

CloudBase Tutorial

Version 1.0

October 16, 2008

Introduction

CloudBase is a data warehouse system build on top of [Map-Reduce](#) architecture. The current CloudBase code has been developed to [Hadoop's](#) map-reduce implementation. CloudBase provides a database abstraction layer on top of flat log files and allows one to query these log files using ANSI SQL. CloudBase comes with a JDBC driver, so one can use any JDBC Database Manager application as a client to connect to CloudBase. CloudBase also allows one to push results of queries to any RDBMS (e.g Oracle, MS SQL Server) that provides a JDBC Driver.

License

CloudBase is developed by Business.com and is released under GPL license version 2.0

Pre-requisites

To run CloudBase, you will need-

- Hadoop preferably version 0.18 or above
- Java version 1.6 or above

Installation and setup

Download and extract CloudBase from –

<http://sourceforge.net/projects/cloudbase/>

The files/directories present in \$CLOUDBASE_HOME directory are explained below-

- *bin*- directory containing scripts e.g. script to start CloudBase server
- *build.xml*- the build file to compile CloudBase from source
- *conf*- directory containing *cloudbase.props* file to configure CloudBase server (e.g one can change the default port of CloudBase via this file)
- *lib*- directory containing third-party jar files
- *COPYING*: the GPL license version 2.0
- *logs*- directory to store CloudBase log files

- *meta*- directory to store meta data. As mentioned above, CloudBase allows one to view and query flat log files as database tables. CloudBase achieves this by associating meta data with the log files. The meta data is stored in this directory.
- *src*- the CloudBase source code
- *test*- directory containing test scripts and utilities.
- *docs*- directory containing release notes, documentation, CloudBase SQL grammar, sample java program to connect to CloudBase using JDBC.

Open `$CLOUDBASE_HOME/bin/cloudbase-env` file and set the `HADOOP_HOME` variable.

To run CloudBase server, run -

```
$CLOUDBASE_HOME/bin/start-cloudbase
```

You will see the output-

```
CloudBase listening on:4444
```

To start querying your log files using CloudBase, you need to connect to CloudBase server. There are two ways to do this-

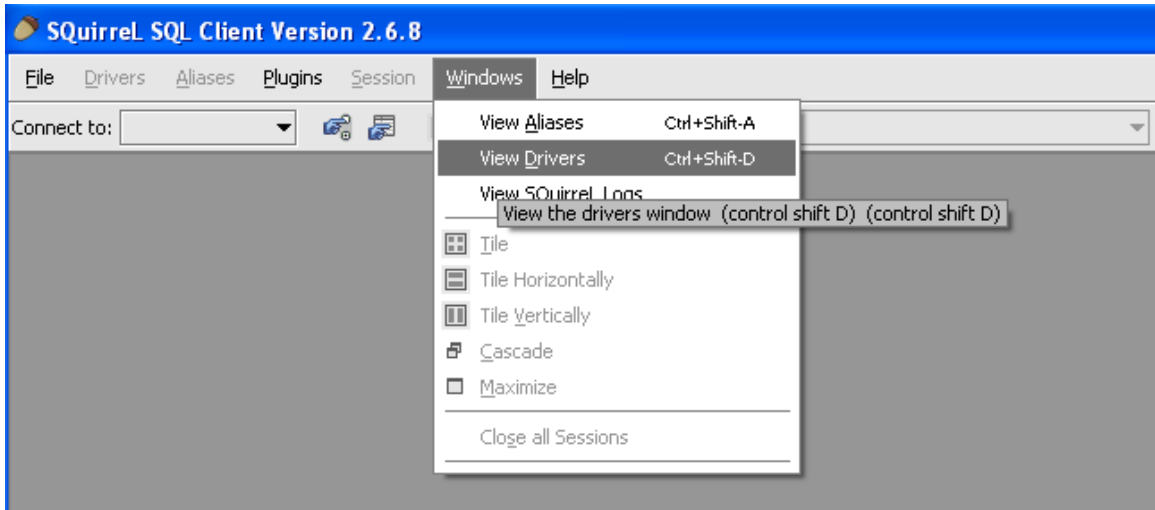
1. Use CloudBase JDBC driver with JDBC Database Manager Applications like Squirrel. We have done most of the testing with Squirrel SQL Client and we suggest that you try it first before trying other JDBC Client applications.
2. Write JDBC program. An example JDBC program is present in *docs* directory in the `$CLOUDBASE_HOME`.

Following steps illustrate how to use Squirrel SQL client to connect to CloudBase server

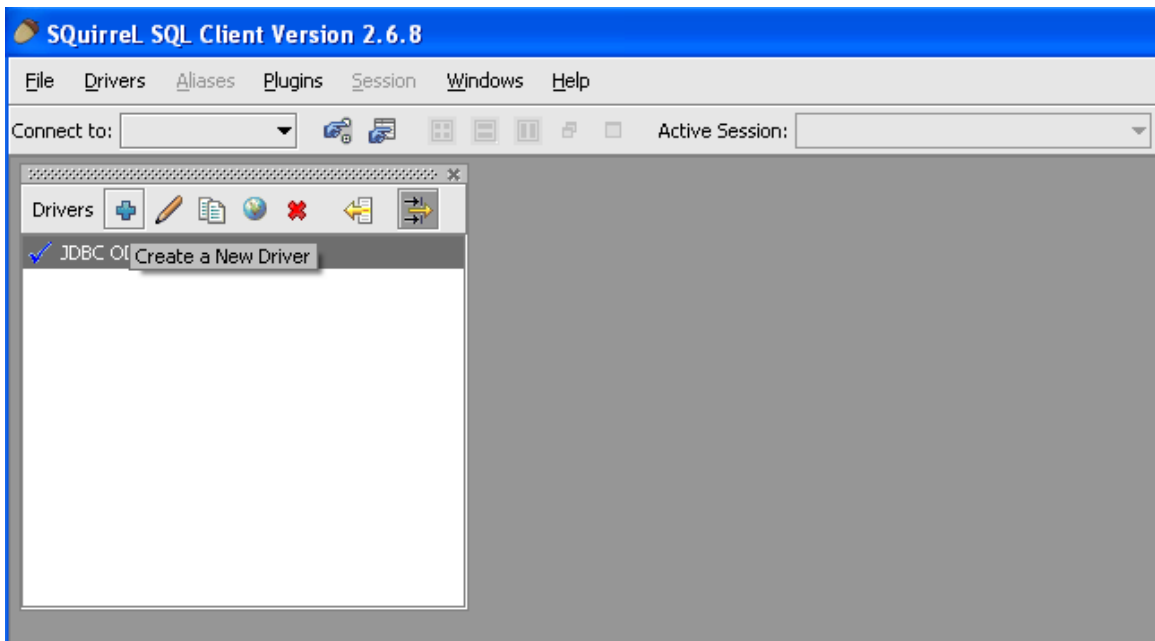
Setting up Squirrel JDBC client to connect to CloudBase server

One can download Squirrel SQL client from – <http://www.squirrelsql.org/>

1. Start Squirrel SQL client and open Drivers windows-



2. Create new Driver-



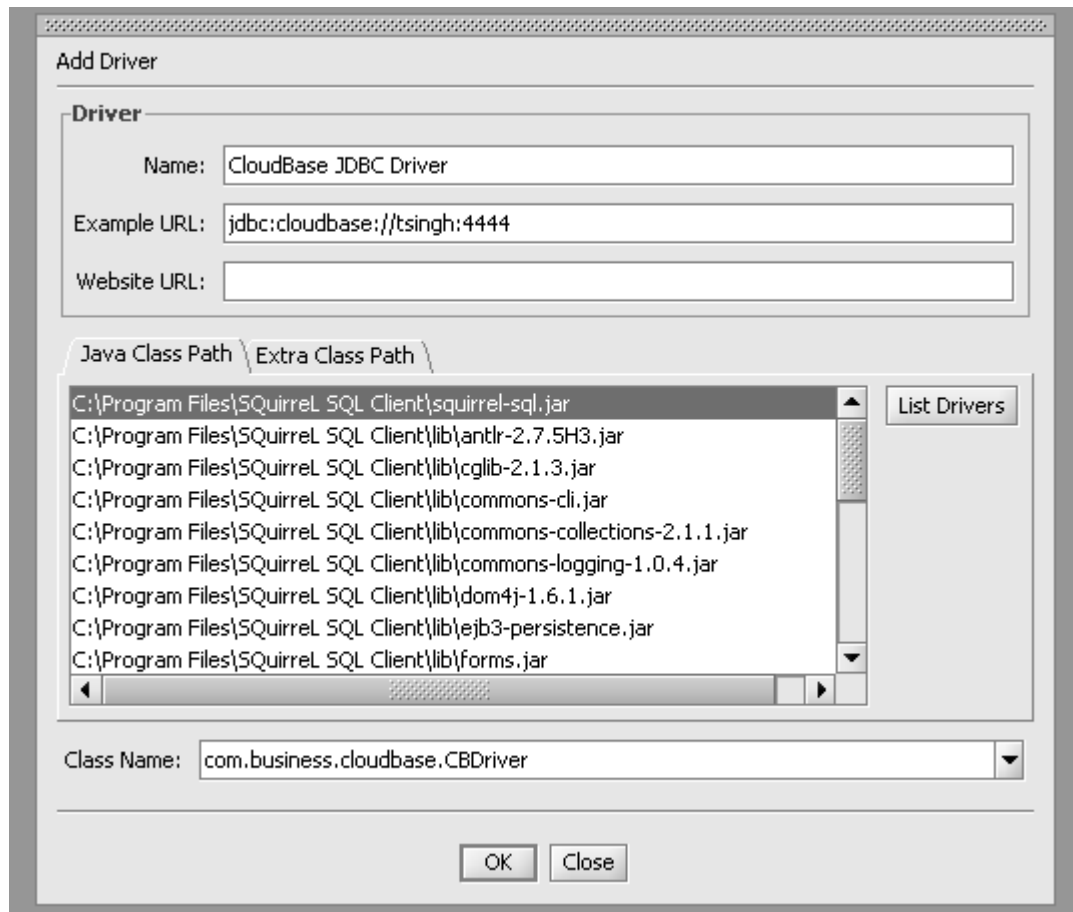
3. In the Add driver window, fill the Name, Example URL and Class Name fields with the following values-

Name: *CloudBase JDBC Driver*

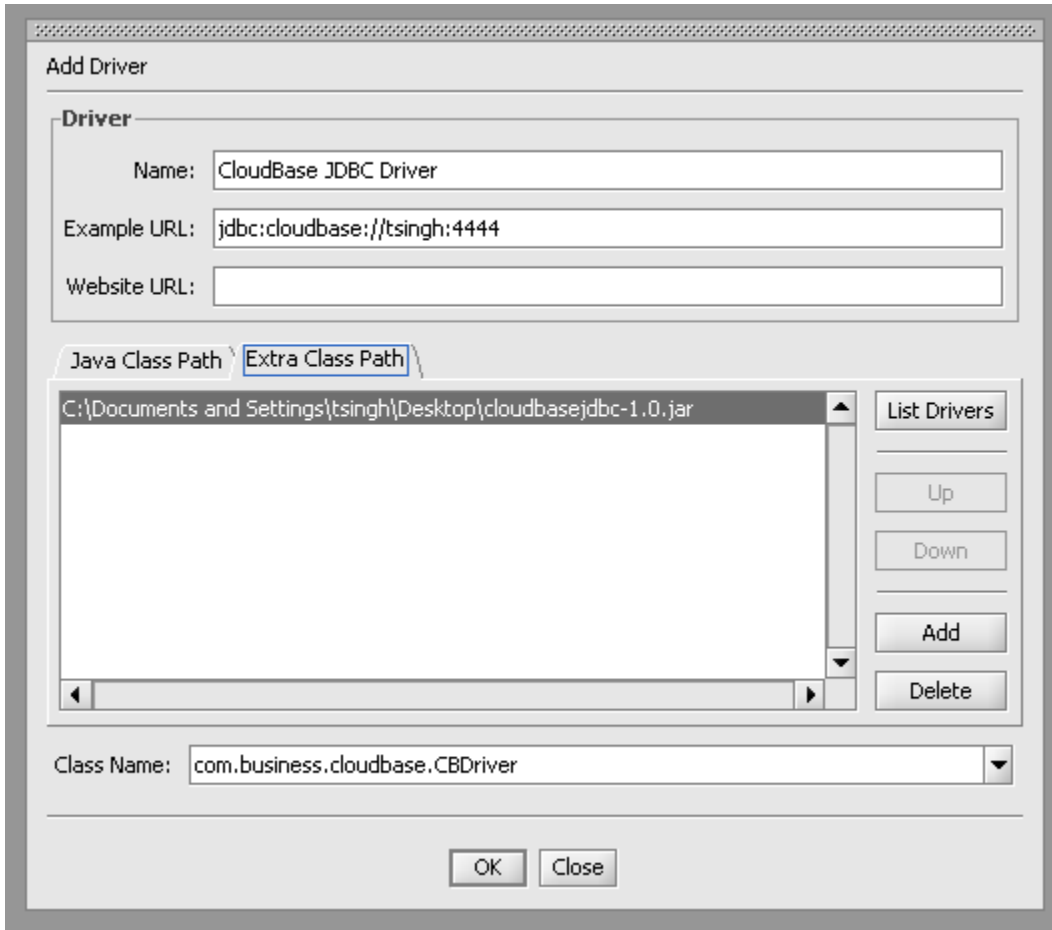
Example URL: *jdbc:cloudbase://hostname:port*

Class Name: *com.business.cloudbase.CBDriver*

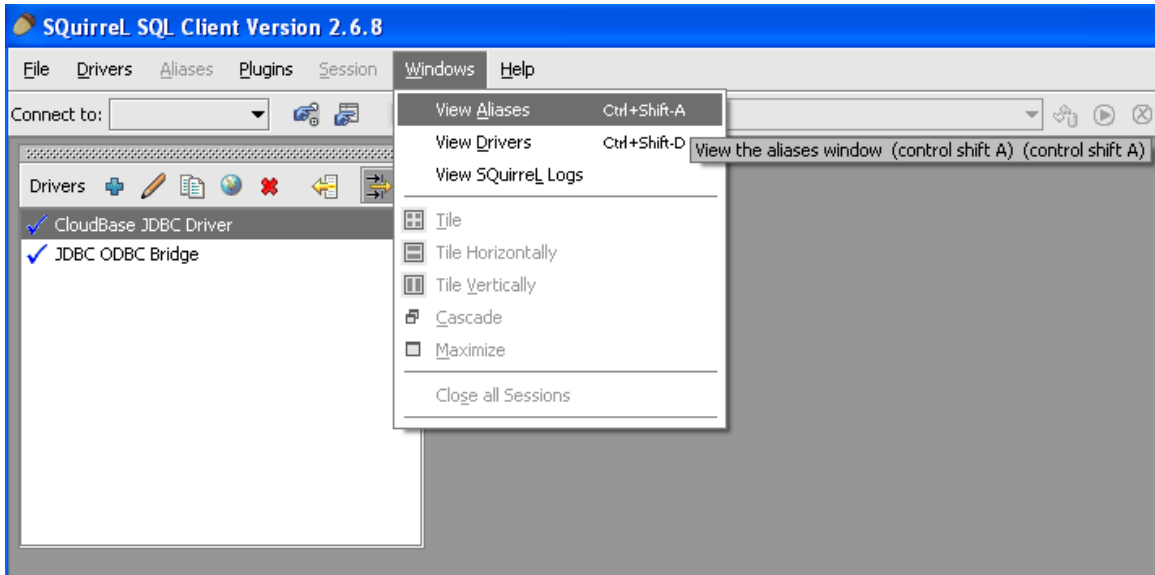
Here hostname is the host running CloudBase Server and port is the port number at which it is listening for incoming JDBC Connections. Default port is 4444



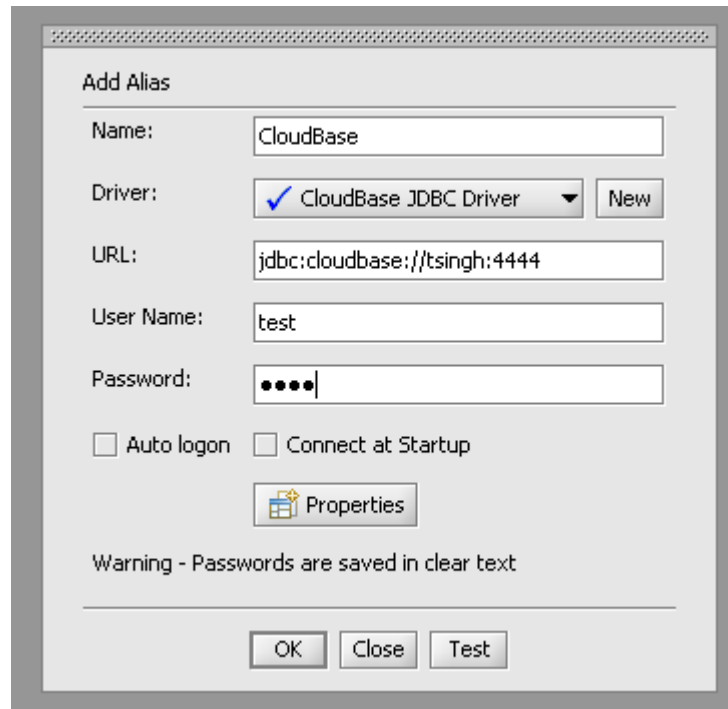
4. Click on Extra Class Path and add the *cloudbase-jdbc-1.0.jar* file.



5. To connect to CloudBase Server, first make an alias- Open alias window –



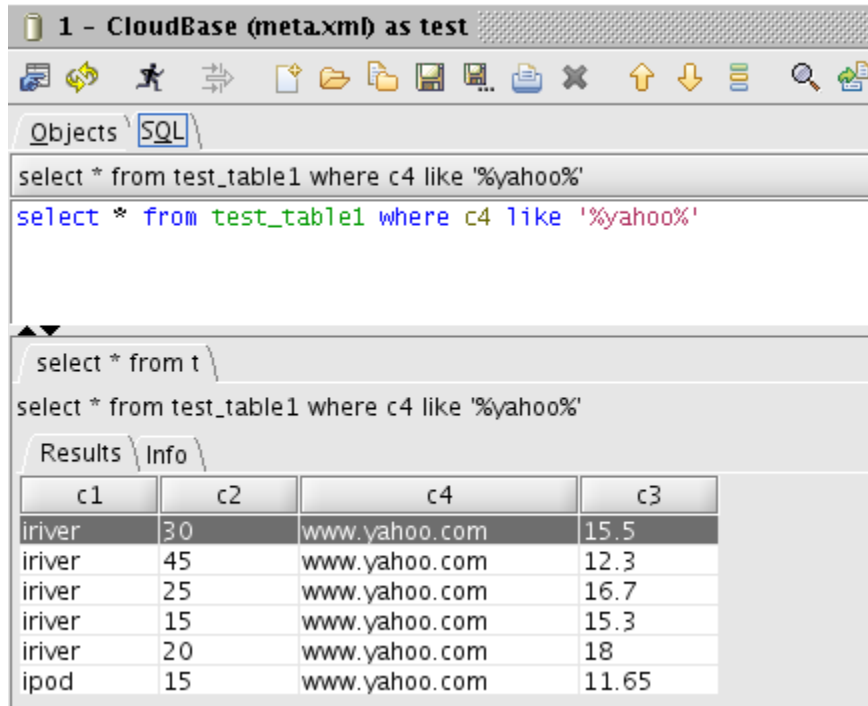
6. Create new alias. Give some name for the alias, say *CloudBase*



7. The URL shown is the one we gave during driver registration. Don't change it.
8. Give *test/test* as the username/password and connect.

You should be connected to CloudBase server. If not, please check if the firewall on the host that is running CloudBase, is allowing port 4444 (or the port you have set)

Fire the queries via the SQL window. You can split the queries in multiple lines if you wish.

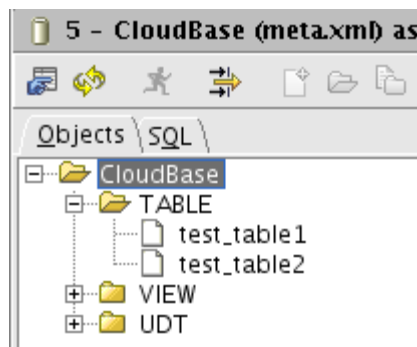


Create sample tables

CloudBase comes with test tables on which you can run various queries and see the output. To create test tables, run-

```
$CLOUDBASE_HOME/test/bin/setup
```

Now you should see the test tables in the Squirrel SQL Client's object browser's window-



CloudBase query language

CloudBase query language is ANSI SQL. Following sections explain how to create tables/views/database-links and query tables/views.

CASE sensitivity

CloudBase is case insensitive. However literal strings (quoted strings like 'LiTeRaL') are treated as such).

String Literals

Single quotes are used to specify string literals, e.g. ' this is a valid string literal ' and " this is not a valid string literal "

Create Statement

CREATE TABLE

A Table in CloudBase is a Path on the Distributed File System. This path can be a file or a directory containing files and/or subdirectories. The CREATE TABLE command associates a name with this path and this name can be used in SELECT or INSERT statements.

One can use CREATE TABLE command to associate a name with an already existing path or one can create a new path and optionally copy files from a local file system into this path. The CREATE TABLE statement has the following syntax –

```
CREATE TABLE <table name>
(<column name> <data type> [, <column name> <data type>, ...] )
COLUMN SEP <string>
[ HDFS PATH <hdfs path> ]
[ IMPORT DATA FROM <local path> ]
[ ON DROP DO NOT PURGE ]
[ COMMENT <string> ]
```

See Data Types section for details on data types supported by CloudBase. Clauses present within '[']' indicate that they are optional. The COLUMN SEP clause is used to indicate the delimiter that separates the fields in the log files. The HDFS PATH clause instructs CloudBase to use this path (on Hadoop File System) instead of creating a new one. The IMPORT DATA FROM clause instructs CloudBase to copy files from the specified location on the local file system to the Hadoop File System. The ON DROP clause should be used if one wants to retain the log files after the table is dropped. By default, once the table is dropped, the log files present in the directory associated with this table are removed from the Distributed File System. The COMMENT clause is used to associate a comment with this table.

As is clear from the above grammar, CloudBase expects log files to be in some structured format, like fields delimited by '|' or tab ('\t'). However, CloudBase is tolerant to structural errors in the log files (see ON ERROR CLAUSE later).

In future releases, support will be added to use even semi-structured log files or totally unstructured log files.

Examples-

```
CREATE TABLE table1 ( c1 VARCHAR, c2 INT) COLUMN SEP '|'
```

This will create an empty table with the name *table1*

```
CREATE TABLE table1 ( c1 VARCHAR, c2 FLOAT) COLUMN SEP '|' HDFS PATH  
'mylogFiles/weblogs'
```

This will create a table with name *table1* and attach this name with the path- 'mylogFiles/weblogs' on Hadoop File System. If this path exists and has log files, then those will constitute the table data. If the path does not exist, it will be created. In that case the table won't have any data.

```
CREATE TABLE table1 ( c1 VARCHAR, c2 FLOAT) COLUMN SEP '|' IMPORT DATA  
FROM '/home/CloudBase/logs/weblogs'
```

This will create table with name *table1* and copy data from the given local path. The data from the local path is loaded on Hadoop File System in the directory - cloudbase/data/<tablename>.

```
CREATE TABLE table1 (c1 VARCHAR, c2 DATETIME ) COLUMN SEP '|' IMPORT  
DATA FROM '/home/logs/weblogs'
```

This will create table with name *table1* with 2 columns c1 and c2. The column c2 has DATETIME data type with default format- 'yyyy/MM/dd'

```
CREATE TABLE table1  
( c1 VARCHAR, c2 DATETIME FORMAT 'dd-MMM-yy')  
COLUMN SEP '|' IMPORT DATA FROM '/home/logs/weblogs'
```

This will create table with name *table1* and set the DATETIME format of column c2 as 'dd-~~MMM~~-yy' e.g. '03-Jul-08'.

CREATE VIEW

A view in CloudBase is just like an RDBMS view (a stored query). The syntax for view creation is as follows-

```
CREATE VIEW <viewname> AS <query>
```

Example-

```
CREATE VIEW view1 AS  
SELECT c1, c2 FROM table1 WHERE c1 > 100 and c2 > 50.55
```

One can also use sub queries in the view, however alias is mandatory while using sub queries-

```
CREATE VIEW view1 AS  
SELECT c1, c2 FROM (SELECT c1, c2 from table1 where c1 > 100 and c2 >  
50.55) AS T
```

See more on Aliases in the Alias section.

CREATE DATABASE LINK

CloudBase can push the result of a query into a RDBMS table via database links. A Database Link defines a JDBC connection between CloudBase and RDBMS. In order to use Database Links, it is required to include the RDBMS JDBC driver in the java CLASSPATH. Syntax for Database Link creation-

```
CREATE DATABASE LINK <linkname>  
CONNECTION URL 'jdbc_conn_url_for_rdbms'  
JDBC_DRIVER 'rdbms_jdbc_driver_class_name'  
[ USER username PASSWORD password ]  
[ jdbc_property val, jdbc_property val ...]
```

For example, to make a link to MS-SQL Server, download the JDBC driver and add it to CLASSPATH. You can save the JDBC driver in the \$CLOUDBASE_HOME/lib directory. CloudBase will add it automatically to the CLASSPATH. Restart CloudBaseserver and run the following SQL-

```
CREATE DATABASE LINK sql_server_link  
CONNECTION URL  
'jdbc:sqlserver://your_host:8888;user=user;password=password;databaseName=your_database'  
jdbcdriver 'com.microsoft.sqlserver.jdbc.SQLServerDriver'
```

Change the hostname, port, user, password according to your MS-SQL server.

Then data can be inserted into an already existent table on MS-SQL server as follows–

```
INSERT INTO tablename@sql_server_link  
Select c1, c2, c3 from table1
```

See INSERT statement for more details.

NOTE: CloudBase Server stores the user name, password as plain text. So use this feature at your own discretion.

DROP Statement

Drop statement is used to remove database objects (TABLE, VIEW, DATABASE LINK). The syntax is given below –

Drop Table

```
DROP TABLE table_name
```

This will remove the given table from CloudBase. It will also delete the log files/directory associated with it from the Distributed File System. If one does not want to delete the log files/directory, then one can use the on drop clause- ON DROP DO NOT PURGE while creating the table.

Drop View

```
DROP VIEW view_name
```

This will remove the given view name.

Drop Database Link

```
DROP DATABASE LINK link_name
```

This will remove the given database link

INSERT Statement

Insert statement is used to insert data into an already existent CloudBase table or into a RDBMS table (via Database link). Syntax –

```
INSERT INTO <tablename> [@database_link_name]
[ ( column_name1, column_name2, ...) ]
<query>
```

The column list is optional. The database link option is used if data needs to be inserted into a RDBMS table. As mentioned in the CREATE DATABASE LINK section, the jdbc driver for the RDBMS should be present in the CLASSPATH.

Example-

```
INSERT INTO table2
Select c1, c2, c3 from table1
```

This will dump the output of the query into table2.

```
INSERT INTO table2
( col2, col1, col3 )
Select c1, c2, c3 from table1
```

This will dump the output of the query into table2 in the specified column order.

```
INSERT INTO table2@database_link_name
Select c1, c2, c3 from table1
```

This will dump the output of the query into table2 in RDBMS as specified by the database_link_name.

If insufficient columns are specified, then INSERT statement will insert NULL values for the remaining columns.

Please note that, the INSERT statement does not support inserting values using VALUES clause. The only way to insert values is via the Query.

SELECT Statement

The select statement is used to query data from CloudBase tables. Its syntax is –

```
SELECT [ DISTINCT ] [ TOP <number> ]  
column1 [ , column2, column3, ... ]  
[ into_clause ]  
FROM <from_list>  
[ join_clause ]  
[ where_clause ]  
[ group_by_clause [ having_clause ] ]  
[ order_by_clause ]  
[ on error clause ]
```

As mentioned above, clauses present in '[']' indicate that they are optional

From clause

One can use tables, views and/or sub queries in the from-clause. Aliases can be optionally set for table names and views. **However, setting aliases for sub queries is mandatory.** Example-

```
SELECT * FROM test_table1  
SELECT * FROM ( SELECT c1, c2 FROM test_table1) AS T  
SELECT DISTINCT c1 FROM test_table1
```

Join clause

CloudBase supports inner and outer (left, right, full) joins via ANSI SQL's **explicit join syntax**-

```
SELECT * FROM test_table1 AS t1  
INNER JOIN test_table2 AS t2 ON t1.c1 = t2.c1
```

Another example-

```
SELECT * FROM test_table1 AS t1  
[ OUTER ] LEFT | RIGHT | FULL JOIN test_table2 AS t2  
ON t1.c1 = t2.c1
```

Sub queries can also be used instead of tables/views –

```
SELECT * FROM test_table1 AS t1  
INNER JOIN
```

```
(SELECT * FROM test_table2) AS t2
ON t1.c1 = t2.c1
```

NOTE: Only equijoins are supported as of now.

Where clause

Rows returned by queries can be filtered by using *Where clause*. At present CloudBase supports the following conditions in the *Where Clause*-

- Arithmetic condition- compare columns using arithmetic operators- >, <, =, >=, <=, !=, <> (either of !=, <> can be used for not equal to)
- Between condition- e.g. *WHERE c1 BETWEEN 10 and 100*. NOT can also be used with BETWEEN- *WHERE c1 NOT BETWEEN 10 and 100*
- Like condition- e.g. *WHERE c1 LIKE '%abc%'*. NOT can also be used- *WHERE c1 NOT LIKE '%abc%'*. To match on a single character use '?'. One can use regular expressions in LIKE clause. See below for details.

Boolean AND / OR can be used to tie conditions together, e.g-

```
SELECT * FROM test_table1 WHERE c1 LIKE 'z%' AND c2 > 10
SELECT * FROM test_table1 WHERE c2 > 40 OR c3 < 20
```

Support for IS NULL, IS NOT NULL and IN condition will be added in the next release.

Regular Expressions in LIKE clause-

CloudBase supports SQL wild cards- '%' and '?' in LIKE conditions. For conditions that can not be expressed using these wild cards, one can use regular expressions. CloudBase supports regular expression similar to Java programming language. One can read about Java regular expressions here – <http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html>

Some example queries using regular expressions are given below-

```
SELECT c6 FROM test_table3 where c6 like 'b[ea]ta'
(matches 'beta', 'bata')
```

```
SELECT * FROM test_table3 WHERE c like '(l|L)earning'
(matches 'learning', 'Learning')
```

```
SELECT c FROM test_table WHERE c LIKE '(z|Z)o{3}+m'
```

(matches all values of column c that begins with z or Z followed by EXACTLY 3 o's and then m)

```
SELECT c FROM test_table WHERE c LIKE '(z|Z)o{3,}+m'
```

(matches all values of column c that begins with z or Z followed by AT LEAST 3 o's and then m)

```
SELECT c FROM test_table WHERE c LIKE '(z|Z)o{3,6}+m'
```

(matches all values of column c that begins with z or Z followed by AT LEAST 3 but NOT MORE THAN 6 o's and then m)

To escape regular expression construct, use '\'. For example-

```
SELECT c FROM test_table WHERE c LIKE '\[abc\]'
```

(matches literal string- '[abc]')

Group by clause

CloudBase supports aggregate functions and group by clause. The aggregate functions supported by CloudBase are- SUM, COUNT, MAX, MIN, and AVG. One can use GROUP BY statement in conjunction with the aggregate functions to group the result-set by one or more columns. In Group by clause, one can use column names, alias or index of column present in the select clause. e.g.-

```
SELECT c4, COUNT(*) FROM test_table1 GROUP BY c4
```

```
SELECT c1, SUM(c2) FROM test_table1 GROUP BY c1
```

```
SELECT c1 as a, SUM(c2) FROM test_table1 GROUP BY a
```

```
SELECT c1, c4, SUM(c2) FROM test_table1 GROUP BY 1,2
```

```
SELECT COUNT(c1),MAX(c2),MIN(c3), c4 FROM test_table1 GROUP BY c4
```

DISTINCT in Aggregate functions

```
SELECT COUNT( DISTINCT c1), MAX(c2) FROM test_table1
```

```
SELECT COUNT( DISTINCT c1), MAX(c2), MIN(c3), COUNT(DISTINCT c4)
```

```
FROM test_table1
```

DISTINCT in Aggregate functions with GROUP BY

```
SELECT COUNT( DISTINCT c1), COUNT( DISTINCT c2),
```

```
COUNT( DISTINCT c3), COUNT( DISTINCT c4), MAX(c2), MIN(c2),
```

```
MAX(c3), MIN(c3), c4 FROM test_table1 GROUP BY c4
```

HAVING clause with GROUP BY

```
SELECT COUNT(c1) cnt_c1, COUNT( DISTINCT c2) cnt_d_c1,  
SUM(c2) sum_c2, SUM(c3) sum_c3, c4 FROM test_table1 GROUP BY c4  
HAVING cnt_d_c1 > 2 AND sum_c3 > 100
```

```
SELECT COUNT(c1) cnt_c1, MAX(c4) max_c4, MIN(c4) min_c4, MAX(c5)  
max_c5, MIN(c5) min_c5, SUM(c4) sum_c4, SUM(c5) sum_c5,  
COUNT(DISTINCT c1), SUM(DISTINCT c4), c6 FROM test_table3 GROUP  
BY c6 HAVING c6 LIKE '%p%'
```

Order by clause

ORDER BY clause can be used to sort result set on one or more columns. One can sort in ascending order (default behavior) or in descending order. Just like Group By clause, one can use column name, alias or column index in the ORDER BY clause-

```
SELECT c1, c2, c3, c4 FROM test_table1 ORDER BY c1  
SELECT c1 as a, c2 as b, c3 as c, c4 as d FROM test_table1 ORDER BY a, b  
SELECT c1, c2, c3, c4 FROM test_table1 ORDER BY 3  
SELECT * FROM test_table1 ORDER BY c4 DESC, c2
```

On error clause

ON ERROR clause is used to tell CloudBase what to do when some error occur while processing log file data. One can specify error handling clause for the following errors-

- **DATA CONVERSION ERROR:** This kind of error occurs when CloudBase can not convert log file field's value into the column's data type (e.g. got some string, when expecting a number or unable to parse date time data type). One can specify one of the following actions-
 - **SKIP:** Skip this record and move further. This is the default action
 - **STOP:** Stop processing. This functionality is broken right now.
 - **LOG:** Log the bad record and continue.
- **LESS COLUMNS ERROR:** This kind of error occurs when CloudBase got fewer columns (fields) while parsing a line of record than expected. Actions supported are- SKIP, STOP, LOG and APPEND NULL. The APPEND NULL will put SQL NULL for the missing fields in the end. Default action is to SKIP the record.

- **MORE COLUMNS ERROR:** This kind of error occurs when CloudBase got more columns (fields) while parsing a line of record than expected. Actions supported are- SKIP, STOP and LOG. Default action is to SKIP the record.

An example is shown below-

```
SELECT * FROM test_table1 WHERE c1 = 'ipod'
ON LESS COLUMNS ERROR APPEND NULL
```

SELECT INTO

SELECT INTO statement is used to dump output of a query into a new CloudBase table or into a file on local file system. Example-

```
SELECT * INTO table2 FROM table1 WHERE c1 > 10
```

This will create a new table- *table2* and populate it with the result of the query - *select * from table1 where c1 > 10*. The column names and data types of the newly created table are same as the columns specified in the SELECT statement.

```
SELECT * INTO '/tmp/output' FROM test_table1
```

This will store the output of the query into a directory '/tmp/output' on local file system (of the machine where CloudBase server is running).

SET operators (UNION, INTERSECT etc)

One can perform set operations on the result sets of 2 or more SELECT statements via set operators. The set operators supported by CloudBase are-

- **UNION:** The union operator combines the results of two or more SELECT statements into a single result set. All the SELECT statements should have the same number of columns and compatible data types. Any duplicate records/rows are automatically removed.

Example-

```
SELECT c1 FROM test_table1
UNION
SELECT c1 FROM test_table2
```

- **UNION ALL:** Same as UNION except that duplicate records are removed. Example-

```
SELECT c1, c2, c3 FROM test_table1
UNION ALL
SELECT c1, c2, c3 FROM test_table2
```

- **INTERSECT:** It takes result sets from two SELECT statements and returns only rows that appear in both the result sets. Example-

```
SELECT c2 FROM test_table1
INTERSECT
SELECT c4 FROM test_table2
```

- **MINUS:** It takes results from the first SELECT statement and then subtract the ones that are present in the second SELECT statement's result set to get the final result set. Example-

```
SELECT c2 FROM test_table1
MINUS
SELECT c2 FROM test_table2
```

Using ALIASES

One can give an alias to a column name used in the select clause and then use that alias in the where clause/order by clause etc. One can also give aliases to tables and sub queries used in the from clause. Please note that, *setting alias is mandatory for a sub query*. The keyword 'AS' is not mandatory while setting aliases for column names or tables or sub queries. Examples-

```
SELECT c1 col1, c2 AS col2 FROM test_table1
SELECT T.c1 col1, T.c2 col2 FROM test_table1 T
SELECT T.c1 col1, T.c2 col2 FROM test_table1 AS T
SELECT T.c1 col1, T.c2 col2 FROM test_table1 AS T where col1 > 10
SELECT c1 FROM ( SELECT c1 FROM test_table1 ) AS T
SELECT T.c1 FROM ( SELECT c1 FROM test_table1 ) T
SELECT c1 + c2 c FROM test_table1 where c > 100
(Here alias c is given to expression c1 + c2)
SELECT c1 + c2 AS c FROM test_table1 where c > 100
SELECT ( c1 + c2 ) AS c FROM test_table1 where c > 100
```

Data Types

The data types supported by CloudBase are listed below. Also listed is their precedence, corresponding java.sql.Type and java data type used internally for expression evaluation. The precedence value is used in evaluating expressions having different data types. A lower precedence data type is promoted to higher precedence data type using java's type promotion rules. For example, while evaluating expression having INT and REAL data types, the INT data type is promoted to REAL. If a data type can not be promoted to higher-precedence data type, then it results in error and the operation is terminated. Further, certain operations are not permitted on certain data types and any attempt to perform these operations will result in error. For example, addition/subtraction/division/multiplication of two VARCHAR data types will result in error.

Data types supported by CloudBase

Data Type	Precedence	java.sql.Type	java data type
TINYINT	2	TINYINT	byte
SMALLINT	3	SMALLINT	short
INT	4	INT	int
BIGINT	5	BIGINT	long
REAL	6	REAL	float
DOUBLE	7	DOUBLE	double
FLOAT	7	FLOAT	double
VARCHAR	8	VARCHAR	String
DATETIME	9	DATE	java.util.Date

When data type of a column is set to DATETIME, default format (yyyy/MM/dd) is assumed, however one can change the format using the FORMAT clause followed by the format string. The format string is specified using date time pattern strings –

Symbol	Meaning	Examples
G	Era Designator	“GG” => “AD”
Y	Year	“yy” => “08” “yyyy” => “2008”
M	Month in year	“M” => “7” “M” => “12” “MM” => “07” “MMM” => “Jul” “MMMM” => “December”
w	Week in year (1-53)	“w” => “7”
W	Week in month (1-5)	“W” => “3”
D	Day in year (1-365 or 1-364)	“D” => “65” “DDD” => “065”
d	Day in month	“d” => “3” “dd” => “03”
F	Day of week in month (1-5)	“F” => “1”
E	Day in week	“EEE” => “Tue” “EEEE” => “Tuesday”
a	Am/pm marker	“a” => “AM” “aa” => “AM”
H	Hour in day (0-23)	“H” => “15” “HH” => “15”
k	Hour in day (1-24)	“k” => “3” “kk” => “03”
K	Hour in am/pm (0-11)	“K” => “15” “KK” => “15”
h	Hour in am/pm (1-12)	“h” => “3” “hh” => “03”
m	Minutes in hour	“m” => “7” “m” => “15” “mm” => “15”
s	Second in minute	“s” => “15” “ss” => “15”
S	Millisecond (0-999)	“SSS” => “007”
z	Time zone	“z” => “EST” “zzz” => “EST” “zzzz” => “Eastern Standard Time”

Example –

```
CREATE TABLE table1
( c1 VARCHAR, c2 DATETIME FORMAT 'dd-MMM-yy')
```

java.Text.SimpleDateFormat is used to parse the date/time format string constructed using the above mentioned patterns. For more details, see java docs for [SimpleDateFormat](#).

Scalar Functions

CloudBase supports String and Date time functions as mentioned in JDBC specifications. These functions are explained below-

String Functions

Function Name	Function Returns
ASCII(string)	Integer representing the ASCII code value of the leftmost character in string e.g.
CHAR(code)	Character with ASCII code value code, where code is between 0 and 255
CONCAT(string1,string2)	Character string formed by appending string2 to string1;
INSERT(string1, start, length, string2)	A character string formed by deleting length characters from string1 beginning at start, and inserting string2 into string1 at start
LCASE(string)	Converts all uppercase characters in string to lowercase
LEFT(string, count)	The count leftmost characters from string
LENGTH(string)	Number of characters in string, excluding trailing blanks
LOCATE(string1, string2[, start])	Position in string2 of the first occurrence of string1, searching from the beginning of string2; if start is specified, the search begins from position start. 0 is returned if string2 does not contain string1. Position 1 is the first character in string2
LTRIM(string)	Characters of string with leading blank spaces removed
REPEAT(string, count)	A character string formed by repeating string count times
REPLACE(string1, string2, string3)	Replaces all occurrences of string2 in string1 with string3
REPLACEALL(string1, string2, string3)	Same as replace, but string2 is interpreted as regular expression
RIGHT(string, count)	The count rightmost characters in string
RTRIM(string)	The characters of string with no trailing blanks
SPACE(count)	A character string consisting of count spaces
SUBSTRING(string, start, length)	A character string formed by extracting length characters from string beginning at start
UCASE(string)	Converts all lowercase characters in string to uppercase

An example query using various string functions is given below-

```
select ascii( c1) as f_ascii,
       char( 65) as f_char,
       left( c4, 3) as f_left,
       ltrim( '  TARAN') as f_ltrim,
       replace( replace( c4, 'com', 'COM'), 'www', 'WWW') as f_replace,
       right( c4, 3) as f_right,
       rtrim( 'TARAN  ') as f_rtrim,
       space(4) as f_space,
```

```

substring( c4, 1, 3) as f_substring,
lcase( 'TARAN') as f_lcase,
ucase( c1) as f_ucase,
length( c4) as f_length,
insert( c4, 1, 3, 'WWW') as f_insert,
locate( 'com', c4) as f_locate1,
locate( 'www', c4) as f_locate2,
locate( 'www', c4, 3) as f_locate3,
locate( 'COM', c4) as f_locate4,
repeat( 'A', 5) as f_repeat
from test_table1

```

Regular expressions in REPLACEALL- As mentioned above regular expressions can be used to replace characters in REPLACEALL. These regular expressions are interpreted as `javax.util.regex`. Some of the regex constructs are shown below-

Construct	Matches
[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
.	Any character
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z 0-9]
\W	A non-word character: [^\w]
^	The beginning of a line
\$	The end of a line
X?	X, once or not at all
X*	X, zero or more times
X+	X, one or more times

For more on regex, see-

<http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Matcher.html>

An example query that you can try on test tables is given below-

```
select replaceall( 'a b      c          d', '\s+', '') as f_replaceall1,
       replaceall( 'ab cd ab cd', '^ab', 'AB') as f_replaceall2,
       replaceall( 'ab cd ab cd', 'cd$', 'CD') as f_replaceall3,
       replaceall( 'abcABC123', '[abc]', '0') as f_replaceall4,
       replaceall( 'abcABC123', '[^abcABC]', '0') as f_replaceall5,
       replaceall( 'abcABC123', '[a-zA-Z]', '0') as f_replaceall6,
       replaceall( 'abcABC123', '.', '0') as f_replaceall7,
       replaceall( 'abcABC123', '\d', '0') as f_replaceall8,
       replaceall( 'abcABC123', '\D', '0') as f_replaceall9_1,
       replaceall( 'abcABC123', '[^0-9]', '0') as f_replaceall9_2,
       replaceall( 'a b c', '\s', '0') as f_replaceall10,
       replaceall( 'a b c', '\S', '0') as f_replaceall10_1,
       replaceall( 'a b c', '[^\s]', '0') as f_replaceall10_2,
       replaceall( 'abc ABC 123', '\w', '0') as f_replaceall11,
       replaceall( 'abc ABC 123', '\W', '0') as f_replaceall12_1,
       replaceall( 'abc ABC 123', '[^\w]', '0') as f_replaceall12_2,
       replaceall( c4, 'www.*com', 'www.site.com') as f_replaceall13,
       replaceall( c4, 'o+', '0') as f_replaceall14
from test_table1
```

Date time functions

CloudBase supports various Date time functions. They are explained below-

Function Name	Function Returns
CURDATE()	The current date as a date value, format- yyyy/MM/dd
CURTIME()	The current local time as a time value, format- HH:mm:ss
DAYNAME(date)	A character string representing the day component of date, format- EEEE e.g. Tuesday
DAYOFMONTH(date)	An integer from 1 to 31 representing the day of the month in date
DAYOFWEEK(date)	An integer from 1 to 7 representing the day of the week in date; 1 represents Sunday
DAYOFYEAR(date)	An integer from 1 to 366 representing the day of the year in date
HOURL(time)	An integer from 0 to 23 representing the hour component of time
MINUTE(time)	An integer from 0 to 59 representing the minute component of time
MONTH(date)	An integer from 1 to 12 representing the month component of date
MONTHNAME(date)	A character string representing the month component of date, format- MMMM e.g. December
NOW()	A timestamp value representing the current date and time, format- yyyy/MM/dd HH:mm:ss
QUARTER(date)	An integer from 1 to 4 representing the quarter in date; 1 represents January 1 through March 31
SECOND(time)	An integer from 0 to 59 representing the second component of time
TIMESTAMPADD(interval, count, timestamp)	A timestamp calculated by adding count number of interval(s) to timestamp; interval may be one of the following: SQL_TSI_FRAC_SECOND, SQL_TSI_SECOND, SQL_TSI_MINUTE, SQL_TSI_HOUR, SQL_TSI_DAY, SQL_TSI_WEEK, SQL_TSI_MONTH, SQL_TSI_QUARTER, or SQL_TSI_YEAR
TIMESTAMPDIFF(interval, timestamp1, timestamp2)	An integer representing the number of interval(s) by which timestamp2 is greater than timestamp1; interval may be one of the following: SQL_TSI_FRAC_SECOND, SQL_TSI_SECOND, SQL_TSI_MINUTE, SQL_TSI_HOUR, SQL_TSI_DAY, SQL_TSI_WEEK, SQL_TSI_MONTH, SQL_TSI_QUARTER, or SQL_TSI_YEAR
WEEK(date)	An integer from 1 to 53 representing the week of the year in date
YEAR(date)	An integer representing the year component of date

An example query that you can try on test tables-

```
SELECT c2, dayname( c2) as f_dayname,  
       dayofmonth( c2) as f_dayofmonth,  
       dayofweek( c2) as f_dayofweek,  
       dayofyear(c2) as f_dayofyear,  
       hour(c2) as f_hour,  
       minute(c2) as f_minute,  
       month(c2) as f_month,  
       monthname(c2) f_monthname,  
       quarter( c2) as f_quarter,  
       second( c2) as f_second,  
       week(c2) as f_week,  
       year(c2) as f_year  
FROM test_table3
```

Sample queries- One can view more sample queries in the directory-
\$CLOUDBASE_HOME/test/queries

Issues/Questions/Bug Reporting

We have created a CloudBaseusers group- <http://groups.google.com/group/cloudbase-users>

For the time being, please send all your questions/issues and report bugs at this group. We will notify all when a more formal bug reporting system is setup. Email for the group: cloudbase-users@googlegroups.com