Information Management

IBM

Informix. software

TimeSeries and Real Time Loader

Key Points

- Describe two complex DataBlades.
  - TimeSeries
  - Real Time Loader
- Why time series data can be a problem.
- How the TimeSeries Datablade solves these problems.
- Applications.
- Why streaming data can be a problem.
- How the RealTime Loader solves these problems.

Timeseries

- TimeSeries:
  - What is a time series ?
    - Regular
    - Irregular
  - Where they are used ?
  - What they are used for ?
  - How they are used ?

Regular Time Series Data

- Timestamps have a regularly repeating pattern of intervals daily, hourly, etc...
- Might be breaks in the pattern:
  - A work week is data captured for 5 days in a row then nothing for 2 days.
- Only one piece of data per interval:
  - If an interval has not been inserted into then it has the value of NULL
  - Intervals not inserted into occupy some space
    - Heuristic: intervals not inserted into at the end of series do not take up space.
- Can be thought of as an array.
- Optimized to return data at offset, not timestamp.

Irregular Time Series Data

- Data in an irregular time series does not have a regularly repeating pattern of intervals:
  - Any interval may have zero or more pieces of data.
- Missing data takes no space on disk:
  - There really is no concept of missing data.
- Only efficient way to access data is by timestamp:
  - You can retrieve the Nth piece of data, but the code does a linear search.
- Data can be stair stepped:
  - Value persists until next value arrives – for example stock prices.
- Data can be discrete points:
  - Value is valid only at the given time – for example heart beats.

The Timeseries "Problem"

- Timeseries are usually stored as "tall – thin" tables with a very large number of rows.
- Need fast access to an ordered set of rows.
- May need one index to enforce uniqueness and another for index only read, more space used for index than data.
- Can be difficult to write SQL to work with the data.

Timeseries in Traditional RDBMS

- Irregular time series of stock "ticks".
- Typical volumes are 50M per day and increasing.
- Could access by "Stock" and "Timestamp" but often need an index on all columns for index only read.

Stock	Timestamp	Price	Volume
ABC	2006-01-01 09:30:00.00000	12.34	100
XYZ	2006-01-01 09:30:00.01000	9.34	1000
ABC	2006-01-01 09:30:00.19000	12.44	10
KKK	2006-01-01 09:30:01.05000	194.00	10000
ABC	2006-01-01 09:30:01.70000	12.54	250

Data	35 Mb
Index on stock + timestamp	28 Mb
Index on all columns	43 Mb

Timeseries in Traditional RDBMS

- Can be difficult to write SQL to process tick data:
  - Applying stock splits.

```
select t.tstamp, t.price * s.fac, t.vol/ s.fact
    from tick t, split s
    where t.stock = s.stock
    and t.tstamp::datetime year to day = s.tstamp
    and t.stock = "ABC"
    and t.tstamp::date between
        ('2006-01-01' and '2006-01-05')
```

  - Calculating Volume Weighted Average Price (VWAP)

```
select stock, ststamp::datetime year to hour,
              sum(price * vol) / sum(vol)
    from tick
    where stock = "ABC"
    and tstamp::date between
        ('2006-01-01' and '2006-01-05')
    group by 1, 2
```

Datablade Solution (1 of 7)

- Store timeseries elements as an ordered set of elements:
  - Uses less space because the "key" is factored out and the time can be represented in 4 bytes (regular).
  - Access is as fast (or faster) than index only read but no additional index.
  - SQL can be made much simpler.

Datablade Solution (2 of 7)

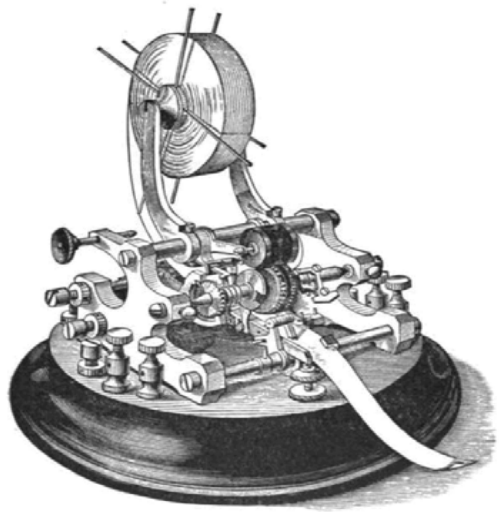
- Store timeseries data as:
  - Start time.
  - Calendar information.
  - Ordered set of elements.

Stock	Ticks
ABC	2006-01-01, daily, {(12.34,1234567), (12.56,950000), (12.34,5555567),..}
KKK	2006-05-05, daily, {(199.08,678900), (198.55,345000), (198.12,850250),..}
XYZ	2003-09-01, daily, {(9.34,890567), (9.56,989000), (9.40,1000780),..}

Datablade Solution (3 of 7)

- How a timeseries is stored:
  - Start time and calendar name.
  - Index (btree) of the first element in each page.
  - Each page contains all the elements between 2 timestamps.

Timeseries	30 Mb
Data	35 Mb
Index on stock + timestamp	28 Mb
Index on all columns	43 Mb



- Traditional methods use 2 to 3 times more space or even more depending on the number of NULL values.

Datablade Solution (4 of 7)

- Performance:
  - Slower for singleton selects.
  - Faster accessing sets of data.
  - Much faster combining timeseries.

	Singleton	1,000 consec.	Aggregate 80,000	1,000 consec. f(t1, t2)
Timeseries	1	1	1	1
Traditional (1)	.5	2	3	5
Traditional (2)	2	400	1000	650

- Traditional(2) = index on key columns.
- Traditional(1) = index on key and all columns.

Datablade Solution (5 of 7)

- Much simpler SQL, consider the "stock split"/calibration problem:

```
select t.tstamp, t.price * s.fac, t.vol/ s.fact
    from tick t, split s
    where t.stock = s.stock
    and t.tstamp::datetime year to day = s.tstamp
    and t.stock = "ABC"
    and t.tstamp::date between ('2006-01-01' and '2006-01-05')
```

```
select func(ticks, splits, '2006-01-01', '2006-01-05')
    from tick
    where stock = "ABC"
```

Datablade Solution (6 of 7)

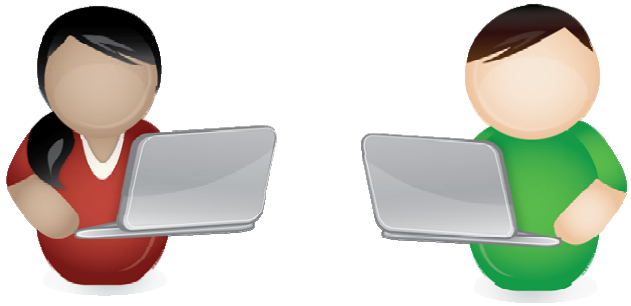
- Much simpler SQL, the VWAP problem:

```
select stock, tstamp::datetime year to hour, sum(price * vol) / sum(vol)
    from tick
    where stock = "ABC"
    and tstamp::date between ('2006-01-01' and '2006-01-05')
    group by 1, 2
```

```
select stock, vwap(ticks, '2006-01-01', '2006-01-05', '01:00:00')
    from tick
    where stock = "ABC"
```

Datablade Solution (7 of 7)

- Other Timeseries Datablade features:
  - Calendar support.
  - VTI to make time series look like traditional tables.
  - C API.
  - Java Class Library.
- Easy to combine with own or 3rd party code.



Datablade Practicalities (1 of 7)

- In order of operations:
- How do I add a Datablade to a database ?

On the command line:

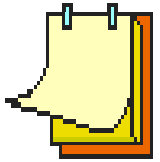
```
blademgr
register TimeSeries.4.10.UC7 in stocks
```

or via dbaccess:

```
execute function
  sysbldprepare('timeseries.4.01.UC7','create');
```

Datablade Practicalities (2 of 7)

- Calendar Patterns – stored in the CalendarPatterns table
  - All time series are associated with these.
  - Used to bound the times at which timeseries data is recorded:
    - Data recorded during 'on' times.
    - Data not recorded during 'off' times.
    - Data recorded for the interval specified within the bounds.
    - Within the calendar definition.



- A Monday to Friday 5 day work week can be represented as 1 day off (Sunday), 5 days on, 1 day off (Saturday):
  - **Insert into CalendarPatterns values ('Five_Day_Work_Week', '{1 off, 5 on, 1 off}, day');**
  - Note the brackets '{ }' above.
  - The trailing day is an interval for a single unit of time.

Datablade Practicalities (3 of 7)

- How do I create a data type that defines each element in a time series ?
  - Must first create a row type whose first column timestamp must be a **datetime year to fraction(5)** value:

```
create row type stockVals (
    tstamp      datetime year to fraction(5),
    price       decimal(16,2),
    high        decimal(16,2),
    low          decimal(16,2),
    median      decimal(16,2),
    volume      integer,
    trades      integer,
    ….
);
```

  - Each timestamp must be unique.
  - Row types cannot contain types:
    - blob, clob, text, serial

Datablade Practicalities (4 of 7)

- Once a calendar pattern is established, create the calendar to use it in the table CalendarTable:

```
insert into CalendarTable ( c_name, c_calendar)
values ('Trading Week','startdate(2010-01-01 00:00:00.00000),
pattstart(2010-01-07 00:00:00.00000),
pattname(Five_Day_Work_Week)');
```

- Regular timeseries calendars are used to guide the mapping between the timestamp and the offset
  - Represented by Calendar Data Type, consisting of:
    - Start date of the calendar.
    - Calendar repeating pattern during the duration of the data type for the valid and invalid recording of the data.
    - Interval – calibration of the calendar pattern.
    - Starting date of the calendar pattern
      - Must be >= Calendar start date and
      - Less than one calendar pattern length after calendar start.

Datablade Practicalities (5 of 7)

- Containers are required data structures that hold data for one or more time series.
- You cannot mix data for regular and irregular time series in the same container.
- How do I create a container to store a time series ?
  - **execute procedure TSContainerCreate("ts_ctrname","dbspace1","stockVals",40000,40000);**
- A container is made up of index pages and data pages.
- Containers allow time series data to be spread onto many disk partitions.

Datablade Practicalities (6 of 7)

- How do I create a table that includes a time series ?
  -
```
create table dailyStocks (
      stockId             integer,
      abbrev              char(4),
      stockData           timeseries(stockVals));
```

- How do I create a time series record ?
  -
```
insert into dailyStocks values (901, 'IBM',
TSCReate('Trading Week','2010-01-03 00:00:00.00000',
20, 0, 0, 'stockCon'));
```

Datablade Practicalities (7 of 7)

- How do I add data to a time series ?
  -
```
update dailyStocks
  set stockData = PutElem(stockData,
  row(NULL::datetime year to fraction(5), 3.3, 4.4,
  2.2, 3.0, 123456, 789, NULL)::stockData)
  where stock = 'XYZ';
```
- How do I query a row with a time series ?
  -
```
select clip(stockData, current – interval(1) month
to month, NULL) from stockData;
```
- Get all of IBM since the beginning of the 2001
  -
```
select clip(series, '2001-01-01 00:00:00.00000',
Current) from S_P_500 where name = 'IBM';
```

Building Applications with Time Series

- Several interfaces are available:
  - SQL
  - VTI
  - SPL
  - Java
  - C-API

- It's a toolkit approach!
- Allow people to build their analytics.

Time Series SQL Interface

- Time series data is usually accessed through user defined routines (UDR's) from SQL, some of these are:
  - Clip() - clip a range of a time series and return it.
  - LastElem(), FirstElem() - return the last (first) element in the time series.
  - Apply() - apply a predicate and expression to a range of elements in a time series.
  - AggregateBy() - change the interval of a time series using a aggregate function.
  - SetContainerName() - move a time series from one container to another.
  - BulkLoad() - load data into a time series from a file.
- See Appendix A for a list of all of SQL Interface UDR's.

Time Series VTI Interface

- Makes time series data look like standard relational data:
  - Useful for programs that can't handle objects.
  - Useful when application connects with ODBC.
- There is a small (10%) penalty for using VTI.
- Restrictions:
  - A VTI table can only reference one time series column from the base table.
  - No secondary indices are allowed.
- SQL to create a VTI table:
  - *execute procedure tscreatevirtualtab('t_vti', 't');*

Stored Procedure (SPL) Example

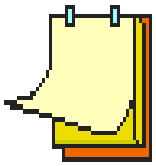
```
-- count non-null elements in a time series
create function spl_test(arg lvarchar) returning integer
    define var daybar;
    define cnt integer;
    let cnt = 0;
    foreach execute function
            transpose((select series from S_P_500 where name = arg))
      into var
          let cnt = cnt + 1;
    end foreach
    return cnt;
end function;
```

Time Series C-API Interface

- Client and server versions of the API.

- Treats a time series like a table (sort of).
- Functions to:
  - Open and close a time series.
  - Scan a time series between 2 timestamps.
  - Create a time series.
  - Retrieve, insert, delete, update.
- Plus another 70 functions defined.
- See Appendix C for a list of these.

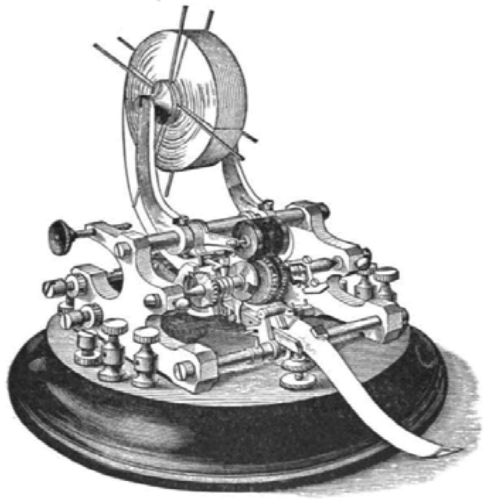
Timeseries Application



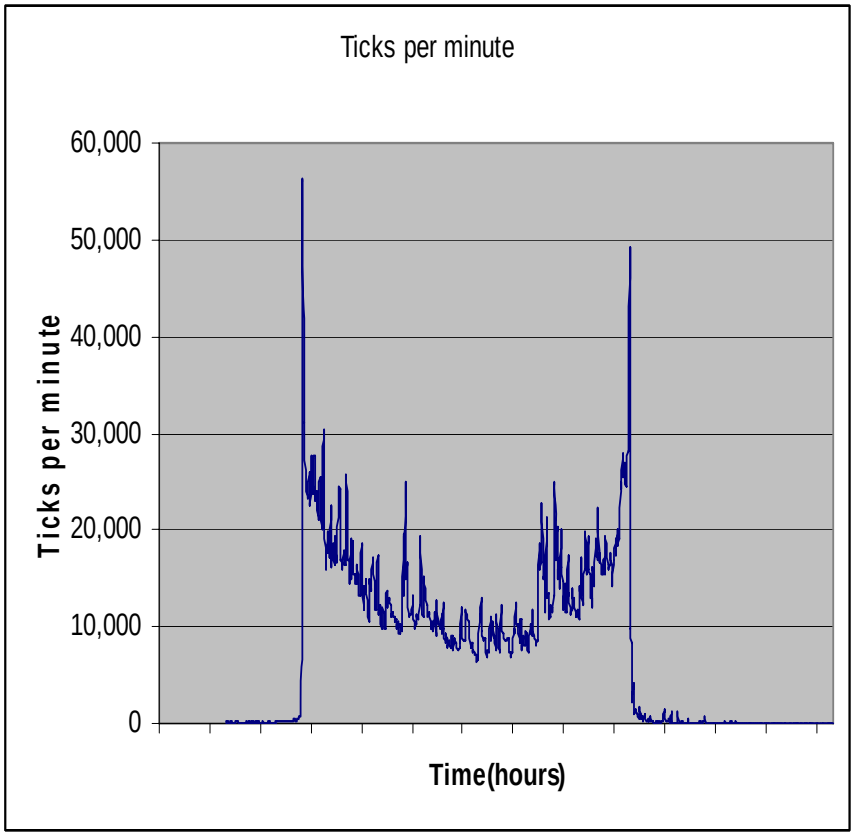
- Existing applications:
  - Finance (Capital Markets).
  - Some in Manufacturing and Environmental.
  - Smart Electrical Grid Usage
- Possible applications:
  - Telco.
    - Real Time Cell Phone Tower Data
    - Sales/sales analysis.
  - Energy
    - Oil Field Geology data.
  - Water/Sewer System Usage
  - Satellite Telemetry
  - Assembly Line Q/A & Analysis
  - Your application ?

RealTime Loader (1)

- Next problem after time series "problem" solved.
- Large number of ticks generated at market open/close and in reaction to events.
- Money is made at these times.
- Need to make the data available as soon as possible.



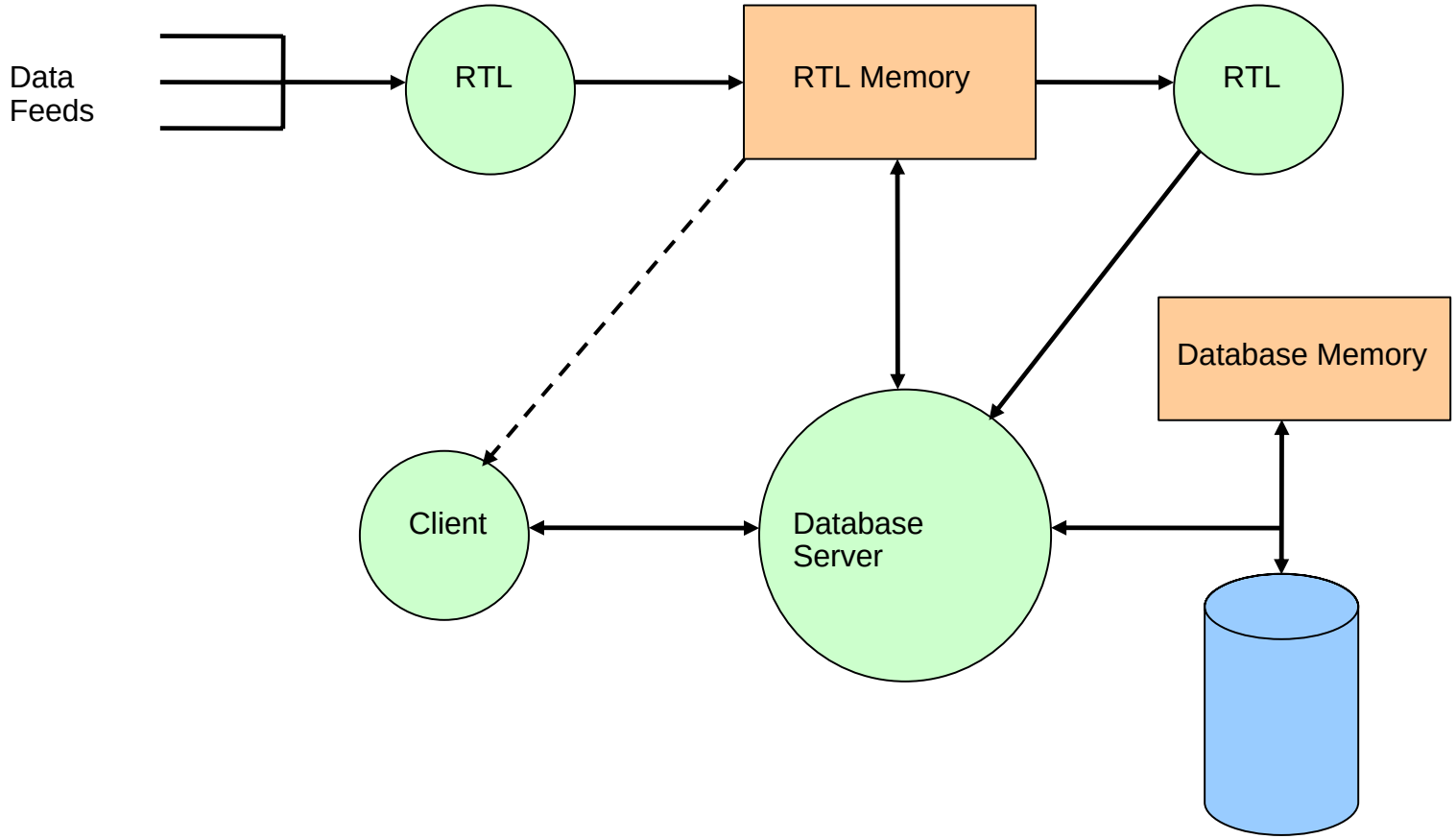
Edison Stock Ticker - 1892



RealTime Loader (2)

- Traditionally data can be loaded very quickly by loading it in batches. This causes problems:
  - It takes time for the data to become available.
  - The table can be locked while the data is loaded.
- The RealTime Loader solves this problem by holding the data in memory and making it available to the SQL layer as if it was already in the database:
  - Allows real-time analysis of data as it come thru the feeds
    - Uncommitted and committed data reads are possible.

RealTime Loader Architecture



RealTime Loader Performance (1 of 3)

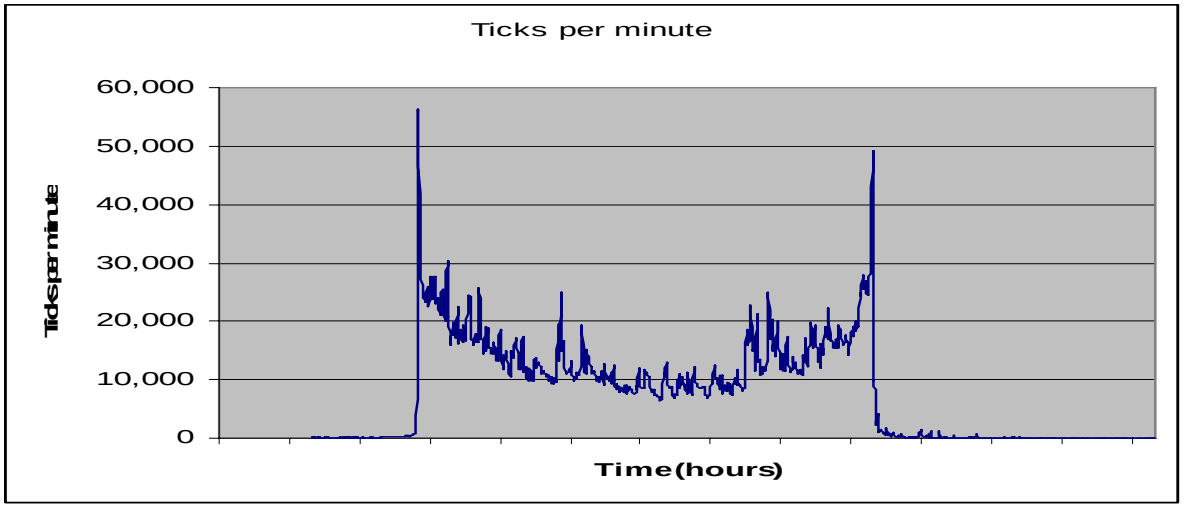
- Load performance of traditional systems depends on the system but rates of 1,000s of ticks per second are reasonable but there is a high latency as the data is batched and loaded.
- With the RTL the data is made accessible within a few ms. There are 2 load rates to consider:
  - How fast the data can be added to shared memory.
  - How fast the data can be loaded into the database.

RealTime Loader Performance (2 of 3)

- How fast can the data be added to shared memory ?
  - Rates of 10,000s per cpu sec are common.
- How fast can the data can be loaded into the database ?
  - This is roughly the same as the traditional RDBS solution.
- The system can run at the "top" rate until the shared memory allocated to RTL is full.

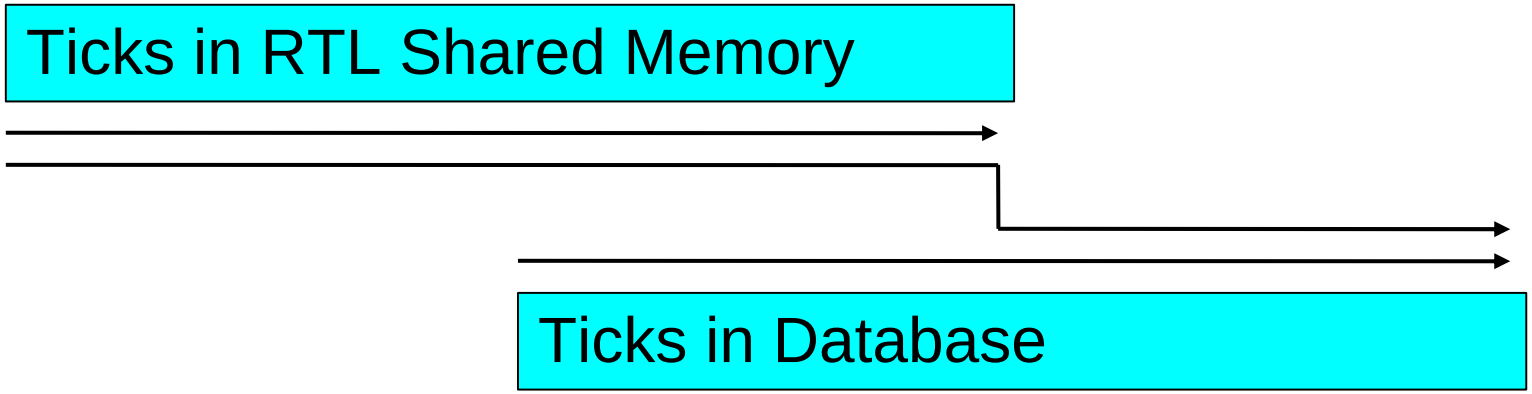
RealTime Loader Performance (3 of 3)

- The peaks don't last very long, the system can be sized for something just above the average rate.
- Traditional systems have to be sized to handle the peak rate or the user has to accept a high latency during the crucial market open/close periods.



RTL Data Access

- Access can be through the usual SQL layer.
- Even faster access through an API that gets the data directly from the shared memory segment.
- The SQL layer can see all the ticks, or just those in RTL, or just those on the database.



RealTime Loader Analytics

- The RealTime Loader can do some simple aggregation and statistics on the incoming data making this available to SQL or through the API:
  - VWAP - Volume Weighted Moving Average Price
    - = total amount spent on a security / total number of shares traded.
    - This needs all of the share purchase history stored unbounded by time.
  - EWAP – Exponentially Weighted Moving Average Price
    - A share price moving average where previous moving average data is considered but dampened by the effect of the new, more recent moving average.
    - Only needs 3 pieces of data:
      - the old and new MAP data.
      - plus the damping factor.
  - max, min etc.

RealTime Loader Applications

- Stock ticker trades data analysis.
- Gaming
- Telco:
  - Cell Phone Tower Call Analysis Apps.
  - Cell Phone Account Apps.
- 24 x 7 Satellite Telemetry Data Applications:
  - Millions of records per minute .....
- Geological Analysis Applications:
  - Oil Exploration Data.
- Land Resource Analysis Applications.
- Water/Sewer Flow Data

Conclusion

- The TimeSeries DataBlade allows time series data to be:
  - Efficiently stored.
  - Quickly accessed.
  - Conveniently queried.
- The RealTimeLoader allows time series data to be:
  - Quickly stored.
  - Accessed by applications with the minimum of delay.

Logo

Information Management

IBM

Informix. software

Appendix A - Time Series Built In Routines

Appendix – Time Series Built In Routines (1 of 9)

Task Type	Description	Routine Name
Get info from a timeseries.	Get the origin.	**GetOrigin**
	Get the interval.	**GetInterval**
	Get the calendar.	**GetCalendar**
	Get the calendar name.	**GetCalendarName**
	Get the container name.	**GetContainerName**
	Get the user defined metadata.	**GetMetaData**
	Get the metadata type.	**GetMetaTypeName**
	Determine whether a time series is regular.	**IsRegular**
	Get the instance Id if the timeseries is stored in a container.	**InstanceId**
Convert between a timestamp and an offset.	Return the offset, given the timestamp.	**GetIndex**
	Return the timestamp, given the offset.	**GetStamp**
Count the number of elements.	Return the number of elements.	**GetNelems**
	Get the number of elements between two timestamps	**ClipGetCount**

Appendix - Time Series Built in Routines (2 of 9)

Select Individual elements.	Get the element associated with a timestamp.	**GetElem**
	Get the element at or before a timestamp.	**GetLastValid**
	Get the element after a timestamp.	**GetNextValid**
	Get the element before a timestamp.	**GetPreviousValid**
	Get the element at a specified position.	**GetNthvalid**
	Get the first element.	**GetFirstElem**
	Get the last element.	**GetLastElem**
	Get the last non null element.	**GetLastNonNull**
	Get the next non null element.	**GetNextNonNull**

Appendix - Time Series Built in Routines (3 of 9)

Modify elements or a set of elements.	Add or update a single element.	**PutElem**
	Add or update a single element.	**PutElemNoDups**
	Add or update a single element at a given offset.	**PutNthElem**
	Add or update an entire set.	**PutSet**
	Delete an element at a given timepoint.	**DelElem**
	Delete all elements in a specified time range.	**DelClip**
	Delete all elements in a specified time range.	**DelTrim**
	Insert an element.	**InsElem**
	Insert a set.	**InsSet**
	Update an element.	**UpdElem**
	Update a set.	**UpdSet**
	Put every element of one time series into another time series.	**PutTimeSeries**
Modify metadata.	Update user-defined metadata.	**UpdMetadata**

Appendix - Time Series Built in Routines (4 of 9)

Make Elements visible or invisible to a scan.	Make an element invisible.	**HideElem**
	Make a range of elements invisible.	**HideRange**
	Make an element invisible.	**RevealElem**
	Make a range of elements visible.	**RevealRange**
Check for null or hidden elements.	Determine if an element is hidden.	**ElemIsHidden**
	Determine if an element is null.	**ElemIsNull**
Extract and use part of a time series.	Extract a period between two timestamps to a set of values and run an expression or function on every entry.	**Apply**
	Extract data between two timepoints.	**Clip**
	Clip a certain number of elements.	**ClipCount**
	Extract a period that includes a given time.	**WithinC**
	Extract a period starting or ending at a given time.	**WithinR**
Apply a new calendar to a time series.	Apply a calendar.	**ApplyCalendar**

Appendix - Time Series Built in Routines (5 of 9)

Create and load a time series.	Load data from a client file.	**BulkLoad**
	Create a regular empty time, a regular populated time series, or a regular time series with metadata.	**TsCreate**
	Create an irregular empty time, an irregular populated time series, or an irregular time series with metadata.	**TsCreateIrr**
Find the intersection or union of time series.	Build the intersection of multiple time series and optionally clip the result.	**Intersect**
	Build the union of multiple time series and optionally clip the result.	**Union**
Iterator functions.	Convert time series data to tabular form.	**Transpose**
Aggregate Functions.	Return a list (collection of rows) containing all elements in a time series.	**TSSetToList**
	Return a list of columns in a timeseries.	**TSColNameToList**
	Return a list of columns in a timeseries.	**TSColNumToList**
	Return a list containing the columns of a time series plus non time series columns.	**TSRowToList**
	Return a list containing the columns of a time series plus non time series columns.	**TsRowNameToList**
	Return a list containing the columns of a time series plus non time series columns.	**TsRowNumToList**

Appendix - Time Series Built in Routines (6 of 9)

Used within the apply function to perform statistical calculations on a time series.	Perform a sum over a time series type.	**Sum**
	Sum SMALLFLOAT or DOUBLE PRECISION values.	**TSAddPrevious**
	Compute the decay function.	**TSDecay**
	Compute a running average over a specified number of values.	**TSRunningArg**
	Compute a running correlation between two time series over a specified number of values.	**TsRunningCor**
	Compute a running median over a specified number of values.	**TsRunningMed**
	Compute a running sum over a specified number of values.	**TSRunningSum**
	Compute a running variance over a specified number of values.	**TSRunningVar**
	Compare SMALLFLOAT or DOUBLE PRECISION values.	**TSCmp**
	Return a previously saved value.	**TSPrevious**

Appendix - Time Series Built in Routines (7 of 9)

Perform an arithmetic operation on one or two time series.	Add two time series together.	**Plus**
	Subtract one time series from another.	**Minus**
	Multiply one time series by another.	**Times**
	Divide one time series by another.	**Divide**
	Raise the first argument to the power of the second.	**Pow**
	Get the absolute value.	**Abs**
	Exponentiate the time series.	**Exp**
	Get the natural logarithm of a time series.	**Logn**
	Get the modulus or remainder of a division of one time series by another.	**Mod**
	Return the argument, is bound to the unary + operator.	**Negate**
	Round the time series to the nearest whole number.	**Positive**

Appendix - Time Series Built in Routines (8 of 9)

Perform an arithmetic operation on one or two time series (cont'd).	Round the time series to the nearest whole number.	**Round**
	Get the square root of the time series.	**Sqrt**
	Get the cosine of the time series.	**Cos**
	Get the sine of the time series.	**Sin**
	Get the tangent of the time series.	**Tan**
	Get the arc cosine of the time series.	**Acos**
	Get the arc sine of the time series.	**Asin**
	Get the arc tangent of the time series.	**Atan**
	Get the arc tangent for two time series.	**Atan2**
	Apply a binary function to a pair of time series, or to a time series and a compatible row type or number.	**ApplyBinaryTsOp**
	Apply a unary function to a time series.	**ApplyUnaryTsOp**
	Apply another function to a set of time series.	**ApplyOpTsSet**
Reset the origin.	Reset the origin.	**SetOrigin**

Appendix - Time Series Built in Routines (9 of 9)

Aggregate values in a time series.	Aggregate values in a time series.	**AggregateBy**
	Aggregate values in a time series over a specified time range.	**AggregateRange**
Create a time series that lags.	Create a time series that lags the source time series by a given offset.	**Lag (regular only)**
Manage containers.	Create a container.	**TsContainerCreate**
	Destroy a container.	**TsContainerDestroy**
	Set the container name.	**SetContainerName**

- Following routines are used only with regular time series:
  - Lag
  - PutNthElem
  - TSCreate
- The TSCreateIrr function is only used with regular time series.

Information Management

IBM

Informix.
software

Appendix B – Calendar Pattern Routines

Appendix - Calendar Pattern Routines

Return the intersection of two calendar patterns.	Calendar pattern that has every interval on that was on in both calendar patterns; the rest are off. If the given patterns do not have the same interval unit, the pattern with the larger interval unit is expanded to match the other.	**AndOp**
Obtain the start date of the pattern for a given calendar.	Returns the start dates of the calendar patterns for each calendar.	**CalPattStartDate**
Convert a calendar from a smaller calendar pattern into a larger calendar pattern.	Collapses the given calendar pattern into destination units, which must have a larger interval unit than that of the given calendar pattern, i.e. Converting hours to days.	**Collapse**
Convert a calendar from a larger calendar pattern into a smaller calendar pattern.	Converts the given calendar pattern into the destination units, which must have a smaller interval unit than that of the given calendar pattern.	**Expand**
Return an inverted calendar intervals pattern.	Turns all on intervals off and all off intervals on in the given calendar pattern.	**NotOp**
Returns the union of the two calendar patterns.	This function returns a calendar pattern that has every interval on that was on in either of the calendar patterns; the rest are off. If the two patterns have different sizes of interval units, the resultant pattern has the smaller of the two intervals.	**OrOp**

Appendix – Calendar Routines

Return the intersection of two calendars.	Returns a calendar that has every interval on that was on in both calendars; the rest are off. The resultant calendar takes the later of the two start dates and the later of the two pattern start dates. If the two calendars have different size interval units, the resultant calendar has the smaller of the two intervals.	**AndOp**
Return the number of valid intervals in the named calendar between the two timestamps.		**CalIndex**
Return a set of valid timestamps within a range.	There are two syntaxes: the first syntax specifies the range as between two given timestamps. The second syntax specifies the number of valid timestamps to return after a given timestamp.	**CalRange**
Return the timestamp at a given number of calendar intervals after a given timestamp.		**CalStamp**
Returns a DATETIME value containing the start date of a given calendar name.		**CalStartDate**
Return a calendar that is the result of two others combined with the OR operator	This function returns a calendar that has every interval on that was on in either calendar; the rest are off. The resultant calendar takes the earlier of the two start dates and the two pattern start dates. If the two calendars have different sizes of interval units, the resultant calendar has the smaller of the two intervals.	**OrOp**

Information Management

Informix. software

Appendix C - Time Series API Information

Appendix - Time Series API's

Application Programmer Interfaces for a Time Series:

- Two types -
  - Client version – *tsfeapi*
    - Sample Location: $INFORMIXDIR/extend/TimeSeries.4.07.UC7/lib/tsfeapi.a (Unix/linux)
    - Similar directory for Windows.
    - Operates on a private copy of timeseries datum.
    - Changed data must be written back into the database.
      - Function *ts_copy* is used prior to the execution of a prepared statement.
  - Server version – *tsbeapi*
    - Sample Location: $INFORMIXDIR/extend/TimeSeries.4.07.UC7/lib/tsbeapi.a (Unix/Linux)
    - Similar directory for Windows.
    - Operates directly on timeseries datum.
      - Direct execution of prepared/unprepared statements.
- Header file – tseries.h -
  - Must be included in header whenever using *bsfeapi* or *tsfeapi*

Appendix - Time Series API – Data Structures

- ts_timeseries – the header for a time series data.
- ts_tscan – allows you to look at no more than two time series elements at a time.
- ts_tsdesc – A structure containing a time series and the data structures it works with.
- ts_tselem – A pointer to on element of a time series.

Appendix – Time Series API Routines

Task type	Description	API Routine	SQL Routine
Open and close a time series.	Open a time series.	**ts_open()**	
	Close a time series.	**ts_close()**	
	Return a pointer to the time series associated with the given time series descriptor.	**ts_get_ts()**	
Create and copy a time series.	Create a time series.	**ts_create()**	**TSCreate, TSCreateIrr**
	Create a time series with metadata.	**ts_create_with_metadata()**	**TSCreate, TSCreateIrr**
	Copy a time series.	**ts_copy()**	
	Free all memory associated with a time series created with ts_copy() or ts_create().	**ts_free()**	
	Copy all elements of one time series into another.	**ts_put_ts()**	**PutTimeSeries**

Appendix – Time Series API Routines (cont'd)

Task type	Description	API Routine	SQL Routine
Scan a time series.	Scan a time series.	**ts_begin_scan()**	
	Retrieve the next element from a scan.	**ts_next()**	
	End a scan.	**ts_end_scan()**	
	Find the timestamp of the last element returned by the ts_next().	**ts_current_timestamp()**	
	Return the offset for the last element returned by ts_next().	**ts_current_offset()**	
Make elements visible or invisible to a scan.	Make an element invisible.	**ts_hide_elem()**	**HideElem**
	Make an element visible.	**ts_reveal_elem()**	**RevealElem**

Appendix – Time Series API Routines (cont'd)

Task type	Description	API Routine	SQL Routine
Select individual elements from a time series.	Get the element closest to a given timestamp.	**ts_closest_elem()**	
	Get the element associated with a given timestamp.	**ts_elem()**	**GetElem**
	Get the element at a specified position.	**ts_nth_elem()**	**GetNthElem**
	Get the first element.	**ts_first_elem()**	**GetFirstElem**
	Get the last element.	**ts_last_elem()**	**GetLastElem**
	Find the next element after a given timestamp.	**ts_next_valid()**	**GetNextValid**
	Find the last element before a given timestamp.	**ts_previous_valid()**	**GetPreviousValid**
	Find the last element at or before a given timestamp.	**ts_last_valid()**	

Appendix – Time Series API Routines (cont'd)

Task type	Description	API Routine	SQL Routine
Update a time series.	Insert an element.	**ts_ins_elem()**	**InsElem**
	Update an element.	**ts_upd_elem()**	**UpdElem**
	Delete an element.	**ts_del_elem()**	**DelElem**
	Put an element in a place specified by a timestamp.	**ts_put_elem()** **ts_put_elem_no_dups()**	**PutElem** **PutElemNoDups**
	Append an element.	**ts_put_last_elem()** (regular only)	
	Put an element in a place specified by an offset.	**ts_put_nth_elem()** (regular only)	
Modify metadata.	Update metadata.	**ts_update_metadata()**	**UpdMetaData**
Convert between an index and a timestamp.	Convert timestamp to index.	**ts_index()**	**GetIndex**
	Put an element in a place specified by an offset.	**ts_time()**	**GetStamp**

Appendix – Time Series API Routines (cont'd)

Task type	Description	API Routine	SQL Routine
Transform an element.	Create an element from an array of values and nulls.	**ts_make_elem()** **ts_make_elem_rowdesc()** **ts_make_elem_no_dups()**	
	Convert an MI_ROW value to an element.	**ts_row_to_elem()**	
	Convert an element to a MI_ROW value.	**ts_elem_to_row()**	
	Free memory from a time series element created by **ts_make_elem()** or **ts_row_to_elem**()	**ts_free_elem()**	
Extract column data from an element.	Get a column from an element by name.	**ts_get_col_by_name()**	
	Get a column from an element by number.	**ts_get_col_by_number()**	
	Pull columns from an element into values and nulls arrays.	**ts_get_all_cols()**	
Create and perform calculations with timestamps.	Compare two timestamps.	**ts_datetime_cmp()**	
	Get fields from a timestamp.	**ts_get_stamp_fields()**	
	Create a timestamp.	**ts_make_stamp()**	

Appendix – Time Series API Routines (cont'd)

Task type	Description	API Routine	SQL Routine
Create and perform calculations with timestamps (cont'd).	Calculate the number of intervals between two timestamps.	**ts_tstamp_difference()**	
	Subtract N intervals from a timestamp.	**ts_row_to_elem()**	
	Add N intervals to a timestamp.	**ts_elem_to_row()**	
Get information about element data.	Find the number of a column.	**ts_col_id()**	
	Return the number of columns contained in each element.	**ts_get_col_cnt()**	
	Get type information for a column specified by number.	**ts_get_colinfo_number()**	
	Get type information for a column specified by name.	**ts_get_colinfo_name()**	
	Determine if an element is hidden.	**ts_ELEM_HIDDEN**	
	Determine if an element is NULL.	**ts_ELEM_NULL()**	

Appendix – Time Series API Routines (cont'd)

Task type	Description	API Routine	SQL Routine
Get information about a timeseries.	Get the name of calendar associated with a timeseries.	**ts_get_calname()**	**GetCalendar Name**
	Return the number of elements in a timeseries.	**ts_nelems()**	**GNelems**
	Return the flags associated with the timeseries.	**ts_get_flags()**	
	Get the name of the container.	**ts_get_containername()**	**GetContainerName**
	Determine if the timeseries is in a container.	**TS_IS_INCONTAINER()**	
	Get the origin of the timeseries.	**ts_get_origin()**	**GetOrigin**
	Get the metadata associated with the timeseries.	**ts_get_metadata()**	**GetMetaData**
	Determine if the timeseries is irregular.	**TS_IS_IRREGULAR()**	

Appendix – Time Series API Routines (cont'd)

Task type	Description	API Routine	SQL Routine
Get information about a calendar.	Return the number of valid intervals between two timestamps.	**ts_cal_index()**	**CalIndex**
	Return all valid timepoints between two timestamps.	**ts_cal_range()**	**CalRange**
	Return a specified number of timestamps starting at a given timestamp.	**ts_cal_range_Index()**	
	Return the timestamp at a given number of intervals after a given timestamp.	**ts_cal_stamp()**	**CalStamp**

Logo