



User Manual

**CSQL**  
Main Memory Database Cache

# Table of Contents

<b>1. BEFORE YOU START.....</b>	<b>5</b>
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
<b>2. OVERVIEW OF CSQL MMDB CACHE SYSTEM.....</b>	<b>8</b>
2.1. WHAT IS MMDB AND DRDB.....	8
2.2. ADVANTAGES OF CSQL DBMS.....	8
2.3. CSQL IS OPEN SOURCE PRODUCT.....	8
2.4. CSQL WEB SITE.....	8
2.5. BASIC FEATURES.....	9
2.5.1. MAIN FEATURES OF CSQL.....	9
2.5.2. STANDARDS COMPLIANCE.....	11
2.5.2.1. SQL 92 ENTRY LEVEL.....	11
2.5.2.2. ODBC LEVEL 2.....	12
2.5.2.3. JDBC 2.0.....	12
2.6. FEATURE OVERVIEW.....	12
2.6.1. ODBC AND JDBC INTERFACES.....	12
2.6.2. DATABASE CONNECTIVITY.....	12
2.6.3. CACHING OPTION FOR DISK BASED DATABASES.....	13
2.7. WHAT IS NEW IN CSQL 2.4.....	13
2.8. WHO CAN USE CSQL.....	13
2.9. CSQL SELECTIVE INFORMATION REFERENCES.....	13
2.9.1. MAILING LISTS.....	13
2.9.2. FORUMS.....	14
<b>3. CSQL INSTALLATION.....</b>	<b>15</b>
3.1. WHERE TO FIND CSQL.....	15
3.2. OS PLATFORM SUPPORT.....	15
3.3. INSTALLATION PREREQUISITES.....	15
3.4. DIRECTORY LAYOUT AND FILES.....	15
3.5. RUNNING EXAMPLES.....	16
3.6. HOW TO CONFIGURE CSQL.....	16

3.6.1.	SETTING UP ENVIRONMENTAL VARIABLES .....	17
3.6.1.1.	CSQL_INSTALL_ROOT .....	17
3.6.1.2.	CSQL_CONFIG_FILE .....	17
3.6.1.3.	LD_LIBRARY_PATH .....	17
3.6.1.4.	PATH .....	17
3.6.1.5.	CLASSPATH .....	18
3.6.2.	CONFIGURATION FILE (CSQL.CONF).....	18
3.6.2.1.	SERVER SECTION VARIABLES .....	18
3.6.2.2.	CLIENT SECTION VARIABLES .....	20
3.6.2.3.	CACHE SECTION VARIABLES .....	20
<b>3.7.</b>	<b>STARTING AND STOPPING CSQLSERVER .....</b>	<b>22</b>
<b>3.8.</b>	<b>TROUBLESHOOTING .....</b>	<b>23</b>
3.8.1.	ERRORS WHILE RUNNING CSQLSERVER .....	23
3.8.1.1.	CSQLSERVER COMMAND NOT FOUND.....	23
3.8.1.2.	UNABLE TO CREATE THE LOG FILE .....	23
3.8.2.	ERRORS WHILE RUNNING CLIENT .....	24
<b>4.</b>	<b><u>GETTING STARTED WITH CSQL.....</u></b>	<b>25</b>
<b>4.1.</b>	<b>CLIENT TOOL FOR COMMUNICATING WITH CSQL DATABASE .....</b>	<b>25</b>
<b>4.2.</b>	<b>USING A DATABASE.....</b>	<b>25</b>
4.2.1.	CREATING NEW TABLE .....	25
4.2.2.	INSERTING DATA INTO A TABLE .....	26
4.2.3.	QUERING A TABLE .....	27
4.2.4.	JOIN BETWEEN TABLES.....	28
4.2.5.	AGGREGATE FUNCTIONS .....	29
4.2.6.	UPDATE.....	29
4.2.7.	DELETE .....	29
<b>4.3.</b>	<b>GETTING INFORMATION ABOUT METADATA USING CATALOG TOOL.....</b>	<b>29</b>
<b>4.4.</b>	<b>SHOW STATEMENT.....</b>	<b>30</b>
<b>4.5.</b>	<b>USING CSQL IN BATCH MODE .....</b>	<b>31</b>
<b>4.6.</b>	<b>CONSTRAINTS.....</b>	<b>31</b>
4.6.1.	NOT-NULL .....	32
4.6.2.	UNIQUE.....	32
4.6.3.	PRIMARY KEY CONSTRAINT.....	33
<b>4.7.</b>	<b>INDEX.....</b>	<b>34</b>
4.7.1.	HASH INDEX.....	34
4.7.2.	TREE INDEX.....	35
4.7.3.	COMPOSITE INDEX .....	36
4.7.4.	UNIQUE INDEX AND PRIMARY INDEX .....	36
<b>5.</b>	<b><u>DATA TYPES.....</u></b>	<b>38</b>
<b>5.1.</b>	<b>NUMERIC TYPES .....</b>	<b>38</b>
5.1.1.	INTEGER TYPES .....	38

5.1.2.	FLOATING POINT TYPES .....	38
<b>5.2.</b>	<b>CHARACTER TYPES .....</b>	<b>39</b>
<b>5.3.</b>	<b>BINARY DATA TYPES .....</b>	<b>39</b>
<b>5.4.</b>	<b>DATE/TIME TYPES .....</b>	<b>39</b>
5.4.1.	DATE .....	40
5.4.2.	TIME .....	40
5.4.3.	TIME STAMP.....	40
<b>6.</b>	<b><u>FUNCTION AND OPERATORS .....</u></b>	<b><u>41</u></b>
<b>6.1.</b>	<b>LOGICAL OPERATORS .....</b>	<b>41</b>
<b>6.2.</b>	<b>COMPARISON OPERATORS.....</b>	<b>42</b>
<b>6.3.</b>	<b>MATHEMATICAL OPERATORS .....</b>	<b>42</b>
<b>6.4.</b>	<b>AGGREGATE FUNCTIONS.....</b>	<b>43</b>
<b>6.5.</b>	<b>SQL OPERATORS.....</b>	<b>44</b>
6.5.1.	IN OPERATOR .....	44
6.5.2.	BETWEEN OPERATOR.....	44
6.5.3.	LIKE OPERATOR .....	44
<b>7.</b>	<b><u>DATABASE RECOVERY .....</u></b>	<b><u>45</u></b>
<b>7.1.</b>	<b>AUTOMATIC RECOVERY .....</b>	<b>45</b>
<b>7.2.</b>	<b>ARCHIVE THE DATABASE.....</b>	<b>45</b>
<b>7.3.</b>	<b>RESTORE THE DATABASE .....</b>	<b>45</b>
<b>8.</b>	<b><u>TOOL REFERENCE .....</u></b>	<b><u>46</u></b>
<b>8.1.</b>	<b>CSQL.....</b>	<b>46</b>
<b>8.2.</b>	<b>CATALOG.....</b>	<b>46</b>
<b>8.3.</b>	<b>CSQLDUMP.....</b>	<b>48</b>

# 1. Before you start

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

## 1.1. What is CSQL

CSQL Main Memory Database is an easily accessible and powerful database management system, which includes the CSQL Main Memory Database, and Cache to MySQL, PostgreSQL and Oracle disk based database management system.

It typically resides in the middle tier alongside 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 houses use CSQL to store current stock price information, which they need to quickly 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 for a wide variety of applications ranging from single-user system to enterprise-wide multi-user installations with hundreds of concurrent connections.

## 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.

### 1.2.2. CSQL Cache

CSQL Cache option enables customers to significantly improve application response times and throughput. Based on the CSQL MMDB, CSQL Cache delivers a real-time, dynamic, updatable cache for frequently accessed data in MySQL, Postgres and Oracle database.

For performance critical applications in industries such as Telecom, Process Control, Airline Reservation, Stock Market, HealthCare, etc.,

the CSQL Cache option delivers application response times in microseconds by bringing the frequently accessed data closer to the application, and by executing SQL requests in the CSQL MMDB.

We found that CSQL and its associated suite of products to be between 10 and 40 times faster than traditional database system.

Visit Benchmark result page at <http://www.csqldb.com> for more information.

### 1.3. Information in this Manual

This manual describes CSQL concepts, components, and basic usages and 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 typographic and syntax rules have been maintained throughout the guide.

#### 1.4.1. Typographic Rule

This manual uses the following typographic rule.

Table 1.1. Typographic Rule

<b>FORMAT</b>	<b>USED FOR</b>
Main Memory Database	This font is used for ordinary text.
<code>SELECT * FROM T1</code>	This font is used for SQL Statements and Program code.
<code>UNIQUE</code>	This font with uppercase letter indicates SQL keywords and data types.
<code>csql.conf</code>	This font indicates file name and path.

<code>build.ksh</code>	This font is used for command lines
<code>SQLFetch()</code>	This font is used for function names
<code>Java.Sql.Statement</code>	This font is used for interface, classes and header files names.
<code>Test</code>	This font is used for directory name.

### 1.4.2. Syntax Annotation

This manual uses the following syntax annotation conventions.

Table 1.2. Syntax Annotation

<b>FORMAT</b>	<b>USED FOR</b>
<code>[]</code>	This square bracket indicates that item in command line is optional
<code>{ }</code>	Curly braces indicates that one must choose one of the items separated by a vertical bar(   ) in a command line.
<code> </code>	A vertical bar separates arguments
<code>...</code>	An ellipsis indicates that arguments can be repeated several times
<code>.</code>	This indicates that continuation of previous lines of code
<code>\$</code>	This dollar sign indicates the Linux prompt.
<code>#</code>	The pound sign indicates the Linux root prompt.

## 2. Overview of CSQL MMDB Cache System

This chapter is intended for features, roadmap, standard compliance, information references, and advantage of CSQL.

### 2.1. What is MMDB and DRDB

The difference between MMDBMS and DRDBMS is their data storage. The MMDBMS is also a kind of RDBMS where data is stored completely in main memory. So the user can access data using the standard query languages and the MMDBMS has the typical features of RDBMS such as controlling data inconsistency, concurrency controls, etc.

By managing data in memory, and optimizing data structures and access algorithms accordingly, database operations execute with maximum efficiency, achieving dramatic gains in responsiveness and throughput, even compared to a fully cached DRDBMS.

### 2.2. Advantages of CSQL DBMS

- It keeps the data in main memory rather than disk.
- It is times faster than any disk based database system.
- No buffer manager overhead is present since all the records are stored in main memory.
- The data structure and algorithms are targeted for memory access.
- Portability via JDBC, ODBC and Gateway interface.
- A powerful proprietary, Storage engine that stores and manages the data in memory optimized for real time queries.
- A high performance proprietary SQL Engine.

### 2.3. CSQL is Open Source Product

CSQL is an Open Source Software at SourceForge([www.sourceforge.net](http://www.sourceforge.net)), world's largest development and download repository of open source code and applications. SourceForge is an Independent, non-profit initiative whose mission is to build user's trust and confidence in the Software Product by keep it open to the users.

### 2.4. CSQL Web Site

By the below link, you may be getting information about the product, like downloading , documentation and other informations .

- [www.csqldb.com](http://www.csqldb.com)

## 2.5. Basic Features

### 2.5.1. Main features of CSQL

The following list draws some of the important features of the CSQL Database.

#### Interoperability and Internals

- Written in C and C++
- A powerful, Storage Engine that store and manage data completely in memory.
- High performance SQL Engine optimized for real time queries.
- Works on Linux Platform.
- APIs for C, C++, and Java are available. *Refer CSQL Programmers Guide for APIs.*
- Multi threaded access to database.
- Provides transactional and non-transactional storage engine.
- Tree Index for faster range look up.
- Hash Index for fast point look up.
- Very fast Joins.
- ACID Compliant (Atomicity, Consistency, Isolation and Durability Properties).
- Highly Concurrent
  - Uses lock free data structures and reduces contention by multifold.
  - Row level locking.
- Multi Granular Locking
  - Database level locking
  - Table level locking
  - Row level locking
- Isolation level support
  - Read uncommitted
  - Read committed
  - Read repeatable
- Constraints
  - Primary key
  - Not Null
  - Unique

- Multi user
- Fault tolerance
  - process clean up for resources in case of application crash
- Archive/Restore
- Primitive ODBC Driver
- Primitive JDBC Driver

## Data Type

- All primitive
  - int
  - real
  - tinyint
  - long
  - smallint
  - double
  - binary
  - char
- Non-Primitive
  - date
  - time
  - timestamp

## Statements

- All arithmetic and logical operator support WHERE clauses of queries.  
For Example:

```
CSQL > select  ename, address
from emp
where  eid >1000 and sal >2000 ;
```

- Support for GROUP BY clause, support for aggregate function COUNT ( ), AVG ( ), MIN ( ), MAX ( ), and SUM ( ).
- Support for INNER JOINS.
- SHOW statement to retrieve table information from the database.

## Connectivity Mechanism

- Client can connect to the CSQL server using proprietary SQLAPI. *Refer the section 3. in CSQL's Programmers Guide*
- Primitive ODBC Driver (support for C, C++ client programs that use Open Database Connectivity Connection.
- Primitive JDBC Driver (support for Java client programs that use JDBC Connection.

## Client tools

- CSQL Interactive tool for SQL statements.
- catalog tool is used to retrieve metadata information of the database such as tables,, index, mutex, etc.
- csqldump tool for backup and restore.

## 2.5.2. Standards Compliance

This section describes about Standard compliance of ANSI/ISO, ODBC, and JDBC Standards.

- SQL 92 Entry Level
- ODBC level 2
- JDBC 2.0

### 2.5.2.1. SQL 92 Entry Level

The SQL 92 standard was accepted in 1992. At the time of writing, SQL 92 is not fully implemented by CSQL. One of our main goals with the product is to continue to work toward compliance with the SQL standard, but without sacrificing speed or reliability.

Though it supports minimal feature set, it shall be used as stand alone main memory database system for your applications. Most of the application may not require rich feature set; rather they require minimal feature set with ultra fast response.

The below list is some of the features of SQL 92 Entry Level implemented by CSQL.

- Support for dynamic SQL
- Support for Schema definition and manipulation statements
- Support for INNER JOIN
- Support for Data Types DATE, TIME, TIMESTAMP with all Primitive types
- Support for the qualifier '\*' in the select lists
- Case Insensitivity for Identifiers
- Table columns involved in a PRIMARY KEY constraint are no longer required to be explicitly declared as NOT NULL.

### **2.5.2.2. ODBC Level 2**

The CSQL ODBC Driver supports ODBC version 2.5 and 3.0 Level 2 only. The Level 2 is all that is necessary to do standard operations and your application might want to limit up to level 2 ODBC calls. Refer CSQL Programmers Guide to know in details about ODBC APIs which comes under Level 2.

### **2.5.2.3. JDBC 2.0**

The CSQL JDBC Driver supports JDBC 2.0 APIs and few higher version APIs are also implemented. Its functionality includes parameters, selects, batch updates, programmatic inserts, updates, deletes, and updates.

## **2.6. Feature Overview**

This section provides brief overview on main features like ODBC, JDBC and Cache connection for disk based databases.

### **2.6.1. ODBC and JDBC Interfaces**

CSQL supports ODBC and JDBC. Though ODBC and JDBC are standard CSQL interfaces it operates directly with the database engine. CSQL supports these APIs that are both fully compliant with the standards and tuned for maximum performance in the CSQL environment.

### **2.6.2. Database Connectivity**

Application written in C, C++ or Java can connect to the database directly through the drivers in Embedded (Server and Client running in same machine) or Client/Server mode. These connection options allow users to choose the best performance functionality for

their applications. Direct driver connections are fastest for ODBC and JDBC applications.

### **2.6.3. Caching option for Disk based Databases**

Caching option for MySQL, Postgres and Oracle is an option to the CSQL Main Memory Database that creates a real-time, updateable cache for target database.

## **2.7. What is new in CSQL 2.4**

- Lock Free Data structures
- Durable Transactions and Recovery
- Client/Server connection mode
- Asynchronous update propagation to target database
- Real time multi node caching for clusters
- Performance fixes

## **2.8. Who can use CSQL**

CSQL has been designed from scratch with a single point benefit – to provide undistruptive performance benefits in application domains where real time access to data is a core necessity. For example –

- Financial and Insurance Industry
- Information Technology
- Telecommunication Industry

## **2.9. CSQL Selective Information References**

For providing support to CSQL users, below communication channels are dedicated for customer queries.

### **2.9.1. Mailing lists**

The primary mechanism for CSQL communication is through its mailing lists. Anyone who uses this product shall participate in user mailing lists. You can search for the archive of past discussions before you post your question to the mailing lists in [http://sourceforge.net/mailarchive/forum.php?forum\\_name=csql-users](http://sourceforge.net/mailarchive/forum.php?forum_name=csql-users). The main channel for user support is [csql-users@lists.sourceforge.net](mailto:csql-users@lists.sourceforge.net) mailing list. As is usual with mailing lists, be prepared to wait for an answer.

Please summarize any off-list knowledge gained and post it for the benefit of all. For example, if a user asks a question and gets response, post that to the above mentioned site also.

### **2.9.2. Forums**

CSQL Users shall use forums for posting their queries for support in the below mentioned URL

[http://sourceforge.net/forum/forum.php?forum\\_id=562614](http://sourceforge.net/forum/forum.php?forum_id=562614)

Issues related to installation and implementation shall be posted here to get answers from CSQL technical team.

## 3. CSQL Installation

This chapter contains configuration information that you need to know before installation as well as complete information about building and installing CSQL. In addition; a complete structure on directory and all the important files are also mentioned.

### 3.1. Where to find CSQL

From the below two web sites, you can download the product.

- [www.sourceforge.net/projects/csqli](http://www.sourceforge.net/projects/csqli)
- [www.csqldb.com](http://www.csqldb.com)

### 3.2. OS Platform support

- CSQL runs on Linux operating system on Intel x86 architecture.

### 3.3. Installation Prerequisites

- UnixODBC
- JDK 1.6

### 3.4. Directory Layout and Files

CSQL directory layout is divided into the following sections.

Table 1.3. Directories and Files

Directories and Files	Contains
Include	Header files
Bin	Client tools
Lib	Shared libraries and jar file.
Docs	Manuals
Sample	Sample configuration files
Examples	Sample programs for ODBC,JDBC and SQLAPI
demo	Performance and caching demonstration
setupenv. ksh	Script to set up environment variables
ChangeLog	
LICENSE	
README	

## 3.5. Running examples

You are now ready to run some examples given in examples directory. This directory contains following subdirectories and each directory have individual Makefile and README file.

Table 1.4. Example Directory

Directory Name	File Name
sqlapi	sqlapiexample.c
isql	create.sql, drop.sql, insert.sql, select.sql, delete.sql
jdbc	jdbcxample.java
odbc	odbcexample.c

Each of these subdirectory contain README file that will guide you to compile and run these examples.

**Note:** Make sure the server is started in another terminal and `setupenv.ksh` is executed before running the examples.

## 3.6. How to Configure CSQL

The default database size for CSQL is 10 MB. If you wish to change the size to more than 10 MB, then the system variable `kernel.shmmax` should be set to either same size or more than the size of the database. Only the superuser has the privilege to run this command. You can ask your system administrator to set it for you in case you are not the super user of the system.

The following command sets the kernel parameter to 1 GB,

```
# /sbin/sysctl -w kernel.shmmax=1000000000
kernel.shmmax=1000000000
```

After setting up the kernel parameter, the user should set the CSQL environment variables using below command.

```
$ cd <CSQL_ROOT>
$ . ./setupenv.ksh
```

This will set all the environmental variables defined in `setupenv.ksh` file present in `<CSQL_ROOT>` directory. In the below section, the environment variables have been discussed in details.

If you are evaluating or trying out CSQL for the first time, you may skip the next section as `setupenv.ksh` script sets the necessary environment variables.

**Note:** Run `setupenv.ksh` in each and every terminal or console you open before you do any operation with CSQL.

### 3.6.1. Setting up environmental variables

CSQL has a set of environmental variables that need to be set before starting the server. To carry out the various database operations, either by using the CSQL tools or using executables which link with CSQL libraries, the following environmental variables need to be set. Let us assume `CSQL_ROOT` is the absolute path where CSQL is installed.

#### 3.6.1.1. CSQL\_INSTALL\_ROOT

This should be set to the full path of where CSQL is installed. If CSQL is installed in `/opt/csql`, set this environment variable as follows

```
$ export CSQL_INSTALL_ROOT=/opt/csql
```

#### 3.6.1.2. CSQL\_CONFIG\_FILE

There is a configuration file called `csql.conf` in the `CSQL_ROOT` directory, which the `csqlserver` reads during loading up. This file has the configuration variables for the CSQL system. Refer the Section 3.5.2. for details.

```
$ export CSQL_CONFIG_FILE=<CSQL_ROOT>/csql.conf
```

#### 3.6.1.3. LD\_LIBRARY\_PATH

This variable is set to locate the CSQL `lib` directory that contains CSQL specific libraries.

```
$ export LD_LIBRARY_PATH=<CSQL_ROOT>/lib:$LD_LIBRARY_PATH
```

#### 3.6.1.4. PATH

This variable is set to locate the CSQL `bin` directory that contains the CSQL executables.

```
$ export PATH=<CSQL_ROOT>/bin:$PATH
```

### 3.6.1.5. CLASSPATH

This variable is set to locate the libraries for the JDBC driver of CSQL.

```
$ export CLASSPATH=<CSQL_ROOT>/lib/CSqlJdbcDriver.jar:.
```

Running the following script from the CSQL root directory will set all the above variables automatically.

```
$ . ./setupenv.ksh
```

### 3.6.2. Configuration File (csql.conf)

CSQL subsystem requires some system parameters that need to be set before starting CSQL database. Hence CSQL defines some of the system configuration variables that need to be defined. 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.

If you are evaluating or trying out CSQL for the first time, you may skip this section as the default csql.conf file present under sample directory of CSQL installation root directory is sufficient for demonstration purposes.

The lines starting with # are ignored as comments and the rest are treated as configuration variables. All the other lines 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

#### 3.6.2.1. Server 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.

#### **PAGE\_SIZE**

Each database is logically divided into pages and allocation happens in this unit of pages. Increasing this value will reduce frequent allocation of pages. This value should be multiple of 1024 bytes. This value may be set to the OS page size which is usually 8192 bytes which is a default value is set.

#### **MAX\_PROCS**

This is the number of processes that can connect and work with the database concurrently. This value can be set anywhere between 10 and 8192 depending on the number of users who may access the database. Default value 100 is set for this variable.

### **MAX\_SYS\_DB\_SIZE**

This is the maximum size of the system database where the metadata are stored. The value can be anywhere between 1 MB and 1GB. It should be multiples of PAGE\_SIZE. Default size is 1048576 bytes.

### **MAX\_DB\_SIZE**

This is the maximum size of the user database where user data are stored. This value can be set anywhere between 1MB and 2GB. It should be multiples of PAGE\_SIZE. Default size is 10485760 bytes.

### **SYS\_DB\_KEY**

Shared memory key to be used by the system to create and locate system database. The value can be anywhere between 10 and 8192. The default key value is 2222.

### **USER\_DB\_KEY**

Shared memory key to be used by the system to create and locate user database. The value can be anywhere between 10 and 8192. This should not be the same as SYS\_DB\_KEY. The default key value is 3333.

### **LOG\_FILE**

It specifies full path for a directory, where CSQL log files are stored. Make sure that this directory exists before you start the server. The default path is /tmp/csql/log/log.out for the log directory.

### **MAP\_ADDRESS**

This is the virtual memory start address at which the shared memory segment will be created and attached. The default value for the variable is set to 4000000000.

### **DURABILITY**

It is a flag which specifies whether to enable durable transactions and recovery in case of server restart. By default, it is set to false.

### **DATABASE\_FILE**

It specifies full path for a directory, where CSQL stores checkpoint and redo log files for providing durable transactions and recovery. Make sure that this directory exists before you start the server. The default path is /tmp/csql/db

### 3.6.2.2. Client section variables

#### **MUTEX\_TIMEOUT\_SECS**

Mutex timeout interval in seconds. When requesting for mutex, if it is acquired by anyone else, then the requester will wait for this specified time interval before it checks whether it is free to acquire.

#### **MUTEX\_TIMEOUT\_USECS**

Mutex timeout interval in microseconds. The cumulative of the seconds and microseconds set will be used for the mutex timeout. The default value is 5000 set for this variable.

#### **MUTEX\_TIMEOUT\_RETRIES**

Number of retries before csql gives mutex timeout error. The default value for this variable is 10.

#### **LOCK\_TIMEOUT\_SECS**

Lock timeout interval in seconds. When requesting for lock, if it is acquired by anyone else, then the requester will wait for this interval before it checks whether it is free to acquire. Default value is zero.

#### **LOCK\_TIMEOUT\_USECS**

Lock timeout interval in microseconds. The cumulative of the seconds and microseconds set will be used for the lock timeout. Default value is 5000.

#### **LOCK\_TIMEOUT\_RETRIES**

Number of retries before csql gives lock timeout error. Default value is 10.

### 3.6.2.3. Cache section variables

#### **CACHE\_TABLE**

It is a flag which specifies whether to enable caching of tables from the target database. Default value is false.

## **DSN**

DSN name to connect to the target database for caching. This name should be present in the `odbc.ini` file with the respective `odbc` library of target database.

## **CACHE\_ID**

It uniquely identifies a cache instance in multi node caching. The default value for this variable is 1.

## **USER ID**

It specifies the username to be used when connecting to target database while caching.

## **PASSWORD**

It specifies the password to be used when connecting to target database while caching.

## **TABLE\_CONFIG\_FILE**

File name where the cached table information is stored. Specify the file name with full path. Default value is set to `/tmp/csql/csqltable.conf`

## **ENABLE\_BIDIRECTIONAL\_CACHE**

It is a flag, which specifies whether to enable bi-directional caching for cached tables. Direct updates to target database will be brought into CSQL cache table automatically making cache coherent.

## **CACHE\_RECEIVER\_WAIT\_SECS**

It specifies the time interval in seconds, for CSQL server to wait if there are no update logs from the target database. This flag is used only in case of bi-directional caching.

## **CACHE\_MODE**

This specifies the mode for updatable caching from CSQL to target database. It can either be `SYNC` for synchronous updates or `ASYNC` for asynchronous updates. By default it is set to `SYNC` mode.

## **CSQL\_SQL\_SERVER**

CSQL supports network access from remote machine using client/server mode. This flag specifies whether to enable client/server mode access. By default it is set to `false`.

## **PORT**

This specifies the port at which CSQL server will be listening for client connections. Default value is 5678.

## **REPLICATION**

It is a flag which specifies whether to enable replication. This should be set to true for multi node coherent caching deployment.

## **REPLICATION\_SITES**

It specifies the maximum number of sites in the replication quorum. Default value is 16

## **NETWORK\_CONFIG\_FILE**

It specifies the file name where the replication quorum information is specified. Specify the file name with full path. Default value is set to /tmp/csql/csqlnw.conf

## **MSG\_KEY**

CSQL uses message queues for communication internally. This specifies the key to be used for creating that message queue. It is an internal parameter. Users need not modify this value.

## **ID\_SHM\_KEY**

CSQL stores control information in separate shared memory segment internally. This specifies the key to be used for creating that shared memory segment. It is an internal parameter. Users need not modify this value.

### **3.7. Starting and Stopping csqlserver**

Once CSQL is configured the server is ready to start. The CSQL Server can be invoked by running the following command:

```
$csqlserver
ConfigValues
getPageSize 8192
getMaxProcs 100
.
.
.
getMaxLogStoreSize 1048576
getNetworkID 1
getCacheNetworkID -1
```

```
sysdb size 1048576 dbsize 10485760
System Database initialized
Database server started
```

If the output on your screen looks similar to the above, then the server is ready for operations. To stop the server, just press <Ctrl + C> from the terminal where the server is running.

```
Received signal 2
Stopping the server
Server Exiting.
```

The above output message is displayed during the exit. This will stop the server gracefully by doing the necessary clean ups.

## 3.8. Troubleshooting

When you are working with CSQL, you may find some problems. This section gives a description about problems and how to fix it.

### 3.8.1. Errors while running csqserver

The following errors may come at the time of starting of the server because the environment variable might not be set up or the server is unable to create the log file.

#### 3.8.1.1. csqserver command not found

This below error is thrown when the \$PATH environmental variable is not set for CSQL.

```
$ csqserver
-bash: csqserver: command not found
```

Run the following command from CSQL\_ROOT directory.

```
$ . ./setupenv.ksh
```

Now run `csqserver` and it should work.

#### 3.8.1.2. Unable to create the log file

When you will start the server you may come across a message like “Unable to start the logger”. Below examples is for the error scenario.

### Example : 1

```
$ csqserver
4822:3086075584:Logger.cxx:101:Unable to create log file.
Check whether server started
Unable to start the logger
```

This above error is thrown when **csqserver** is not able to create the log file. Create the directory defined for LOG\_FILE in csq.conf file, which is present in CSQL\_ROOT directory. For example, LOG\_FILE=/tmp/csq/log1/log.out

```
$ mkdir -p /tmp/csq/log
```

If you wish to create the log file in different directory then create that directory and change the value of LOG\_FILE in csq.conf file. Now start the server and it would work fine.

### 3.8.2. Errors while running Client

If you start csqserver with one user and try to access it as another user you may get this error as shown below.

```
$csql
22845:3085903568:Logger.cxx:101:
Unable to create log file.Check whether server started
```

**csqserver** and its clients should be run by the same user.

## 4. Getting started with CSQL

This chapter discusses the SQL statements available in CSQL.

For each statement, this chapter gives the supported syntax; explain the parameters, any other relevant information.

We start with CSQL Interactive tool for SQL Statements, and then explain briefly about DDL and DML statements including JOIN and GROUP BY.

The middle part lists about accessing the meta-data information using Catalog tool and at the end it describes about the batch mode operation with CSQL tool.

### 4.1. Client tool for communicating with CSQL database

CSQL provides a tool called `csql`, which is a sub-shell used to access the CSQL database. It supports most of the standard SQL statements. Type `csql` command in command line to get Interactive prompt named "`CSQL>`" to access Database. Make sure that `csqlserver` is running prior to running this tool.

In order to log into CSQL we must pass information to the CSQL client program when we start it.

This is done with the following commands and syntax.

```
$csql -u root -p manager
CSQL>
```

Now the tool is ready to be used by the user. Here "root" is the default user name and "manager" is the password for log on. Once you have the CSQL prompt then the tool is ready to access the database.

### 4.2. Using a database

This section will demonstrate about database table and indexes, including all other supported and provided functionality by CSQL.

#### 4.2.1. Creating new table

You can create a new table by specifying the table name, along with all column names and their types:

## Syntax

```
CREATE TABLE <table_name> (  
col1      Datatype[<Size>] <default value>[NOT NULL],  
col2      Datatype[<Size>] < default value >[NOT NULL],  
col3      Datatype[<Size>] < default value >[NOT NULL],  
col4      Datatype[<Size>] < default value >NOT NULL],  
.....  
.....  
[ PRIMARY KEY(col1, col2, ...)]  
);
```

### Example : 1

```
CSQL> CREATE TABLE stud (  
      roll      INT NOT NULL,  
      name      CHAR(20) NOT NULL,  
      course_fee  DOUBLE,  
      dept_no  INT  
);
```

**NOTE :** Except CHAR and BINARY data type other data types does not require size.

We can apply "primary key" on all data types except FLOAT, REAL, DOUBLE.

### Description

The above SQL Statement will create table ' stud ' having 4 fields "roll" is an integer field , "name" is a string field which can store a string ,"course\_fee" is a floating point with double precision field which can store values with decimal point and "dept no" is an integer field.

## 4.2.2. Inserting data into a table

The INSERT statement is used to populate a table with rows ,You can insert data into the table in two ways.

### First Form :

#### Syntax

```
INSERT INTO <table_name> VALUES ( val1, val2, val3, val4,  
...);
```

### Example : 1

```
CSQL> INSERT INTO stud VALUES ( 1, 'Rajesh Kumar', 3000, 20);
```

### Description

This will insert one record into table "stud". If data is not available for any of the 4 fields at the time of insertion, you can put NULL there.

### Example : 2

Suppose you have no data for the field "course\_fee" then you can have the following insert Statement.

```
CSQL> INSERT INTO stud VALUES ( 2, 'Rakesh Chandra', NULL, 10);
```

### Second Form

#### Syntax

```
INSERT INTO <table_name> (col1, col2, col3, ...) VALUES (val1, val2, val3, ...);
```

Generally this form is useful when data for many fields are to be NULL.

### Example : 3

```
CSQL> INSERT INTO stud (roll, name) VALUES ( 3, 'Kishore Kumar');
```

### Description

The above Statement does not include "course\_fee" and "dept\_no" field. So After execution of the above statement NULL values will be automatically inserted into "course\_fee" and "dept\_no" field. This will insert one record into table "stud" having "course\_fee" and "dept\_no" field as NULL value.

## 4.2.3. Querying a table

To retrieve data from a table, the table is queried. An SQL SELECT statement is used to do this. The statement is divided into a SELECT list (the part that lists the columns to be

returned), a table list (the part that lists the tables FROM which to retrieve the data), and an optional qualification (the part that specifies any restrictions).

For example, to retrieve all the rows of table "stud", type:

```
SELECT * FROM stud;
```

Here \* is a shorthand for "all columns".

```
SELECT roll, name, course_fee, dept_no FROM stud;
```

The output would be:

stud.roll	stud.name	stud.course_fee	stud.reg_no
1	Rajesh Kumar	3000.000000	20
2	Rakesh Chandra	NULL	10
3	Kishore Kumar	NULL	NULL

You can also retrieve selective data as per your requirement using different logical and relational Operators.

#### 4.2.4. Join between tables

A query that accesses rows from multiple tables is called a join query.

##### Example :1

Let's say you wish to list all the students' records together with the dept\_name of the associated stud. To do that, we need to compare the "roll" column of each row of the "stud" table with the "roll" column of all rows in the "dept" table, and SELECT the pairs of rows WHERE these values match.

We need another table say "dept" whose table definition is given below.

```
CSQL> CREATE TABLE dept(  
dept_no INT NOT NULL,  
dept_name CHAR(10) NOT NULL  
);
```

Insert some records into stud and dept table with matching values for dept\_no column and run below query to get the result.

```
CSQL>SELECT * FROM stud, dept where stud.dept_no =  
dept.dept_no;
```

### 4.2.5. Aggregate functions

Like most other relational database products, CSQL supports aggregate functions. An aggregate function computes a single result FROM multiple input rows. For example, there are aggregates to compute the count, sum, avg (average), max (maximum) and min (minimum) over a SET of records.

As an example, we can find the average of course\_fee field. Aggregates are also very useful in combination with GROUP BY clauses.

```
CSQL>SELECT avg(course_fee) FROM stud;
-----
AVG(course_fee)
-----
2714.285714
```

### 4.2.6. Update

You can modify the data values of existing rows using the UPDATE command. Let's you want to modify the dept\_no for stud having roll 3 to 10.

```
CSQL>UPDATE stud SET dept_no=10 WHERE roll=3;
Statement Executed: Rows Affected = 1
```

### 4.2.7. Delete

Using the DELETE command you can delete the existing rows from a table.

Suppose you need not require the record having dept\_no 40. You can use DELETE command for this.

```
CSQL>DELETE FROM stud WHERE dept_no=40;
Statement Executed: Rows Affected = 2
```

## 4.3. Getting Information about Metadata using Catalog tool

Catalog tool, which provides information about system metadata and user metadata of tables stored in the CSQL database.

We are going to run catalog tool on table 't1', which contains two fields, f1 which is integer type and f2 which is character type with size 10 bytes.

Example : The below catalog command with no option will retrieve all the tables present in the database.

```
$catalog
<TableNames>
  <TableName> t1 </TableName>
</TableNames>
```

Example : To get all the information about table, you can use the `-l` option.

```
$ catalog -u root -p manager -l
<Table Information of all tables>
  <TableInfo>
    <TableName> t1 </TableName>
    <FieldInfo>
      <FieldName> f1 </FieldName>
      <Type> 0 </Type>
      <Length> 4 </Length>
      <Primary> 0 </Primary>
      <Null> 0 </Null>
      <Default> 0 </Default>
      <DefaultValue> </DefaultValue>
    </FieldInfo>
    <FieldInfo>
      <FieldName> f2 </FieldName>
      <Type> 30 </Type>
      <Length> 11 </Length>
      <Primary> 0 </Primary>
      <Null> 0 </Null>
      <Default> 0 </Default>
      <DefaultValue> </DefaultValue>
    </FieldInfo>
  </TableInfo>
</Table Information of all tables>
```

*Refer section 9.2.* for complete information on catalog tool and its all arguments and options in detail .

## 4.4. SHOW Statement

To display all the tables present in database, use this statement.

```
CSQL>show tables;
=====TableName=====
  t1
=====
```

At the time of executing this SHOW statement only 't1' table is present in the database.

## 4.5. Using CSQL in batch mode

You can use csql tool with -s option for executing more than one statement at a time.

1. Create a file (example1.sql file as per convention) which will contain all the valid necessary SQL statements.

2. To execute all the statements in the file, type 'csql -s example1.sql'.

### Example :1

Suppose you want to create a table named "example1", Insert 4 records into it and you want to do it in batch mode.

For this create a file for example1.sql and write all sql statement you want to execute in it.

```
$cat > example1.sql
create table example1(f1 int,f2 int);
insert into example1 values(1,10);
insert into example1 values(2,20);
insert into example1 values(3,30);
insert into example1 values(4,40);
ctrl+d
```

Execute the file with **csql -s**

```
$ csql -s example1.sql
Statement Executed
Statement Executed: Rows Affected = 1
Statement Executed: Rows Affected = 1
Statement Executed: Rows Affected = 1
Statement Executed: Rows Affected = 1
```

## 4.6. Constraints

This section talks about CSQL database constraints such as Primary Key, NOT NULL, and UNIQUE.

CSQL and its associated suite of products have a single design objective and that is undistruptive performance, Because of this main goal it supports some of the important constraints (handles them in database level) and rest of the constraints when necessary

are being handled at application level. The following are the important constraints supported by the CSQL database.

### 4.6.1. Not-Null

A not-null constraint specifies that a column must have some value. That means you should not store NULL value for a NOT NULL field.

#### Example: 1

```
CSQL>CREATE TABLE products (  
product_no INTEGER NOT NULL,  
name CHAR(20) NOT NULL,  
price FLOAT  
);
```

If you want to insert NULL value in NOT NULL specified field it will raise NOT NULL constraint violation error.

#### Example : 2

```
CSQL>insert into products values(1,NULL,7000);  
  
8160:3086649568:TableImpl.cxx:681:NOT NULL constraint  
violation for field name  
8160:3086649568:TableImpl.cxx:443:Unable to copy values  
from bind buffer  
Statement execute failed with error -22
```

NOT NULL constraint is a Domain-Integrity constraint. That means it can restrict a column from storing NULL value. A not-null constraint is always written as a column constraint.

### 4.6.2. Unique

Unique constraints ensure that the data contained in a column or a group of columns is unique with respect to all the rows in the table. We can use UNIQUE constraint when working with Index.

After table creation we can apply UNIQUE key on specific field or group of fields for index creation.

#### Example : 1

Suppose one want to create unique index on "roll" field of "stud" table, it can be created as follows:

```
CSQL>CREATE INDEX idx1 ON stud(roll) UNIQUE;
Statement Executed
```

You can create INDEX on multiple fields of an existing table. This is called Composite Unique index.

```
CSQL>create index idx2 on stud(roll,name) unique;
Statement Executed
```

In general, a unique constraint is violated when trying to insert a row with index field values that is already present in the table.

### 4.6.3. Primary Key Constraint

Relational database theory says that every entity of a relational schema must be uniquely identified from each other. A primary key indicates that a column or group of columns can be used as a unique identifier for rows in the table. As per the relational Database theory every table must have a primary key. This rule is not enforced by CSQL, but it is usually best to follow it. A table can have at most one primary key. Primary key is the combination of NOT NULL and UNIQUE constraints.

#### Example : 1

```
CSQL>CREATE TABLE emp (
    emp_no INT,
    name CHAR(20) NOT NULL,
    dept_no INT,
    PRIMARY KEY(emp_no)
);
Statement Executed
```

You can also apply primary key by combining two or more fields. This is called Composite primary Key.

#### Example : 2

```
CSQL>create table emp(ename char(20),dept_no int,job
char(10),age int,primary key(ename,dept_no));
Statement Executed
```

#### Description:

Here we have emp table having four fields out of which ename and dept\_no together behave as PRIMARY KEY.

## 4.7. Index

A database index is a data structure that improves the speed of operations on a database table. Indexes can be created using one or more columns of a database table. CSQL uses Hash index for faster point look up and Tree index for range look up.

CSQL supports two types of indexes:

- **Hash Index** : This is used for making point lookup faster
- **Tree Index** : This is used for making range lookup faster

### 4.7.1. Hash Index

Hash index is used, when you are working with "=" operator with where clause for searching records based on certain key value. This type of searching is called as point lookup in database.

#### Syntax

```
CREATE INDEX <INDEX_NAME> ON <TABLE_NAME> (<COL1, COL2, . . .>) [HASH];
```

In the above syntax, the hash keyword specifies to create the hash index. Even if it is omitted in the statement, CSQL creates hash index as it is the default indexing mechanism used in CSQL.

Lets create a 'T1' table with three fields using the below statements.

```
CSQL> CREATE TABLE T1 (F1 INT, F2 SMALLINT, F3 CHAR(15));
```

#### Example :1 Creating default hash index

The above statement will create a hash index named idx1, which is default . you can also see the metadata information about the index using catalog tool.

```
CSQL> CREATE INDEX idx1 ON T1 (F1);
```

#### Example : 2 Accessing index information using catalog tool

Catalog tool has a -I option which gives the information related to that index. Here the created index 'idx' is specified with -I option. The below output is shows the information for 'idx1' index.

```
$catalog -u root -p manager -I idx1
<Index Info>
<IndexName> idx1 </IndexName>
<Unique> 0 </Unique>
```

```

<Type> Hash Index </Type>
<HashBucket>
  <TotalPages> 1 </TotalPages>
  <TotalBuckets> 1009 </TotalBuckets>
</HashBucket>
<IndexNodes>
  <TotalPages> 1 </TotalPages>
  <TotalNodes> 0 </TotalNodes>
<IndexNodes>
<Index Info>

```

### Example : 3 Drop the Index

If you want to drop the index from database you can use the DROP INDEX command. Below statement drops the 'idx1' index in the database which was created on 'T1' table.

```
CSQL> DROP INDEX idx1;
```

## 4.7.2. Tree Index

Tree index is used, when you are working with range operators like < , > ,<=, >= , BETWEEN operator with where clause for searching records based on key value. This type of searching is called as range lookup in database.

If you want to create a tree index it is mandatory to specify the 'TREE' keyword at the time of creating the tree index.

### Syntax :

```
CREATE INDEX <INDEX_NAME> ON <TABLE_NAME> (<COL1, COL2, . . . .>) TREE;
```

Lets create a 'T1' table having three fields F1,F2 and F3 as follows

```
CSQL> CREATE TABLE T1 (F1 INT, F2 SMALLINT, F3 CHAR(15));
```

### Example :1 Creating tree index on 'F2' field

```
CSQL> CREATE INDEX idx2 ON T1 (F2) TREE;
```

After execution of the above statement idx2 index will be created in the database. You can also see the metadata information about the index using catalog tool.

### Example :2 Index information

```

$catalog -u root -p manager -I idx2
<Index Info>
<IndexName> idx2 </IndexName>

```

```

<Unique> 0 </Unique>
<Type> Tree Index </Type>
<HashBucket>
  <TotalPages> 1 </TotalPages>
  <TotalBuckets> 1009 </TotalBuckets>
</HashBucket>
<IndexNodes>

```

### 4.7.3. Composite index

CSQL also supports Composite index. That means you can apply index on two or more fields.

**Example :1** Composite index on F1 and F2 field

```

CSQL> CREATE INDEX idx3 on T1(F1,F2) HASH;

```

Note: CSQL restricts creating composite index with Tree index.

### 4.7.4. Unique Index and primary index

You can apply Unique and Primary Key on both Hash Index as well as Tree Index

**Syntax for Unique Index :**

```

CREATE INDEX <index_name> ON <table_name>(<col1,col2,...>)
[HASH | TREE] UNIQUE;

```

**Example :** Creating two different index

```

CSQL> CREATE INDEX idx1 ON T1(F1) HASH UNIQUE;
CSQL> CREATE INDEX idx2 ON T1(F2) TREE UNIQUE;

```

The unique indexes 'idx1' and 'idx2' will be created on 'F1' and 'F2' fields respectively.

**Syntax for primary key index :**

Before creating Primary Key on any fields, you should make sure that the field should be NOT NULL at the time of table creation.

```

CREATE INDEX <index_name> ON <table_name>(<col1,col2,...>)
[HASH | TREE] PRIMARY;

```

**Example:1**

```
CSQL> CREATE INDEX idx1 ON T1(F1) HASH PRIMARY;  
CSQL> CREATE INDEX idx2 ON T1(F2) TREE PRIMARY;
```

After executing the above statements, hash and tree indexes will be created in the database which will allow unique values.

## 5. Data Types

CSQL supports a number of data types in several categories: numeric types, string (character) types, binary types, date/time types. This chapter first gives an overview of these data types, and then provides a more detailed description of the properties of the types in each category, and a summary of the data type storage requirements.

### 5.1. Numeric types

Numeric type includes all integer types and floating point data types. These consist of two-, four-, and eight-byte integers, four- and eight-byte floating-point numbers, and selectable-precision decimals.

#### 5.1.1. Integer Types

The types `smallint`, `tinyint`, `int`, `integer` and `bigint` store whole numbers, i.e numbers without fractional components, of various ranges. Attempts to store values outside of the allowed range will result in an error.

`SMALLINT` type is generally only used if disk space is at a premium. The range of `smallint` is -32768 to 32767.

`TINYINT` type is generally only used if disk space is at a more premium. the range of this is -128 to 127.

`INTEGER` (or `int`) (or `long`) is the usual choice, as it offers the best balance between range, storage size, and performance. The range of integer type is -2147483648 to 2147483647.

`BIGINT` type should only be used if the integer range is not sufficient, because the latter is definitely faster. The range of `bigint` type is -9223372036854775808 to 9223372036854775807.

#### 5.1.2. Floating point types

These types are used for working with fractional numbers. There are 3 types of floating point types are supported in CSQL.

`FLOAT` : A single-precision floating-point number takes four bytes. and `REAL` value is same as float.

`DOUBLE` : A double-precision floating-point number takes 8 bytes.

## 5.2. Character types

A field of characters can consist of any kinds of alphabetical symbols in any combination. If you want a column to hold a fixed number of characters, such as the book shelf numbers of a library, use the char data type for such a column. In such a case, all fields under that column would use the same number of character. The desired number of characters must be typed in the parentheses of the char data type. The syntax used is:

```
char (Number)
```

In this case, the Number factor represents the number of characters that each field would use. The Number factor must be a positive number between 0 and 65536.

## 5.3. Binary Data Types

CSQL supports fixed-length binary data with a maximum length of 65536 bytes. it is only used to keep info made up of Hexadecimal characters such as (0,1,2,...,9 and A,B,C,D,E,F,a,b,c,d,e,f). Two input characters take 1 byte.

### Syntax

```
<field name> BINARY( length )
```

```
where length < 65536
```

### Example :1

```
CSQL>CREATE TABLE BINTABLE(f1 BINARY(5));  
Statement Executed
```

```
CSQL>INSERT INTO BINTABLE VALUES('ABCDEF123');  
Statement Executed: Rows Affected = 1
```

```
CSQL>SELECT * FROM BINTABLE;
```

```
-----  
BINTABLE.f1
```

```
-----  
ABCDEF123
```

## 5.4. Date/Time Types

Date and time types are a convenient way to store date and time related data in a uniform SQL data structure, without having to worry about the conventions involved with storage (e.g., if you were to try to store such information in a character data type). CSQL supports three different data types for date and time. They are Date, Time, Timestamp.

Generally maximum people are confused about what is the syntax of inserting the values into the date, time and timestamp fields. So in the below section you will find detail syntax for inserting values into three fields.

### Example :1

```
CSQL>CREATE TABLE T1(f1 DATE,f2 TIME,f3 TIMESTAMP);
Statement Executed

CSQL>insert into T1 values('2001-1-1', '01:01:01', '2001-
11-30 01:01:01');
Statement Executed: Rows Affected = 1

CSQL>select * from T1;
-----
T1.f1      T1.f2      T1.f3
-----
2001/1/1  1:1:1.0    2001/11/30 1:1:1.0
```

#### 5.4.1. Date

This describes a date using the fields YEAR, MONTH and DAY in the format YYYY-MM-DD. The length is 8.

```
Syntax : <field name> DATE
```

```
Format : 'YYYY-MM-DD'
```

#### 5.4.2. Time

This describes a time in an unspecified day, with seconds precision s, using the fields HOUR, MINUTE and SECOND in the format HH:MM:SS[.sF] where F is the fractional part of the SECOND value.

```
Syntax : <field name> TIME
```

```
Format : 'HH:MM:SS'
```

#### 5.4.3. Time Stamp

This describes both a date and time, with seconds precision s, using the fields YEAR, MONTH, DAY, HOUR, MINUTE and SECOND in the format YYYY-MM-DD HH:MM:SS[.sF] where F is the fractional part of the SECOND value.

```
Syntax : <field name> TIMESTAMP
```

```
Format : 'YYYY-MM-DD HH:MM:SS'
```

## 6. Function and Operators

This Chapter describe about various logical and comparison operators and Aggregate functions supported by CSQL.

### 6.1. Logical operators

In the previous section we saw how a single condition could be added to a query using a WHERE clause. While this is useful, usually more than a single condition is required to produce the correct result. To support multiple rules we need to make use of NOT, OR, AND logical operators.

Following are logical operators CSQL supports:

- AND
- OR
- NOT

#### AND

The basic way of supporting multiple conditions in a single query is by making use of AND. It provides a way of connecting two rules together such that all the conditions must be true before the row is shown. For example,

```
SELECT * FROM stud
WHERE stud.roll=2 AND stud.name='Rakesh Chandra' ;
```

#### OR

Either Condition can apply. For example,

```
SELECT * FROM stud
WHERE stud.dept_no=20 OR stud.dept_no=40 ;
```

#### NOT

The NOT operator does the opposite of whatever comparison is being done.

```
SELECT * FROM stud
WHERE stud.roll !=3 ;
```

```
SELECT * FROM stud
WHERE NOT(stud.name='Rakesh Chandra') ;
```

## 6.2. Comparison operators

Comparison operators test whether two expressions are same or not. These are used with character, numeric, or date type or can be used in the WHERE clause. Comparison operators evaluate to a boolean value and return TRUE or FALSE based on the outcome of the tested condition.

Following comparison operators are supported:

<b>Operator</b>	<b>Name</b>	<b>Example</b>	<b>Description</b>
=	Equal	Gender='Male'	Check if selection is equal to Male
>=	Greater than or equal to	Age >=18	Check if Age is greater than or equal to 18.
<=	Less than or equal to	Age <= 50	Checks if Age is greater than or equal to 50.
>	Greater than	Age > 24	Checks if Age is greater than 24.
<	Less than	Age < 45	Checks if Age is less than 45.
!=	Not equal	Gender!= 'Male'	Checks if Gender is not equal to Male.

## 6.3. Mathematical operators

Below table shows the basic arithmetic operators supported in CSQL.

<b>Operator</b>	<b>Description</b>
+	Addition
-	Subtraction
*	Multiplication
/	Division

**Note :** CSQL restricts these operators to use only with UPDATE statements to modify the data.

## 6.4. Aggregate functions

Aggregate functions perform a calculation on a set of values and return a single value. And it is frequently used with the GROUP BY clause of the SELECT statement.

CSQL supports the below aggregate functions :

- AVG() – Returns the average value.
- COUNT() – Returns the number of rows.
- MAX() – Returns the largest value.
- MIN() – Returns the smallest value
- SUM() – Returns the sum.

### The AVG() Function

It returns the average value of a numeric column. And it accepts the numeric field names to perform the query. For example,

```
SELECT AVG(stud.course_fee) FROM stud ;
```

### The COUNT() Function

It returns the number of rows that matches specified criteria and returns the number of values of the specified column except NULL values which is integer data type value. For example,

```
SELECT COUNT(stud.roll) FROM stud ;
```

If the argument is the '\*' qualifier then it returns the number of records and includes NULL values and duplicates also. For example,

```
SELECT COUNT(*) FROM stud ;
```

### The MAX() Function

The MAX() function returns the largest value of the selected column and can be used with numeric, character, date, and time column. It returns a value same as expression and ignores any null values. In the below examples, the function accepts both numeric and character fields.

```
SELECT MAX(stud.course_fee) FROM stud ;  
SELECT MAX(stud.name) FROM stud ;
```

## The MIN() Function

To calculate the minimum and smallest value present in a particular column you can use the MIN function. It also accepts the numeric and character in its arguments and ignores NULL values. For example,

```
SELECT MIN(stud.couse_fee) FROM stud ;
```

The above functions can be used with GROUP BY clause.

## 6.5. SQL Operators

CSQL supports SQL operators such as IN, BETWEEN and LIKE.

### 6.5.1. IN operator

The IN operators allows you to specify multiple values in WHERE clause.

```
SELECT * FROM stud
WHERE stud.roll IN (10,40) ;
```

### 6.5.2. BETWEEN operator

Range operators are used to create ranges and to see if a value is within the range created. The BETWEEN operator is used in a WHERE clause to select a range of data between two values.

```
SELECT * FROM stud
WHERE stud.roll BETWEEN 10 AND 40 ;
```

### 6.5.3. LIKE operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. It makes the searching task by allowing for unknown characters (using underscore,-) and partial strings (using a wildcard, %). It takes as input any string or partial string and attempts to find a match in the data being searched.

Here is the example to search the students name from stud table using both underscore and wild card. As underscore character( \_) matches exactly one character and percentile character (%) matches zero or more characters.

```
SELECT * FROM stud
WHERE stud.name LIKE 'R%' ;

SELECT * FROM stud
WHERE stud.name LIKE '_i%' ;
```

## 7. Database Recovery

CSQL provides both redo logs based automatic recovery and archive/restore based manual recovery mechanisms.

### 7.1. Automatic Recovery

CSQL recovers the database automatically after a crash or system failure. By default, durability mode is set to false in `csql.conf`. In order to recover the data till the last committed transaction, this mode should be set to true when starting the server.

`DATABASE_FILE` configuration parameter specifies the directory in which checkpoint and redo log files are stored. Make sure that enough space (twice the size of the database size) is available in disk which contains this directory.

### 7.2. Archive the database

In case of hard disk failure, CSQL won't be able to recover the database from its checkpoint and redo log files. To recover database in this scenario, user needs to take backup of the CSQL database periodically using `csqldump` tool and store in file in another machine, In case of any disk failure, this archive file shall be used to restore the database. The interval at which the archive needs to be taken is decided based on the application requirement.

`csqldump` tool is used to take archive of the database. This command stores all the tables and its records into the file specified.

```
$csqldump >/tmp/archive1
```

### 7.3. Restore the database

Start the server and run the following command to restore the database from the archive.

```
$csql -s /tmp/archive1
```

If `DURABILITY` option is set in `csql.conf` file, then remove all the files under the directory specified in `DATABASE_FILE` before starting the server.

## 8. Tool Reference

CSQL provides certain tools to access data from CSQL database. These are

- csql
- catalog
- csqldump

### 8.1. csql

The CSQL tool contains three argument and all are optional. If you type only CSQL in command prompt without user name and password, the tool responds

#### Arguments :

`[-u username]` username of the user  
`[-p password]` password of the user  
`[-s sqlfile]` This file should contain SQL statements to be executed

CSQL tool, which is a sub-shell used to access the CSQL database. It supports most of the standard SQL statements.

Make sure that csqserver is running prior to running this tool. Run csql to get CSQL sub shell.

```
$ csql
CSQL>
```

### 8.2. catalog

This tool displays meta-data information stored in system and user database.

#### Arguments:

`-u username` username of the user  
`-p password` password of the user  
`-l` lists all the tables with field information  
`-i` reinitialize catalog tables dropping all the tables  
`-d` prints the database usage statistics for system and user database  
`-T table` prints the table information  
`-I index` prints the index information  
`-D lock | trans | proc | chunk` prints debug information for system tables.

If the username is not mentioned then it will list all the tables with only their names. If multiple options are specified then only the last option is considered for processing. Let us understand some of the outputs of the command.

Create two tables in the database as follows with the help of CSQL tool.

```
$ csql
CSQL> create table t1(f1 int, f2 char(20), f3 float);
Statement Executed

CSQL>create table emp(eid int, name char(20), sal float);
statement Executed

CSQL>quit;
```

We have two tables, t1 and emp created. Let us see how the catalog tool displays the details of the two tables.

```
$ catalog -l
<TableNames>
<TableName> t1 </TableName>
<TableName> emp </TableName>
</TableNames>
```

This is a default behavior as mentioned before since there is no username provided.

#### **Explanation for tool arguments:**

- List all the tables with field information and index information

```
$ catalog -u root -p manager -l
```

- This will print the database usage statistics

```
$ catalog -u root -p manager -d
```

- Field and Index information of the specified table

```
$ catalog -u root -p manager -T <table-name>
```

- This will list index information of the index specified.

```
$ catalog -u root -p manager -I <index-name>
```

- This will list process table information

```
$ catalog -u root -p manager -D proc
```

- This will list lock table information

```
$catalog -u root -p manager -D lock
```

- This will list transaction table information

```
$catalog -u root -p manager -D trans
```

- List all the chunk information for both system and user

```
$catalog -u root -p manager -D chunk
```

- This is same as -d option

```
$ catalog -u root -p manager -ild
```

- This will drop all the tables from the database

```
$ catalog -u root -p manager -i
```

### 8.3. csqldump

csqldump is a tool that generates a SQL file, which is a dump of all the tables with records in CSQL database.

#### Syntax :

```
csqldump  
  
[-u username]  
[-p password]  
[-c]  
[-n numberOfStmtPerCommit]  
[-T <tableName>]
```

## Usage:

`[-u username]`

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

`[-p passwd]`

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

`[-c]`

It includes all the cache tables in the dump output.

`[-n noOfStmtsPerCommit]`

This option is worked for number of statements per commit. Default value is 100. If system database size is bigger, then it shall be increased.

`-t <table_name>`

Will dump only the table specified with this option

**Note:** csqldump does not output cache tables by default. Use `-c` option to include cache tables.

Now let us create some tables and insert some of the tuples into those tables. Run the csqserver in one terminal. Open another terminal, and run csq tool.

```
$ csq1
```

```
CSQL> set autocommit off;  
AUTOCOMMIT Mode is set to OFF
```

```
CSQL> create table t1(f1 int, f2 char(30), primary  
key(f1));  
Statement Executed
```

```
CSQL> insert into t1 values(1, 'Lakshya');  
Statement Executed: Rows Affected = 1
```

```
CSQL> insert into t1 values(10, 'Uttara');  
Statement Executed: Rows Affected = 1
```

```
CSQL> commit;
```

```
CSQL> create table emp(empId int, empName char(40), empSal  
float, primary key (empId));  
Statement Executed
```

```
CSQL> insert into emp values(1001, 'Jitendra', 1000.00);
Statement Executed: Rows Affected = 1

CSQL> insert into emp values(1002, 'Dharmendra', 2000.00);
Statement Executed: Rows Affected = 1

CSQL> commit;
CSQL> quit;
```

#### **\$ csqldump**

```
CREATE TABLE t1 (f1 INT NOT NULL , f2 CHAR (30));
CREATE INDEX t1_idx1_Primary on t1 ( f1 ) UNIQUE;
CREATE TABLE emp (empId INT NOT NULL , empName CHAR (40),
empSal FLOAT );
CREATE INDEX emp_idx1_Primary on emp ( empId ) UNIQUE;
SET AUTOCOMMIT OFF;
INSERT INTO t1 VALUES(1, 'Lakshya');
INSERT INTO t1 VALUES(10, 'Uttara');
COMMIT;
INSERT INTO emp VALUES(1001, 'Jitendra',1000.000000);
INSERT INTO emp VALUES(1002, 'Dharmendra',2000.000000);
COMMIT;
```

csqldump displays records as SQL INSERT statements. For each record which is present in CSQL database table, it displays INSERT statement with respective values of that record.

