



Cache Guide

CSQL
Main Memory Database Cache

Table of Contents

<u>1. BEFORE YOU START.....</u>	<u>5</u>
1.1. WHAT IS CSQL.....	5
1.2. CSQL COMPONENTS	5
1.2.1. CSQL MAIN MEMORY DATABASE.....	5
1.2.2. CSQL CACHE.....	5
1.3. INFORMATION IN THIS MANUAL.....	6
1.4. CONVENTIONS.....	6
1.4.1. TYPOGRAPHIC RULE	6
1.4.2. SYNTAX ANNOTATION	7
<u>2. OVERVIEW OF CSQL CACHE SYSTEM.....</u>	<u>8</u>
2.1. ARCHITECTURAL OVERVIEW.....	8
2.2. MAIN FEATURES	9
2.3. DEPLOYMENT SCENARIOS.....	11
2.3.1. ACTIVE RESULT SET CACHE	11
2.3.2. TRANSPARENT CACHING WITH GATEWAY	11
2.3.3. READ INTENSIVE CACHE.....	12
2.3.4. READ/WRITE INTENSIVE CACHE	13
2.3.5. WRITE INTENSIVE CACHE	13
2.3.6. MULTI NODE CACHE	14
2.3.7. COHERENT MULTI NODE CACHE.....	15
<u>3. CSQL CACHE INSTALLATION</u>	<u>17</u>
3.1. CONFIGURING CACHE.....	17
3.1.1. CACHE SECTION VARIABLES	17
<u>4. CACHE OPTION FOR MYSQL.....</u>	<u>20</u>
4.1. SETTING UP CSQL AND MYSQL	20
4.1.1. PREREQUISITES.....	20
4.1.2. MYSQL DATABASE	20
4.1.3. MYSQL ODBC CONNECTOR.....	20
4.1.4. UNIXODBC.....	21
4.1.5. CONFIGURE – ODBC.INI FILE.....	21
4.2. UNIDIRECTIONAL CACHE.....	22
4.2.1. CONFIGURE THE CACHE	22
4.2.2. STEPS FOR UNIDIRECTIONAL CACHE	22

4.2.3.	CREATE A MYSQL TABLE	23
4.2.4.	LOAD MYSQL TABLE INTO CACHE	23
4.2.5.	CSQL - TO - MYSQL UPDATES	24
4.2.6.	UNLOAD THE CACHE TABLE	25
4.3.	BI-DIRECTIONAL CACHE.....	25
4.3.1.	CONFIGURING CSQL. CONF FILE.....	25
4.3.2.	SETTING UP MYSQL.....	26
4.3.3.	CREATE A MYSQL TABLE FOR CACHING	26
4.3.4.	CREATE LOG TABLE ON MYSQL	26
4.3.5.	EXECUTE TRIGGER.....	27
4.3.6.	STEPS FOR BI-DIRECTIONAL CACHE	28
4.3.7.	LOAD MYSQL TABLE TO CACHE	28
4.3.8.	MYSQL – TO –CSQL UPDATE PROPAGATION	28
4.3.9.	BIDIRECTIONAL CACHE ON NON-INTEGER PRIMARY KEY.....	29
5.	<u>CACHE OPTION FOR POSTGRES</u>	<u>30</u>
5.1.	SETTING UP CSQL AND POSTGRES.....	30
5.1.1.	PRE-REQUISITES	30
5.1.2.	POSTGRES DATABASE.....	30
5.1.3.	UNIXODBC.....	30
5.1.4.	CONFIGURE –ODBC.INI FILE.....	31
5.2.	UNIDIRECTIONAL	31
5.2.1.	CONFIGURE CSQL.CONF FILE	31
5.2.2.	STEPS FOR UNIDIRECTIONAL CACHE.....	32
5.3.	BI-DIRECTIONAL CACHE.....	32
5.3.1.	CONFIGURING CSQL. CONF FILE.....	32
5.3.2.	SETTING UP POSTGRES	33
5.3.3.	CREATE A TABLE IN POSTGRES FOR CACHING	33
5.3.4.	CREATE CSQL_LOG_INT TABLE IN POSTGRES	33
5.3.5.	EXECUTE TRIGGER.....	33
5.3.6.	STEPS FOR BI-DIRECTIONAL CACHE.....	35
6.	<u>CACHE OPTION FOR ORACLE</u>	<u>36</u>
6.1.	SETTING UP CSQL AND ORACLE.....	36
6.1.1.	PRE-REQUISITES	36
6.1.2.	ORACLE DATABASE.....	36
6.1.3.	UNIXODBC.....	36
6.1.4.	CONFIGURE –ODBC. INI FILE.....	36
6.2.	UNIDIRECTIONAL	37
6.2.1.	CONFIGURE CSQL. CONF FILE	37
6.2.2.	STEPS FOR UNIDIRECTIONAL CACHE.....	38
6.3.	BI-DIRECTIONAL CACHE.....	38
6.3.1.	CONFIGURING CSQL.CONF FILE.....	38

6.3.2.	SETTING UP ORACLE.....	39
6.3.3.	CREATE A TABLE IN ORACLE FOR CACHING.....	39
6.3.4.	CREATE CSQL_LOG_INT TABLE IN ORACLE.....	39
6.3.5.	EXECUTE TRIGGER.....	39
6.3.6.	STEPS FOR BI-DIRECTIONAL CACHE.....	40
7.	<u>MULTI NODE CACHE.....</u>	41
7.1.	CONFIGURE CSQL.....	41
7.2.	CONFIGURE MYSQL.....	41
7.3.	CONFIGURE MULTIPLE NODES.....	43
7.3.1.	LOAD MYSQL TABLE TO CACHE	43
7.3.2.	MYSQL – TO – CSQL UPDATE PROPAGATION.....	44
8.	<u>CSQL CACHE MODES.....</u>	45
8.1.	UPDATEABLE CACHE	45
8.2.	READ ONLY CACHE.....	45
8.3.	DIRECT CACHE	46
8.4.	ASYNCHRONOUS CACHE.....	47
8.5.	SYNCHRONOUS CACHE	48
8.6.	UNIDIRECTIONAL CACHE.....	48
8.7.	BI-DIRECTIONAL CACHE.....	48
9.	<u>WORKING WITH CSQL GATEWAY.....</u>	49
9.1.	USING CSQL TOOL.....	49
9.2.	USING ODBC INTERFACE.....	49
9.3.	USING JDBC INTERFACE.....	50
9.4.	USING SQL INTERFACE	51
10.	<u>TOOLS FOR CACHE.....</u>	52
10.1.	CACHETABLE.....	52
	UNLOAD THE CACHE TABLE USING –U OPTION	53
	CACHE RECORDS WHICH SATISFY CONDITION	53
	CACHE ONLY SPECIFIED FIELDS.....	53
	RELOADING THE CACHE TABLE.....	53
10.2.	CACHEVERIFY	54

1. Before you Start

Before you dip into this Guide, this chapter gives a brief introduction such as what is CSQL, Information in this manual, and Conventions used in this manual.

1.1. What is CSQL

CSQL Main Memory Database Cache is an easy to use and powerful database management system, which includes the CSQL Main Memory Database, and caching for MySQL, PostgreSQL or Oracle database.

It typically resides in the middle tier (also called as business logic tier) along with the applications and store data in main memory rather than disk, allowing it to be retrieved very quickly. It is used for applications where extremely fast response times are critical. Investment applications could use CSQL to store current stock price information, which needs to be accessed quickly to make trading decisions. Mobile operators use the software for billing, so that they can quickly determine whether a caller has enough prepaid credit to place a call.

CSQL could be used around the world for a wide variety of applications ranging from single-user system to enterprise-wide multi-user installations with thousands of concurrent connections with very less contention as it employs lock free data structures.

1.2. CSQL Components

This Introduction provides an overview of the two major components that make up CSQL. These components include:

- CSQL Main memory database
- CSQL Cache

1.2.1. CSQL Main Memory Database

CSQL Main Memory Database derives much of its power from its unique architecture. At the core, the CSQL database storage engine provides the complete set of services – including data storage, concurrency management, transactions, and process management needed to build efficient database management system.

CSQL delivers 10 to 40 times faster response times for real time queries than traditional database systems.

1.2.2. CSQL Cache

CSQL Caching enables applications to significantly improve its throughput. As data is cached on the main-memory database, cache delivers a real-time, dynamic, updatable

cache for frequently accessed data in the disk based databases such as Oracle, MySQL or Postgres.

For the performance critical applications in industries such as Telecom, Process Control, Airline Reservation, Stock Market, Health Care, etc., the Cache option delivers application response times in microsecond by bringing the frequently accessed data closer to the application, and by executing SQL requests in the main-memory database.

1.3. Information in this Manual

This manual describes CSQL concepts, components, and basic usage. This manual could be useful for the following CSQL Users.

- Database Administrators
- Application Designers
- Programmers

Before reading this manual, understanding of following background knowledge is recommended.

- Basic knowledge required for Linux Operating System and Operating System Commands.
- Experience in using the relational database or understanding of the database concepts.

1.4. Conventions

The below mentioned conventions have been maintained throughout this manual.

1.4.1. Typographic Rule

This manual uses the following typographic rule.

Table 1.1. Typographic Rule

FORMAT	USED FOR
MAIN MEMORY DATABASE	This font is used for ordinary text.
SELECT * FROM T1	This font is used for SQL Statements and Program code.
UNIQUE	This font with uppercase letter indicates SQL keywords and data types.
csql.conf	This font indicates file name and path.
build.ksh	This font is used for command lines
SQLFetch()	This font is used for function names
Java.Sql.Statement	This font is used for interface, classes and header files names.

1.4.2. Syntax Annotation

This manual uses the following syntax annotation conventions

Table 1.2. Syntax Annotation

FORMAT	USED FOR
[]	THIS SQUARE BRACKET INDICATES THAT ITEM IN COMMAND LINE IS OPTIONAL
{ }	Curly braces indicates that one must choose one of the items separated by a vertical bar () in a command line.
	A vertical bar separates arguments
...	An ellipsis indicates that arguments can be repeated several times
.	This indicates that continuation of previous lines of code
.	
.	
\$	This dollar sign indicates the Linux prompt.
#	The pound sign indicates the Linux root prompt.

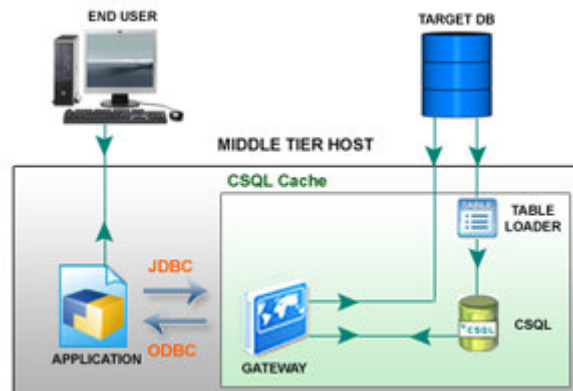
2. Overview of CSQL Cache System

CSQL Cache is employed in real time applications for accelerating and speeding up access to disk-based databases. It comprises a high performance main-memory database (performs 10-40X faster than traditional disk database management systems), which provides microsecond response times for data manipulation operations.

2.1. Architectural Overview

CSQL Cache is a high performance, bi-directional updateable data-caching infrastructure that sits between the clustered application process and back-end data sources. It provides unprecedented high throughput to your application by offloading the computing cycles from expensive backend systems along with reduction in costly network calls, thereby enabling real time application to provide faster and predictive response time.

CSQL Cache uses the fastest Main Memory Database (CSQL MMDB) designed for high performance and high volume data computing for caching the table and provides most flexible and cost-effective way to cache and manage enterprise information without compromising on transactional and indexed access to the data. This main memory database is 20-40 times faster than traditional disk based database system as the database completely resides in main memory and developed to be used on real time high computing data platforms.



Caching happens at table level granularity in CSQL, which means that the tables, which are frequently accessed by the applications, shall be cached in CSQL MMDB. Once these tables are loaded, they are treated like normal CSQL tables and any DML operation is allowed on these tables.

The difference between normal CSQL table and cached table is that, the CSQL server propagates DML operations INSERT, UPDATE, DELETE on cached tables to target database automatically. These updates can be propagated to target database in two modes,

SYNC and ASYNC from CSQL. In SYNC modes, execute() returns after performing the operation at both cache and original table. In case of ASYNC mode, execute() returns immediately after inserting into the cache, this operation is propagated to original table in target database by another process.

Apart from caching tables from target database, CSQL Cache also acts as a transparent gateway, which allows the application to access the tables, which are not cached in CSQL. Application is completely unaware of this caching as CSQL Cache provides a unified interface to the applications and it does not need to deal with the location of the table, making the underlying database caching architecture transparent to the application layer. This minimizes the code changes for existing applications to adapt CSQL caching.

2.2. Main Features

The following are some of the features of CSQL Cache

- **Updateable Cache Tables**

Most of the existing cache solutions are read only which limits their usage to small segment of the applications, i.e. non-real time applications. In CSQL, updates are allowed on cached tables and these updates are automatically applied on the original table in the target database.

- **Bi-Directional Updates**

For updateable caches, updates that happen directly on the original table on the target database come to cache automatically.

- **Synchronous and Asynchronous update propagation**

The updates on cache table shall be propagated to target database in two modes. Synchronous mode makes sure that after the database operation completes the updates are applied at the target database as well. In case of Asynchronous mode the updates are delayed to the target database providing better performance for modification operations.

Synchronous mode gives high cache consistency and is suited for real time applications. Asynchronous mode gives high throughput and is suited for near real time applications where applications can tolerate inconsistency of records at cache and original table for few milliseconds.

- **Multiple cache granularity**

CSQL supports Table level and Result-set caching. Major portions of corporate databases are historical and infrequently accessed. But, there is

some information that should be instantly accessible like premium customer's data, records of one zone, etc

- **Recovery for cached tables**

In case of system or power failure, during the restart of caching platform all the committed transactions on the cached tables can be recovered.

- **Tools to validate the coherence of cache**

In case of asynchronous mode of update propagation, cache at different cache nodes and target database may diverge. This needs to be resolved manually and the CSQL Cache provides tools to identify the mismatches and take corrective measures if required.

- **Horizontally Scalable**

Clustering is employed in many solutions to increase the availability and to achieve load balancing. CSQL Cache works in a clustered environment spanning to multiple nodes and also keeps the cached data coherent across the nodes.

- **Transparent access to non-cached tables reside in target database**

CSQL Cache keeps track of queries and intelligently route to the database cache or to the origin database based on the data locality without any application code modification.

- **Transparent Fail over**

There will not be any service outages, incase of caching platform failure. Client connections are automatically routed to the target database.

- **Minimal Code Change to adapt caching**

No or very minimal changes to application for the caching solution.

- **Support for standard interfaces**

Caching is supported for standard interfaces including JDBC and ODBC that will make the application to work seamlessly without any application code changes. It intelligently routes all stored procedure calls to target database so that they don't need to be migrated.

- **Support for caching from any target database**

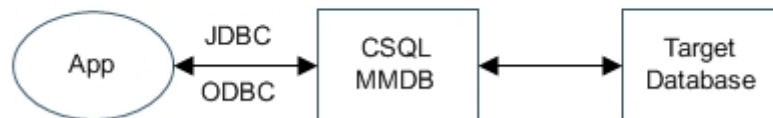
CSQL can cache tables from any DBMS provided they have ODBC driver to access the data. Though CSQL can cache tables from any database, it is tested for MySQL, Postgres and Oracle databases currently.

2.3. Deployment Scenarios

2.3.1. Active Result Set Cache

In result set based caching strategy adopted by applications, one of the major drawbacks is changes to the source table will not be reflected in the cached result set. This limits the strategy to be used only for tables, which are never updated. In active result set caching, applications are allowed to update result set either directly in CSQL MMDB or on the target database in which case it is automatically pushed to cache table at CSQL MMDB. It also allows querying on the cached result set (as it is stored in MMDB) using SQL queries.

ACTIVE RESULTSET CACHING

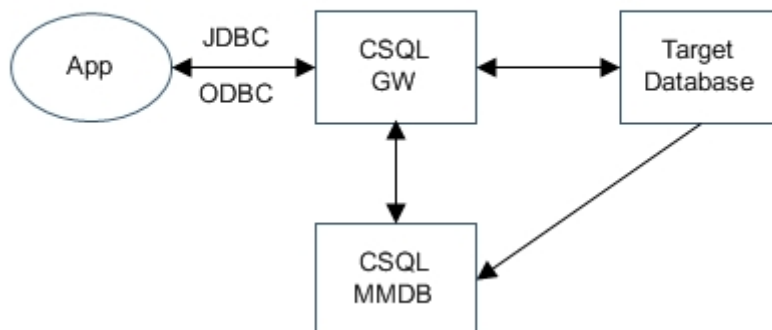


In this architecture, there will be one CSQL MMDB and one target database. The application talks to CSQL cache through the standard JDBC or ODBC interface. The tables to be cached say “customer_info” (from telephone billing database) is loaded from the target database into CSQL MMDB. If this table is updated in the target database by some other backend application, the updated/inserted record is pushed to CSQL cache by employing bi-directional mode.

2.3.2. Transparent Caching with Gateway

In this architecture also, CSQL gateway module provides a unified interface to applications through the standard JDBC or ODBC. Application changes the DSN name for ODBC application or connection URL for JDBC applications to employ caching.

GATEWAY CACHE



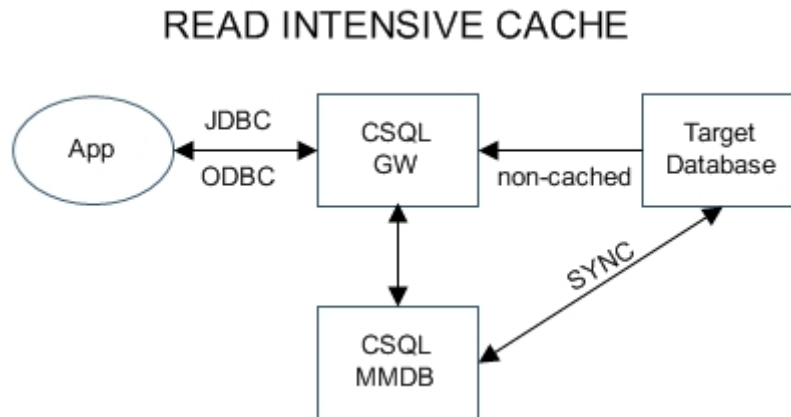
Frequently accessed tables are cached from target database into CSQL MMDB and gateway component enables access/modifications to cached tables in CSQL MMDB and non-cached tables, which reside only in the target database.

If the application sends a select query, requesting some records from the table “customer_info” , the gateway checks whether the table resides in the CSQL cache. If it is so, it returns records from CSQL. If the table is not found in the Cache, the gateway returns the records from the target database. If our application inserts/updates any record in the cached table, that operation is effected in both cache and the target database by the gateway.

This gateway cache can be configured in three ways. They are read intensive cache, read/write intensive cache and write intensive cache.

2.3.3. Read Intensive Cache

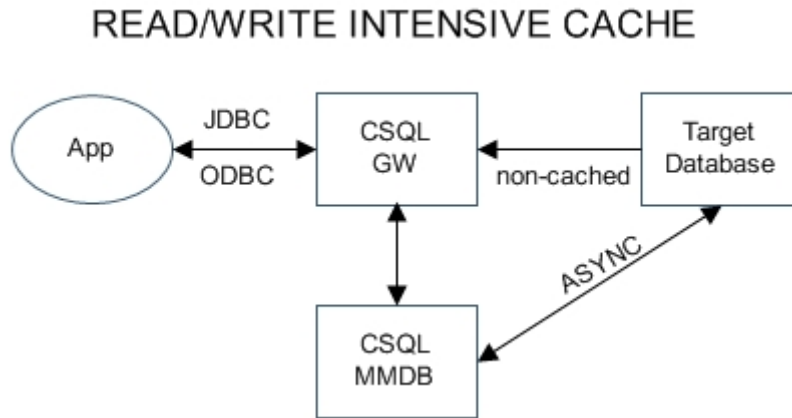
This deployment mode is useful when the reads are more than writes on cached tables. In flight booking management system, flight schedules are read-only and are not updated frequently. They shall be cached in CSQL cache, so that applications accessing these timetables will get ultra-fast response.



Application connected to gateway fetches records from either CSQL MMDB (in case of cached table) or from target database (in case of non-cached table). If application does updates (which happens very rarely as the application is read intensive) in the cached table, CSQL MMDB will do the same operation on target database before the commit operation returns. So in this architecture, reads are ultra fast and writes are comparatively slow as it involves updating both cache and target database before commit returns.

2.3.4. Read/Write Intensive Cache

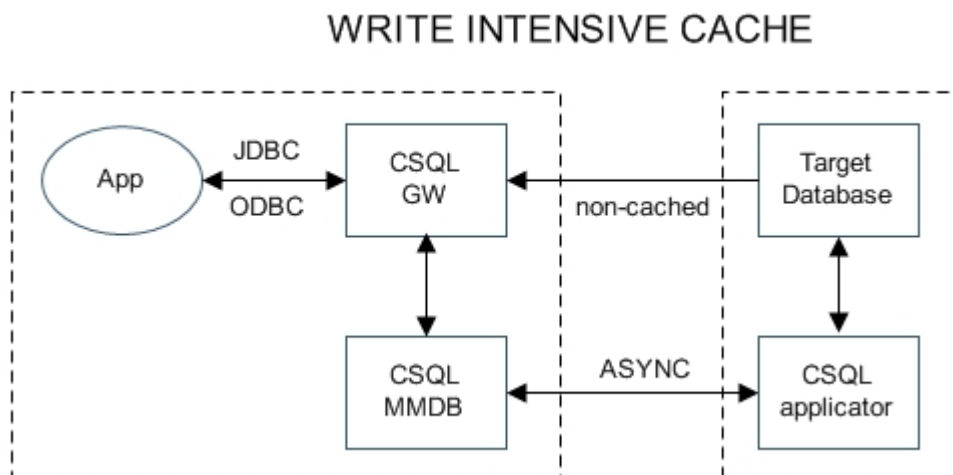
This deployment is suitable for applications, which have equal probability for read and write operations to database.



Here everything is same as the read intensive cache except the update propagation mode between CSQL MMDB and target database. If a transaction tries to update the cached table, CSQL MMDB carries out the operation in its cached table and then just informs another internal process (applier) to perform the same operation in the target database. Commit operation returns immediately after updating the cache, it delivers better performance for modification operations.

2.3.5. Write Intensive Cache

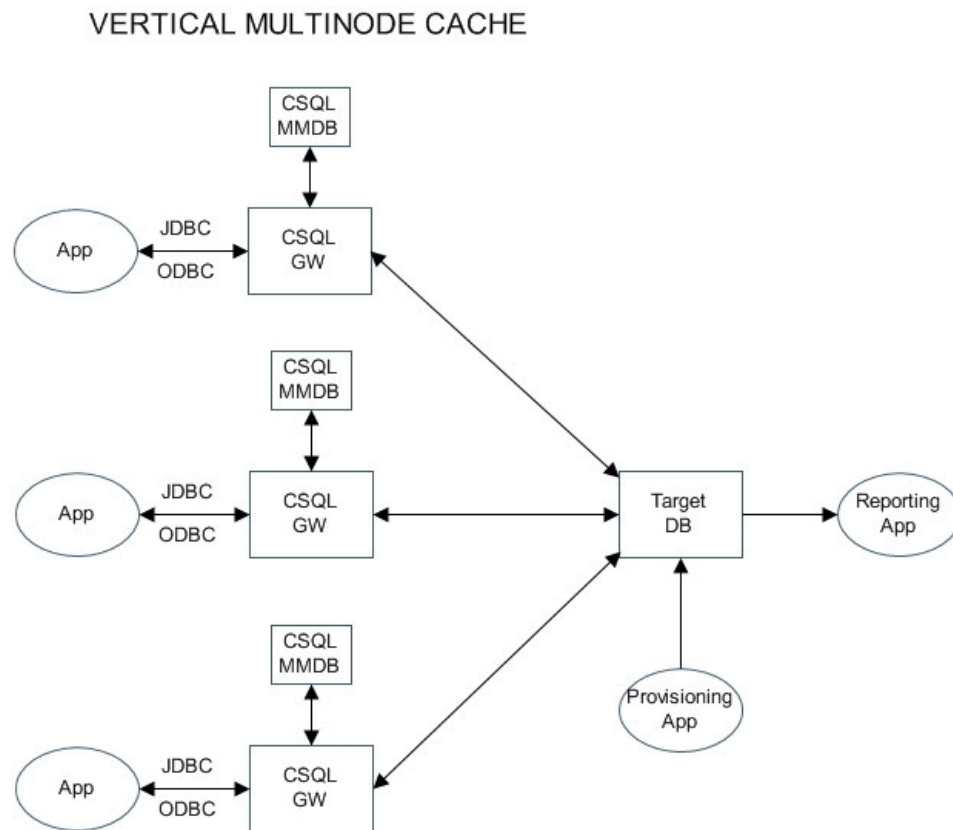
CSQL MMDB performs write operations faster than disk-based databases. If you have performance bottleneck on write operations, then applications can use this deployment option to improve write operations



Here the application, Gateway and CSQL MMDB lies in one machine and another applier process runs in the host where target database is running. Updates on cached tables are propagated to another process called (applier) which runs in target database host and that process takes care of executing the operations on behalf of CSQL MMDB. This reduces the computing cycles for the application, as it does not perform execution at the host where application resides.

2.3.6. Multi Node Cache

Applications can scale up by employing multiple cache nodes to cache tables. This architecture consists of many CSQL gateways, each having its own CSQL cache. All the gateways are connected to a single target database. Applications shall be connected to any one of the gateways through the standard JDBC/ODBC interface.



In this configuration, each CSQL MMDB shall cache complete table at all nodes or a subset of records in each node. For example, for caching 'customer_info' table using this architecture, all the zone 1 customer details will be cached in the first CSQL cache, the second one caches the zone 2 customer details and the last one caches the zone 3 customer details. In this way, we can add any number of nodes, thus improving the performance of the entire system. All the gateways are then connected to a centralized server in which the actual 'customer_info' table resides. In case of telecom domain, each

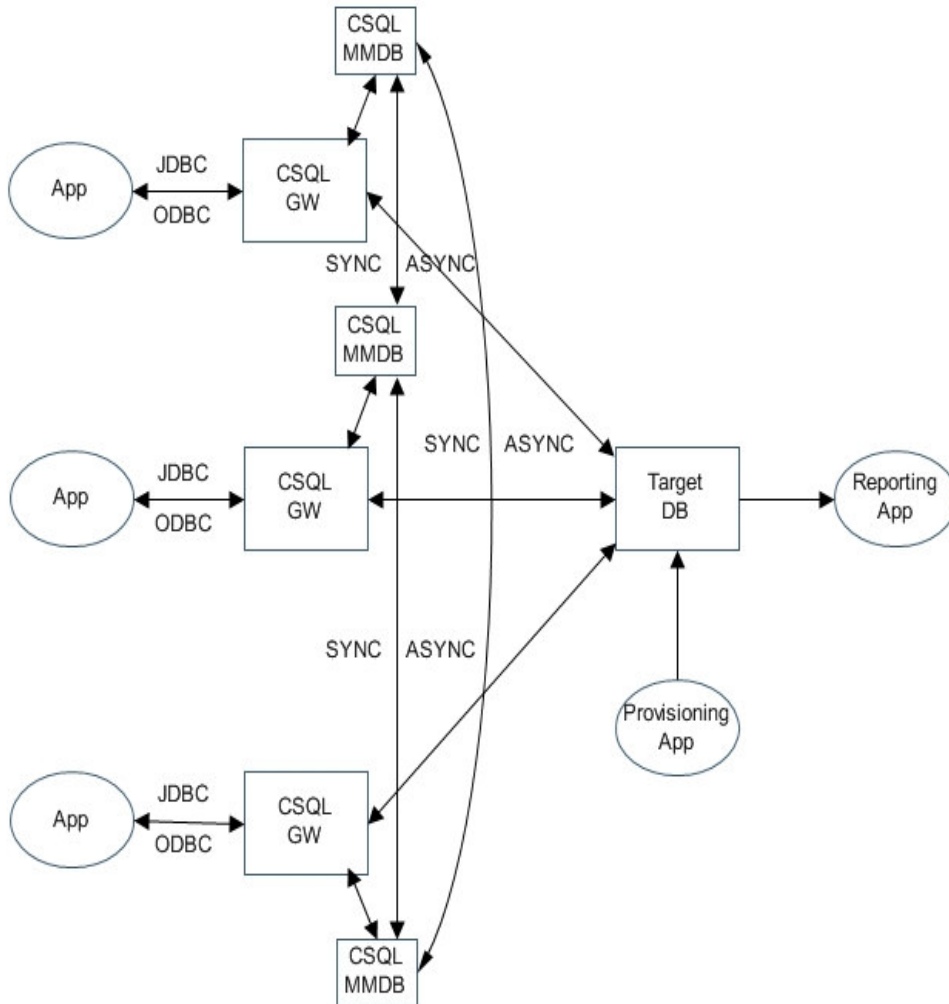
zone (Bangalore, Chennai, Delhi, Mumbai etc.,) contains CSQL cache with only their zone customer details and all these servers are connected to a centralized database server(target database) which contains information about all the customers in India.

By enabling bi-directional caching at all cache nodes, updates on any of the cache instances will be propagated to other cache nodes automatically. In the above deployment scenario, provisioning application and the reporting application talks directly to the target database. The provisioning application inserts/updates the original table in target database, which is then pushed into the corresponding CSQL MMDB only. For example, new customer added to Chennai zone will be available only in the Chennai Zone CSQL Instance.

2.3.7. Coherent Multi Node Cache

This deployment option is useful for application with stringent cache consistency requirements. When update operation is performed on one cache node, it should be reflected in all the other cache nodes as well before it returns to the application. This can be used for banking applications to cache account table for accessing balance amount before performing any withdrawal. When the withdrawal transaction completes, the balance amount should be updated in all the other cache nodes as well.

COHERENT MULTINODE CACHE



Each cache server can communicate to the other caches instances either synchronously or asynchronously. In synchronous communication, when application updates the cached table, the updates are applied to all the cache instances before returning to the application. In asynchronous communication, when application updates the table, the updates are applied to CSQ MMDB and returns to the application. Another process “applier” takes care of applying these updates on other cache instances asynchronously thereby improving the throughput of updates.

3. CSQL Cache Installation

For complete information about installing CSQL and its directory structure, *Refer Section 3. in the CSQL User Manual.*

CSQL can be downloaded from <http://www.csqldb.com>

3.1. Configuring Cache

CSQL requires some system parameters that need to be set before starting CSQL database. These configuration variables are defined in a file called `csql.conf`. Some of the parameters mentioned in this file may have to be tweaked based on the requirements.

The lines starting with `#` are ignored as comments and the rest are treated as configuration variables. These variable values are read from this file during server start up.

These configuration variables are divided logically into following classes.

- Server section variables
- Client section variables
- Cache section variables

It is very important to note that for Server section parameters, the value should be the same for the server process and all the CSQL client processes, which connects to it. Otherwise, behavior is undefined. For Server and Client section variables, *please Refer Section 3.5.2.2 in CSQL User's Manual.*

3.1.1. Cache section variables

The configuration file, `csql.conf` has nine parameters for caching. They can be found in cache section of the `csql.conf` file.

- **Set the `CACHE_TABLE` configuration parameter to 'true' .**

```
[ CACHE_TABLE=true ]
```

This is a Boolean parameter and should be set to true in order to enable caching of tables from target database.

- **set the `CACHE_ID` parameter to 1.**

```
[ CACHE_ID=1 ]
```

This is an integer parameter and it identifies a cache node instance in case of multiple cache nodes. If you have single cache node instance, then set it to 1.

For multi node caching, set this to a unique value for all the nodes. For example node instance-1 will have this parameter set to 1 and node instance-2 will have this parameter set to 2 and so on.

- **Set the DSN configuration parameter with respect to the target database.**

```
[ DSN = myodbc3 ]
```

This is set to connect to target database and this data source name should present in the `~/odbc.ini` file. The above DSN name (myodbc3) is for MySQL database, it could be changed for Postgres or Oracle database based on the requirement. This is a string parameter.

- **Set the USER and PASSWORD parameter with respect to the target database**

```
[ USER = root ]  
[ PASSWORD = root123 ]
```

Username and password specified in the above parameters are used when CSQL Cache connects to target database. In case of ORACLE, this needs to be scott and tiger respectively for connecting to the default oracle schema.

- **Set the CACHE_MODE parameter**

This is a string parameter, which takes values either SYNC or ASYNC. This determines the update propagation mode to target database for modification operations (insert, update and delete) on cached tables.

- **Set the ENABLE_BIDIRECTIONAL_CACHE configuration parameter to true**

```
[ ENABLE_BIDIRECTIONAL_CACHE=true ]
```

This boolean parameter is needed in order to enable “two way caching” (Bi-directional update propagation). Any updates on the target database will be applied to the cache and makes the cache coherent with the target database.

- **Set the CACHE_RECEIVER_WAIT_SECS parameter to 10 seconds.**

```
[ CACHE_RECEIVER_WAIT_SECS=10 ]
```

This is an integer parameter, which should be set to interval the server waits if there are no update logs from the target database in case of bi-directional caching.

- **Set the TABLE_CONFIG_FILE parameter to '/tmp/csql/csqltable.conf'**

```
[ TABLE_CONFIG_FILE=/tmp/csql/csqltable.conf ]
```

This is a string parameter, which contains the complete path to the file, which holds the cache table information. You can specify your own path for this file.

4. Cache option for MySQL

This chapter shows a step-by-step method for caching MySQL data into CSQL and demonstrates automatic update propagation to MySQL.

4.1. Setting up CSQL and MySQL

In this section we outline and demonstrate how CSQL can be configured to work as data cache for MySQL database.

4.1.1. Prerequisites

For CSQL Cache to work you need to install MySQL Server, MySQL ODBC Connector, unixODBC packages on your system. Please make sure that these packages are installed in your system before you proceed.

It can be downloaded from below URLs:

- ❑ [MySQL Database](http://dev.mysql.com/downloads/mysql/5.0.html#downloads)
(<http://dev.mysql.com/downloads/mysql/5.0.html#downloads>)
- ❑ [MySQL ODBC Connector](http://dev.mysql.com/downloads/connector/odbc/3.51.html)
(<http://dev.mysql.com/downloads/connector/odbc/3.51.html>)
- ❑ [UnixODBC](http://rpm.pbone.net/index.php3/stat/3/srodzaj/1/search/unixODBC)
(<http://rpm.pbone.net/index.php3/stat/3/srodzaj/1/search/unixODBC>)

4.1.2. MySQL Database

After you installed MySQL, it automatically starts the server. If it is not started for any reason, you can start it using the following command after logging in with user ‘mysql ‘

```
$/etc/init.d/mysqld start
```

4.1.3. MySQL ODBC Connector

The driver library of MySQL ODBC connector is usually named as libmyodbc3.so and will be available under the installation directory.

4.1.4. UnixOdbc

The driver library of unixODBC is named as `odbc.so` and will be available at `/usr/lib` or `/usr/local/lib` directory.

4.1.5. Configure – odbc.ini file

The `odbc.ini` file is a sample data-source configuration information file. This file contains list of data sources and properties for each database those are present in your system. There are two types of `odbc.ini` files.

- System Configuration File
- User Configuration File

The System `odbc.ini` file is present at `/etc/odbc.ini` while the user file is usually present at `~/.odbc.ini`.

Below `odbc.ini` file contains DSN(Data Source Name), named `myodbc3` along with driver path and some other information.

The `~/.odbc.ini` file which is present in your home directory should contain the below lines.

```
[ODBC Data Sources]
myodbc3           = MyODBC 3.51 Driver DSN

[myodbc3]
Driver            = /usr/lib/libmyodbc3.so
Description       = Connector/ODBC 3.51 Driver DSN
SERVER            = localhost
PORT              = 3306
USER              = lakshya
Password          = lakshya123
Database          = test
OPTION            = 16
SOCKET            = /var/lib/mysql/mysql.sock
```

You can find the complete path for the Driver is `/usr/lib/libmyodbc3.so`. If the file `libmyodbc3.so`, which has been downloaded by you, is present in some other path then you have to specify the complete pathname accordingly.

This guide assumes that the driver library is present in `/usr/lib/libmyodbc3.so`.

You can use isql tool, to check whether the above configuration for MySQL ODBC Connector will succeed or not.

```
$isql myodbc3
It should display the following
+-----+
| Connected! |
|           |
| sql-statement |
| help [tablename] |
| quit |
|           |
+-----+
SQL>
```

If you get the above output then you have configured properly.

4.2. Unidirectional Cache

The default caching in CSQL is one-way (or unidirectional cache), which means all updates (INSERT, UPDATE, DELETE statement execution) on cached tables will be automatically propagated to target database.

4.2.1. Configure the Cache

You must have to modify some cache variables in conf file. Refer 3.1.1. for Cache Section variables.

Cache section variables should contain below specified value for unidirectional cache.

```
CACHE_TABLE=true
CACHE_ID=1
DSN = myodbc3
USER = lakshya
PASSWORD = lakshya123
CACHE_MODE=SYNC
ENABLE_BIDIRECTIONAL_CACHE=false
CACHE_RECEIVER_WAIT_SECS=0
TABLE_CONFIG_FILE=/tmp/csql/csqltable.conf
```

4.2.2. Steps for Unidirectional Cache

This section describes how to create a unidirectional cache that caches the contents of a single table from MySQL database. Although CSQL supports multiple cache tables, we cache only one MySQL table to keep the example simple.

4.2.3. Create a MySQL table

Connect to your MySQL account and create a table.

Using the command `isql` in `myodbc3`, you will be able to get SQL prompt. Through this prompt you can execute DDL and DML statements in MySQL database.

Connect to your database and create a table:

```
$isql myodbc3;  
SQL> CREATE TABLE T1 ( f1 int, f2 char(196), primary  
key(f1) ) ;
```

Now insert some records and commit the changes:

```
SQL> INSERT INTO T1 VALUES( 1, 'Hi' ) ;  
SQL> INSERT INTO T1 VALUES( 2, ' All' ) ;  
SQL> COMMIT ;
```

4.2.4. Load MySQL table into Cache

Use the `cachetable` tool to load the MySQL table into Cache. Make sure that CSQL Server is running before you use `cachetable` tool. Refer the Section 3.8. in CSQL UserManual for starting and stopping the CSQL Server.

Open another terminal and make sure that you are in CSQL root directory and set up the environment using the command. `./setupenv.ksh`.

```
$ cachetable -U root -P manager -t T1
```

Now check the contents of the cache table, which is present in main memory database using CSQL tool.

```
CSQL>SELECT * FROM T1;
```

```
-----  
      f1      f2  
-----  
      1      Hi  
      2      All
```

Refer the section 10.1. for more information on cachetable tool.

4.2.5. CSQL - to - MySQL Updates

The unidirectional feature makes sure that if we perform any modification operations (INSERT, UPDATE, DELETE) on the table in Cache the original table at MySQL will be modified as well.

Now update one record of cache table, it should propagate to the MySQL. The 'T1' table would have been updated in both the database.

Run csql tool with -g option, it creates an isql session which acts as gateway for CSQL and MySQL and propagates changes to the original tables at target database.

Follow the below instructions:

- **Create gateway connection**

```
$csql -g
CSQL>
```

- **Update first row in CSQL**

```
CSQL>update T1 set f2='Hi Smith' where f1=1;
Statement Executed: Rows Affected = 1
```

- **Checks the update in CSQL**

```
CSQL>select * from T1;
-----
      f1      f2
-----
      1      Hi Smith
      2      All
```

- **Checks the update in MySQL**

```
mysql> select * from T1;
+-----+-----+
| f1  | f2      |
+-----+-----+
| 1   | Hi Smith|
| 2   | All     |
```

```
+-----+-----+
```

If you perform INSERT or DELETE operation, the effect of that operation will be done at both cache and target database. For Example, if you insert new record in the cache, it will be inserted in target database as well.

4.2.6. Unload the cache table

Cached tables can be unloaded (removed from the cache) using `-u` option of `cachetable` tool

- **Unload the Table using “cachetable -u” option.**

```
$cachetable -U root -P manager -t T1 -u
```

Note: CSQL restricts for dropping the cache table using DROP statement. Application is expected to unload the table first and then drop it.

4.3. Bi-directional Cache

The default caching in CSQL is unidirectional caching, which means all updates (INSERT, UPDATE, DELETE) on cached tables will be automatically propagated to target database. CSQL also supports bi-directional caching in which, direct updates on target database are also propagated to CSQL cache automatically.

Bi-directional caching is implemented using triggers of the target database. This requires additional tables and triggers, which needs to be installed on the tables in target database.

Sample trigger code is available in the file `trigger.sql` under the CSQL root directory.

4.3.1. Configuring `csql.conf` file

For bi-directional cache, some modification needs to be made in cache section in `csql.conf` file. Refer 3.5.2.3. Cache Section variables.

Cache section variables should contain below specified value

```
CACHE_TABLE=true  
CACHE_ID=1  
DSN = myodbc3  
USER = lakshya  
PASSWORD = lakshya123  
CACHE_MODE=SYNC  
ENABLE_BIDIRECTIONAL_CACHE=true
```

```
CACHE_RECEIVER_WAIT_SECS=10
TABLE_CONFIG_FILE=/tmp/csql/csqltable.conf
```

The only difference between unidirectional and bi-directional configuration is the value of `ENABLE_BIDIRECTIONAL_CACHE` parameter, which is set to true for bi-directional caching.

4.3.2. Setting up MySQL

After configuring `csql.conf` file, a log table needs to be created in target database to hold the operation log records for all the cached tables. You also need to install triggers on cached table for INSERT, UPDATE and DELETE operation.

4.3.3. Create a MySQL table for caching

```
$isql myodbc3;
SQL> CREATE TABLE p1 ( f1 int, f2 char(196), primary
key(f1 ) ) ;
```

Now insert some records and commit the changes :

```
SQL> INSERT INTO T1 VALUES( 1, 'Hi' ) ;
SQL> INSERT INTO T1 VALUES( 2, ' All' ) ;
SQL> COMMIT ;
```

4.3.4. Create log table on MySQL

As mentioned before, for propagating direct updates on MySQL, we need to create log table to hold operation logs for all cached tables.

Create the `csql_log_int` table with five fields, Follow below statements.

```
SQL>CREATE TABLE csql_log_int (
tablename CHAR(64),
pkid INT,
operation INT,
cacheid INT,
id INT NOT NULL UNIQUE AUTO_INCREMENT ) engine='innodb';
```

4.3.5. Execute trigger

Following is the format for creating the triggers for cached table 'p1' having primary key 'f1' in MySQL database. A sample file (trigger.sql) is available at the CSQL root directory; you shall modify that with your cached table name and its primary key field.

```
use test;
drop trigger if exists triggerinsertp1;
drop trigger if exists triggerupdatep1;
drop trigger if exists triggerdeletep1;

DELIMITER |
create trigger triggerinsertp1
AFTER INSERT on p1
FOR EACH ROW
BEGIN
Insert into csql_log_int (tablename, pkid, operation,
cacheid) values ('p1', NEW.f1, 1, 1);
End;

create trigger triggerupdatep1
AFTER UPDATE on p1
FOR EACH ROW
BEGIN
Insert into csql_log_int (tablename, pkid, operation,
cacheid) values ('p1', OLD.f1, 2, 1);
Insert into csql_log_int (tablename, pkid, operation,
cacheid) values ('p1', NEW.f1, 1, 1);
End;

create trigger triggerdeletep1
AFTER DELETE on p1
FOR EACH ROW
BEGIN
Insert into csql_log_int (tablename, pkid, operation,
cacheid) values ('p1', OLD.f1, 2, 1);
End;
|
```

Note: Trigger name ends with the table name. Replace 'p1' in the above script to the cached table name and 'f1' to the primary key fieldname of the cached table as per your table and field name.

After a successful editing of trigger.sql file, now it's ready for execution. Follow the below command to execute the trigger which is present under your csql root directory.

```
$ mysql -u root -p <trigger.sql
```

4.3.6. Steps for Bi-directional cache

Follow the below steps to see bi-directional caching in action

4.3.7. Load MySQL table to Cache

Now run the csqlserver and use the cachetable tool to cache MySQL table into cache.

Create another terminal make sure that you are in csql root directory and set up the environment (. ./setupenv.ksh)

```
$ cachetable -U root -P manager -t p1
```

Now check the contents of the cache table which is present in Cache using CSQL tool.

```
CSQL>SELECT * FROM p1;
-----
      f1      f2
-----
      1      Hi
      2      All
```

Refer the section 10.1. for more details about cachetable tool.

4.3.8. MySQL – to –CSQL Update propagation

Insert some records in p1 table in MySQL, and the record should be propagated to the cached table at CSQL automatically based on the time specified in CACHE_RECEIVER_WAIT_SECS variable. The update will appear in cache before CACHE_RECEIVER_WAIT_SECS.

After this query the cached table in CSQL and the original table in MySQL database will contain the same number of records..

- o **Insert Records in MySQL**

```
$isql myodbc3;
SQL> insert into p1 values(3, 'NoOne');
SQLRowCount returns 1
SQL> insert into p1 values(4, 'EveryOne');
SQLRowCount returns 1

two more records now are in inserted.
```

- **Query the p1 table in MySQL**

```
mysql> select * from p1;
+-----+-----+
| f1 | f2      |
+-----+-----+
| 4  | Everyone|
| 3  | NoOne   |
| 2  | All     |
| 1  | Hi      |
+-----+-----+
```

- **Query the p1 table in CSQL**

```
CSQL>select * from p1;
-----
          f1      f2
-----
          1      Hi
          2      All
          3      NoOne
          4      Everyone
```

You will find the records in cache, those are inserted into the original table at MySQL.

4.3.9. Bidirectional cache on non-integer primary key

Bi-directional cache on table having non-integer primary key field

Let us consider a table having non-integer (say char) primary field to be cached as bi-directionally .For example:

```
$isql myodbc3
create table t1 (f1 int, f2 char(10), f3 float, primary
key(f2) );
```

For the bi-directional cache on table t1 having non-integer primary key field f2, you need to add a extra key field f4 which is not null unique auto_increment field .

```
$ alter table t1 add column f4 int not null unique
auto_increment ;
```

After adding an extra key field make necessary changes on trigger file table f4 as key field .Run the trigger and do the operation as per requirement as mentioned in binary cache section

5. Cache Option for Postgres

This chapter shows a step-by-step method for caching Postgres database into CSQL and demonstrates automatic update propagation to Postgres database.

5.1. Setting up CSQL and Postgres

Setting requires for CSQL and for Postgres for caching. Below sections illustrate it.

5.1.1. Pre-requisites

The “ unixODBC ” package and “ Postgres “ have to be installed in your system and you can follow the below links for downloading if you don't have.

- ❑ [Postgres Database](http://www.postgresql.org/download/)
(<http://www.postgresql.org/download/>)
- ❑ [UnixODBC](http://rpm.pbone.net/index.php3/stat/3/srodzaj/1/search/unixODBC)
(<http://rpm.pbone.net/index.php3/stat/3/srodzaj/1/search/unixODBC>)

5.1.2. Postgres Database

Follow the below commands for starting the database server. The following command assumes that Postgres is installed under /usr/local/pgsql ' directory.

```
$export PGDATA=/usr/local/pgsql/data  
$ /usr/local/pgsql/bin/postmaster /tmp \  
>/postgreslog 2>/tmp/postgreslog &
```

Unlike MySQL, postgres installs the ODBC driver library while installing the server software. Usually this library is named as libodbcpsql.so and will be present in /usr/lib or /usr/local/lib directory.

5.1.3. UnixOdbc

The driver library of unixODBC is named as odbc.so and will be available at /usr/lib or /usr/local/lib directory.

5.1.4. Configure –odbc.ini file

Below odbc.ini file contains the properties for DSN (Data Source Name), named psql. You should append these lines in your odbc.ini file at your home directory (~/.odbc.ini)

```
[psql]
Description = Postgres ODBC Data source
Driver = /usr/lib/libodbcpsql.so
Database = test
Servername = localhost
Username = postgres
Password =
Port = 5432
Protocol = 6.4
```

This guide assumes that the driver library is present in /usr/lib directory. If it is installed in some other directory change the DRIVER parameter value accordingly.

Check the connection using DSN name with isql tool.

```
$isql psql
It should display the following
+-----+
| Connected! |
|          |
| sql-statement |
| help [tablename] |
| quit      |
|          |
+-----+
SQL>
```

5.2. Unidirectional

5.2.1. Configure csqldb.conf file

You must have to modify some cache variables in csqldb.conf file. [Refer 3.1. for Cache Section variables](#). Cache section variables should contain below specified value.

```
CACHE_TABLE=true
CACHE_ID=1
DSN = psql
USER = postgres
PASSWORD =
CACHE_MODE=SYNC
```

```
ENABLE_BIDIRECTIONAL_CACHE=false
CACHE_RECEIVER_WAIT_SECS=0
TABLE_CONFIG_FILE=/tmp/csql/csqltable.conf
```

5.2.2. Steps for unidirectional cache

This section describes how to create a unidirectional cache that caches the contents of a single table from Postgres database. Although CSQL supports multiple cache tables, we cache only one Postgres table to keep the example simple.

Follow the section 4.2.2.1. for discussion on “Cache MySQL data”. The step-by-step procedure has been discussed for caching a table and to make some update and see caching in action. The difference is only that Postgres will be placed instead of MySQL when you follow those sections.

5.3. Bi-directional Cache

Bi-directional caching is implemented using triggers of the target database. This requires additional tables and triggers, which needs to be installed on the tables in target database. Sample trigger code is available in the file trigger.psql under the CSQL root directory

5.3.1. Configuring csql.conf file

For bi-directional cache, some changes are required in cache section parameters of csql.conf file. [Refer 3.5.2.3. for Cache Section variables.](#)

Cache section variables should contain below specified value

```
CACHE_TABLE=true
CACHE_ID=1
DSN = psql
USER = postgres
PASSWORD =
CACHE_MODE=SYNC
ENABLE_BIDIRECTIONAL_CACHE=true
CACHE_RECEIVER_WAIT_SECS=10
TABLE_CONFIG_FILE=/tmp/csql/csqltable.conf
```

The only difference between unidirectional and bi-directional configuration is the value of ENABLE_BIDIRECTIONAL_CACHE parameter, which is set to true for bi-directional caching.

5.3.2. Setting up Postgres

After configuring `csql.conf` file, a log table needs to be created in target database to hold the operation log records for all the cached tables. You also need to install triggers on cached table for INSERT, UPDATE and DELETE operation.

5.3.3. Create a table in Postgres for Caching

```
$isql psql
SQL> CREATE TABLE t1 ( f1 int, f2 char(196), primary
key(f1 ) ) ;
```

Now insert some records and commit the changes :

```
SQL> INSERT INTO p1 VALUES( 1, 'Hi' ) ;
SQL> INSERT INTO p1 VALUES( 2, ' All' ) ;
SQL> COMMIT ;
```

5.3.4. Create `csql_log_int` table in Postgres

As mentioned before, for propagating direct updates on MySQL, we need to create log table to hold operation logs for all cached tables.

Follows the below statements for creating log table :

```
$isql psql
SQL>CREATE TABLE csql_log_int(
tablename varchar(64),
pkid int,
operation int,
cacheid int);

ALTER TABLE csql_log_int
add id serial;
```

5.3.5. Execute trigger

Let's say for cached table 'p1' having primary key field 'f1', following is the format for creating the triggers in postgres database. A sample file (`trigger.psql`) is available at the CSQL root directory; you shall modify that with your cached table name and its primary key field.

```
CREATE LANGUAGE plpgsql;
```

```

CREATE FUNCTION log_insert_t1() RETURNS trigger AS
$triggerinsertt1$
BEGIN
insert into csql_log_int (tablename, pkid, operation,
cacheid) values ('t1', NEW.f1, 1, 1);
RETURN NEW;
END;
$triggerinsertt1$ LANGUAGE plpgsql;
create trigger triggerinsertt1
AFTER INSERT on t1
FOR EACH ROW
EXECUTE PROCEDURE log_insert_t1();

CREATE FUNCTION log_update_t1() RETURNS trigger AS
$triggerupdatet1$
BEGIN
insert into csql_log_int (tablename, pkid, operation,
cacheid) values ('t1', OLD.f1, 2, 1);
insert into csql_log_int (tablename, pkid, operation,
cacheid) values ('t1', NEW.f1, 1, 1);
RETURN NEW;
END;
$triggerupdatet1$ LANGUAGE plpgsql;

create trigger triggerupdatet1
AFTER UPDATE on t1
FOR EACH ROW
EXECUTE PROCEDURE log_update_t1();

CREATE FUNCTION log_delete_t1() RETURNS trigger AS
$triggerdeletet1$
BEGIN
insert into csql_log_int (tablename, pkid, operation,
cacheid) values ('t1', OLD.f1, 2, 1);
RETURN NEW;
END;
$triggerdeletet1$ LANGUAGE plpgsql;

create trigger triggerdeletet1
AFTER DELETE on t1
FOR EACH ROW
EXECUTE PROCEDURE log_delete_t1();

```

Note: Trigger name ends with the table name. Replace 't1' in the above script to the cached table name and 'f1' to the primary key fieldname of the cached table.

After editing the trigger.psql file as per your need, you shall execute it by running the below command under 'postgres' login.

Note: Trigger name ends with the table name. Replace 'p1' in the above script to the cached table name and 'f1' to the primary key fieldname of the cached table as per your table and field name.

After a successful editing of trigger.sql file, now its ready for execution. Follow the below command to execute the trigger which is present under your csq root directory.

```
$ isql psql <trigger.psql
```

5.3.6. Steps for bi-directional cache

After set up the configurations, you can now cache the t1 table from Postgres and use bi-directional cache. That means you can make modification to original table, which is present in Postgres, and they are automatically propagated to the cache table in CSQL.

Follow the [section 4.3.3](#) for bi-directional caching in action. Note that you should use Postgres DSN and tools instead of MySQL.

6. Cache Option for Oracle

This chapter shows a step-by-step method for caching Oracle database into CSQL and demonstrates automatic update propagation to Oracle database.

6.1. Setting up CSQL and Oracle

6.1.1. Pre-requisites

The “unixODBC” package and “Oracle” have to be installed in your system

6.1.2. Oracle Database

Oracle environment script needs to be run before starting the server. This file will be present in \$ORACLE_HOME/bin and name of the script is “oracle_env.sh”

Usually this library is named as libsqlora.so and will be present in lib directory of ORACLE installation directory.

6.1.3. UnixOdbc

The driver library of unixODBC is named as odbc.so and will be available at /usr/lib or /usr/local/lib directory.

6.1.4. Configure –odbc. ini file

Below odbc.ini file contains the properties for DSN (Data Source Name), named oracle. You should append these lines in your odbc.ini file at your home directory (~/.odbc.ini)

```
[oracle]
Description = Oracle ODBC driver for Oracle 10g
Driver =
/usr/lib/oracle/xe/app/oracle/product/10.2.0/server/lib/lib
sqora.so.10.1
DSN = oracle
ServerName =
UserId=
```

This guide assumes that the driver library is present in /usr/lib directory. If it is installed in some other directory change the DRIVER parameter value accordingly.

Check the connection using DSN name with isql tool.

```
$isql oracle
It should display the following
+-----+
| Connected! |
|           |
| sql-statement |
| help [tablename] |
| quit       |
|           |
+-----+
SQL>
```

If it does not show the above message then try with as follow

```
$isql oracle <user> <password>
It should display the following
+-----+
| Connected! |
|           |
| sql-statement |
| help [tablename] |
| quit       |
|           |
+-----+
SQL>
```

NOTE: You can take the help of script file to generate `odbc.ini` and `odbcinst.ini` which is present in `$ORACLE_HOME/odbc/utl/odbc_update_ini.sh`

6.2. Unidirectional

Any updates on cache table are automatically propagated to the target database.

6.2.1. Configure `csql.conf` file

You must have to modify some cache variables in `csql.conf` file. [Refer 3.5.2.3 for Cache Section variables](#). Cache section variables should contain below specified value.

```
CACHE_TABLE=true
CACHE_ID=1
DSN = oracle
USER = scott
```

```
PASSWORD = tiger
CACHE_MODE=SYNC
ENABLE_BIDIRECTIONAL_CACHE=false
CACHE_RECEIVER_WAIT_SECS=0
TABLE_CONFIG_FILE=/tmp/csql/csqltable.conf
```

Note : Put Oracle username and password instead of scott and tiger.

6.2.2. Steps for unidirectional cache

This section describes how to create a unidirectional cache that caches the contents of a single table from Oracle database.

Follow the section 4.2.2.1. for discussion on “Cache MySQL data”. The step-by-step procedure has been discussed for caching a table and to make some update and see caching in action. The difference is only that Oracle will be placed instead of MySQL when you follow those sections.

6.3. Bi-directional Cache

Bi-directional caching is implemented using triggers of the target database. This requires additional tables and triggers, which needs to be installed on the tables in target database.

Sample trigger code is available in the file `trigger.osql` under the CSQL root directory

6.3.1. Configuring csql.conf file

For bi-directional cache, some changes are required in cache section parameters of `csql.conf` file. Refer 3.5.2.3. for Cache Section variables. Cache section variables should contain below specified value

```
CACHE_TABLE=true
CACHE_ID=1
DSN = oracle
USER = scott
PASSWORD = tiger
CACHE_MODE=SYNC
ENABLE_BIDIRECTIONAL_CACHE=true
CACHE_RECEIVER_WAIT_SECS=10
TABLE_CONFIG_FILE=/tmp/csql/csqltable.conf
```

The only difference between unidirectional and bi-directional configuration is the value of `ENABLE_BIDIRECTIONAL_CACHE` parameter, which is set to true for bi-directional caching.

6.3.2. Setting up Oracle

After configuring `csql.conf` file, a log table needs to be created in target database to hold the operation log records for all the cached tables. You also need to install triggers on cached table for INSERT, UPDATE and DELETE operation.

6.3.3. Create a table in Oracle for Caching

```
$isql oracle
SQL> CREATE TABLE t1 ( f1 int, f2 char(196), primary
key(f1 ) ) ;
```

Now insert some records and commit the changes :

```
SQL> INSERT INTO p1 VALUES( 1, 'Hi' ) ;
SQL> INSERT INTO p1 VALUES( 2, ' All' ) ;
SQL> COMMIT ;
```

6.3.4. Create `csql_log_int` table in Oracle

As mentioned before, for propagating direct updates from Oracle to cache, we need to create log table to hold operation logs of all cached tables.

Use the below statements for creating log table:

```
$isql oracle scott tiger
SQL> create table csql_log_int(tablename char(64), pkid
int, operation int,cacheid int, id int not null );
SQL> CREATE SEQUENCE csql_id MINVALUE 1 INCREMENT BY 1
START WITH 1 ;
```

6.3.5. Execute trigger

Lets say for cached table 't1' having primary key field 'f1', following is the format for creating the triggers in Oracle database. A sample file (`trigger.osql`) is available at the CSQL root directory; you shall modify that with your cached table name and its primary key field.

```
CREATE OR REPLACE TRIGGER triggert1
AFTER insert or update or delete on t1
FOR EACH ROW declare rid PLS_INTEGER;
begin
```

```

SELECT CSQL_ID.NEXTVAL INTO rid FROM dual;
if inserting then insert into csq_log_int values
('t1',:new.f1,1,1,rid);
end if;

if updating then insert into csq_log_int values
('t1',:old.f1,2,1,rid); SELECT CSQL_ID.NEXTVAL INTO rid
FROM dual; insert into csq_log_int values
('t1',:new.f1,1,1,rid); end if;

if deleting then insert into csq_log_int values
('t1',:old.f1,2,1,rid);
end if;
end;

```

Note: Trigger name ends with the table name. Replace 't1' in the above script to the cached table name and 'f1' to the primary key fieldname of the cached table.

After editing the trigger. osql file as per your need, you shall execute it by running the below command.

```
$ isql oracle <trigger.osql
```

After executing this file the trigger would be executed on the table to whom you are going to cache.

6.3.6. Steps for bi-directional cache

After set up the configurations, you can now cache the t1 table from Oracle and use bi-directional cache. That means you can make modification to original table, which is present in Oracle, and they are automatically propagated to the cache table in CSQL.

Follow the section 4.3.3 to see bi-directional caching in action. Note that you should use Oracle DSN and tools instead of MySQL.

7. Multi Node Cache

Applications can scale up by employing multiple cache nodes to cache tables. [Refer Section 2.3](#), “Deployment Scenarios” for more information on this.

7.1. Configure CSQL

The `CACHE_ID` variable value must be unique for each node (host containing CSQL installation). For the first node the value is set up as 1 and for second node the value is 2. This number can go as per your node creation with unique numbers.

FIRST NODE :

```
CACHE_TABLE=true
CACHE_ID=1
DSN=myodbc3
USER=lakshya
PASSWORD=lakshya123
CACHE_MODE=SYNC
ENABLE_BIDIRECTIONAL_CACHE=true
CACHE_RECEIVER_WAIT_SECS=10
TABLE_CONFIG_FILE=/tmp/csql/csqltable.conf
```

SECOND NODE :

```
CACHE_TABLE=true
CACHE_ID=2
DSN=myodbc3
USER=lakshya
PASSWORD=lakshya123
CACHE_MODE=SYNC
ENABLE_BIDIRECTIONAL_CACHE=true
CACHE_RECEIVER_WAIT_SECS=10
TABLE_CONFIG_FILE=/tmp/csql/csqltable.conf
```

7.2. Configure MySQL

To set multiple Bi-Directional caching,

- First create the table in MySQL to hold the log records.
- Triggers are installed in the target database for all the DML operations on cached table to generate log entries in the log table.

The below statements is executing with mysql tool or isql tool.

Create Log Table

```
mysql> CREATE TABLE csql_log_int (tablename CHAR(64), pkid
INT, operation INT, cacheid INT, id INT NOT NULL UNIQUE
AUTO_INCREMENT) engine='innodb';
```

Create Trigger

Lets say for a cached table p1 with primary key f1 ,write a trigger(trigger. sql) as below.

```
use test;
drop trigger if exists triggerinsertp1;
drop trigger if exists triggerupdatep1;
drop trigger if exists triggerdeletep1;
DELIMITER |
create trigger triggerinsertp1
AFTER INSERT on p1
FOR EACH ROW
BEGIN
Insert into csql_log_int (tablename, pkid,
operation,cacheid )values ('p1', NEW.f1, 1,1);
Insert into csql_log_int (tablename, pkid,
operation,cacheid )values ('p1', NEW.f1, 1,2);
End;

create trigger triggerupdatep1
AFTER UPDATE on p1
FOR EACH ROW
BEGIN
Insert into csql_log_int (tablename, pkid, operation,
cacheid ) values ('p1', OLD.f1, 2,1);
Insert into csql_log_int (tablename, pkid,
operation,cacheid ) values ('p1', NEW.f1, 1,1);
Insert into csql_log_int (tablename, pkid, operation,
cacheid ) values ('p1', OLD.f1, 2,2);
Insert into csql_log_int (tablename, pkid,
operation,cacheid ) values ('p1', NEW.f1, 1,2);
End;

create trigger triggerdeletep1
AFTER DELETE on p1
FOR EACH ROW
BEGIN
Insert into csql_log_int (tablename, pkid, operation,
cacheid )values ('p1', OLD.f1, 2,1);
```

```
Insert into csql_log_int (tablename, pkid, operation,
cacheid )values ('p1', OLD.f1, 2,2);
End;
|
```

Note that in above triggers, for each operation it inserts two logs into the log table, one for cache node-1 and another for cache node-2. After execution of the below command, triggers are installed on the p1 table and we are ready to cache the MySQL table and do some operations.

```
$ mysql -u root -p <trigger.sql
```

7.3. Configure multiple nodes

Install CSQL in two machines (say host-1 and host-2) and set the csql.conf file as specified in section 7.1.1. Make sure that CACHE_ID parameter at host-1 is set to 1 and CACHE_ID parameter at host-2 contains CACHE_ID set to 2.

7.3.1. Load MySQL table to Cache

Now run the csqlserver and use the cachetable tool for caching MySQL table into cache on host-1(cache node-1)

Create another terminal make sure that you are in csql root directory and set up the environment using the command `./setupenv.ksh`.

```
$ cachetable -U root -P manager -t p1
```

Now check the contents of the cache table which is present in Cache using CSQL tool.

```
CSQL>SELECT * FROM p1;
```

```
-----
f1  f2
-----
1   Hi
2   All
```

Refer the section 10.1. for syntax and uses in details about cachetable tool and its arguments.

Perform the above said operations (start csqlserver and run cachetable to cache 'p1' table) on host-2.

7.3.2. MySQL – to – CSQL update propagation

Insert some records in p1 table which is present in MySQL and the record should be propagated to the MySQL as well as other host (host-2) within the time interval specified in CACHE_RECEIVER_WAIT_SECS configuration variable. After this interval, query the table in both CSQL instances and MySQL database, you should get the same records in both the cache instances and target database.

INSERT RECORDS IN MYSQL

```
$isql myodbc3;

SQL> insert into p1 values(3, 'NoOne');
SQLRowCount returns 1
SQL> insert into p1 values(4, 'EveryOne');
SQLRowCount returns 1
```

two more records now are in inserted.

QUERY 'P1' TABLE IN CSQL

```
CSQL>select * from p1;

-----
      f1      f2
-----
      1      Hi
      2      All
      3      NoOne
      4      EveryOne
```

If we run the above query on host-2, it will also contain 4 records. If you perform operations on any cache node, it will be automatically propagated to target database as well as all the other cache nodes.

8. CSQL Cache Modes

In this section, different cache modes have been discussed. Applications can use any of the modes as per their requirements.

In the below examples we will go to cache the table 't1' which is present in MySQL , Postgres or Oracle database.

The records present in the table t1 are:

```
SQL>SELECT * FROM t1;

+-----+-----+
| f1    | f2                |
+-----+-----+
| 1     | Hello World      |
| 2     | Hi                |
| 3     | Hi                |
| 4     | No One            |
| 5     | Everyone          |
+-----+-----+
```

8.1. Updateable Cache

In this mode, application can perform all DML operations on the cached table. Updates on the cached tables are automatically applied in the target database. This is the default-caching mode in CSQL.

8.2. Read only Cache

The READ ONLY CACHE enforces a caching behavior in which applications only can read the data from the cache table not update any data. When partial records are cached in CSQL, then the Cache behaves as read only.

There is two types of partial caching which comes under READ ONLY Cache,

- Record Level (Horizontal Cache)
- Column Level (Vertical Cache)

Record Level Cache:

Use -c option to specify the condition for partial caching. Records which satisfy the condition will be cached in CSQL from the target database.

```
$cachetable -U root -P manager -t t1 -c "f1 < 3"
```

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

```
-----  
      f1      f2  
-----  
      1      Hello World  
      2      Hi
```

Column Level Cache :

Use `-f` option for specifying the field name list to retrieve only values of those fields from the original table

```
$cachetable -U root -P manager -t t1 -f "f1"
```

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

```
-----  
      f1  
-----  
      1  
      2  
      2  
      3  
      4  
      5
```

It caches only record values for 'f1' field in table 't1' from the target database.

8.3. Direct Cache

In this cache mode, updates to cache directly go to the target database table and will not update the cached table. This mode is useful when there is auto increment field in cached table. Use `-D` option to cache table in this mode

```
$cachetable -t T1 -D
```

```
CSQL>select * from T1;
```

```
-----  
      f1      f2  
-----  
      1      Hello World  
      2      All  
      3      Hello
```

INSERT ONE RECORD IN CACHE TABLE AND QUERY THE TABLE

```
$csql -g
CSQL> insert into T1 values ( 4, 'Hi');
$cachetable -t T1 -D

CSQL>select * from T1;
-----
          f1          f2
-----
          1          Hello World
          2           All
          3           Hello
```

Though we inserted one more record, it is not inserted in cache table; rather, the record is inserted in target database.

QUERY THE T1 TABLE IN MYSQL

```
mysql> select * from T1;
+-----+-----+
| f1    | f2                |
+-----+-----+
|     1 | Hello World      |
|     2 | All              |
|     3 | hello            |
|     4 | Hi               |
+-----+-----+
```

8.4. Asynchronous cache

An asynchronous cache mode enforces the same caching behavior as Synchronous cache mode, in which cached data is updated in CSQL and propagated to target database. This mode provides better response times than Synchronous because the CSQL commit occurs asynchronously for the target database commit. This allows an application to continue executing without having to wait for the target database transaction to commit. You can also perform updates on cached tables in Async mode even when the target database is down. When the target database restarts, the updates will be applied to the target database automatically.

8.5. Synchronous cache

A Synchronous cache enforces a caching behavior in which cached data is updated in CSQL and also propagates to target database. Synchronous mode gives high cache consistency and is suited for real time applications. When target database goes down, transactions fail till the target database restarts. This is the default mode for propagating updates to target database.

8.6. Unidirectional Cache

The default caching in CSQL is unidirectional caching, which means all updates (INSERT, UPDATE, DELETE) on cached tables will be automatically propagated to target database.

8.7. Bi-directional Cache

CSQL supports bi-directional caching in which, direct updates on target database are propagated to CSQL cache automatically.

9. Working with CSQL Gateway

CSQL Cache allows access to tables which are stored in target database and are not cached in CSQL MMDB. ODBC and JDBC applications can connect using the gateway option to get his behavior.

9.1. Using CSQL tool

In another terminal run csql tool with -g option. This creates an isql session which acts as gateway to csql and target database.

```
$ csql -g
CSQL>
```

You can also retrieve records in tables, which are not cached in CSQL and are present in target database

```
$ csql -g
CSQL>select * from t3;

It will display

24083:3086153424:DatabaseManagerImpl.cxx:599:Table not
exists t3
24083:3086153424:SelStatement.cxx:245:Unable to open the
table:Table not exists
-----
          f1          f2
-----
          103          103
```

First it displays that the table is not present in CSQL, then it checks with target database whether the table is present there and if present it retrieves the records from target database.

9.2. Using ODBC Interface

ODBC DSN name should specify the mode to connect through gateway. Sample code snippet to connect to gateway is given below.

```
void checkrc (int rc, int line)
{
    if (rc)
    {
        printf("Failed with rc:%d at line:%d\n", rc, line);
    }
}
```

```

        exit (1);
    }
}
int main()
{
    SQLHENV henv;
    SQLHDBC hdbc;
    SQLHSTMT hstmt;
    int rc = SQLAllocHandle (SQL_HANDLE_ENV,
                            SQL_NULL_HANDLE, &henv);
    checkrc (rc, __LINE__);
    SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                 (void *) SQL_OV_ODBC3, 0);

    rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
    checkrc (rc, __LINE__);
    rc = SQLConnect (hdbc, (SQLCHAR *)
"DSN=test;MODE=Gateway;SERVER=localhost;PORT=5678;",
                    SQL_NTS,
                    (SQLCHAR *) "root",
                    (SQLSMALLINT) strlen ("root"),
                    (SQLCHAR *) "manager",
                    (SQLSMALLINT) strlen (""));

    //Place your statements here to
    //perform DML operations on cached
    //and non-cached tables

    SQLDisconnect (hdbc);
}

```

9.3. Using JDBC Interface

JDBC URL needs to be modified to “jdbc::gateway” to connect through gateway. Sample code snippet to connect to gateway is given below

```

import java.sql.*;
public class gwexample
{
    public static void main(String[] args)
    {
        try
        {
            Class.forName("csql.jdbc.JdbcSqlDriver");

```

```

Connection con =
DriverManager.getConnection("jdbc:gateway", "root",
"manager");

Statement cStmt = con.createStatement();
PreparedStatement stmt = null;

//Place your statements here to
//perform DML operations on cached
//and non-cached tables

con.close();

    }catch(Exception e) {
        System.out.println("Exception in Test: "+e);
        e.printStackTrace();
    }
}
}

```

9.4. Using SQL Interface

From SQL API, use CSqlGateway flag to create connection and statement objects using SqlFactory class.

```

AbsSqlConnection *con = SqlFactory::
                        createConnection(CSqlGateway);

rv = con->connect("root", "manager");

if (rv != OK) return 1;

AbsSqlStatement *stmt = SqlFactory::
                        createStatement(CSqlGateway);

stmt->setConnection(con);

//Place your statements here to
//perform DML operations on cached
//and non-cached tables

stmt->disconnect();

```

10. Tools for Cache

The cachetable tool is used when we cache a table from target database into CSQL. This needs to be invoked when the csqserver process is running.

Cacheverify tool is used to display the missing records in the specified cached table either in CSQL or in target database, if any.

10.1. cachetable

cachetable is a tool to cache the table from the target database into CSQL. This needs to be invoked when the csqserver process is running.

```
Example : cachetable -t Employee
```

In this example, cachetable is used with `-t` argument to cache the Employee table from target database. `-t` argument is mandatory for cachetable tool. Except this one all other arguments are optional, you can choose and use them as per the application requirement.

Syntax :

```
cachetable  
  
[-U username]  
[-P passwd]  
-t tablename  
[-c "condition"]  
[-f "field names"]  
[-p fieldname]  
[-D]  
[-R]  
[-s]  
[-r]
```

-U username

This is a mandatory argument. Username is required for authentication.

-P passwd

This is also a mandatory argument. Password for the above username to connect to the csq.

-t tablename

Table name to be cached in csq from target db.

-c "condition"

At the time of cache a table you can cache with condition value. This is optional argument.

-f "fieldnames"

Field name list for doing partial caching with specific fields

-D

Enable direct access option to target database

-R

Recover all cached tables from the target database.

-s

Load only the records from target db. Assumes table is already created in csql.

-r

Reload the table. get the latest image of table from target db.

-u

Unload the table. if used with -s option, removes only records and preserves the schema.

Unload the cache table using -u option

```
$ cachetable -U root -P manager -t t1 -u
```

Note : It is not permissible to use DROP statement to drop the cache table.

Cache records which satisfy condition

```
$cachetable -U root -P manager -t t1 -c "f1<3"
```

Cache only specified fields

```
$cachetable -U root -P manager -t t1 -f "f1"
```

Reloading the Cache table

```
$cachetable -t t1 -r
```

10.2. cacheverify

cacheverify tool displays the missing records in the specified cached table either in CSQL or in target database, if any. This tool should be used only when CACHE_TABLE option is set in the csql.conf file.

Syntax:

```
cacheverify
```

```
[-U username]
```

```
[-P passwd]
```

```
-t tablename
```

```
[-p]
```

```
[-f]
```

-U username

username to connect with csql.

-P passwd

password for the above username to connect with csql.

-t tablename

cached table name in csql from target db.

-p

verification at primary key field level

-f

verification at record level

?

help

Verification based on number of records

```
$cacheverify -t t1
```

```
Number of Records:
```

Data	CSQL	TargetDB
No. Of Records	4	4

Find missing records

```
$cacheverify -t t1 -p
```

Number of Records:

Data	In CSQL	In TargetDB
No. Of Records	3	3

Primary key field name is 'f1'

Missing Records: Marked by 'X'

Primary Key	In CSQL	In Target DB
1	X	
4		X

The above output shows that the record having value 1 is missing from CSQL and the record having value 4 is missing from Target DB.

Find mismatching field values

```
$ cacheverify -t t1 -f
```

Number of Records:

Data	In CSQL	In TargetDB
No. Of Records	3	3

Primary key field name is 'f1'

Missing Records: Marked by 'X'

Primary Key	In CSQL	In Target DB
1	X	
4		X

Inconsistent Records for the same key:

```
-----+-----+-----+-----
```

Primary Key	Field Name	CSQL	Trgt DB
2	t1.f2	World	Hi All
3	t1.f2	Hi	India

This tool is useful at the time of caching happen. cacheverify tool directly helps the users to detect the missing records as we discussed in above example.