TimeDB 2.0

Version Beta 4
September 1999

A TimeConsult Product

www.TimeConsult.com

This documentation describes the installation and use of TimeDB 2.0 Beta 4.

Information presented here is accurate as of the time of writing, but is subject to change without notice.

Please send any questions, comments, suggestions and bug reports to steiner@timeconsult.com.

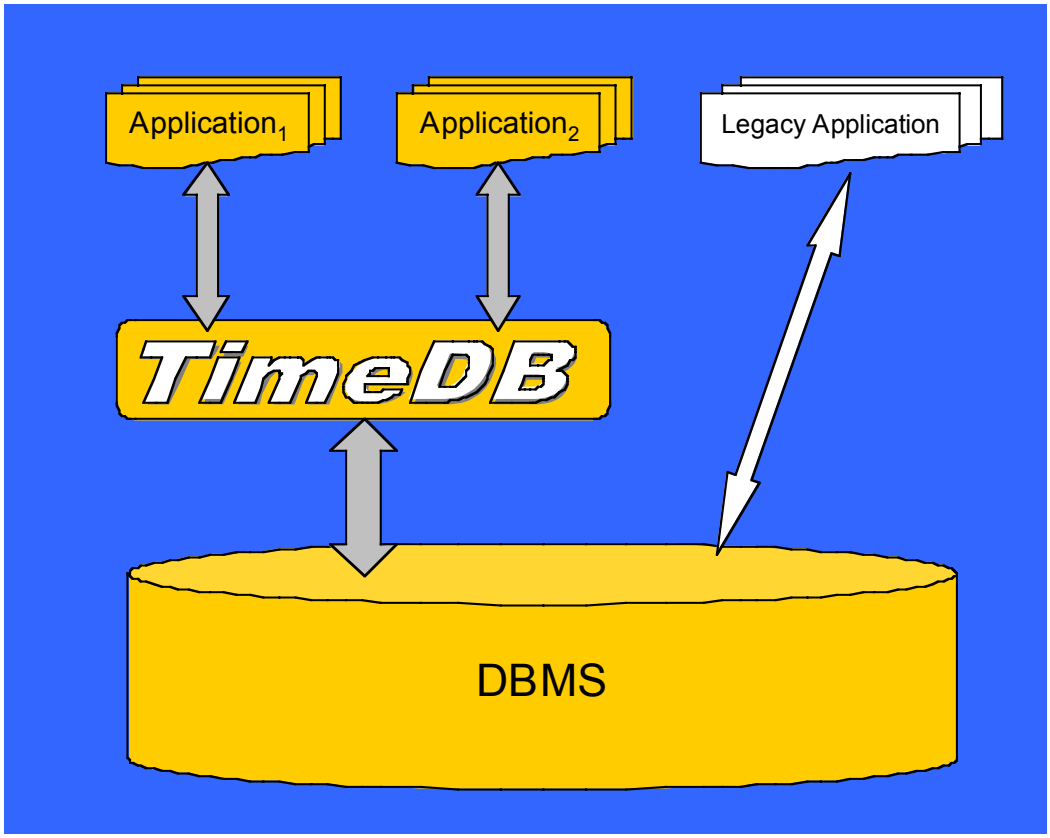
In no event shall TimeConsult or any persons working for TimeConsult be liable for any consequential, incidental or special damages whatsoever (including without limitation damages for loss of critical data, loss of profits, interruption of business, and the like) arising out of the use or inability to use this software.

Table of Contents

What is TimeDB?

TimeDB supports a temporal version of SQL called ATSQL2 [SBJS96a, SBJS96b, SBJS98] by translating temporal SQL statements into standard SQL statements which then are evaluated using a commercial database management system (DBMS). TimeDB thus supports a uniform way to implement applications dealing with historical (or *temporal*) data. By using TimeDB, it is possible to store and manage not only a single database state (as it is done in all the currently available commercial DBMS) but several ones. Research in the area of temporal databases has shown that while it is usually no problem to store the validity time periods of data in commercial DBMS in one way or another, it is very cumbersome to query and update such data and keep it consistent. These drawbacks are eliminated if a *temporal DBMS* is used.

TimeDB, however, is not a temporal DBMS itself but is a frontend to a relational DBMS. By translating temporal SQL into standard SQL statements, TimeDB supports *temporal functionality for a non-temporal relational DBMS*. The advantage of this approach is that existing databases stored in a commercial DBMS and applications accessing this data still can be used while new applications dealing with temporal data can be added. These applications then access the databases via TimeDB. This is depicted in the following figure :



Features of TimeDB

TimeDB 1.0 was implemented during the design of ATSQL2 [SBJS96a, SBJS96b]. It helped refining the language and eliminating weaknesses. This prototype system was implemented at the Swiss Federal Institute of Technology (ETH Zürich) as part of a Ph. D. thesis [S98]. The language implemented in TimeDB 1.0 supports

- temporal queries
- temporal insert, update and delete statements
- temporal tables and views
- temporal table constraints and assertions

It supports valid time (when was a fact true in the real world) and transaction time (when was a fact stored in the database). These time lines are treated orthogonally which means that for each valid-time statement a corresponding transaction-time statement exists.

What is new in TimeDB 2.0?

There are several important differences between TimeDB 1.0 and TimeDB 2.0 :

- TimeDB 2.0 was implemented in **Java** and thus is platform independent
- TimeDB 2.0 uses **JDBC** and thus can be used with many different DBMS
- TimeDB 2.0 has a **GUI** and thus is easier to install and use
- TimeDB 2.0 is **optimised** with respect to the creation of auxiliary tables
- TimeDB 2.0 has a **native call interface (TDBCI)** which Java applications can use to execute ATSQL2 statements

TimeDB 2.0 is a re-implementation of TimeDB 1.0. There were several reasons why we implemented TimeDB 2.0 from scratch. *First*, with the spreading of the object-oriented programming language Java, it becomes possible to run the same code on different platforms without extra effort. Thus, we decided to go for a platform independent implementation for the next release of TimeDB in order to supply it to as many users as possible. *Second*, there were many inquiries of users who wanted to use TimeDB together with a commercial relational DBMS. TimeDB 1.0 could only be used with the product of a single DBMS vendor, namely Oracle, since
TimeDB 1.0 used the native Oracle Call Interface (OCI). However, using JDBC [HCF97], a standardised way to access data in different DBMS is possible. Thus, the DBMS interface of TimeDB 2.0 is based on JDBC and hence is independent of any specific DBMS. *Third*, a graphical user interface (GUI) is helpful to simplify the installation procedure and use of TimeDB. *Fourth*, TimeDB 2.0 generates less auxiliary tables during statement evaluation, e. g. snapshot queries do not generate any auxiliary tables anymore. *Last but not least*, TimeDB 2.0 Beta 4 supports a native API (TimeDB Call Interface, TDBCI) to allow the development of temporal applications based on TimeDB. Additionally, we plan to add query

optimisation (e. g. semantical query optimisation) to provide faster evaluation of ATSQL2 statements.

What is new in the beta version 4 of TimeDB 2.0

The following features were added to TimeDB 2.0 since release 2.0 Beta 1:

- Aggregate Functions
- GROUP BY clause[1]
- HAVING clause[1]
- Table and column constraints: Primary Key, Referential Integrity, Check
- TDBCI, a native call interface for ATSQL2 statements

What is missing in the beta version 4 of TimeDB 2.0

The **beta version 4** of TimeDB 2.0 does not support all of the features found in TimeDB 1.0. The **full version** of TimeDB 2.0, however, will subsume the functionality of TimeDB 1.0. The features missing in the beta version 4 are:

- No update operation (only *insert* and *delete*)
- No transaction time and bitemporal operations (snapshot, nonsequenced valid and valid time operations are supported)
- Only a single minimal calendar is supported

Differences to ATSQL2

The temporal SQL supported in TimeDB 2.0 is slightly different from ATSQL2 as it is proposed in [SBJS96a, SBJS96b]. In the beta version, interval expressions after timeflags (as shown in the example below) may only refer to constant values. References to timestamps of tables are not allowed.

```
validtime period [1980-1990) select ...
```

[1] GROUP BY and HAVING clause are not supported for Cloudscape's JBMS

Software Requirements

In order to run TimeDB, the following software is needed:

- Java Runtime Environment 1.1 (or newer)
- A DBMS, e.g. Oracle (Version 8), Sybase (Version 11.5) or Cloudscape's JBMS (Version 1.1 or newer)
- A JDBC driver for the DBMS

You also need a login and password for the database you will use, the JDBC driver name and the URL to connect to your database (this information should be provided in the documentation of the JDBC driver).

If you plan to develop applications which access temporal data via TimeDB, you also need a Java development kit (JDK).

Compatibility

Supported DBMS

While we developed TimeDB using the Oracle DBMS (Version 8), we also tested it on Sybase's DBMS (Adaptive Server Enterprise 11.5) and Cloudscape's JBMS (Version 1.1). We further plan to support DBMS such as

- Microsoft's Access
- SQL Server 7.0
- Informix

Other DBMS may be supported on demand.

Restrictions caused by underlying DBMS

Apart from the different data types supported in the different commercial DBMS, there is another restriction you should be aware of. Cloudscape's JBMS and Sybase do not support the non-temporal set operations *intersect* and *except*. These operations thus are not available in TimeDB, too, if it is used with one of these DBMS. However, you still can calculate *temporal* intersect and except operations.

Using the GUI of TimeDB 2.0

Installing TimeDB 2.0

If you would like to run TimeDB 2.0 using its graphical user interface (GUI), the first step is to set up the Java environment correctly. To the classpath of your Java environment add the path to the directory containing the *TimeDB* classes (e. g. C:\TimeDB2.0B4\classes) and the path to the classes containing the JDBC driver. Start TimeDB 2.0 using a command which looks like

java -classpath <set your classpath here> TimeDB

After a few seconds, the main window of TimeDB should open up (see *Figure 1 : Main Window*). The next step is to configure TimeDB. Select item *Preferences* in menu*TimeDB*. A new window opens where you can set the path to the TimeDB directory (*application dir*), the JDBC driver, the URL to your database and the DBMS you are using. The path to the TimeDB directory can be set by clicking on the corresponding text area which opens up a file selection dialog box. Select any file in the main directory of TimeDB. Write the name of the JDBC driver and the URL in the corresponding text areas and select the DBMS you are using. Press *Save* to save this data.

Now you can connect to your database account. Select *Open DB* in menu *TimeDB*. A window appears where you can enter your login and password (if there is one needed). Click the ok button, and after a short while the status information *Database opened* will be displayed in the result window.

The next step is to add the metadata needed by TimeDB to your database account. You have to select the *Create DB* menu item in menu*TimeDB* which starts to create the necessary tables and inserts metadata to your database.

If all of the above steps have been successfully completed, you can use TimeDB to store and query temporal data. The directory *demos* contains example queries and statements.

Using the Input Text Area

Temporal SQL statements can be written to the input text area in the main window of TimeDB. Note that in any case only the first statement will be executed. Each statement must end with a semicolon. If you would like to execute several statements at once, you can write them to a file and execute the file.
The first statement in the input text area can be executed by pushing button*Execute* in the button panel. Push *Clear* to clear the input text area. The results of your statement are displayed in the result window.

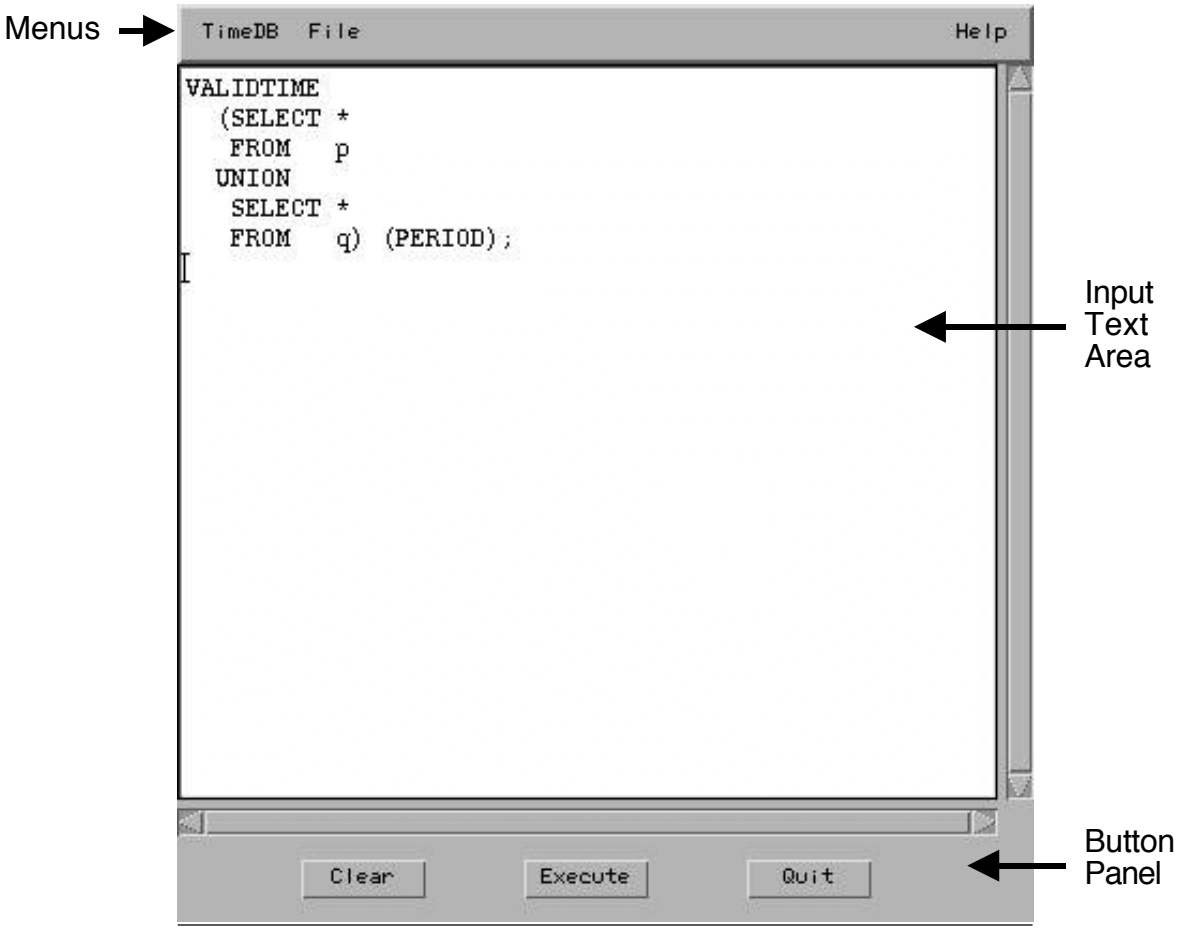


Figure 1 : Main Window of TimeDB

Executing a File

To execute temporal SQL statements stored in a file, choose item *Execute File* in menu *File*. A file selection dialog box opens up where you can select the file to be executed. The output will be written into the result window.

Using the API of TimeDB 2.0

In order to use TimeDB 2.0 to support temporal queries, temporal data definition and temporal data manipulation statements in your Java applications, you can access the temporal data via TimeDB's native call interface TDBCI. TDBCI is a simple way to execute any ATSQL2 statement via the TimeDB frontend on any of the supported commercial DBMS.
Note that if you use the API, a log file is written containing useful information in case the system does not properly run.

Setting up the environment

First, you have to add TimeDB's Java classes to the classpath which is used by your Java application. Add the path to the directory containing TimeDB's classes, e. g.

C:\TimeDB2.0B4\classes

and the path to the JDBC driver for the DBMS you use to the corresponding classpath.
The functionality supported by TDBCI is described in the following sections.

The TimeDB Call Interface (TDBCI)

In folder docu/Interface, you find the interface class which describes the TDBCI object of TimeDB 2.0.It contains 6 public instance methods you can use once you have created such an object in your Java application. These are

```
/* Setting the preferences : */
public boolean setPrefs(String Path, int DBMS,
                        String JDBCDriver, String URL);


/* Initialise DB with Metadata : */
public boolean createDB();
public boolean clearDB();


/* Open/Close Database : */
public boolean openDB(String Login, String Password);
public void closeDB();

/* Execute an ATSQL statement : */
public ResultSet execute(String stmt);
```

The first time you use TimeDB 2.0, you have to set the preferences and create the metadata in the database you will use. This needs to be done only once. To do this, you have to use the methods *setPrefs* and *createDB*.
For the preferences, you must specify four items: the path to the directory containing TimeDB, the type of DBMS you use, the JDBC driver and the URL. After that, you can access the database using the *openDB* method. Next thing to do is creating the metadata for TimeDB. Just call the method *createDB*. This automatically creates the necessary metadata tables and fills them with metadata.

Now you can execute ATSQL2 statements using the method *execute*. It returns an object of type ResultSet (note that TimeDB's class *ResultSet* does not correspond to class *ResultSet* implemented for JDBC drivers). In case you execute a query, the ResultSet object contains the rows of the result table. Otherwise it contains a message telling you that the statement has been properly execute or – in case of an error – returns the error message.

Method *clearDB* can be used to delete the metadata from your database. Method *closeDB* should be used to log out from the database.

The following section gives examples for the use of each of these methods.

An Example

In folder docu/Interface, you find the file Demo.java which contains examples of how you should use the TDBCI. We now shortly discuss the main steps of this demo file.

Creating a TDBCI Object

In order to use the TDBCI call interface in your applications, you first have to create an interface object. This is done the following way:

```
TDBCI t = new TDBCI();
```

Object t is the TDBCI object we will use in the following to access the database via TimeDB.

Setting the Preferences

The first time you use TimeDB 2.0, you have to set the preferences in order to tell TimeDB where it can find important files (path), which DBMS you are using (1=Oracle, 2=Sybase, 3=Cloudscape's JBMS), what the corresponding JDBC driver is (JDBCDriver) and where TimeDB can find the running DBMS instance (URL).

In our demo, we have installed TimeDB on a Windows NT platform in directory C:\TimeDB2.0B4\. We use an Oracle DBMS and the JDBC driver is oracle.jdbc.driver.OracleDriver. The port to access the DBMS instance is 1521 and the corresponding TNS name is ORCL. This leads to the following settings:

```
t.setPrefs(
   "C:\\TimeDB2.0B4\\",              // Path to TimeDB2.0 directory
   1,                                // Using Oracle DBMS
   "oracle.jdbc.driver.OracleDriver", // Oracle's JDBC driver
   "jdbc:oracle:thin:1521:ORCL"      // URL to access DBMS instance
)
```

Opening and closing a Database

The following statement opens the database for user *scott* (password is *tiger*):

```
t.openDB("scott", "tiger")
```

To close the database, simply execute the command

```
t.closeDB()
```

Creating the Metadata

The first time you use TimeDB for a specific database (e.g. scott's database), you have to create the necessary metadata by executing the following command:

```
t.createDB()
```

Once you want to delete the metadata again, simply execute the command

```
t.clearDB()
```

Queries and Query results

Executing queries against your temporal database consists of the following steps: execute the query and access the resulting table.
Executing a query (or any other statement) is done using the execute method of the TDBCI object:

```
ResultSet output = t.execute("VALIDTIME SELECT * FROM p;");
```

The result is returned in a ResultSet object.

Accessing the resulting table can be done in two ways: create a string representing the result in a tabular format, or iterate through the resulting rows and columns. To

create a string containing the data simply use the createString method of the ResultSet object:

```
output.createString()
```

Iterating through the result table needs more Java code to be written. Two loops – one going through the rows and one going through the corresponding columns – have to be implemented:

```
ResultRow row = output.firstRow(); // Get first row of result table

while (row != null) {
  int i = 0;
  String value = null;

  while ((value = row.getColumnValue(i)) != null) // Get next value in row
  {
    // Print value
    System.out.print(value);
    // Print type of value
    System.out.print(" [" + row.getColumnType(i) + "]  ");
    i++; // Next column
  }

  row = output.nextRow(row); // Get next row of result table
  System.out.println();
}
```

In the above Java code, the outer loop iterates through the rows of the result set. The loop variable *row* is initialized using method *firstRow*. The next row is accessed using method *nextRow*. In case of an error, method *firstRow* returns *null*. The corresponding error message can be accessed using method *createString* (see above).

The inner loop iterates through the columns of a specific row. Method *getColumnValue*(i) returns the value of column *i* (*i* >= 0). If the specified column is not available (*i* < 0 or *i* >= *row.getLength()*), method *getColumnValue* returns *null*. Method *getColumnType* returns the type of the column specified. The following column types are supported:

- number
- numeric
- longint
- date
- varchar
- smallint
- integer
- real
- period
- float
- double
- interval
- char

Note that for the time being, the returned values of method *getColumnValue* are always of Java type *String*, however.

The Temporal Aspects of TimeDB 2.0

Timeflag semantics

The language ATSQL2 distinguishes three different modes to evaluate an SQL statement: *snapshot* semantics, *sequenced* and *nonsequenced* semantics. Snapshot semantics means that only the database state valid at time instant *now* is evaluated. This corresponds to evaluating a non-temporal SQL statement over a non-temporal database containing data about the current state of the real world. In ATSQL2, a statement without a time flag has snapshot semantics.
Sequenced semantics means that an SQL statement is evaluated over all database states stored in the temporal database. A query with sequenced semantics thus returns temporal data. In ATSQL2, a sequenced valid-time statement starts with timeflag *validtime*.
Statements with nonsequenced semantics treat the timestamps just as any other user defined attribute. The algebra operations have non-temporal semantics. This allows the comparison of different database states with each other. In ATSQL2, a nonsequenced valid-time statement starts with timeflag *nonsequenced validtime.*

Table 1 gives an overview of the different timeflags together with the semantics of the corresponding statements.

Nesting of Timeflags

Usually, timeflags are propagated from the outside to the inside of nested queries. For example, in the query

```
validtime
  (select * from employees)
union
…
```

the timeflag *validtime* is propagated to the inner select statement.
Timeflags, however, may also be overwritten. In the query

```
validtime
  (nonsequenced validtime period [1980-1990)
   select * from employees)
union
…
```

the inner query has a different timeflag than the outer query. First, the inner select statement is evaluated using nonsequenced semantics. Due to the interval

expression in the timeflag, it returns a valid-time table. The outer query then calculates the valid-time union of this table and the rest of the outer query.

Timeflag	Semantics
No flag	***Snapshot semantics*** Algebra operations have non-temporal semantics. Queries only refer to the currently valid database state and return non-temporal tables. Modification statements only refer to the currently valid database state.
nonsequenced validtime	***Nonsequenced semantics*** Algebra operations have non-temporal semantics. Queries refer to all database states and return non-temporal tables. Modification statements do not interpret timestamps.
nonsequenced validtime <interval exp>	***Nonsequenced semantics*** Algebra operations have non-temporal semantics. Queries refer to all database states and return valid-time tables where each tuple's valid-time corresponds to <interval exp>. Modification statements do not interpret timestamps and set timestamps of modified tuples to <interval exp>.
validtime	***Sequenced semantics*** Algebra operations have temporal semantics. Queries refer to all database states and return valid-time tables. Modification statements update each database state separately.
validtime <interval exp>	***Sequenced semantics*** Algebra operations have temporal semantics. Queries refer to the database states valid during <interval exp> and return valid-time tables. Modification statements update each database state during <interval exp>.

Table 1 : Timeflags

Temporal expressions and comparison operations

TimeDB supports spans (a duration of time, e. g. two years and one month), events (a time instant, e. g. June 12, 1964) and time intervals (e. g. from 1980 to 1990). Spans, events and time intervals are treated just as any other values such as strings, integers etc. and thus may appear anywhere in select and where clauses where expressions are allowed.

According to the syntax given at the end of this document, a legal time span - specified as a constant value - is, for example,

interval 2 year 1 month.

Additionally, values of type span stored in tables may be referenced. Last but not least, new spans may be calculated using the operators +, -, * and /. Allowed are the following combinations:

span	+	span	-> span
span	-	span	-> span
number	*	span	-> span
span	/	number	-> span

Spans may be compared with other spans using the comparison operations =, <, >, <=, >= and <>.

The expressions

date '1964-06-12',
timestamp '1964-06-12 12:30:24' and
date 1964/6/12~12:30:24

are legal event values. While the first two correspond to the SQL standard, the third is used for output of event values and may also be used for input. It is special in the sense that only the significant part of an event is displayed. For example, 1964 actually is shorthand for 1964/1/1~00:00:00.
New events may be calculated by adding or subtracting a time span :

event	+	span	-> event
event	-	span	-> event

Events may be compared with each other using the comparison operations *precedes* and =.

The constant

period [1980-1990)

is a legal time interval. TimeDB displays time intervals as [1980-1990). Time intervals are closed on the lower and open on the upper bound. Time intervals

may be compared either with other time intervals using the comparison operations *precedes*, *overlaps*, *meets*, *contains* and =, or they may be compared with events. In the latter case, the following combinations are supported:

interval	*contains*	event	-> boolean
interval	*precedes*	event	-> boolean
event	*precedes*	interval	-> boolean

Calendar

TimeDB 2.0 supports a simple minimal calendar. The calendar starts with year 1. Each month has 30 days and each day 24 hours (0 to 23). Expressions calculating new events may lead to illegal values which are represented as << NAD >> (not a date). The smallest non-decomposable time unit is a second.

A. Literature

[HCF97] G. Hamilton, R. Cattell, M. Fisher : JDBC Database Access with Java. Addison Wesley. July 1997.

[SBJS96a] R. Snodgrass, M. Böhlen, C. Jensen, A. Steiner : Adding Valid Time to SQL/Temporal (Change Proposal). ANSI X3H2-96-501r2, ISO/IEC JTC1/SC 21/WG 3 DBL-MAD-146r2. November 1996.

[SBJS96b] R. Snodgrass, M. Böhlen, C. Jensen, A. Steiner : Adding Transaction Time to SQL/Temporal (Change Proposal). ANSI X3H2-96-502r2, ISO/IEC JTC1/SC 21/WG 3 DBL-MAD-147r2. November 1996.

[SBJS98] R. Snodgrass, M. Böhlen, C. Jensen, A. Steiner : Transitioning Temporal Support in TSQL2 to SQL3. In *Temporal Databases : Research and Practice*. O. Etzion, S. Jajodia and S. Sripada, editors. LNCS 1399, Springer Verlag. March 1998.

[S98] Andreas Steiner : A Generalisation Approach to Temporal Data Models and their Implementations. Ph. D. Thesis, ETH Zürich. November 1997.

B. Grammar

The following syntax defines the legal temporal SQL statements supported in TimeDB 2.0 Beta 4:

```
statement ::= (query | ddl | dml | control) ';'
```

Query

```
timeFlag ::= [ 'nonsequenced' ] 'validtime' [ scalarExp ]

coal ::= '(' 'period' ')'


query       ::= [ timeFlag ] queryExp
queryExp    ::= queryTerm { ('union' | 'except') queryTerm }
queryTerm   ::= queryFactor { 'intersect' queryFactor }
queryFactor ::= '(' query ')' [ coal ] | sfw

sfw         ::= 'select' selectItemList
                'from' tableRefList
                [ 'where' condExp ]
                [ 'group' 'by' groupByList ]  /* Not supported for JBMS */
                [ 'having' condExp ]          /* Not supported for JBMS */


selectItemList ::= '*' | selectItem { ',' selectItem }
selectItem     ::= scalarExp [ alias ]

tableRefList ::= tableRef { ',' tableRef }
tableRef     ::= '(' query ')' [ coal ] alias [ colList ] |
                 identifier [ coal ] [ alias ]

alias ::= ['as'] identifier

condExp          ::= condTerm { 'or' condTerm }
condTerm         ::= condFactor { 'and' condFactor }
condFactor       ::= [ 'not' ] simpleCondFactor
simpleCondFactor ::=
     '(' condExp ')'                                          |
     'exists' '(' query ')'                                   |
     scalarExp condOp scalarExp                               |
     scalarExp condOp ('all' | 'any' | 'some') '(' query ')'  |
     scalarExp [ 'not' ] 'between' scalarExp 'and' scalarExp  |
     scalarExp [ 'not' ] 'in' '(' query ')

condOp ::= '<' | '>' | '<=' | '>=' | '<>' | '=' |
           'precedes' | 'overlaps' | 'meets' | 'contains'

groupByList ::= colRef { ',' colRef }

scalarExp    ::= term { ('+' | '-') term }
term         ::= factor { ('*' | '/') factor }
factor       ::= [ ('+' | '-') ] simpleFactor
```

```
simpleFactor ::= colRef                  |
                 const                   |
                 '(' scalarExp ')'       |
                 'abs' '(' scalarExp ')'

colRef ::= identifier [ '.' identifier ]

const ::= integer        |
          float          |
          ''' string ''' |
          interval       |
          event          |
          span


interval    ::= 'validtime' '(' identifier ')' |
                'period' intervalExp |
                'period' '(' scalarExp ',' scalarExp ')'
intervalExp ::= '[' time '-' time ')'
time        ::= timeDBDate | eventExp


event    ::= ( 'begin' | 'end' ) '(' scalarExp ')'               |
             ( 'first' | 'last' ) '(' scalarExp ',' scalarExp ')' |
             eventExp

eventExp ::= 'now'                       |
             'beginning'                 |
             'forever'                   |
             'date' dateString           |
             'date' timeDBDate           |
             'timestamp' timestampString

dateString      ::= ''' YYYY '-' MM '-' DD '''
timestampString ::=  ''' YYYY '-' MM '-' DD ' ' HH ':' MM ':' SS '''
timeDBDate      ::= YYYY [ '/' MM [ '/' DD
                    [ '~' HH [ ':' MM [ ':' SS ]]]]]


span      ::= 'interval' spanExp
spanExp   ::= integer qualifier { integer qualifier }
qualifier ::= 'year'   |
              'month'  |
              'day'    |
              'hour'   |
              'minute' |
              'second'
```

Data Definition

```
ddl ::=  ddlTable | ddlView | 'drop' 'table' | 'drop' 'view'

ddlTable  ::= 'create' 'table' identifier ( tableDef | ddlQuery )
ddlView  ::= 'create' 'view' identifier ddlQuery

tableDef ::= '(' colDefList ')' [ 'as' 'validtime' ]
ddlQuery ::= [ '(' colList ')' ] 'as' query
```

```
colDefList ::= colDef { ',' (colDef | tableConstraint) }
colDef     ::= identifier dataType [ columnConstraint ]

columnConstraint ::= ['constraint' identifier]
                     [ 'validtime' ] (primKeyCol   |
                                      refIntegrity |
                                      checkConstraint)
tableConstraint  ::= [ 'validtime' ] (primKeyTab |
                                      foreignKey |
                                      checkConstraint)

primKeyCol ::= 'primary' 'key'
primKeyTab ::= 'primary' 'key' '(' colList ')'

refIntegrity ::= 'references' identifier '(' identifier ')'
foreignKey ::= 'foreign' 'key' '(' colList ')'
               'references' identifier '(' colList ')'

checkConstraint ::= 'check' '(' condExp ')'

colList ::= col { ',' col }
col     ::= identifier

dataType ::= 'number'  [ typeLength ] |  /* Oracle */
             'numeric' [ typeLength ] |  /* Sybase */
             'smallint'               |  /* Cloudscape's JBMS */
             'longint'                |  /* Cloudscape's JBMS */
             'integer'                |
             'real'                   |
             'float'                  |
             'interval'               |
             'date'                   |
             'period'                 |
             'char'    [ typeLength ] |
             'varchar' [ typeLength ]

typeLength ::= '(' integer ')'
```

Data Manipulation

```
dml ::= [ timeFlag ] ( insert | delete )

insert ::= 'insert' 'into' identifier valExp
valExp ::= 'values' '(' valList ')' | query

delete ::= 'delete' 'from' identifier [ 'where' condExp ]
```

Control

```
control ::= 'commit' | 'rollback'
```

C. TimeDB's API

The interface class of the TimeDB Call Interface (TDBCI) is:

```
public interface TDBCInterface {

  /*****************************************************/
  /************** PUBLIC INSTANCE METHODS *************/
  /*****************************************************/

  /* Set TimeDB Preferences */
  public boolean setPrefs(String Path, int DBMS,
                          String JDBCDriver, String URL);
  /* Oracle            : DBMS = 1 */
  /* Sybase            : DBMS = 2 */
  /* Cloudscape's JBMS : DBMS = 3 */


  /* Initialise DB with Metadata : */
  public boolean createDB();
  public boolean clearDB();


  /* Open/Close Database : */
  public boolean openDB(String Login, String Password);
  public void closeDB();


  /* Execute an ATSQL statement : */
  public ResultSet execute(String stmt);

}
```

The interface class of class ResultSet is:

```
public interface ResultSetInterface {

  /*****************************************************/
  /************** PUBLIC INSTANCE METHODS **************/
  /*****************************************************/

  public ResultRow firstRow();

  public ResultRow nextRow(ResultRow row);

  public String createString();

}
```

The interface class of class ResultRow is:

```
public interface ResultRowInterface {

  /*****************************************************/
  /************** PUBLIC INSTANCE METHODS **************/
  /*****************************************************/

  public int getLength(); // Get number of columns in row

  public String getColumnValue(int col);
  // Returns null if col < 0 or col >= length

  public String getColumnType(int col);
  // Returns null if col < 0 or col >= length

}
```