at the src root directory with name csql.conf.

4.1.1 Server Section

This contains parameters which affect the functioning of the csql server process. For Server section parameters, make sure that the value is same for the server process and all the csql client process which connects to it. otherwise, behavior is undefined

`PAGE_SIZE`

Default Value :8192
Description: Each database is logically divided into pages and allocation happens in unit of pages. This defines the size of the allocation unit. Increasing this value will reduce frequent allocation of pages

`MAX_TRANS`

Default Value :100
Description: Total number of active transactions that can run concurrently in the system

`MAX_PROCS`

Default Value :100
Description: Total number of client process which can connect and work with the database concurrently

`MAX_SYS_DB_SIZE`

Default Value :1048576
Description: Maximum size of the system database. Catalog information and control information(lock, logs, etc) are stored in this internal database.

`MAX_DB_SIZE`

Default Value :10485760
Description: Maximum size of the user database. Rows and Index information of tables are stored in this database.

`SYS_DB_KEY`

Default Value :2222
Description: Shared memory key to be used by the system to create and locate system database. Change this value if u want to run two csql instances in same host

`USER_DB_KEY`

Default Value :5555
Description: Shared memory key to be used by the system to create and locate user database

`LOG_FILE`

Default Value :/tmp/log/log.out
Description: Give full path for the log file. This is where important system actions are stored for tracing application events

`MAP_ADDRESS`

Default Value :400000000
Description: The virtual memory start address at which the shared memory segment will be created and attached

4.1.2 Client Section

This contains parameters which affect the functioning of the client process which connects to the database.

`MUTEX_TIMEOUT_SECS`

Default Value : 0
Description: Mutex interval in seconds

`MUTEX_TIMEOUT_USECS`

Default Value : 1000
Description: Mutex interval in milli seconds

`MUTEX_TIMEOUT_RETRIES`

Default Value : 10
Description: Mutex retry count in case it fails to acquire

`LOCK_TIMEOUT_SECS`

Default Value : 0
Description: Lock interval in seconds

`LOCK_TIMEOUT_USECS`

Default Value : 1000
Description: Lock interval in milli seconds

LOCK_TIMEOUT_RETRIES

Default Value : 10
Description: Lock retry count in case it fails to acquire

5 Tools

5.1 csql tool

```
Usage: csql [-u username] [-p passwd] [-s sqlfile]
       username -> username to connect to database
       password -> password to connect to database
       sqlfile -> filename containing sql statements
```

Example:

```
$csql -u root -p manager -s /tmp/import.sql
```

5.2 catalog tool

```
Usage: catalog [-u username] [-p passwd] [-l] [-i] [-d] [-T table] [-I index] [-D <lock|
       l -> list all table with field information
       i -> reinitialize catalog tables. Drops all tables.
       d -> print db usage statistics
       T -> list table information
       I -> list index information
       D -> print debug information for system tables
Note: If multiple options are specified, last one will be considered.
```

Example:

```
$catalog -u root -p manager -T table1
```

Sample output:

```
<Table Info>
  <TableName> table1 </TableName>
  <TupleCount> 0 </TupleCount>
  <PagesUsed> 1 </PagesUsed>
  <SpaceUsed> 16 </SpaceUsed>
  <Indexes> 1 <Indexes>
  <TupleLength> 20 </TupleLength>
  <Fields> 2 </Fields>
  <Indexes>
<IndexName> t1_idx1_Primary </IndexName>
  </Indexes>
</Table Info>
```

6 Common Error Messages

Message: Shared memory open failed
Check whether the csqlserver process is running