InfiniDB®
Concepts Guide

Release: 4.6
Document Version: 4.6-1

InfiniDB Concepts Guide
July 2014
Copyright © 2014 InfiniDB Corporation. All Rights Reserved.

Contents

1 Introduction

Welcome to the InfiniDB Concepts Guide. This manual is intended to introduce you to the InfiniDB analytic database and provide you with a general understanding of how InfiniDB is architected and how the database functions. After finishing this manual you should have a solid understanding of InfiniDB and be equipped to be productive with the database, both from an administration point of view as well as a development standpoint.

1.1 Audience

This guide is intended for a number of different roles, including:

- Database Administrators
- Application and Web Developers
- Data Architects
- System and Web Administrators

1.2 List of Documentation

The InfiniDB Database Platform documentation consists of several guides intended for different audiences. The documentation is described in the following table:

Document	Description
InfiniDB Administrator's Guide	Provides detailed steps for maintaining InfiniDB.
InfiniDB Apache Hadoop^{TM} Configuration Guide	Installation and Administration of an InfiniDB for Apache Hadoop system.
InfiniDB Minimum Recommended Technical Specifications	Lists the minimum recommended hardware and software specifications for implementing InfiniDB.
InfiniDB Installation Guide	Contains a summary of steps needed to perform an install of InfiniDB.
InfiniDB Multiple UM Configuration Guide	Provides information for configuring multiple User Modules.
InfiniDB SQL Syntax Guide	Provides syntax native to InfiniDB.
Performance Tuning for the InfiniDB Analytics Database	Provides help for tuning the InfiniDB analytic database for parallelization and scalability.
InfiniDB Windows Installation and Administrator's Guide	Provides information for installing and maintaining InfiniDB for Windows.

1.3 Obtaining documentation

These guides reside on our http://www.infinidb.co website. Contact support@infinidb.co for any additional assistance.

1.4 Documentation feedback

We encourage feedback, comments, and suggestions so that we can improve our documentation. Send comments to support@infinidb.co along with the document name, version, comments, and page numbers.

1.5 Additional resources

If you need help installing, tuning, or querying your data with InfiniDB, you can contact support@infinidb.co.

2 What is the InfiniDB Analytic Database?

The InfiniDB analytic database is a software solution specially designed to manage and retrieve large volumes of collective information, commonly referred to as data warehouses, data marts, and analytic databases/data stores. InfiniDB is architected to overcome the limitations found in traditional relational databases that are designed more for transactional processing operations rather than analytic or business intelligence work. Through its multi-threaded, column-oriented architecture, InfiniDB is able to utilize inexpensive commodity hardware to effectively manage terabytes of data in a scale up (adding more CPU's and RAM on one machine) configuration and deliver very fast response times for queries that would otherwise take longer to complete on conventional relational database management systems (RDBMS's). In addition, it allows a scale out design to be deployed in massive parallel processing (MPP) fashion. This configuration parallelizes work between all participating machines and has the capability of providing linear performance gains as more nodes are added to the system.

More recently, InfiniDB added native HDFS integration so that InfiniDB can be used as a SQL-on-Hadoop query engine. Because InfiniDB was built from the ground up for high performance analytics at scale it dramatically outperforms other SQL-on-Hadoop solutions that were built as incremental add-ons on top of the batch processing focused Hadoop platform.

Because business intelligence and data analytics have become such a key mechanism for businesses to use in making critical day-to-day decisions and staying competitive, InfiniDB believes that powerful analytic database capabilities should be made available to everyone. This being the case, InfiniDB offers a free and open source version of the InfiniDB analytic database (InfiniDB) that anyone can download and use free of charge under the GPLv2 license. InfiniDB also makes available a commercial version of its analytic database (Enterprise) that provides a couple of advanced features, 24 x 7 technical support, and more.

2.1 Introduction to Analytic Databases

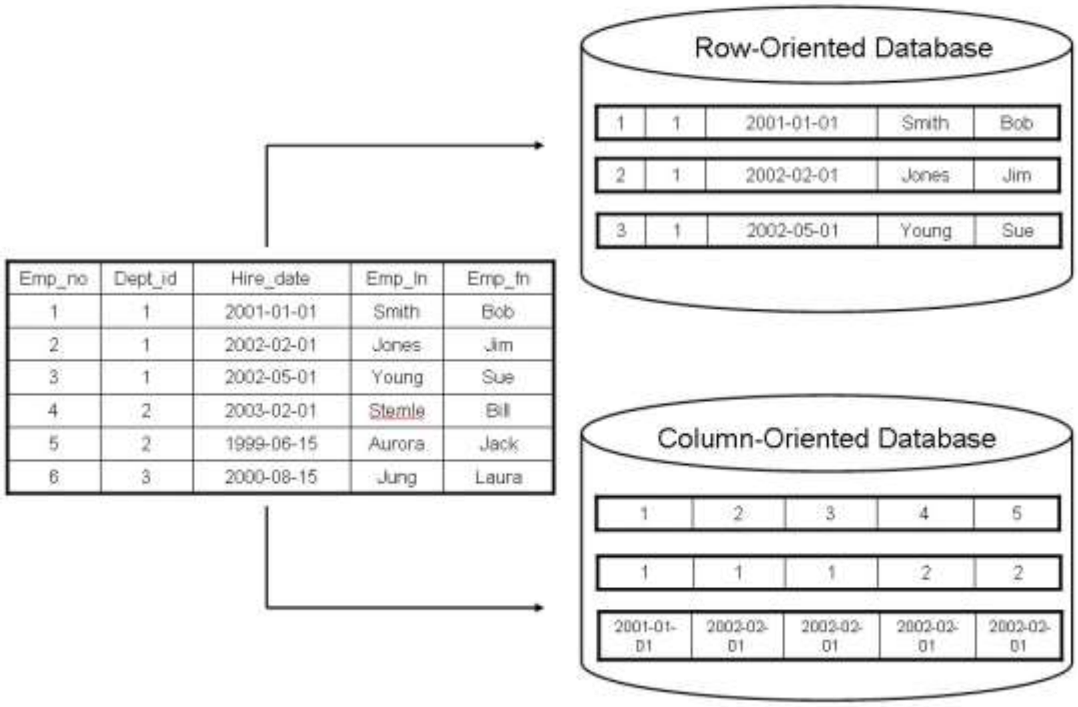
Nearly all legacy relational databases currently being offered today were primarily designed to handle online transactional processing (OLTP) workloads. A transaction (e.g. an online order for a product) typically maps to one or more rows in a relational database, and the vast majority of RDBMS designs are based on a per row paradigm. For transactional-based systems, this architecture is well suited to handle the input of incoming data and row-by-row removals of information.

However, for applications that are very read intensive and selective in the information being requested, the OLTP database design isn't the best model. Whereas transactions are row-based, most database queries are column-based. Inserting and deleting transactional data are well served by a row-based system, but selective queries that are only interested in a few columns of a table are handled much better by a column oriented architecture. On average, a row-based system expends 5 to 10 times the physical I/O that a column-based database does to retrieve the same information. Taking into account that physical I/O is typically the slowest part of a query, and that an analytical query typically touches significantly more rows of data that a typical transactional database operation, the performance gap between row-oriented architectures and column-oriented architecture oftentimes widens as the database grows.

To get around their selective query inefficiencies, row-based RDBMS's utilize indexing, horizontal partitioning, materialized views, summary tables, and parallel processing, all of which can help complex queries perform better, but each comes with their own set of drawbacks as well. For example, while indexing can certainly help queries complete faster in some cases, they also require more storage, impede insert/update/delete and bulk load operations (because the indexes must be maintained as well as the underlying table), and can actually degrade performance when they become heavily fragmented.

Moreover, in business intelligence/analytic environments, the ad-hoc nature of such scenarios makes it nearly impossible to predict which columns will need indexing, so tables end up either being over-indexed (which causes load and maintenance issues) or not properly indexed and so many queries end up running much slower than desired.

Column-oriented databases designed especially for analytics overcome the limitations that exist in traditional RDBMS systems by storing, managing, and querying data based on columns rather than rows. Because only the necessary columns in a query are accessed rather than entire rows, I/O activities as well as overall query response times can be reduced. The end result is the ability to interrogate and return query results against either moderate amounts of information (tens or hundreds of GB's) or large amounts of data (1-n terabytes) in less time that standard RDBMS systems can.



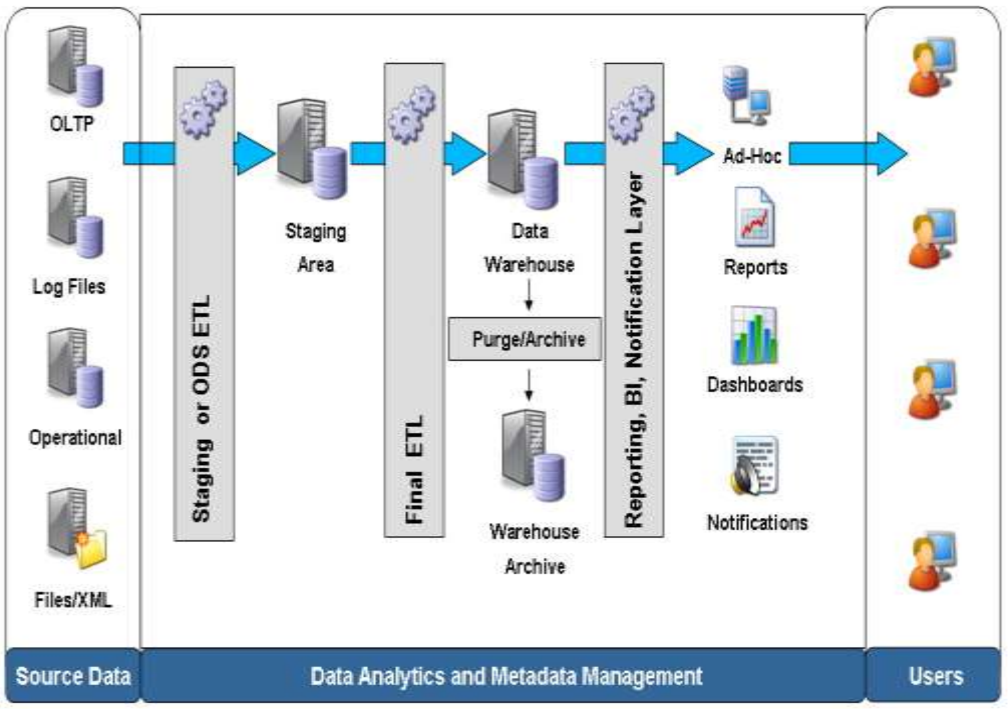
Emp_no	Dept_id	Hire_date	Emp_ln	Emp_fn
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Stemle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura

2.2 *Basic Analytic Database Use Cases*

When it comes to what applications and systems will benefit from an analytic database, there are three common use cases where analytic databases typically excel:

1. Data warehouses, data marts, and other business intelligence (BI) data stores
2. General purpose reporting databases
3. Read-intensive segments of an overall application where read/search/lookup portions of the application are served by an analytic/query database and the transactional segments (e.g. order entry, etc.) are managed by a traditional relational database

Business intelligence (BI) and data analytics concern themselves with smartly utilizing information to make critical business decisions. The lifecycle of BI and data analytics typically resembles the following:



The graphic above illustrates how operational/transactional data is fed into an extract-transform-load (ETL) phase, which then ultimately populates an analytic database or data warehouse. The analytic database is then used by various business intelligence processes to service decision makers that access the data through ad-hoc queries, pre-built dashboards and reports, or through automatic notifications sent when key data changes occur.

Nearly all the relational databases currently being offered today were and are primarily designed to handle online transactional processing (OLTP) workloads. A transaction (e.g. an online order for a book through Amazon or another Web-based book dealer) typically maps to one or more rows in a relational database, and the vast majority of RDBMS designs are based on a per row paradigm. For transactional-based systems, this architecture is well-suited to handle the input of incoming data.

However, for applications that are very read intensive and selective in the information being requested, the OLTP database design isn't a model that typically holds up well. Whereas transactions are row-based, most database queries are column-based. Inserting and deleting transactional data are well served by a row-based system, but selective queries that are only interested in a few columns of a table are handled much better by a column-oriented architecture. On average, a row-based system does 5-10x the physical I/O that a column-based database does to retrieve the same information. Taking into account that physical I/O is typically the slowest part of a query, and that an analytical query typically touches significantly more rows of data that a typical transactional database operation, the performance gap between row-oriented architectures and column-oriented architecture oftentimes widens as the database grows.

To get around their selective query inefficiencies, row-based RDBMS's utilize indexing, horizontal partitioning, materialized views, summary tables, and parallel processing, all of which can provide benefits for intensive queries, but each comes with their own set of drawbacks as well. For example, while indexing can certainly help queries complete faster in some cases, they also require more storage, impede insert/update/ delete and bulk load operations (because the indexes must be maintained as well as the underlying table), and can actually degrade performance when they become heavily fragmented. Moreover, in business intelligence/analytic environments, the ad-hoc nature of such scenarios makes it nearly impossible to predict which columns will need indexing, so tables end up either being over-indexed (which causes load and maintenance issues) or not properly indexed and so many queries end up running much slower than desired.

Column-oriented databases designed especially for analytics overcome the limitations that exist in traditional RDBMS systems by storing, managing, and querying data based on columns rather than rows. Because only the necessary columns in a query are accessed rather than entire rows, I/O activities as well as overall query response times are dramatically reduced. The end result is the ability to interrogate and return query results against either moderate amounts of information (tens or hundreds of GB's) or large amounts of data (1-n terabytes) in a fraction of the time that standard RDBMS systems can.

There are clear benefits that can be obtained when choosing a database designed for analytics over a legacy row-based RDBMS. In his report on column-oriented databases, Philip Howard (Research Director for Bloor Research) says:

> "Columns provide better performance at a lower cost with a smaller footprint: it is difficult to understand why any company seriously interested in query performance would not consider a column-based solution."[1]

2.3 Core Features of InfiniDB

The InfiniDB analytic database offers a number of features for those looking to improve performance in intensive read environments, which include read-only Web applications, data marts, data warehouses, and other business intelligence, analytic, or query-centric use cases.

InfiniDB offers a free and open source (FOSS) edition of its database that is designed to run on a single server and take advantage of modern multi-CPU/core machines to process both queries and write activity in a parallel, multi-threaded manner. The core features of the open source InfiniDB server are as follows:

- **Column-oriented architecture**: InfiniDB is architected in a column-based manner vs. a legacy row-based format. This allows InfiniDB to service queries that need to read anywhere from a few hundred GB to many TB's of information faster than row-oriented databases.
- **Multi-threaded design**: InfiniDB is multi-threaded and can make use of today's modern hardware that is multi-CPU/core based. More CPU's and/or cores allow InfiniDB to increase performance with no application modifications being necessary.

1 Philip Howard, *What's Cool About Columns*, Bloor Research. March 2008.

- **Automatic vertical and horizontal partitioning**: Being column-oriented, InfiniDB transparently uses vertical partitioning to store data and only read the needed columns to satisfy a query. But InfiniDB also uses a form of logical horizontal range partitioning that does not require special storage placements or design. Using both vertical and logical horizontal range partitioning allows InfiniDB to reduce I/O in both directions (column and row). Both vertical and horizontal partitioning are automatically handled by the InfiniDB database and require no user intervention.
- **High concurrency**: InfiniDB's limits as far as concurrency is concerned are only restricted by the server machine's capabilities; no theoretical concurrency limit exists.
- **High-speed data loader**: InfiniDB supports traditional insert/update/delete operations, but to load massive amounts of data, a high-speed load utility is made available. InfiniDB's loader allows TB's of data to be loaded much more quickly into an InfiniDB database than other traditional methods.
- **DML support:** In addition to supporting the high-speed bulk loading of data, InfiniDB supports full DML (insert, update, delete) operations as well. Non-transactional "LOAD DATA INFILE" and "INSERT INTO SELECT FROM" SQL statements have the ability to use the high-speed data loader for significant performance increases – allowing you to easily transfer data from other tables in your InfiniDB or MySQL environment or quickly load data from a file within a MySQL operation.
- **Transactional support:** ACID-compliant transactional support is provided in the InfiniDB database. Transactions can easily be committed or rolled back, and deadlock detection support is also provided to handle conflict resolution.
- **Crash recovery:** InfiniDB provides for full crash recovery capabilities. In the event of a system crash, InfiniDB automatically maintains data integrity and upon system restart, InfiniDB supports roll forward and back operations to return the database to a consistent state.
- **MVCC design:** InfiniDB supports multi-version concurrency control (MVCC) or "snapshot read" so query operations are never blocked on a table; a query will always see the data as it existed at the time the query was issued.
- **No need for indexing:** Because of InfiniDB's transparent use of both vertical and logical horizontal partitioning, there is no need for indexing. In essence, the data in a column-oriented database is the index. In addition, InfiniDB also automatically maintains a small, important structure called the Extent Map, which is used to reduce I/O. The Extent Map also removes the need for any manual data partitioning.
- **Low Maintenance:** In addition to removing the need to index tables, InfiniDB also doesn't require other objects such as materialized views and summary tables to achieve fast performance. This helps remove the need for typical database administration maintenance functions as well as cuts down on the complexity of the overall system.
- **Dictionary String Compression:** InfiniDB uses a token-dictionary strategy to manage strings longer than 8-bytes fixed length, and 7 bytes variable length. The column file stores tokens (pointers) to a separate dictionary file containing the longer strings. Repeated string values within the table are stored at a given location within the dictionary with repeated tokens pointing to a shared string entry.
- **Alter table support:** Columns may also be added and dropped from a table with the `ALTER TABLE` command, with specialized support for online column additions.

- **Performance diagnostics:** To help tune performance, InfiniDB supplies monitoring and diagnostic utilities that help a user monitor their database and troubleshoot poorly running SQL.
- **MySQL front end:** InfiniDB utilizes MySQL for its front end. This allows anyone familiar with MySQL to become immediately productive with InfiniDB. For those not acquainted with MySQL, the learning curve is minimal as MySQL supports almost all standard SQL operations. Moreover, there are many freely supplied GUI tools from MySQL as well as other vendors that may be used to develop against and administer an InfiniDB database.
- **Compression with Real-Time Decompression:** InfiniDB uses a compression strategy optimized for read performance from disk while still offering excellent compression. This allows for systems that are I/O bound when reading from disk to improve performance. Compression is on by default, but can be turned on or off at the instance, level, table level or the column level. Columnar storage offers excellent compressibility because similar data is stored within each column file.
- **Partition Disable and Drop:** InfiniDB uses an automatic partitioning strategy that does not require the DBA to name or define the partitions up front. When dropping partitions, a show partitions command will display the ranges of values that exist within each partition of data. For partitions that no longer contain any rows of interest to the business, the partitions can be disabled to remove the data logically, or dropped to remove the associated files as well.
- **User Defined Functions (UDFs):** InfiniDB allows for creation of fully parallel and distributed UDFs that run as an integrated function within the InfiniDB engine. This enables creation of custom functionality that leverage the full benefits of InfiniDB's integrated map reduction capability, distributing the custom functionality across all available cores within the distributed layer.
- **Massive parallel processing (MPP) capable**: InfiniDB can use multiple commodity hardware machines to achieve linear increases in overall performance. Adding inexpensive hardware to an InfiniDB configuration allows the database to increase processing power so response times can sometimes be reduced in half just with the addition of new nodes. This makes it easy to adjust for growing data volumes, increased user activity, or the desire to meet better overall performance goals. Moreover, this can be accomplished in a dynamic fashion where the InfiniDB system doesn't go down or be taken offline when new nodes are added.
- **Distributed shared-nothing data cache**: In a multiple-node InfiniDB configuration, data is distributed among the various nodes and their data caches. No node shares data with the other, however all are accessed in the InfiniDB MPP architecture when data is read to satisfy queries. In essence then, InfiniDB creates one large logical data cache that is accessed in a distributed fashion in parallel by all participating nodes. This allows InfiniDB to literally cache large databases when enough nodes are present with generous amounts of memory.
- **Automatic failover**: InfiniDB's architecture contains built-in automatic failover for the software modules responsible for managing user queries and performing the actual database I/O operations.
- **Automatic concurrency scale-out**: Nodes can be added to a configuration that allows concurrency to be automatically scaled across 1-n machines.
- **Automated software patch management:** When patches or upgrades need to be applied to more than one InfiniDB server, an automatic process takes the software from the first node where an upgrade is applied and automatically upgrades all other participating nodes.

All the above help novices create and manage fast running data marts, warehouses, and other analytic databases without specialized knowledge of analytic/data warehouse design and implementation.

2.4 Hardware/Software Support

The InfiniDB database supports the following hardware and software configurations:

- Commodity Intel and AMD hardware, 64-bit.
- Linux, which includes:
  - Red Hat Enterprise
  - CentOS
  - Debian
  - Ubuntu
  - AWS (Amazon) EC2

3 The InfiniDB Architecture

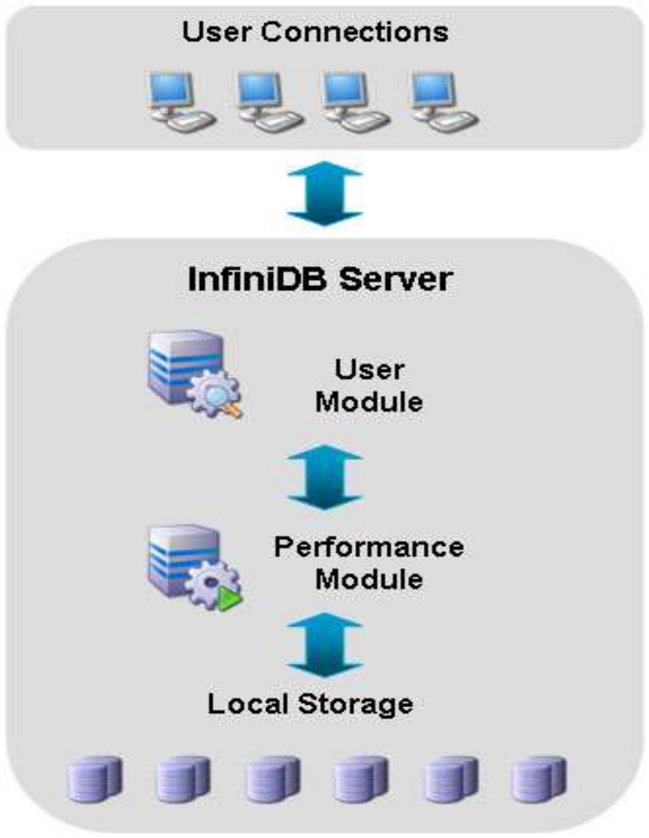
This section discusses the InfiniDB architecture and the different components that comprise the database.

3.1 *Overview of InfiniDB*

InfiniDB's architecture consists of a number of different components, all of which work together to comprise an InfiniDB database. These components include:

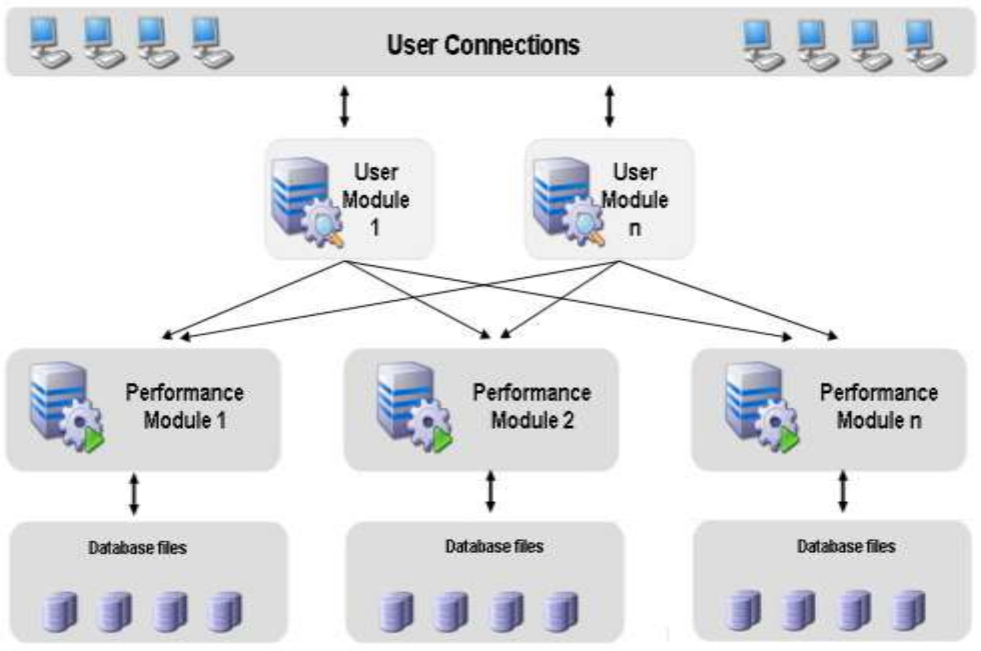
- **User Module**: The User Module is made up of a small MySQL instance (sometimes referred to separately as the Director Module) and a number of InfiniDB processes that handle concurrency scaling. The User Module is also responsible for breaking down SQL requests and distributing the various parts to one or more Performance Modules that actually retrieve requested data from either memory caches or disk. Finally, the User Module assembles all the query results from the various participating Performance Modules to form the complete query result set that is returned to the user. Note there can more than one User Module , which provides a high availability configuration and a method for query workload balancing.
- **Performance Module**: The Performance Module is responsible for storing, retrieving, and managing data, processing block requests for query operations, and passing it back to the User Module(s) to finalize query requests. The Performance Module selects data from disk and caches it in a shared nothing data cache that is part of the server on which the Performance Module resides. MPP is accomplished by allowing the user to configure as many Performance Modules as they would like; each additional Performance Module adds more cache to the overall database as well as more processing power.
- **Storage**: InfiniDB is extremely flexible with respect to the storage system. When running on-premise, InfiniDB can use either local storage or shared storage (e.g. SAN) to store data. In the Amazon EC2 environment, InfiniDB can use ephemeral or Elastic Block Store (EBS) volumes. When data redundancy is required for a shared-nothing deployment, InfiniDB is built to integrate with GlusterFS and the Apache Hadoop Distributed File System (HDFS).

There are a number of different configurations that can be architected for an InfiniDB database. The InfiniDB database can be configured with either single server or MPP scale-out. The single server is depicted the following way:



In the above configuration, all modules run on the same server and take advantage of all the CPU processing power on the machine allowing scale-up processing power.

In a MPP scale-out option, there are a number of configurations that can be designed. One is a shared nothing disk architecture:



In the above configuration, concurrency scale-out is accomplished by additional User Module nodes whereas I/O scale-out is accomplished via adding Performance Module nodes.

Note that User and Performance Modules can be in close proximity to each other or be separated out in different data centers and geographic locations.

3.2 InfiniDB Components

This section discusses each major component of the InfiniDB database in detail as well as participating structures that contribute to the database's overall operation.

3.2.1 User Module

In brief, the InfiniDB User Module manages and controls the operation of end user queries. It maintains the state of each query, issues requests to one or more Performance Modules to perform work on the behalf of a query, and performs end resolution of a query by aggregating the various result sets from all participating Performance Modules into one that is ultimately returned to the end user.

3.2.1.1 Director Module/MySQL Interface

A separate component of the InfiniDB User Module is sometimes referred to as the Director Module. In essence, this is a MySQL instance that InfiniDB relies on to perform the following basic functions:

- Connection validation
- Parsing of SQL statements
- General SQL plan generation
- Final result set distribution activities

Within this component, InfiniDB supplies a few processes to manage the server. The Process Manager (ProcMgr) is responsible for starting, monitoring and re-starting all InfiniDB processes. It utilizes another process, called Process Monitor (ProcMon) on each machine to keep track of InfiniDB processes.

InfiniDB also uses a number of connection libraries that take query plans in MySQL format and converts them to InfiniDB format. The InfiniDB format is still essentially a parse tree, but execution hints from the optimizer are added to assist the User Module in converting the parse tree to an InfiniDB Job List. There is also a MySQL connector library (libcalmysql.so), which provides a MySQL Storage Engine API (handler) interface to MySQL for InfiniDB.

3.2.1.2 Primary User Module Operations

The User Module performs these core functions for the InfiniDB database:

- Transform MySQL plan into an InfiniDB Job List
- Perform InfiniDB OID (object ID) lookups from the InfiniDB system catalog
- Inspects the Extent Map (described in detail later in this document) to reduce I/O, which is accomplished via the elimination of unnecessary extents
- Issues instructions (sometimes referred to as 'primitive operations') to the Performance Modules
- Executes hash joins as needed (depending on size of smaller table in the join). Helps manage distributed hash joins by sending any hash maps needing processing to Performance Modules
- Execute cross-table-scope functions and expressions that happen after a hash join
- Receives data from the Performance Modules and re-transmits back to them if needed
- Executes follow-up steps for all aggregation and distinct processing
- Return data back to the MySQL interface

The primary job of the User Module is to handle concurrency scaling. It never directly touches database files and does not require visibility to them. It uses a machine's RAM in a transitory manner to assemble partial query results into a complete answer that is ultimately returned to a user. As of this time, a query lives in one and only one User Module, but a query can be broadcast to all available Performance Modules to scale out actual I/O work.

3.2.1.3 User Module Processes

The User Module contains several processes: the Execution Manager or ExeMgr, and the DML, DDL and import distribution managers (DMLProc, DDLProc and cpimpotrt). ExeMgr listens on a TCP/IP port for query parse trees from the Director Module (or MySQL instance). ExeMgr is responsible for converting the query parse tree into an InfiniDB job list, which is the sequence of instructions necessary to answer the query. ExeMgr walks the query parse tree and iteratively generates job steps, optimizing and re-optimizing the job list as it goes. The major categories of job steps are application of a column filter, processing a table join, and projection of returned columns. DMLProc and DDLProc distribute DML and DDL to the appropriate Performance Module. Cpimport, when run on the User Module, distributes source files to the Performance Modules.

Each node in a plan is executed in parallel by the job list itself and has the capability of running entirely on the User Module, entirely on the Performance Module or in some combination. Each node uses the Extent Map to determine which Performance Modules to send work orders to (see the later section on the Extent

Map for more information on how this happens). All of these nodes are user-level threads, and each thread can spawn additional threads based on the distribution of data and available system resources.

3.2.1.4 User Module Scale Out

InfiniDB offers transparent User Module scale out. An administrator can configure an InfiniDB system to have more than one User Module so that a system can be scaled to support more user connections.

3.2.2 Performance Module

This section discusses the various aspects of InfiniDB's Performance Module.

3.2.2.1 General Operations

The Performance Module is responsible for performing I/O operations in support of query and write processing. It receives its instructions from a User Module with respect to the work that it does. A Performance Module doesn't see the query itself, but only the set of instructions given it by a User Module. In the respect, a Performance Module is more concerned with performing block oriented I/O operations and not running queries in the standard sense.

The massive parallel processing or scale out capabilities are achieved by adding additional Performance Modules to an InfiniDB configuration. Linear performance gains are realized when more nodes are added to particular setup because of the parallelization of query work among the different Performance Modules. The Performance Module delivers three critical behaviors key to scaling out database behavior: distributed scans, distributed hash joins, and distributed aggregation. The combination of these three behaviors enables true MPP behavior for query intensive environments.

There are three processes that run on the Performance Module: PrimProc, which handles query execution; WriteEngineServer, which coordinates parallel writes; and cpimport, which performs the actual database file updates.

3.2.2.2 Load and Write Processing

A Performance Module node is given the task of performing loads and writes to the underlying persistent storage. WriteEngineServer is responsible for coordinating DML, DDL and imports on each Performance Module. Cpimport is aware of which module It is running on and, when running on the Performance Module, handles the actual updates of the database disk files. In this manner, InfiniDB supports fully parallel load capabilities. DDL changes are persisted within the InfiniDB System Catalog which keeps track of all InfiniDB metadata. Note that InfiniDB provides failover capabilities for the write node to support high availability configurations.

3.2.2.3 Shared Nothing Data Cache

InfiniDB utilizes a shared nothing data cache on multiple Performance Modules. When data is first accessed a Performance Module acts upon the amount of data that it has been instructed to by a User Module and caches them in an LRU-based cache for subsequent access. On dedicated InfiniDB machines, the majority of the box's RAM can be dedicated to a Performance Module's data cache.

Being that the Performance Module cache is a shared nothing design provides at least two key performance benefits. First, there is no data block pinging between participating Performance Module nodes as sometimes occurs in other multi-instance/ shared disk database systems. Second, as more Performance

Module nodes are added to a system, the overall cache size for the database is greatly increased; this allows (in some cases) for an entire database to be cached in RAM for very fast access.

Note that when InfiniDB is used on a standalone server, the data cache acts in a fashion as other standard LRU-based data caches. In a single-server implementation, all requested data from the database passes through the cache because there is only one Performance Module.

3.2.2.4 Adding Additional Performance Capacity

Adding additional Performance Modules is accomplished via a basic configuration set of steps on the User Module(s) for the additional server. The process is transparent to the overall InfiniDB system and usually the new Performance Module node is picked up within 60 seconds.

3.2.2.5 Performance Module Failover

In a multi-node Performance Module configuration, a heartbeat mechanism ensures all nodes are all online and there is transparent failover if a particular Performance Module fails. If a Performance Module abnormally terminates, any in-process queries will error. Any users receiving an error due to a Performance Module failure can simply resubmit their query and have the work performed by the remaining Performance Modules.

In the event of a failover, when the underlying storage data is externally mounted (e.g. EC2 EBS or SAN), the mapping of data blocks to the Performance Modules is re-organized across the working Performance Modules, and the Extent Maps on the User Modules are re-evaluated so that queries are sent to the appropriate remaining Performance Modules. This process is transparent to the user and no manual intervention is required.

When the failed Performance Module is brought back online, InfiniDB will auto-adopt it back into the configuration and begin using it for work.

3.2.2.6 Overview of Query Processing through User and Performance Modules

A top-down view of how InfiniDB processes an end user query is as follows:

1. A request comes in through the MySQL interface. MySQL performs a table operation for all tables needed to fulfill the request and obtains the initial query execution plan from MySQL.
2. InfiniDB utilizes the MySQL storage engine interface to convert the MySQL table objects to InfiniDB objects. These objects are then sent to a User Module.
3. The User Module converts the MySQL execution plan and optimizes these objects into an InfiniDB execution plan. The User Module determines the steps needed to run the query and when they can run.
4. The User Module consults the Extent Map for the locations of the data needed to satisfy the query and performs extent elimination based on the information contained within the Extent Map.
5. The User Module sends commands to one or more Performance Modules to perform block I/O operations.
6. The Performance Module(s) carry out predicate filtering, join processing, initial aggregation of data, and sends data back to the User Module for final result set processing.
7. The User Module performs final result set aggregation and composes the final result set for the query.

8. The User Module returns the result set back for delivery to the user.

3.3 InfiniDB Storage Concepts

This section covers the basic storage concepts of the InfiniDB database.

3.3.1 General Storage Concepts for InfiniDB Data

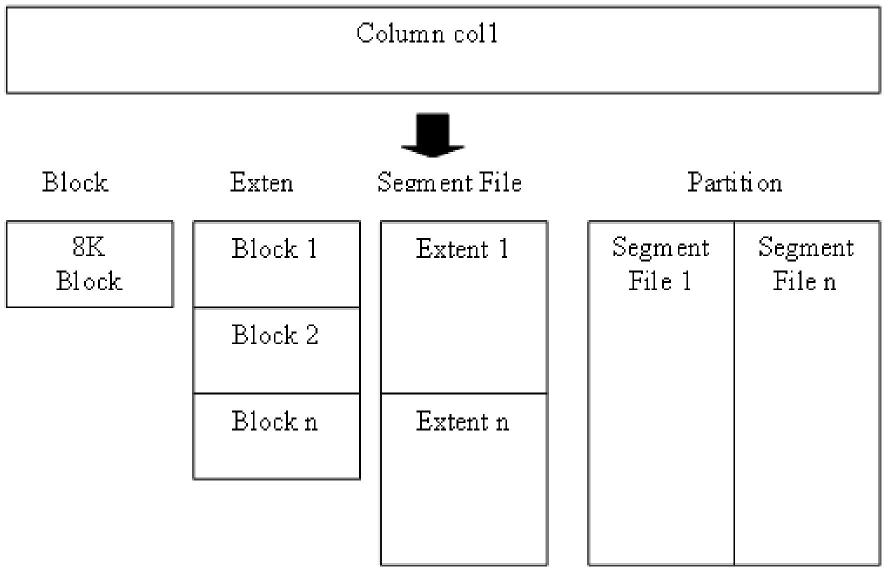
Being a column-oriented database, when a table is created on InfiniDB, the system creates at least one file per column for the table. So, for example, a table created with three columns would have at a minimum, three separately-addressable logical objects created on a SAN or a server's disk drives.

The storage building blocks of an InfiniDB database – from bottom to top – are the following:

- **Block** – a block (also referred to as a page in some RDBMS's) is a physical object, which is 8KB in size. Every database block in a database is uniquely identified by its Logical Block Identifier (LBID). While the InfiniDB block size is not configurable, the number of blocks read by InfiniDB's read ahead mechanism can be customized.
- **Extent –** an extent is a logical measure of 8 million rows that exists within one physical segment file. An extent for 1 byte datatypes consume 8MB; 2 byte datatypes consume 16MB; 4 byte datatypes consume 32MB; 8 byte datatypes consume 64MB; and variable size datatypes consume 64MB. Once an extent becomes full, a new extent is automatically created.
- **Segment File –** a segment file is a physical file on disk that holds a column's data. Once a segment file reaches its maximum number of extents, a new segment file is automatically created.
- **Partition –** a partition is a different concept than that used by standard row-based databases that use partitioning to horizontally divide up the data that is in a table or index. In InfiniDB, a partition is a logical object, which is comprised by one or more segment files. The `FilesPerColumnPartition` parameter controls how many segment files a single partition can have. There is no limit to the number of partitions that a column can have.

A graphical representation of the InfiniDB storage concepts might be:

```
create table t1 (col1 int, col2 varchar(20), col3 date) engine=infinidb;
```



With respect to the placement of segment files on a disk subsystem, directories can be specified within the InfiniDB configuration file as to which disk resources should be used for InfiniDB storage. InfiniDB automatically takes care of spreading all segment files across all specified disk resources so physical read rate of data is maximized.

3.3.2 Compression with Real-Time Decompression

InfiniDB uses a compression strategy optimized for read performance from disk while still offering excellent compression. This allows for systems that are I/O bound when reading from disk to improve performance.

Compression is on by default, but can be turned on or off at the table level or the column level and in addition can be controlled at a session level by setting a variable: infinidb_compression_type. The default setting enables compression for all tables created after the 2.0 release. Note that previously existing tables are left unchanged. To create a compressed version of a previously un-compressed table simply create a new version of the table as compressed, then select the data out and use the high-speed import method (cpimport) to load the new, compressed table. The tables can then be re-named and the uncompressed table dropped.

Columnar storage offers excellent compressibility because similar data is stored within each column file. Most data sets will show excellent compression rates, saving between 65% and 95% of space, however the actual space savings depends on the randomness of the data and the number of distinct values that exist.

The compression strategy for InfiniDB is tuned to accelerate the decompression rate, maximizing the performance benefits when reading from disk. This is ideal for standard Data Warehouse uses of the system that are typically write once, read many. Accelerating reads from disk are ideally suited for systems that follow best-practices and minimize manipulation of existing data. For columns that are frequently updated after the initial load, InfiniDB may offer better performance for updates if that column is defined as un-compressed. Adding or removing columns do not require rebuilding the table for a columnar database, so evaluation of multiple options can be done in parallel.

3.3.3 Version Buffer and Files

InfiniDB utilizes a structure termed a Version Buffer to store the disk blocks which are being modified. It also is used to manage rollback activities for transactions as well as service the MVCC (multi-version concurrency control) or “snapshot read” function of the database so that InfiniDB can offer a query consistent view of the database. All statements in InfiniDB run at a particular version (or snapshot) of the database, which the system refers to as the System Change Number (SCN). Although it is called the Version Buffer, it is composed of both memory and disk structures.

The Version Buffer utilizes in-memory hash tables to supply memory access to in-flight transaction information. The initial size upon startup is 4MB with the memory region growing from that amount to handle blocks that are being modified by a transaction. Each entry in the hash table is a 40-byte reference to an 8K block that is being modified.

The number of rows being updated is not the limiting factor for the Version Buffer, rather the number of disk blocks that are being updated. The size can be increased but caution should be used since updating more disk blocks can allow the update/delete statements to run for long periods of time and if a problem is encountered a rollback would also take a long period of time.

The Version Buffer files default to a 1GB-sized file on each DBRoot, which is configurable via the VersionBufferFileSize parameter. The Version Buffer files are spread across each DBRoot in the system.

NOTE: when InfiniDB is configured to run as a Hadoop query engine over HDFS, the MVCC function (and hence usage of the Version Buffer files) is disabled. HDFS is a write-only file system and hence block-level versioning in the MVCC model is not operationally practical. InfiniDB supports statement level tracking and rollback capability for DML operations when running over HDFS. Queries over HDFS operate in more of an “eventual consistency” mode where queries will converge to the proper result at a point in time when updates are completed, but queries issue concurrently with updates will use whatever block happens to be current at the time the block-level execution occurs.

3.3.4 Transaction Log

To manage transactions and perform roll forward actions for the database in the event of a crash recovery situation, InfiniDB utilizes a Transaction Log. The Transaction Log contains entries for DDL/DML statements as well as markers for bulk load actions. In the current implementation of InfiniDB on Linux, the Transaction Log is named `data_mods.log` and is installed on the `/var` filesystem.

For backup and recovery purposes, an archive of the transaction log is performed on a time-based interval, configurable in the InfiniDB configuration file (`TransactionArchivePeriod`). Each transaction log dump has a timestamp appended to the end so it is a unique file on the machine. After the transaction log is dumped, the active log is truncated and ready for new entries.

To view Transaction Log information, a user can employ the ReplayTransactionLog utility, which reports on DDL/DML statements that have been executed. Note that bulk load operations are not included in the Transaction Log.

3.4 The Extent Map

To increase speed of query processing, InfiniDB utilizes a smart structure known as the Extent Map, which removes the need for indexing, manual table partitioning, materialized views, summary tables, and other structures and objects that row-based databases must implement to overcome their row-based architectures.

As previously stated, an extent is a logical block of space that exists within a physical segment file, and is anywhere from 8-64MB in size. Each extent supports the same number of rows, with smaller data types using less space on disk. The Extent Map catalogs all extents and their corresponding blocks (LBID's). The Extent Map also maintains minimum and maximum values for a column's data within an extent.

Note that InfiniDB utilizes both the Extent Map and the Version Buffer to service query requests. The combination of these two structures is called the Block Resolution Manager (BRM).

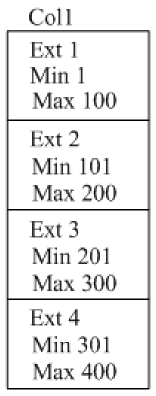
A master copy of the Extent Map exists on the primary Performance Module. Upon system startup, the file is read into memory and then physically copied to all other participating User and Performance Modules for disaster recovery and failover purposes. All nodes keep the Extent Map resident in memory for quick access. As extents are modified, updates are broadcasted to all participating nodes as well.

3.4.1 How the Extent Map Works

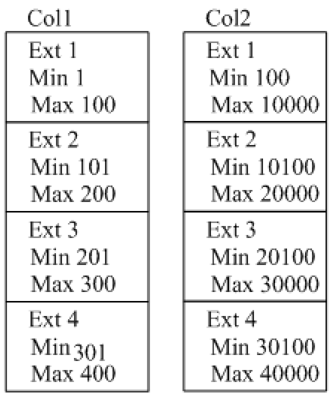
The Extent Map provides the ability for InfiniDB to only retrieve the blocks needed to satisfy a query, but it also provides another benefit – that of logical range partitioning. This is accomplished via the minimum and maximum values for each extent that are contained within the Extent Map. Extent elimination is first accomplished in InfiniDB via the column-oriented architecture (only needed columns are scanned), but the process is accelerated because of this logical horizontal partitioning that is implemented in the Extent Map.

This automatic extent elimination behavior is well suited for series, ordered, patterned, or time-based data where data is loaded frequently and often referenced by time. Near real-time loads with queries against the leading edge of the data can easily show good extent elimination for all of the date/time columns as well as an ascending key value. Any column with clustered values is a good candidate for extent elimination.

To eliminate an extent when a column scan involves a filter, that filter value is compared to the minimum and maximum values stored in each extent for the column:



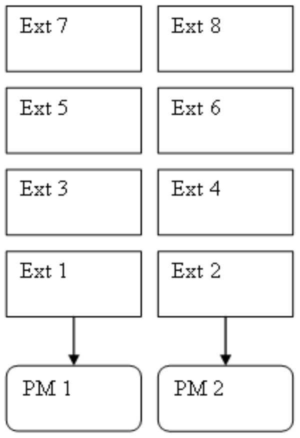
In the above figure, if a `WHERE` column filter of "`COL1 BETWEEN 220 AND 250`" is specified, InfiniDB will eliminate extents 1, 2 and 4 from being scanned, saving ¾ of the I/ O and many comparison operations. If the extent cannot possibly contain any rows that would match the filter, that extent is ignored entirely. Additionally, since each extent eliminated for a column also eliminates the same extents for all the other columns in a table, impossible filter chains can be quickly identified without any I/O being required. For example, take the following two columns and their Extent Map information:



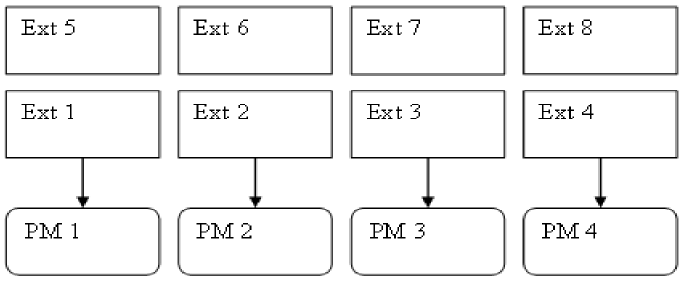
If a column WHERE filter of "`COL1 BETWEEN 220 AND 250 AND COL2 < 10000`" is specified, InfiniDB will eliminate extents 1, 2 and 4 from the first column filter, then, looking at just the matching extents for COL2 (i.e. just extent 3), it will determine that no extents match and return zero rows without doing any I/O at all.

3.4.2 I/O Processing and Workload Distribution

Because blocks are allocated within extents, and extents map to contiguous areas on disk (segment files), a very even distribution of work across any number of Performance Modules can be accomplished. For example, in a two Performance Module system, a scan of a column with 8 extents would have its distribution look very much like:



A four Performance Module system would look like:



Such an architecture allows InfiniDB to deliver linear performance results for most queries as work is smartly divided up among all participating Performance Modules.

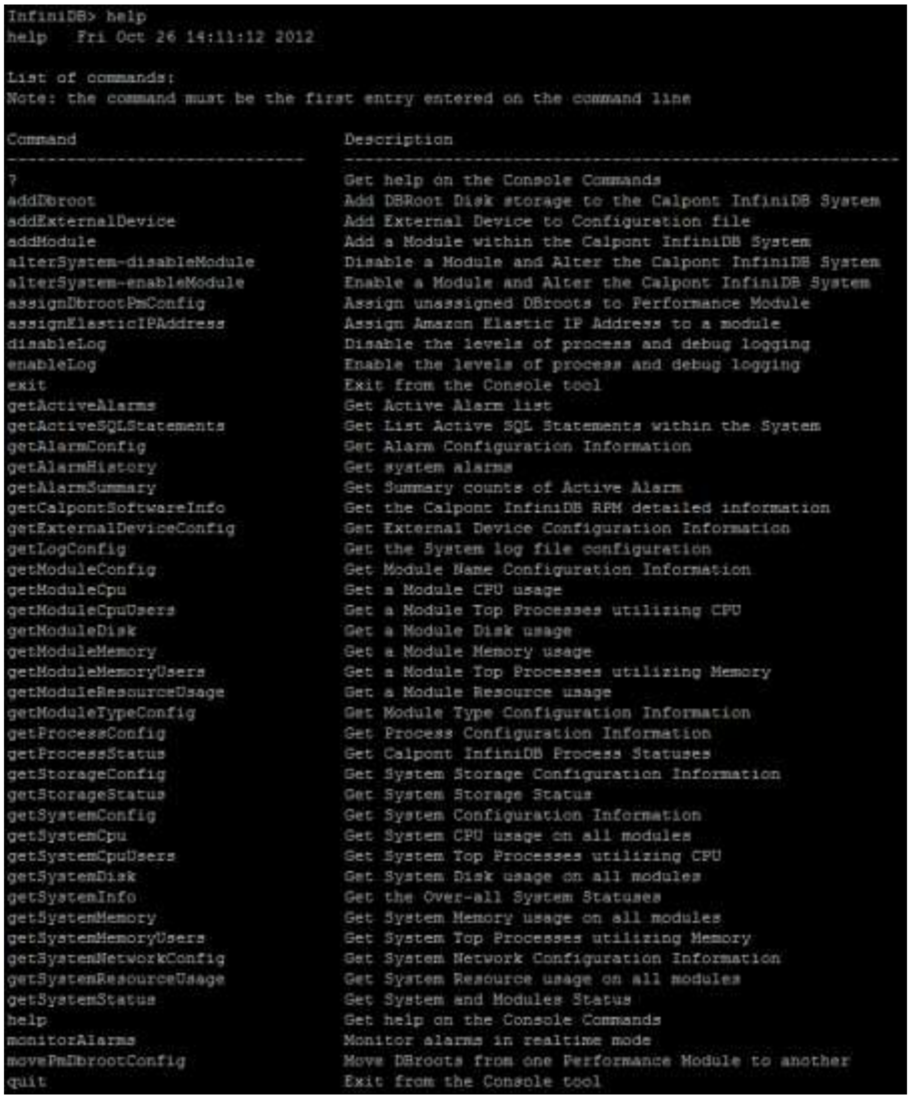
To assist in large scan operations, InfiniDB utilizes a read ahead mechanism that is configured via the `ColScanReadAheadBlocks` parameter, which is currently defaulted to 512. However, to prevent from reading data that may not be necessary to satisfy a query, InfiniDB also makes use of a prefetch threshold parameter (`PrefetchThreshold`, defaulted to 5) so that if the system determines the number of blocks needed for a query is below the threshold, only the needed blocks will be read.

One other item of note is that InfiniDB supports hash joins, which at the time of this writing, MySQL does not. Moreover, InfiniDB's hash joins can be distributed and processed in parallel when a multi-node, MPP configuration is used.

4 Database Administration

InfiniDB ships with the InfiniDB management or command console, which is used to issue administration commands to the InfiniDB database. It allows an administrator to configure, monitor, and manage one or more InfiniDB servers. The utility name is cmconsole and can be invoked by any operating system user with execute privileges on InfiniDB binaries.

Obtaining help for supported commands is available by simply typing help and reviewing the list of commands. Specific command help can be obtained by typing help followed by the desired command.



```
InfiniDB> help
help   Fri Oct 26 14:11:12 2012

List of commands:
Note: the command must be the first entry entered on the command line

Command                           Description
------------------------------    --------------------------------------------------------
?                                 Get help on the Console Commands
addDbroot                         Add DBRoot Disk storage to the Calpont InfiniDB System
addExternalDevice                 Add External Device to Configuration file
addModule                         Add a Module within the Calpont InfiniDB System
alterSystem-disableModule         Disable a Module and Alter the Calpont InfiniDB System
alterSystem-enableModule          Enable a Module and Alter the Calpont InfiniDB System
assignDbrootPmConfig              Assign unassigned DBroots to Performance Module
assignElasticIPAddress            Assign Amazon Elastic IP Address to a module
disableLog                        Disable the levels of process and debug logging
enableLog                         Enable the levels of process and debug logging
exit                              Exit from the Console tool
getActiveAlarms                   Get Active Alarm list
getActiveSQLStatements            Get List Active SQL Statements within the System
getAlarmConfig                    Get Alarm Configuration Information
getAlarmHistory                   Get system alarms
getAlarmSummary                   Get Summary counts of Active Alarm
getCalpontSoftwareInfo            Get the Calpont InfiniDB RPM detailed information
getExternalDeviceConfig           Get External Device Configuration Information
getLogConfig                      Get the System log file configuration
getModuleConfig                   Get Module Name Configuration Information
getModuleCpu                      Get a Module CPU usage
getModuleCpuUsers                 Get a Module Top Processes utilizing CPU
getModuleDisk                     Get a Module Disk usage
getModuleMemory                   Get a Module Memory usage
getModuleMemoryUsers              Get a Module Top Processes utilizing Memory
getModuleResourceUsage            Get a Module Resource usage
getModuleTypeConfig               Get Module Type Configuration Information
getProcessConfig                  Get Process Configuration Information
getProcessStatus                  Get Calpont InfiniDB Process Statuses
getStorageConfig                  Get System Storage Configuration Information
getStorageStatus                  Get System Storage Status
getSystemConfig                   Get System Configuration Information
getSystemCpu                      Get System CPU usage on all modules
getSystemCpuUsers                 Get System Top Processes utilizing CPU
getSystemDisk                     Get System Disk usage on all modules
getSystemInfo                     Get the Over-all System Statuses
getSystemMemory                   Get System Memory usage on all modules
getSystemMemoryUsers              Get System Top Processes utilizing Memory
getSystemNetworkConfig            Get System Network Configuration Information
getSystemResourceUsage            Get System Resource usage on all modules
getSystemStatus                   Get System and Modules Status
help                              Get help on the Console Commands
monitorAlarms                     Monitor alarms in realtime mode
movePmDbrootConfig                Move DBroots from one Performance Module to another
quit                              Exit from the Console tool
```

For a full list of InfiniDB commands, please see the InfiniDB Administrator's Guide.

4.1 SQL Interface

SQL commands to an InfiniDB database may be entered via the `idbmysql` utility. This command line tool mirrors the standard MySQL mysql command line utility in terms of use and functionality. All standard SQL statements (select, insert, update, delete, grant, etc.) may be issued through this utility.

4.2 Starting and Stopping an InfiniDB Instance

Starting and stopping all participating User and Performance Modules is done very easily with the `cmconsole` utility. The three core commands are:

- `startsystem`
- `stopsystem`
- `restartsystem`

An administrator can also utilize the basic operating system service start commands. For example, on an individual Linux machine, an administrator can issue a `service infinidb start` command to start an InfiniDB instance. A `service infinidb stop` will stop an InfiniDB instance.

4.3 Security Concepts

This section briefly discusses InfiniDB security concepts and practices.

4.3.1 Operating System Security

InfiniDB utilizes standard operating system security with respect to securing the executables and underlying database files. Nothing special or out of the ordinary is employed at this level. An administrator installing and managing an InfiniDB database should observe standard security procedures as to what operating system accounts have access to file systems/directories that contain InfiniDB binaries or database files.

4.3.2 Database Login Access

InfiniDB makes use of standard MySQL security commands to add and remove users from an InfiniDB database. User accounts are granted access rights to the overall system and any specific databases with which to work.

As an example, to grant a new user named ‘rms’ access to the system and full rights on a database named ‘db1’, an administrator would enter the following command:

```
mysql> grant all on db1.* to rms identified by ‘password’;
Query OK, 0 rows affected (0.00 sec)
```

To remove a user account from an InfiniDB database, an administrator makes use of the `drop user` command.

For a more complete understanding of adding and removing user accounts, please see the most current online version of the MySQL Reference Manual.

4.3.3 Object Security

As with database/system access security, InfiniDB uses standard MySQL security commands. This includes standard SQL `grant` and `revoke` commands to grant and remove object level privileges to selected user accounts. For example, to grant a user named 'rms' all rights on a table named 't1' in the 'db1' database, an administrator would issue this command:

```
mysql> grant all on db1.t1 to rms;
Query OK, 0 rows affected (0.00 sec)
```

For a full list of `grant` and `revoke` capabilities, please see the most current online version of the MySQL Reference Manual.

4.4 The InfiniDB Configuration File

The InfiniDB configuration file contains parameters that control the operation of the InfiniDB database. It is an XML-based file that is automatically replicated on all participating nodes in an InfiniDB system during system startup and shutdown (or explicit commands issued to replicate the file). The master copy of the configuration file is kept on the primary Performance Module. Upon startup, the system reads the information from the configuration file and allocates the necessary resources (e.g. data caches, etc.) needed for system activity.

It is not recommended that the configuration file be directly edited via a text editor. Instead, a command line tool – `configxml` – is supplied to make changes to the file. Various parameters are used to the script to view and set configuration information.

For example, if a DBA wanted to see the value for the period of time (set in minutes) that archive logs of the active Transaction Log are written to disk, they would enter the following:

```
[root@server1 bin]$ ./configxml.sh getconfig SystemConfig
TransactionArchivePeriod
Current value of SystemConfig / TransactionArchivePeriod is 10
```

To change the archive log time period to 60 minutes, they would use the `setconfig` option and enter:

```
[root@server1 bin]$ ./configxml.sh setconfig SystemConfig
TransactionArchivePeriod 60
Old value of SystemConfig / TransactionArchivePeriod is 10

/usr/local/Calpont/etc/Calpont.xml backed up to /usr/local/
Calpont/etc/Calpont.xml.1255027253

Old value of SystemConfig / TransactionArchivePeriod is 10
SystemConfig / TransactionArchivePeriod now set to 60
```

For a complete list of the configuration variables used by InfiniDB, please see the InfiniDB Administration Guide.

4.5 Creating New Databases

InfiniDB has a similar paradigm as MySQL and Microsoft SQL Server with respect to an instance of the server that is running and databases that are managed under that instance. There is a one to many relationship with an instance and databases, which differs from Oracle where an instance and a database are normally one to one.

Creating a new database (or schema as it is sometimes referred to in MySQL) is very easy in InfiniDB:

```
mysql> create database db1;
Query OK, 1 row affected (0.00 sec)
```

Once created, a database may then be targeted for the creation of tables, as in this example:

```
mysql> use db1
Database changed
mysql> create table t1 (c1 int, c2 date) engine=infinidb;
Query OK, 0 rows affected (0.03 sec)

mysql> desc t1;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c1    | int(11) | YES  |     | NULL    |       |
| c2    | date    | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

4.6 Administration Utilities

There are a number of different, command-line administration utilities that are provided with InfiniDB. Some of the more commonly used include:

- **colxml**: creates a job file that is used by the bulk load utility, cpimport
- **configxml.sh**: used to read and update configuration file parameters
- **cpimport**: used to bulk load data into a database
- **dumpcol:** outputs column data
- **edititem**: used to report and edit the Extent Map
- **edittxn**: outputs the system change number (SCN) and can help verify consistency across a multi-node configuration
- **file2oid.sh**: outputs the object ID for a column given one of the column's datafiles
- **oid2file**: provides directory information for an object's first file
- **replaytransactionlog**: creates a file containing all the transaction information from archived and current transaction logs

4.7 System Logs

InfiniDB writes system information to a number of logs including:

- An information log (`info.log`) that contains general information about the server's operation
- An error log (`err.log`) that contains data about errors that have occurred on the server
- A warning log (`warning.log`) that is used for notifications regarding conditions on the server that may signal future system errors (e.g. running out of server disk space, etc.)

InfiniDB stores these files in a general logging directory (e.g. on Linux, the location is `/var/log/Calpont`).

4.8 Backup and Recovery

This section describes InfiniDB backup and recovery operations.

4.8.1 Overview of Backup Procedures

To make a full backup of an InfiniDB system, the basic steps are as follows:

- Temporarily suspend write activity on the system (done by using the `cmconsole` utility)
- Backup the MySQL instance
- Backup the InfiniDB database files
- Restore write activity on the system (done by using the `cmconsole` utility)

4.8.2 Suspending Database Writes

To suspend database writes on an InfiniDB system, the following command would be issued inside of `cmconsole`:

```
calpont> suspendDatabaseWrites
suspenddatabasewrites   Wed Sep 30 08:58:06 2009

This command suspends the DDL/DML writes to the InfiniDB Database
           Do you want to proceed: (y or n) [n]: y

Suspend InfiniDB Database Writes Request successfully completed
```

4.8.3 Backing up the MySQL Instance

The MySQL database may be backed up using a couple of methods.

Method #1:
By using MySQL-supplied utilities such as `mysqldump` or `mysqlhotcopy`. Note that any backup of the MySQL instance should run very quickly as InfiniDB does not store any database data within MySQL.

For a full treatment of MySQL backup and recovery, please see the most current online version of the MySQL Reference Manual.

Method #2:
InfiniDB does not implement anything unique into the MySQL front-end and the following directory may be backed up in lieu of previously established procedures:

```
/usr/local/Calpont/mysql/db
```

Example:

```
cp -rp /usr/local/Calpont/mysql/db /mnt/InfiniDB/backup/frontend
```

Note: The -rp options are for copying directories recursively and saving ownership information.

If selected databases are to be backed up only, then the database directories within the above directory may be backed up instead.

4.8.4 Backing Up InfiniDB Data Files

Backing up InfiniDB data files involves simply copying them to a backup location or using system or vendor-supplied backup or snapshot utilities to take a snapshot of the directories where InfiniDB data is located. The files from these locations should be backed up:

- All data and DBRM files. As an example, on a standard Linux install, all files in the `/usr/local/Calpont/data`*N* (where *N* represents unique directories such as data1,data2, and so forth) for each PM.

NOTE: when InfiniDB is configured to run over HDFS, the /usr/local/Calpont/data*N* directories exist in HDFS and not the Linux host filesystem.

4.8.5 Resuming Database Writes

To resume database write activity after a backup has been completed, the following command would be issued inside of `cmconsole`:

```
calpont> resumeDatabaseWrites
resumedatabasewrites   Wed Sep 30 08:58:23 2009

This command resumes the DDL/DML writes to the InfiniDB Database
           Do you want to proceed: (y or n) [n]: y

Resume InfiniDB Database Writes Request successfully completed
```

4.9 Overview of Recovery Procedures

To do a full restore of an InfiniDB database, the following steps are performed:

- Restore the MySQL instance
- Restore the InfiniDB database

4.10 Restoring the MySQL Instance

The MySQL database may be restored in a number of different ways, with the most common being taking a backup file created by mysqldump and running the file through the mysql command line utility, which executes all of the SQL statements contained within the file.

For a full treatment of MySQL backup and recovery, please see the most current online version of the MySQL Reference Manual.

4.10.1 Restoring the InfiniDB Data Files

Restoring InfiniDB data files involves simply copying them from their backup location or using system or vendor-supplied snapshot utilities to restore a snapshot of the directories where InfiniDB data was located. The files restored include:

- All data and DBRM files. As an example, on a standard Linux install, all files would be returned to `/usr/local/Calpont/data`*N* (where *N* represents unique directories such as data1,data2, and so forth) for each PM.

4.11 GUI Tool Support

InfiniDB currently does not supply GUI tools to work with the InfiniDB database. For general data modeling, query development, and general MySQL security and object management, a developer or DBA can download the free GUI tools from MySQL (MySQL Workbench and the older GUI tools bundle).

In addition, a number of third-party software vendors make tools that enable a developer or DBA to query or manage InfiniDB database objects. Vendors such as SQLYog, Quest Software (TOAD for MySQL), Mentat Technologies (DreamCoder for MySQL), and many others supply both free versions of the GUI tools and pay-for editions that include advanced functionality.

5 Object Management

This section covers information to the creation and general management of InfiniDB database objects.

5.1 Tables

Being a relational database, InfiniDB stores data in logical table structures, with the primary difference between InfiniDB and most other RDBMS's is that InfiniDB utilizes a column-oriented structure, which is more suitable for analytic or data warehousing environments than legacy row-based implementations. However, in regards to creating database tables, there is no surface-level difference between InfiniDB and other relational databases. Standard SQL/DDL/DML statements are used to create, populate, manipulate, and view data inside of tables.

For example, to create a new table inside a database called 'db1' with three columns, a user might enter something like the following:

```
mysql> use db1
Database changed
mysql> create table t1 (c1 int, c2 date, c3 varchar(20)) engine=infinidb;
Query OK, 0 rows affected (3.03 sec)
```

Note that because InfiniDB pre-allocates contiguous space for each table, a create table command – even though the table is created empty – will take a small amount of time longer to execute than other database systems that do not pre-allocate space for their tables.

Once created, a user can make use of InfiniDB's bulk load utility to rapidly load large volumes of data into the table, use MySQL utilities to load data into the table, or use single DML statements to insert, update, and delete data. Note there is nothing the user needs to do with respect to space allocation for the table. For a more thorough discussion of how InfiniDB allocates space, please see the previous section that discusses InfiniDB storage concepts.

In the current implementation of InfiniDB, there is no support for primary or foreign keys or column-level constraints. Such support will be added in a future release of the server.

For a more detailed treatment of the `CREATE TABLE` command, please see either the InfiniDB SQL syntax guide or the MySQL Reference Manual.

5.1.1 Modifying and Dropping Tables

InfiniDB supports the typical `ALTER TABLE` command for adding and dropping columns in a table. Table modifications tend to run quicker in InfiniDB than standard MySQL (or other databases) because the operation does not need to perform any index maintenance as InfiniDB does not use indexing.

InfiniDB also supplies a special method for adding new columns to a table in an online fashion. During the operation, read operations can continue with the table uninterrupted. The `calonlinealter` function call is used to add a new column table a table in an online manner, and then the MySQL data dictionary is synchronized once the operation is complete. For example, to add a new integer column (c7) to a table (t1) online, a user would issue the following commands:

```
select calonlinealter('alter table t1 add column c7 int;');
alter table t1 add column c7 int comment 'schema sync only';
```

Dropping tables are accomplished via the simple `DROP TABLE` command. When InfiniDB drops a table, it also removes all underlying physical data files from the server so there is no manual cleanup an administrator has to do afterwards.

For a more detailed treatment of the `ALTER TABLE` and `DROP TABLE` commands, please see either the InfiniDB SQL syntax guide or the MySQL Reference Manual.

5.2 Stored Procedures and SQL Functions

InfiniDB does support the use of MySQL stored procedures and SQL functions. Stored procedures are procedural code objects stored within the server itself and are oftentimes used to do programmatic processing of data and to provide an extra layer of security over sensitive data. The DBA can restrict what and how columns in tables are accessed by granting execute privileges on a stored procedure but not the underlying tables themselves.

For example, if a DBA wished to create a stored procedure that retrieved two columns from a table named 'customer' when a user supplied a customer number, they might write a procedure as follows:

```
delimiter //
create procedure p_cust (p1 int)
begin
            select c_name,c_address from customer
            where c_custkey = p1;
end
//
delimiter ;
```

Calling the stored procedure is accomplished as follows:

```
mysql> call p_cust(1);
+----------------------------+---------------+
| c_name                     | c_address     |
+----------------------------+---------------+
| Customer#000000001         | IVhzIApeRb    |
+----------------------------+---------------+
1 row in set (0.26 sec)
```

Note that a current restriction in using stored procedures and InfiniDB is that only one open cursor at a time may be used.

SQL Functions may also be created and used – a simple function example might be:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
    -> RETURNS CHAR(50) DETERMINISTIC
    -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select hello('world');
+-----------------+
| hello('world') |
+-----------------+
| Hello, world! |
+----------------+
1 row in set (0.03 sec)
```

For more information on stored procedures and functions, please see the most current version of the MySQL Reference Manual.

5.3 *Unsupported Database Objects*

The current version of InfiniDB does not support the following objects:

- Triggers
- Primary keys
- Foreign keys
- Column constraints other than NOT NULL and default values

5.4 *InfiniDB System Catalog*

When InfiniDB is installed, a small system catalog is configured that is used to track metadata as it relates to InfiniDB objects. This system catalog is used in addition to the standard objects found in the MySQL `mysql` and `information_schema` databases. The database/schema name is `calpontsys`.

The objects contained within the InfiniDB system catalog database include the following:

```
mysql> desc systable;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| tablename     | varchar(128) | YES  |     | NULL    |       |
| schema        | varchar(128) | YES  |     | NULL    |       |
| objectid      | int(11)      | YES  |     | NULL    |       |
| createdate    | date         | YES  |     | NULL    |       |
| lastupdate    | date         | YES  |     | NULL    |       |
| init          | int(11)      | YES  |     | NULL    |       |
| next          | int(11)      | YES  |     | NULL    |       |
| numofrows     | int(11)      | YES  |     | NULL    |       |
| avgrowlen     | int(11)      | YES  |     | NULL    |       |
| numofblocks   | int(11)      | YES  |     | NULL    |       |
| autoincrement | int(11)      | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
11 rows in set (0.00 sec)
```

```
mysql> desc syscolumn;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| schema          | varchar(128) | YES  |     | NULL    |       |
| tablename       | varchar(128) | YES  |     | NULL    |       |
| columnname      | varchar(128) | YES  |     | NULL    |       |
| objectid        | int(11)      | YES  |     | NULL    |       |
| dictobjectid    | int(11)      | YES  |     | NULL    |       |
| listobjectid     | int(11)      | YES  |     | NULL    |       |
| treeobjectid     | int(11)      | YES  |     | NULL    |       |
| datatype         | int(11)      | YES  |     | NULL    |       |
| columnlength     | int(11)      | YES  |     | NULL    |       |
| columnposition   | int(11)      | YES  |     | NULL    |       |
| lastupdate       | date         | YES  |     | NULL    |       |
| defaultvalue     | varchar(64)  | YES  |     | NULL    |       |
| nullable         | int(11)      | YES  |     | NULL    |       |
| scale            | int(11)      | YES  |     | NULL    |       |
| prec             | int(11)      | YES  |     | NULL    |       |
| autoincrement    | char(1)      | YES  |     | NULL    |       |
| distcount        | int(11)      | YES  |     | NULL    |       |
| nullcount        | int(11)      | YES  |     | NULL    |       |
| minvalue         | varchar(64)  | YES  |     | NULL    |       |
| maxvalue         | varchar(64)  | YES  |     | NULL    |       |
| compressiontype  | int(11)      | YES  |     | NULL    |       |
| nextvalue        | bigint(20)   | YES  |     | NULL    |       |
+------------------+--------------+------+-----+---------+-------+
22 rows in set (0.00 sec)
```

6 Loading and Manipulating Data

This section covers the loading and manipulation of data within an InfiniDB database.

6.1 *InfiniDB Bulk Loader*

InfiniDB supplies a high-speed bulk load utility that imports data into InfiniDB tables in a fast and efficient manner. The utility accepts as input any flat file containing data that contains a delimiter between fields of data (i.e. columns in a table). The default delimiter is the pipe ('|') character, but other delimiters such as commas may be used as well.

The InfiniDB bulk load utility – called `cpimport` – performs the following operations when importing data into an InfiniDB database:

- Data is read from specified flat files
- Data is transformed to fit InfiniDB's column-oriented storage design
- Redundant data is tokenized and logically compressed
- Data is written to disk

The 2 most-common ways to use cpimport are: 1) from the UM: cpimport will distribute rows to all Performance Modules; and 2) from a PM: cpimport will load the imported rows only on the PM from which is was invoked. Cpimport has a number of other, less popular ways to load data; see the InfiniDB Administrator's Guide for a detailed explanation of cpimport usage.

There are two primary steps to using the `cpimport` utility:

1. Optionally create a job file that is used to load data from a flat file into multiple tables
2. Run the cpimport utility to perform the data import

Note that bulk loads are an append operation to a table so they allow existing data to be read and remain unaffected during the process. Also, bulk loads do not write their data operations to the transaction log; they are not transactional in nature but are considered an atomic operation at this time. Information markers, however, are placed in the transaction log so the DBA is aware that a bulk operation did occur.

Upon completion of the load operation, a high water mark in each column file is moved in an atomic operation that allows for any subsequent queries to read the newly loaded data. This append operation provides for consistent read but does not incur the overhead of logging the data.

6.1.1 Simple Bulk Load Example

Suppose that a DBA wants to load a table named 't1' in a database named 'db1' and use a load file called 't1.tbl'. The file has the following data and uses the pipe delimiter to delimit the data:

```
1|Smith|2000-01-01
2|Jones|2000-02-01
3|Fields|2000-03-01
4|Johnson|2000-04-01
5|Simms|2000-05-01
```

In the current version of InfiniDB, the file location will default to the current directory unless the -f pathname option is used. Also the utility currently defaults the naming format of <table name>.tbl.

The table structure is the following:

```
mysql> desc t1;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| c1    | int(11)     | YES  |     | NULL    |       |
| c2    | varchar(20) | YES  |     | NULL    |       |
| c3    | date        | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.03 sec)
```

If loading only one table, the cpimport utility may be called directly. The DBA executes the `cpimport` utility (found in the InfiniDB bin directory) to actually load the data into the table:

```
[root@server1 test]# pwd
/home/test

[root@server1 bin]# /usr/local/Calpont/bin/cpimport db1 t1 t1.dbl

Bulkload root directory : /home/test
job description file : Job_501.xml
2009-10-06 15:08:16 (17995) INFO : successfully load job file /usr/
local/Calpont/data/bulk/job/Job_501.xml
2009-10-06 15:08:16 (17995) INFO : PreProcessing check starts
2009-10-06 15:08:16 (17995) INFO : PreProcessing check completed
2009-10-06 15:08:16 (17995) INFO : preProcess completed, total run time : 0
seconds
2009-10-06 15:08:16 (17995) INFO : No of Read Threads Spawned = 1
2009-10-06 15:08:16 (17995) INFO : No of Parse Threads Spawned = 3
2009-10-06 15:08:17 (17995) INFO : For table db1.t1: 5 rows processed and 5
rows inserted.
2009-10-06 15:08:17 (17995) INFO : Bulk load completed, total run time : 1
seconds
```

If loading multiple tables, see the Multi-Table Bulk Load Example below.

6.1.1.1 STDIN Bulk Load Example

InfiniDB supports bulk loading data from standard in with the simple usage of cpimport. Using the same load file defined above, a file can be piped into the cpimport process. For example, the t1.tbl file can be restricted to only import the first 100,000,000 rows, which is then piped into cpimport.

```
head -100000000 t1.tbl | /usr/local/Calpont/bin/cpimport db1 t1   -s '\t'
```

6.1.1.2 Bulk Insert with Select From Table(s)

Standard in can also be used to directly pipe the output from an arbitrary select statement into our high speed bulk loader. Here the db2.source_table is selected from, using the -N flag to remove non-data formatting.

```
idbmysql -e 'select * from source_table;' -N db2 | /usr/local/
Calpont/bin/cpimport db1 t1 -s '\t'
```

6.1.2 Multi-Table Bulk Load

Multiple tables may be simultaneously imported using the import utility. With the simple bulk load, multiple jobs may be simultaneously submitted as long as the tables are unique per import. With the traditional bulk load, multiple tables may be imported by either importing all tables within a schema or listing specific tables using the -t option in colxml. Multiple jobs may be submitted with the traditional interface as long as the tables and jobids are unique.

6.1.3 Traditional Bulk Load Example

A DBA can also use the traditional bulk load to load an entire schema or multiple tables in one bulk load job. For example, if a DBA wants to load data into the following tables that comprise a database name 'tpch2':

```
mysql> show tables;
+----------------+
| Tables_in_rms  |
+----------------+
| customer       |
| lineitem       |
| nation         |
| orders         |
| part           |
| partsupp       |
| region         |
| supplier       |
+----------------+
8 rows in set (0.00 sec)
```

They can use the default pipe ('|') field delimiter in their load files, place the load files for all the tables in the import directory, and then execute the `colxml` utility for the load job for the 'tpch2' database as such:

```
[user1@server1 bin]$ ./colxml tpch2 -j500
Running colxml with the following parameters:
2009-10-07 15:14:20 (9481) INFO :
        Schema: tpch2
        Tables:
```

```
        Load Files:
        -b      0
        -c      1048576
        -d      |
        -e      10
        -f      CSV
        -j      500
        -n
        -p      /usr/local/Calpont/data/bulk/job/
        -r      5
        -s
        -u
        -w      10485760
        -x      tbl

File completed for tables:
        tpch2.customer
        tpch2.lineitem
        tpch2.nation
        tpch2.orders
        tpch2.part
        tpch2.partsupp
        tpch2.region
        tpch2.supplier

 Normal exit.
```

Then the actual bulk load operation can be run:

```
[user1@server1 bin]$ ./cpimport -j 500

Bulkload root directory : /usr/local/Calpont/data/bulk
job description file : Job_500.xml
2009-10-07 15:14:59 (9952) INFO : successfully load job file /usr/
local/Calpont   data/bulk/job/Job_500.xml
2009-10-07 15:14:59 (9952) INFO : PreProcessing check starts
2009-10-07 15:15:04 (9952) INFO : PreProcessing check completed
2009-10-07 15:15:04 (9952) INFO : preProcess completed, total run time : 5
seconds
2009-10-07 15:15:04 (9952) INFO : No of Read Threads Spawned = 1
2009-10-07 15:15:04 (9952) INFO : No of Parse Threads Spawned = 3
2009-10-07 15:15:06 (9952) INFO : For table tpch2.customer: 150000 rows
processed and 150000 rows inserted.
2009-10-07 15:16:12 (9952) INFO : For table tpch2.nation: 25 rows processed
and 25 rows inserted.
2009-10-07 15:16:12 (9952) INFO : For table tpch2.lineitem: 6001215 rows
processed and 6001215 rows inserted.
2009-10-07 15:16:31 (9952) INFO : For table tpch2.orders: 1500000 rows
processed and 1500000 rows inserted.
2009-10-07 15:16:33 (9952) INFO : For table tpch2.part: 200000 rows
processed and 200000 rows inserted.
2009-10-07 15:16:44 (9952) INFO : For table tpch2.partsupp: 800000 rows
processed and 800000 rows inserted.
2009-10-07 15:16:44 (9952) INFO : For table tpch2.region: 5 rows processed
and 5 rows inserted.
2009-10-07 15:16:45 (9952) INFO : For table tpch2.supplier: 10000 rows
processed and 10000 rows inserted.
```

```
        2009-10-07 15:16:45 (9952) INFO : Bulk load completed, total run time : 106
        seconds
```

6.1.4 STDIN Bulk Load Example

InfiniDB supports bulk loading data from standard in with the –f STDIN flag on the multi-table cpimport. Using the same load file defined above with the -j501 command, a file can be piped into the cpimport process. For example, the t1.tbl file can be restricted to only import the first 100,000,000 rows, which is then piped into cpimport.

```
head -100000000 t1.tbl | /usr/local/Calpont/bin/cpimport -j501 -s '\t' -fSTDIN
```

6.1.5 Bulk Insert with Select From Table(s)

Standard in can also be used to directly pipe the output from an arbitrary select statement into our high speed bulk loader. Here the db2.source_table is selected from, using the -N flag to remove non-data formatting.

```
idbmysql -e 'select * from source_table;' -N db2 | /usr/local/
Calpont/bin/cpimport -j501 -s '\t' -fSTDIN
```

6.1.6 Bulk Insert with Binary Source

A binary file with fixed length records may also be used as input to cpimport. This may be accomplished using the -I1 (NULLs accepted) or -I2 (NULLs saturated) options. The following is an example simple import method load of a binary file to load data that includes NULLs:

```
[root@server1 bin]# /usr/local/Calpont/bin/cpimport -I1 mydb mytable mytablesource.bin
```

6.2 MySQL Utilities for Loading Data

MySQL supplies the `load data infile` command to load data into database tables. This command/utility is supported by InfiniDB and may be used, although it is not as fast as using the InfiniDB bulk load utility, `cpimport`.

As an example, for the following table:

```
        mysql> desc t3;
        +-------+---------+------+-----+---------+-------+
        | Field | Type    | Null | Key | Default | Extra |
        +-------+---------+------+-----+---------+-------+
        | c1    | int(11) | YES  |     | NULL    |       |
        | c2    | int(11) | YES  |     | NULL    |       |
        +-------+---------+------+-----+---------+-------+
```

The following file:

```
        [root@server1 local]# more t3.tst
        1,2
        2,3
        3,4
```

May be loaded using the following:

```
mysql> load data infile '/usr/local/t3.tst' into table t3 fields terminated
by ',';
Query OK, 3 rows affected (0.96 sec)
```

6.3 DML Support

Standard DML support is provided with InfiniDB so users may issue SQL `insert`, `update`, and `delete` statements against InfiniDB tables. Note that because InfiniDB is a column-oriented database, insert and delete statements will not normally run as fast on InfiniDB as they will on row-oriented databases (more blocks must be touched). Update statements, however, should run fairly quickly.

Note that for insert statements, InfiniDB does support the MySQL multi-insert operation. For example, for the following table:

```
mysql> desc t3;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c1    | int(11) | YES  |     | NULL    |       |
| c2    | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
```

A user can issue the following `insert` statement:

```
mysql> insert into t3 values (1,2),(3,4),(5,6);
Query OK, 3 rows affected (0.34 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

For more information on DML syntax, see the InfiniDB SQL Syntax Guide.

6.4 Transaction Management

InfiniDB supports ACID-compliant transactions, with complete commit, rollback, and deadlock detection capability. The default mode for InfiniDB is auto-commit, where every DML statement is automatically made permanent. This setting can be easily changed per session in InfiniDB. For example, the following example sets auto-commit mode off, updates a table, rolls back the change, updates the table again, and commits the transaction to the database:

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select c1 from t3;
+------+
| c1   |
+------+
|    1 |
|    2 |
|    3 |
|    1 |
|     1 |
|     3 |
|     5 |
+-------+
7 rows in set (0.26 sec)

mysql> update t3 set c1=4;
Query OK, 7 rows affected (0.26 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> rollback;
Query OK, 0 rows affected (0.23 sec)

mysql> select c1 from t3;
+-------+
| c1    |
+-------+
|     1 |
|     2 |
|     3 |
|     1 |
|     1 |
|     3 |
|     5 |
+-------+
7 rows in set (0.26 sec)

mysql> update t3 set c1=5;
Query OK, 7 rows affected (0.14 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.21 sec)

mysql> select c1 from t3;
+-------+
| c1    |
+-------+
|     5 |
|     5 |
|     5 |
|     5 |
|     5 |
|     5 |
|     5 |
+-------+
7 rows in set (0.14 sec)
```

6.5 Concurrency Control

InfiniDB uses page level locking and employs the Read Committed lock isolation level to ensure that only committed data is read by a user.

InfiniDB also uses Multiversion Concurrency Control (MVCC), which supplies InfiniDB with its versioning infrastructure. This is sometimes referred to in other databases as 'snapshot read', which means a query will always see the data as it existed the moment the query was issued. This allows a consistent read across all columns from all tables involved in a query.

The most important benefit of InfiniDB's versioning is that *reads are never blocked*. That is, one session can read (an older) versioned block while another session is simultaneously updating that block. All statements in InfiniDB run at a particular version (or snapshot) of the database, which InfiniDB refers to as the System Change Number (SCN). A circular version buffer and the Version Substitution Structure (VSS) and Version Buffer Block Manager (VBBM) structures are used to locate a "suitable" version of the block requested. The sizes of these objects are configurable to allow avoiding buffer overrun during anticipated workloads.

Note that versioning only applies to transactional DML statements and does not apply to bulk load which is accomplished via the append operation.

7 Application and Business Intelligence Development

This section briefly discusses working with InfiniDB in application development and business intelligence environments.

7.1 Connecting to InfiniDB with Application and BI Tools

Because InfiniDB uses a MySQL front end, any standard MySQL connector may be used to connect to InfiniDB. Freely available connectors include the following:

- ADO.NET
- ODBC
- JDBC
- C++
- C
- PHP
- Perl
- Python
- Ruby

MySQL connectors may be freely downloaded and used from the MySQL web site, with the current URL location being: http://www.mysql.com/products/connector/.

7.2 Application Development and Business Intelligence Tools

Again, because of InfiniDB's MySQL front end, any application development or business intelligence tool that currently supports MySQL should work with InfiniDB. Application development tools such as Eclipse, Microsoft Visual Studio, and many more all support MySQL and therefore should work well with InfiniDB.

Business intelligence tools that currently connect to MySQL include all the major proprietary vendors such as Business Objects, Cognos, Microsoft Analysis Servicesand many more. Open Source BI tools should also work with InfiniDB; these include Pentaho, Jaspersoft, Talend, and others.

8 Performance Tuning

This section contains information regarding the monitoring and tuning of an InfiniDB database.

8.1 Monitoring and Diagnostic Utilities

The `cmconsole` utility allows a DBA or sysadmin to perform a number of monitoring operations across a single or multi-node InfiniDB configuration. The `cmconsole` utility allows a DBA to monitor both the database and the servers on which the User and Performance Modules operate.

Below is a listing of the various monitoring and diagnostic commands, many of which are self-explanatory from their name:

```
Command                              Description
------------------------------       ---------------------------------------
?                                    Get help on the Console Commands
addDbroot                            Add DBRoot Disk storage to the Calpont InfiniDB System
addExternalDevice                    Add External Device to Configuration file
addModule                            Add a Module within the Calpont InfiniDB System
alterSystem-disableModule            Disable a Module and Alter the Calpont InfiniDB System
alterSystem-enableModule             Enable a Module and Alter the Calpont InfiniDB System
assignDbrootPmConfig                 Assign unassigned DBroots to Performance Module
assignElasticIPAddress               Assign Amazon Elastic IP Address to a module
disableLog                           Disable the levels of process and debug logging
enableLog                            Enable the levels of process and debug logging
exit                                 Exit from the Console tool
getActiveAlarms                      Get Active Alarm list
getActiveSQLStatements               Get List Active SQL Statements within the System
getAlarmConfig                       Get Alarm Configuration Information
getAlarmHistory                      Get system alarms
getAlarmSummary                      Get Summary counts of Active Alarm
getCalpontSoftwareInfo               Get the Calpont InfiniDB RPM detailed information
getExternalDeviceConfig              Get External Device Configuration Information
getLogConfig                         Get the System log file configuration
getModuleConfig                      Get Module Name Configuration Info
getModuleCpu                         Get a Module CPU usage
getModuleCpuUsers                    Get a Module Top Processes utilizing CPU
getModuleDisk                        Get a Module Disk usage
getModuleMemory                      Get a Module Memory usage
getModuleMemoryUsers                 Get a Module Top Processes utilizing Memory
getModuleResourceUsage               Get a Module Resource usage
getModuleTypeConfig                  Get Module Type Configuration Info
getProcessConfig                     Get Process Configuration Information
getProcessStatus                     Get Calpont InfiniDB Process Statuses
getStorageConfig                     Get System Storage Configuration Info
getStorageStatus                     Get System Storage Status
getSystemConfig                      Get System Configuration Information
getSystemCpu                         Get System CPU usage on all modules
getSystemCpuUsers                    Get System Top Processes utilizing CPU
getSystemDisk                        Get System Disk usage on all modules
getSystemInfo                        Get the Over-all System Statuses
getSystemMemory                      Get System Memory usage on all modules
getSystemMemoryUsers                 Get System Top Processes utilizing Memory
getSystemNetworkConfig               Get System Network Configuration Info
getSystemResourceUsage               Get System Resource usage on all modules
getSystemStatus                      Get System and Modules Status
help                                 Get help on the Console Commands
monitorAlarms                        Monitor alarms in realtime mode
movePmDbrootConfig                   Move DBroots from one Performance Module to another
quit                                 Exit from the Console tool
```

```
removeDbroot                      Remove DBRoot Disk storage from the Calpont InfiniDB System
removeExternalDevice              Remove External Device to Config file
removeModule                      Remove a Module from the Calpont InfiniDB System
resetAlarm                        Resets an Active Alarm
restartSystem                     Restarts a stopped or shutdown Calpont InfiniDB System
resumeDatabaseWrites              Resume performing writes to the Calpont InfiniDB Database
setAlarmConfig                    Set a Alarm Configuration parameter
setExternalDeviceConfig           Set a External Device Configuration parameter
setModuleTypeConfig               Set a Module Type Configuration parameter
setProcessConfig                  Set a Process Configuration parameter
setSystemConfig                   Set a System Configuration parameter
shutdownSystem                    Shuts down the Calpont InfiniDB System
startSystem                       Starts a stopped or shutdown Calpont InfiniDB System
stopSystem                        Stops the processing of the Calpont InfiniDB System
suspendDatabaseWrites             Suspend performing writes to the Calpont InfiniDB Database
switchParentOAMModule             Switches the Active Parent OAM Module to another Performance
                                   Module
system                            Execute a system shell command
unassignElasticIPAddress          Unassign Amazon Elastic IP Address

For help on a command, enter 'help' followed by command name
```

Many system diagnostics can be monitored and analyzed using `cmconsole`; query activity, overall system status, disk utilization, CPU and memory resource consumption, and much more can be queried and reported on. For example, a DBA or sysadmin can get a quick status on an overall configuration by doing the following:

```
calpont> getSystemStatus
getsystemstatus   Wed Oct 26 12:34:28 2012

System qperfd01

System and Module statuses

Component     Status                Last Status Change
------------  --------------------  ------------------------
System        ACTIVE                Wed Oct  7 15:10:47 2009
Module um1    ACTIVE                Tue Oct  6 15:50:09 2009
Module pm1    ACTIVE                Tue Oct  6 15:50:19 2009
Module pm2    ACTIVE                Wed Oct  7 15:10:45 2009
Module pm3    MAN_DISABLED          Tue Oct  6 13:25:09 2009
Module pm4    MAN_DISABLED          Tue Oct  6 13:25:39 2009
Module pm5    MAN_DISABLED          Tue Oct  6 11:19:38 2009
Module pm6    MAN_DISABLED          Tue Oct  6 11:20:10 2009
Module pm7    MAN_DISABLED          Tue Oct  6 11:20:44 2009
Module pm8    MAN_DISABLED          Tue Oct  6 11:21:18 2009

Active Parent OAM Performance Module is 'pm1'
```

The output above shows that one User Module and two Performance Modules are currently online and available to process queries while Performance Modules 3-8 are currently offline. If a DBA or sysadmin wants to see the CPU, disk, and memory consumption on one of the nodes in a configuration, they can enter commands such as the following:

```
calpont> getModuleCpu pm1
getmodulecpu   Wed Oct 26 12:35:36 2012
```

```
Module 'pm1' CPU Usage % = 47

calpont> getModuleDisk pm1
getmoduledisk   Wed Oct 26 12:36:02 2012
Module 'pm1' Disk Usage (in 1K blocks)

Mount Point                Total Blocks  Used Blocks  Usage %
-------------------------  ------------  ------------ -------
/                          66719320      57662764        87
/boot                      101086        19987           20
/usr/local/Calpont/data1   1124868224    830641000       74
/usr/local/Calpont/data2   1124868224    835241732       75
/usr/local/Calpont/data3   1124868224    834809896       75
/usr/local/Calpont/data4   1124868224    833940516       75
/usr/local/Calpont/data5   1124868224    835826024       75
/usr/local/Calpont/data6   1124868224    833620112       75
/usr/local/Calpont/data7   1124868224    832053424       74
/usr/local/Calpont/data8   1124868224    832324408       74
/home/qa/srv               235332176     88043896        38

calpont> getModuleMemory pm1
getmodulememory   Wed Oct 26 12:37:20 2012

Module Memory Usage (in K bytes)

Module  Mem Total  Mem Used cache   Mem Usage % Swap Total Swap Used Swap
                                                                      Usage%
------  ---------  -------  -------- ----------  ---------- --------- ----
pm1     16440436   15472352  136236      94       2031608     0         0
```

Being able to monitor local and remote machines with `cmconsole` helps save time and provides DBAs and sysadmins with one tool they can use to run diagnostics across both their database and physical servers.

8.2 Alarms and Notifications

InfiniDB also allows a DBA or sysadmin to configure a variety of system alarms that can be used to notify them of issues in advance of a database or system failure. Administrators can set thresholds on disk, CPU, memory, system swap space, incompatible software versions, and more.

Alarms and notifications are managed with the `cmconsole` utility. To see a listing of alarms that can be configured, an administrator can enter the following command:

```
calpont> getAlarmConfig
getalarmconfig   Wed Oct  7 15:45:50 2009

Alarm Configuration

Alarm ID #1 Configuration information
BriefDesc = CPU_USAGE_HIGH
DetailedDesc = The usage on the indicated CPU has exceeded its high threshold
Severity = CRITICAL
Threshold = 100

Alarm ID #2 Configuration information
```

```
BriefDesc = CPU_USAGE_MED
DetailedDesc = The usage on the indicated CPU has exceeded its medium
threshold
Severity = MAJOR
Threshold = 70

Alarm ID #3 Configuration information
BriefDesc = CPU_USAGE_LOW
DetailedDesc = The usage on the indicated CPU has exceeded its low threshold
Severity = MINOR
Threshold = 50

Alarm ID #4 Configuration information
BriefDesc = DISK_USAGE_HIGH
DetailedDesc = The usage on the indicated Disk Drive has exceeded its high
threshold
Severity = CRITICAL
Threshold = 100
.
.
.
```

An administrator can alter an alarm’s current setting by using the setAlarmConfig command and referencing the alarm number and its threshold. An example of setting the CPU_USAGE_HIGH alarm to notify on CPU that rises about 95%, the administrator would do the following:

```
calpont> setAlarmConfig 1 Threshold 95
setalarmconfig   Wed Oct  7 15:55:47 2009

   Successfully set Threshold = 95

Alarms that have been triggered can be viewed with the getActiveAlarms
command:

calpont> getActiveAlarms

AlarmID            = 1
Brief Description = CPU_USAGE_HIGH
Alarm Severity     = CRITICAL
Time Issued        = Tue Oct  6 14:50:06 2009
Reporting Module   = pm2
Reporting Process = ServerMonitor
Reported Device    = CPU

AlarmID            = 30
Brief Description = CONN_FAILURE
Alarm Severity     = CRITICAL
Time Issued        = Tue Oct  6 15:48:08 2009
Reporting Module   = um1
Reporting Process = ExeMgr
Reported Device    = 10.100.7.5 PrimProc
```

Alarm notifications can also be configured to send SNMP traps to global system monitoring software such as Nagios, HP OpenView or Tivoli.

9 SQL Monitoring and Diagnostic Utilities

Poorly structured SQL is oftentimes the number one cause of performance issues on a database. To help find and fix poorly running SQL statements, InfiniDB provides a number of commands and utilities.

First, from a general MySQL support standpoint, a user can see brief information regarding who is logged on and their currently running SQL using the `show processlist` command:

```
mysql> show processlist;
+----+------+-----------+-------+---------+------+-------+--------------+
| Id | User | Host      | db    | Command | Time | State | Info         |
+----+------+-----------+-------+---------+------+-------+--------------+
| 73 | root | localhost | ssb10 | Query   |    0 | NULL  | show processlist
+----+------+-----------+-------+---------+------+-------+--------------+
1 row in set (0.01 sec)
```

InfiniDB's `cmconsole` utility also provides a command – `getActiveSQLStatements` – that shows what SQL is currently being executed on the database:

```
calpont> getActiveSQLStatements
getactivesqlstatements   Wed Oct  7 08:38:32 2009

Get List of Active SQL Statements
=================================

Start Time         Time (hh:mm:ss)   Session ID             SQL Statement
----------------   ----------------  --------------------   ---------------------
---------------------------------------
Oct  7 08:38:30       00:00:03       73                     select c_name,
sum(lo_revenue) from customer, lineorder where lo_custkey = c_custkey and
c_custkey = 6 group by c_name
```

For troubleshooting individual SQL statement performance, there are three utilities that may be used. First is the standard MySQL EXPLAIN utility, which is somewhat less than helpful for InfiniDB as InfiniDB does not use indexes or make use of MySQL I/O functionality. Some information can be had in an EXPLAIN, however, as the following example shows:

```
mysql> explain select select count(*) from dateinfo,customer, part, supplier, lineorder
where s_suppkey = lo_suppkey and d_datekey = lo_or and p_partkey = lo_partkey
and c_custkey = lo_custkey and s_nation = 'BRAZIL';

+----+-------------+-----------+------+---------------+------+---------+------+-
-----+-------------------------------------------------------+
| id | select_type | table     | type | possible_keys | key  | key_len | ref  |
rows | Extra                                                 |
+----+-------------+-----------+------+---------------+------+---------+------+-
-----+-------------------------------------------------------+
|  1 | SIMPLE      | dateinfo  | ALL  | NULL          | NULL | NULL    | NULL |
2000 |                                                       |
|  1 | SIMPLE      | customer  | ALL  | NULL          | NULL | NULL    | NULL |
2000 | Using join buffer                                     |
|  1 | SIMPLE      | part      | ALL  | NULL          | NULL | NULL    | NULL |
2000 | Using join buffer                                     |
|  1 | SIMPLE       | supplier  | ALL  | NULL          | NULL | NULL    | NULL |
2000 | Using where with pushed condition; Using join buffer |
|  1 | SIMPLE       | lineorder | ALL  | NULL          | NULL | NULL    | NULL |
2000 | Using where; Using join buffer                       |
+----+-------------+-----------+------+---------------+------+--------+------+-
-----+-------------------------------------------------------+
```

Much better diagnostic information can be obtained for a SQL statement's performance by using InfiniDB's supplied SQL statistics command (`calgetstats`) and tracing utility (`caltrace`).

To see SQL performance statistics for a query just run through the SQL interface, a DBA or developer can just issue a call to `calgetstats`. For example:

```
mysql> select c_name, sum(l_quantity) from customer, orders, lineitem where
c_custkey = o_custkey and l_orderkey = o_orderkey and l_shipdate = '1992-01-02'
group by c_name;
+--------------------+-----------------+
| c_name             | sum(l_quantity) |
+--------------------+-----------------+
| Customer#000094199 |           35.00 |
| Customer#000009263 |           19.00 |
.
.
17 rows in set (1.21 sec)

mysql> select calgetstats()\G
*************************** 1. row ***************************
calgetstats(): Query Stats: MaxMemPct-0; NumTempFiles-0; TempFileSpace-0MB;
ApproxPhyI/O-0; CacheI/O-7977; BlocksTouched-7977; PartitionBlocksEliminated-0;
MsgBytesIn-5MB; MsgBytesOut-0MB; Mode-Distributed| 1255540221 457489
1 row in set (0.00 sec)
```

The statistical output contains information on:

- MaxMemPct - Peak memory utilization on the User Module, likely in support of a large based hash join operation.
- NumTempFiles - Report on any temporary files created in support of query operations larger than available memory, typically for unusual join operations where the smaller table join cardinality exceeds some configurable threshold.
- TempFileSpace - Report on space used by temporary files created in support of query operations larger than available memory, typically for unusual join operations where the smaller table join cardinality exceeds some configurable threshold.
- PhyI/O - Number of 8k blocks read from disk, SSD, or other persistent storage. In most cases, the quantity of individual I/O operations are far fewer than the number of blocks; InfiniDB will read as many as 512 blocks together in one I/O operation.
- CacheI/O - Approximate number of 8k blocks processed in memory, adjusted down by the number of discrete Physical I/O calls required.
- BlocksTouched - Approximate number of 8k blocks processed in memory.
- CalPartBlks - The number of block touches eliminated via the InfiniDB
- Extent Map elimination behavior.

- MsgBytesIn, MsgByteOut - Message bytes in MB sent between nodes in support of the query.

The basic steps to using InfiniDB's SQL tracing utility, `caltrace`, are:

1. Start the trace for the particular session
2. Execute the SQL statement in question
3. Review the trace collected for the statement

As an example, the following session starts a trace, issues a query against a 6 million row fact table and 300,000 row dimension table, and then reviews the output from the trace:

```
mysql> select calsettrace(1);
+-----------------+
| calsettrace(1) |
+-----------------+
|              1 |
+-----------------+
1 row in set (0.04 sec)

mysql> select c_name, sum(lo_revenue)
    -> from customer, lineorder
    -> where lo_custkey = c_custkey and
    -> c_custkey = 5
    -> group by c_name;
+--------------------+-----------------+
| c_name             | sum(lo_revenue) |
+--------------------+-----------------+
| Customer#000000005 |    552483078.00 |
+--------------------+-----------------+
1 row in set, 0 warning (1.23 sec)

mysql> select calgettrace();
+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
------------------------------------------------+
| calgettrace()
|
+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
------------------------------------------------+
|
Desc Mode Table     TableOID ReferencedOIDs PIO  LIO   PBE Elapsed Rows
BPS  PM   customer  3327      (3328,3336)    0    150   0   0.017   1
BPS  PM   lineorder 3278      (3281,3291)    4096 29808 0   0.766   63
HJS  PM   lineorder 3278      -              -    -     -   0.000   -
ADS  UM   -         -         -              -    -     -   0.724   -
 |
+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
-------------------------------------------------+
1 row in set (0.02 sec)
```

Information on physical and logical I/O for each step, elapsed time, and more are provided.

10 Migrating to InfiniDB

This section discusses how to migrate to the InfiniDB database from other databases.

10.1 General Migration Concepts

Migrating to InfiniDB generally involves moving schema/data from another database and possibly stored code objects such as stored procedures. In general, the following roadmap is followed:

1. Document the source database objects
2. Design InfiniDB objects
3. Move data from source system to InfiniDB
4. Transform code objects to InfiniDB (optional)

Documenting an existing database structure can be challenging if it is attempted by standard manual processes, which normally involves manual data dictionary lookups and the like. While most databases have very good metadata dictionaries to work with, the process of manually pulling all metadata (table, column etc.) can be a very time consuming event for large databases.

The best approach most times is to use a computerized reverse engineering process that automatically catalogs all applicable metadata that is needed for conversion. A good third party data modeling tool can be used for this, such as PowerDesigner, ERWin, and ER/Studio; all support the reverse engineering of multiple database types.

Moving schema structures over to InfiniDB isn't normally a challenging task as InfiniDB supports most of the important datatypes. For a complete description of datatype support in InfiniDB, see the InfiniDB SQL Syntax Guide.

Once the source database metadata has been obtained and understood, the next step is to design the InfiniDB target database. This involves translating the source objects and their properties (such as column datatypes) to InfiniDB complements. Some modeling tools allow a user to automatically transform one database schema to another simply by selecting an option to change the underlying database in a physical design. All datatypes are automatically altered, however, some many need manual tweaking if there isn't a one-to-one datatype match.

The conversion of code objects, such as stored procedures, is a different matter. There are a few third party tools such as Ispirer's SQLWays product that will convert stored procedures from Oracle, Sybase, SQL Server, and others to MySQL stored procedures.

Turning the attention back to the conversion of schema objects, one possible strategy is to make use of the free MySQL Migration Toolkit that automatically converts any source database reverse engineered through the tool to a MySQL-based counterpart, complete with all datatype translations. A user has complete control over exactly what objects are migrated from the source database as well as how MySQL specific designations are made.

To use the free MySQL Migration Toolkit for InfiniDB, a user would generally follow these steps:

1. Download and install the migration toolkit. It can be found on the MySQL web site under the GUI tools download section (currently: http://dev.mysql.com/downloads/gui-tools/5.0.html)
2. Connect to the source database with the tool
3. Connect to the InfiniDB target database with the tool
4. Select the schemas and objects that will be migrated to InfiniDB
5. On the object mapping screen, select the User defined radio button and enter in 'engine=infinidb' for the SQL script that will be created
6. Rather than automatically create the objects online, choose to send the SQL creation to a script
7. Review the creation script in an editor and make any needed datatype or other changes needed for InfiniDB
8. Run the script through the InfiniDB SQL editor and create the InfiniDB table objects

10.2 MySQL Data Migration Strategies

To move data from a generic MySQL database to InfiniDB, a couple of different methods can be used. First, a user can use the MySQL SELECT INTO OUTFILE command to spool a set of table output to a flat file. For example, if a user wanted to write out data from a table named 't1' with two columns into a file that was delimited by the pipe ('|') character, they could do the following:

```
mysql> select c1, c2
    -> from t1
    -> into outfile 't1.tbl' fields terminated by '|';
```

Once the MySQL file has been written, it can be moved to the InfiniDB import directory and imported into InfiniDB by using the cpimport utility.

Another strategy is to make use of free open source ETL (extract-transform-load) tools to move data from one MySQL server to an InfiniDB server. Tools are available from Pentaho, Jaspersoft, Talend, and others.

10.3 Oracle Data Migration Strategies

There are a number of ways to move data from Oracle to InfiniDB. Any data movement tool that can connect to MySQL is a candidate for moving data to InfiniDB. For example, a DBA can freely download and use an open source ETL tool such as Talend or Kettle from Pentaho to move data and do transformations if needed.

Oracle does not supply an unload tool for its database, so unless they possess a third party unload tool to unload Oracle data, then a DBA is oftentimes reduced to spooling a flat file out from a SQL*Plus session. For example, if a DBA wanted to create a flat file from Oracle to load into InfiniDB that contained a table named 'workorder' with three columns, and delimit the data with the pipe ('|') character, they might use a script such as the following:

```
SET TERMOUT OFF
set newpage 0
set space 0
set linesize 500
set pagesize 0
set echo off
set feedback off
set heading off
set trimspool on
col wonum     format a10
col leadcraft format a8
col location  format a50
spool c:\workorder.txt
SELECT
WORKNUM || '|' ||
CONTACT || '|' ||
PRIMARY_LOCATION
FROM SCHEMA1.WORKORDER;
spool off
exit
```

10.4 Microsoft SQL Server Data Migration Strategies

Microsoft provides Integration Services with its SQL Server product, which is a bundled data movement and BI platform. A SQL Server DBA can use it to connect and move data to InfiniDB.

Unlike Oracle, SQL Server does have a fast unload tool for its database called `bcp`. The `bcp` utility can be used to create flat files of SQL Server table data that can be used to load into InfiniDB using its `cpimport` utility. For large volumes of data, a `bcp` out of SQL Server data followed by a `cpimport` of data into InfiniDB is likely the best strategy.