SQL Execute

Table of Contents

1 Introduction

HTML SQL Library is a set of classes for retrieving and transforming data between different sources using SQL. It doesn't depend on any external library or DB server and can work with almost any source including files, strings and Delphi classes.
SQL query can be written in any modern dialect - Oracle, Firebird, MS SQL Server, etc. No need to setup connection or create query instance, just call one of the class methods. Example:

```
THtSQL.FromFile('c:\sample.xlsx', 'select * from sheet1 where _row > 0').ToJSONFile('
```

What it can be used for:
- Convert to/from CSV, JSON, XML, HTML, Excel, SQL script.
- Upload files/objects to database
- Export data from database
- Check data consistency
- Display data/objects in DB-aware controls, HtPanel, StringGrid, VirtualTreeview, ControlList
- Extend database capabilities (f.e. use Pivot in Firebird).
- Serialize and deserialize objects
- Reduce calls to REST server by using already loaded data.

Supported data sources:
- CSV file
- JSON file
- JSON string
- XML file
- HTML file
- Text file with fixed size fields
- SQL script
- ZIP archive
- XLS/XLSX Excel sheet (requires HTML Office Library).
- Outlook PST/OST file (requires HTML Office Library).
- TList<T>
- TObjectList
- Any class with IEnumerable or GetEnumerator support
- TDataset
- array of T
- SQL Query

Custom sources can be added by implementing IDMDatasource interface.

Supported dialects:
- SQL 92

- Oracle
- MS SQL Server
- Firebird
- MySQL
- PostrgesSQL
- SQLite
- ElevateDB

Result can be exported to
- CSV
- XML
- JSON
- SQL script
- TDataset
- TStringList
- TObjectList
- SQL table
- HTML (plain table or using template)
- Array of variant
- Binary file

or any other format, using custom processing of result data.

For example, the following line will extract 'movies' table data from SQL script and save it to JSON file:

```
THtSQL.FromFile('c:\movies.sql', 'select * from movie', []).ToJSONFile('c:\movies.jso
```

Library supports almost all SQL features, including joins, order, group by, having, cube, rollup, pivot, select from select, subselects, etc.

2 Getting started

To execute a query use one of the following **THtSQL** class methods:

- FromFile
- FromFolder
- FromDataset
- FromDatamodule
- FromList
- FromArray
- FromSchema
- FromJSON.

Common parameters are:

- **ASQL**: SQL query to execute
  AParams: Query parameters (:param)
- **ADialect** - SQL dialect class, by default TSQLDialectAll is used, it is a superset of Firebird, Oracle and MS SQL functions and statements.

```
function FromFile(const AFile, ASQL: string; const AParams: array of variant; ADialect
```

Query data from file or ZIP archive. AFile should contain full file (zip) name including path and extension. When file do not have nested tables (and is not zip) it is registered as 'data' table.

Example of selecting from CSV file: table name is 'data' because CSV do not have nested tables.

```
THtSQL.FromFile('c:\c1000.csv', 'select first 10 * from data order by c', []);
```

Example of selecting from XLSX file: table name is Sheet1 because XLSX can contain several sheets

```
THtSQL.FromFile('c:\sample.xlsx', 'select a, b, c from Sheet1 where e =:1', ['Seattle
```

```
function FromFolder(const AFolder, ASQL: string; const AParams: array of variant; ADia
```

Query data from files in a folder. Complex files are referenced using file name without extension and table name. Example of selecting from Excel file sample.xlsx:

```
THtSQL.FromFolder('c:\demo', 'select * from sample.Sheet1 order by c', []);
```

```
function FromDataset(ADataset: TDataset; const ASQL: string; const AParams: array of v
```

```
Select from singl dataset
```

```
function FromDatamodule(ADatamodule: TComponent; const ASQL: string; const AParams: a
```

Execute query using datasets from datamodule or form.

```
function FromList(ASource: TObject; const ASQL: string; const Params: array of variant
```

Query data from object list or any object with IEnumerble / GetEnumerator support. All elements should be of the same type. Example:

```
THtSQL.FromList(CustList, 'select name, address from data order by name', []).ToDatase
```

```
function FromArray(var ASource; TypeInfo: PTypeInfo; const ASQL: string; const Params:
```

Query data from array of objects or records. All array elements should be of the same type. Example:

```
type
  TCustArray = array of TCustomer;
var A: TCustArray;
THtSQL.FromArray(A, TypeInfo(TCustArray), 'select name, address from data order by nam
```

```
function FromSchema(ASchema: TDMSchema; const ASQL: string; const AParams: array of v
```

Query data from virtual schema containing datasources of different types. See Virtual

```
function FromJSON(const AJSON, ASQL: string; const AParams: array of variant; const A
```

Query data from JSON string.

3 Sources

3.1 CSV

CSV files are referenced by name without extension when using **FromFolder** or by "data" when using **FromFile** method.
Supported field dividers are: comma, semicolon and tab.
Column naming convention is the same as for Excel files - A, B, C, etc. To get row number use _row field.
If first row is header row, columns can be referenced by its names from first row and firs row is skipped. To use this mode pass true as first parameter, f.e.

```
select * from sample(true) where City='Barcelona'
```

3.2 XML

XML files may have complex structure and contains objects of different types. To select only necessary objects use XPath query passed in first parameter. When no parameters are passed, query will iterate children nodes of root nod. Example: if we have XML file cust.xml with the following structure

```
<Root>
  <Customers>
    <Customer CustomerID="GREAL">
      <CompanyName>Great Lakes Food Market</CompanyName>
      <ContactName>Howard Snyder</ContactName>
      <ContactTitle>Marketing Manager</ContactTitle>
      <Phone>(503) 555-7555</Phone>
      <FullAddress>
        <Address>2732 Baker Blvd.</Address>
        <City>Eugene</City>
        <Region>OR</Region>
        <PostalCode>97403</PostalCode>
        <Country>USA</Country>
      </FullAddress>
    </Customer>
  </Customers>
  <Orders>
    <Order>
      <CustomerID>GREAL</CustomerID>
      <EmployeeID>6</EmployeeID>
      <OrderDate>1997-05-06T00:00:00</OrderDate>
      <RequiredDate>1997-05-20T00:00:00</RequiredDate>
      <ShipInfo ShippedDate="1997-05-09T00:00:00">
        <ShipVia>2</ShipVia>
        <Freight>3.35</Freight>
        <ShipName>Great Lakes Food Market</ShipName>
```

```
        <ShipAddress>2732 Baker Blvd.</ShipAddress>
        <ShipCity>Eugene</ShipCity>
        <ShipRegion>OR</ShipRegion>
        <ShipPostalCode>97403</ShipPostalCode>
        <ShipCountry>USA</ShipCountry>
      </ShipInfo>
    </Order>
  </Orders>
</Root>
```

we can select only customers:

```
select * from cust('//Customer')
```

or only orders:

```
select * from cust('//Order')
```

Columns are mapped to node attributes. To get node text or reference nested nodes use XPath expession as column name. Example:

```
select "/CompanyName", "/FullAddress/City" from cust('//Customer')
```

To refer to current node use "/current", parent node - "/.."

Joined customers and orders:

```
select o."/OrderDate", c."/CompanyName", c."/FullAddress/City"
from cust('//Customer') c join cust('//Order') o on o."/CustomerID" = c.CustomerID
```

3.3 HTML

Tables in HTML file can be referenced using index passed in first parameter. Example:

```
select * from countries(0)
```

This code will extract data from first table in countries.html file.

3.4 JSON

JSON files may have complex structure and contains objects of different types. To select only necessary objects use XPath query passed in first parameter. When no parameters are passed, query will iterate top nodes, if file contain list of JSON objects, or children of root node.
Column names may also contains XPath to reference child nodes or array elements (array elements has name "value").
Example: if we have JSON file animals.json with the following structure

```
{
   "name": "Meowsy",
   "species" : "cat",
   "foods": {
     "likes": ["tuna", "catnip"],
     "dislikes": ["ham", "zucchini"]
   }
 },
 {
   "name": "Barky",
   "species" : "dog",
   "foods": {
     "likes": ["bones", "carrots"],
     "dislikes": ["tuna"]
   }
 },
 {
   "name": "Purrpaws",
   "species" : "cat",
   "foods": {
     "likes": ["mice"],
     "dislikes": ["cookies"]
   }
 }
```

We can select all top objects:

```
select name, species, "/foods/likes/value" from animals
```

Select all foods elements and first "likes" array value

```
select "/likes/value[0]" from animals('//foods')
```

3.5 Text

To extract data from text files containing fixed with fields, pass field sizes in parameters. Example:

```
select * from fixed(true, 12,12,15)
```

If first row is header row, columns can be referenced by its names from first row and firs row is skipped. To use this mode pass true as first parameter, otherwise columns are named in Excel style.

3.6 Excel

Sheets may be referenced by name or index. Example:

```
Sample.Sheet1
Sample."Sheet Name with spaces"
Sample.Sheets(0)
```

File name without postfix means first sheet.

Column naming convention is A, B, C, etc. To get row number use **_row** field.
If first row is header row, columns can be referenced by its names from first row and firs row is skipped. To use this mode pass true as first parameter, f.e.

```
select * from sample(true)
```

All cell values are treated as strings, to convert it to other types use conversion functions or **cast** statement.

3.7 Outlook

Folders are referenced by full name. Example:

```
select FromName, FromAddress, Subject, Received, TextBody
from enron."lokay-m.MLOKAY (Non-Privileged).TW-Commercial Group"
```

3.8 SQL script

SQL script can contain several tables so table name should be specified explicitly. I.e. when using FromFolder:

select * from sqlfile.tablename

When script file contain **create table** statement it is used to determine column datatypes.

3.9 TList<T>

Any Delphi object with support for IEnumerable or GetEnumerator can be passed to FromList method.
Note that all objects in list should be of the same type.
Query columns can reference public/published object fields, properties and methods. Methods can have parameters. Example:

```
type
  TCustomer = class
  public
    CustomerID: integer;
    Active: boolean;
    Name, Address, City, Country, Email: string;
    function OnSubscription(ADate: TDateTime): boolean;
  end;

  TCustList = TObjectList<TCustomer>;

  CL := TCustList.Create;
```

```
...

THtSQL.FromObject(CL, 'select Name, Address, OnSubscription(CURRENT_DATE) from data',

Nested objects/records can also be referenced. For example

select d.Owner.Caption from data d
```

3.10 array of T

Any array of records or objects can be used as query source. Note, that elements in an array should be of the same type.
Query columns can reference public/published object fields, properties and methods. Methods can have parameters. Example:

```
type
  TCustomer = class
  public
    CustomerID: integer;
    Active: boolean;
    Name, Address, City, Country, Email: string;
    function OnSubscription(ADate: TDateTime): boolean;
  end;

  TCustArray = array of TCustomer;

var A: TCustArray;

THtSQL.FromArray(A, TypeInfo(TCustArray), 'select name, address, OnSubscription(CURREI
```

3.11 ZIP

It is possible to select from files inside ZIP archive. F.e. if we have sales_data.zip containing four JSON files - customers, orders, products and sales, we can write:

```
select c.name, o.orderId, o.date, p.name, s.quantity
from
  sales_data.sales s
  join sales_data.orders o on o.orderid=s.orderid
  join sales_data.customers c on c.customerid=o.customerid
  join sales_data.products p on p.productid=s.productid
```

3.12 Virtual Schema

Virtual schema allows to combine different data sources. For example we can join JSON table and object list.

```
var
   VS: TDMVirtualSchema;
   CL: TCustList;
```

```
begin
  VS := TDMVirtualSchema.Create;
  try
    VS.RegisterDatasource('orders', THtJSONDatasource.CreatefromFile('c:\orders.json'
    VS.RegisterDatasource('cust', THtRttiDatasource.CreatefromList(CL));
    THtSQL.FromSchema(VS, 'select c.name, o.order_date from orders o join customers c
  finally
    VS.Free;
  end;
```

Note that schema should not be destoyed until query is released.

3.13 SQL Query

Example of performing query on another query from SQL databse.

```
Schema := TDMSchema.Create('', TDMFDProvider.Create('localhost:d:\mydb.fdb@sysdba;mast
try
  Schema.Provider.Q.Select('select first 100 * from customers');
  s := THtSQL.FromDataset(Schema.Provider.Q.Dataset, 'select * from data', []).AsJSON
finally
  Schema.Free
end;
```

4 Export

Query result has the following methods:

```
function SQLData: THtSQLData;
function AsString: string;
function AsDouble: double;
function AsInteger: integer;
function AsValue: TValue;
function AsRow: TVarDynArray;
function AsRowSet: TVar2DynArray;
function AsDataset: TDataset;
function AsStringList: TStringList;
function AsDatapacket: THtXMLNode;
function AsHTML(const ATemplate: hstring): hstring;
procedure ToStringList(AValue: TStrings; const ADelimiter: string = ';');
procedure ToObjectList(AValue: TList);
procedure ToDataset(AValue: TDataset);
procedure ToCSVFile(const AFileName: string; AddHeader: boolean = true;
ADelimiter: char = ',');
procedure ToJSONFile(const AFileName: string);
procedure ToXMLFile(const AFileName: string; const ARoot: string = 'Root';
const ATag: string = 'R');
procedure ToSQLFile(const AFileName, ATable: string);
procedure ToSQLTable(ASchema: TDMSchema; const ATable: string);
procedure Unserialize(AConstructor: THtUnserializeCreate);
```

Also it supports IDMDatasource interface, so can be registered as datasource for another query.

Single value result:

```
function AsString: string;
function AsInteger: integer;
function AsDouble: double;
function AsValue: TValue;
```

Single row result:

```
function AsRow: TVarDynArray;
```

4.1 CSV

```
procedure ToCSVFile(const AFileName: string; AddHeader: boolean = true; ADelimiter: ch
```

Export query result to CSV file.

AddHeader - add row with collumn names
ADelimiter - column delimiter.

4.2 XML

procedure ToXMLFile(const AFileName: string; const ARoot: string = 'Root'; const ATag: string = 'R');

Export query result to XML file with UTF8 encoding.
ARoot - name of root tag
ATag - name of row tag.

4.3 JSON

```
function AsJSON: hstring;
procedure ToJSONFile(const AFileName: string);
```

Export query result to JSON string or UTF8 encoded JSON file

4.4 Dataset

```
procedure ToDataset(AValue: TDataset);
```

Load query result to dataset. Before adding records, method creates dataset fields.
Note that destination dataset should not be used as source dataset in query.

```
function AsDataset: TDataset;
```

Export query result as memory table. Result should be destoyed after use.

4.5 HTML

```
function AsHTML(const ATemplate: hstring): hstring;
```

Process result records using HTML template (see HTML Report Library manual for details).

4.6 Objects (Deserialization)

Query can be used for unmarshalling (deserialization) of objects. Use Unserialize method and pass reference to function which creates and return object instance
Example:

JSON:

```
{"CustomerId": 1,"Active": true,"Name": "Jason Orr","Address": "Ap #387-8229 Nullam R
{"CustomerId": 2,"Active": true,"Name": "Devin Herman","Address": "P.O. Box 905, 9608
```

Class declaration:

```
  TCustomer = class
  public
    CustomerID: integer;
    Active: boolean;
    Name, Address, City, Country, Email: string;
  end;

  TCustList = TObjectList<TCustomer>;
```

Code:

```
  var CL: TCustList;
  begin
    CL := TCustList.Create;
    THtSQL.FromFolder(SelectedFolder, 'select * from customers', []).Unserialize(
      function: TObject begin
       Result := TCustomer.Create;
       CL.Add(TCustomer(Result));
      end);
```

4.7 ForEach

Iterate SQL Result, stop if funciton returns false.

```
procedure ForEach(AProcessor: THtSQLResultProc);

THtSQLResultProc = reference to function(Row: integer): boolean;
```

Example:

```
var V: PVarDynArray;
begin
  for V in THtSQL.FromFolder(SelectedFolder, 'select name from customers', []) do
    Memo1.Lines.Add(V^[0]);
```

4.8 SQL script

```
procedure ToSQLFile(const AFileName, ATable: string);
```

Export query result as SQL script file.

```
procedure ToSQLTable(ASchema: TDMSchema; const ATable: string);
```

Load query result records to existing table in database.

4.9 SQL table

procedure .ToSQLTable(ASchema: TDMSchema; const ATable: string);

Export SQL result to database table (using INSERT statements).

4.10 String List

```
procedure ToStringList(AValue: TStrings; const ADelimiter: string = ';');
```

Export query result to TStringList

4.11 Binary file

procedure ToBinaryFile(const AFileName: string);

Export data to binary file. All fields are written in a binary format. Integer takes 4 bytes, Date 8 bytes, etc.
Strings are written in 2 bytes unicode format with fixed length (same as field size), tail is filled with zeros.

4.12 UI controls

Query result can be displayed in UI controls. To enable this, add sqlgui unit to uses list and pass control to ToObject method.
Following VCL controls can be used:

- DBGrid
- StringGrid
- ValueEditor
- ListView
- ControlList
- Chart
- HtPanel
- VirtualStringTree

Note that reference to IHtSQLResult interface should be stored in some variable while working with control, otherwise UI controller will be destroyed (except TChart).

Correct:
MyFormVar: IHTSQLResult;
..

MyFormVar := THtSQL.From*(..);
MyFormVar.ToObject(DBGrid1);

Wrong:
THtSQL.From*().ToObject(DBGrid1);

Common rules

Hidden fields
Fields starting with _underscore are not displayed.

Display format
To set display format, add it in parentheses at the end. Example: "Start Date (dd.mm.yyyy)"
Format is supported for
- Date/Time fields: any format supported by **FormatDateTime** function
- Number fields: any format supported by **FormatFloat** function and special format **bytes** to display size with KB, MB, GB postfix.
- String fields: %s is replaced by field value.

Format can contain pascal script expressions when it starts with =. Example:

```
Received "Received(=iif(value > Date(), 'hh:nn', 'dd MMM yyyy hh:nn'))",
```

HTML fields
When field contains HTML, to display it as HTML (unescaped) enclose field name in brackets: **"<HTML Field>"**

Multiline fields.
To display multiline text from field, add + at the end. Example: "Message Body+"

Field templates
To set template for a field, add field with the same name but ending with **_template**. Example: **customer_template.**
Template is in Mustache template format and can contain any field. For a value of current field use {{value}}.
Example:
' {{value}} ' body_template

Row and column styles
To setup style for rows and columns use row_style and column_style field. Styles may contain calculated expressions, to refer to row and column index use col and row variables.
Example or row style for highlighing odd rows:

'=iif(odd(row), ''background:#f8faff;'', '''')' row_style,

Groups

Field named **group_field** is treated as group name. When component supports groups (VirtualTreeView, HtPanel) it is displayed above group.

Selected row

To get values if selected row, use **SQLResult.Selected[FieldIndex]** property.

Filtering

To apply filter simply start typing.

DBGrid

After export to DBGrid, it is possible to sort grid by clicking on column title and filter grid by entering text in any column.
Simple operators are also supported in filter, for example >100, <>50, <= 5.5.

ControlList

Set field aliases to control name and property, f.e.

```
select c.name "label1.caption", address "memo1.lines.text" from customers
```

Some properties names: Caption, Text, HTML, Lines can be omitted, so query above can be written as following:

```
select c.name label1, address memo1 from customers
```

List can be filtered by typing search text.

Chart

Export to TeeChart has two modes. When query has more than two columns, and first column is not float, it uses multiseries mode where each row is separate series and values are in float fields.
Otherwise first column is treated as value and second as label. Third (optional) as color.
ToObject parameter may be TChart or TChartSeries instance.

Example:

```
THtSQL.FromFolder(SelectedFolder,
  'select b, a from sample.sheet1 where _row > 0 '+
  'pivot(sum(substring(e from 2)) for a in (select distinct a from sample.sheet1 where
```

```
[]).ToObject(Chart1);
```

5 SQL features

select from select
join, left join
union, union all, intersect
distinct, order by, group by, having
first skip last rows limit fetch
sum, max, min, avg, list/listagg
= <> > >= < <= + - * / mod || in between like not like starting with and or
exists, any, some, all
subselects
cast, decode, iif, case, coalesce
pivot, rollup, cube

5.1 Datatypes conversion

Conversion between datatypes can be performed using SQL cast statement, f.e.

```
select cast(A as integer) from data
```

Some functions and operators perform explicit conversion, f.e. **sum** and **avg** convert its argument to double, same for **+, -, *, /, div, mod** operators.
Note that **min** and **max** aggregate functions do not perform implicit conversion.

Text to number conversion

By default text to number conversion expect '.' as decimal separator and no thousands separator. Explicit separators can be passed as second argument to oracle **TO_NUMBER** function and **float**() funtction. Example:

```
TO_NUMBER('1,230.15', '.,')
float('3 334,5', ', ')
```

To convert hex string like 0xabc or abc to number, use HextoDec function.

Text to date conversion

Explicit conversion supports following formats:

```
YYYY-MM-DD
YYYY-MM-DD HH:NN:SS
DD.MM.YYYY
DD.MM.YY
DD.MM.YYYY HH:NN:SS
DD.MM.YYYY HH:NN
DD.MM.YYYY H:NN
```

```
DD-MMM-YYYY HH:NN:SS
DD-MMM-YYYY
DD-MMM-YY
```

For implicit conversion use TO_DATE(String, Format) function.

5.2 Pivot (cross)

Examples:

```
select b, a from sample.sheet1 where _row > 0
pivot(count(*), sum(e) for a in (select distinct a from sample.sheet1 where _row>0))

select b, a from sample.sheet1 where _row > 0
pivot(count(*) for a in ('Government', 'Midmarket'))
```

5.3 Rollup/Cube

Example:

```
select a, b, count(*), grouping(a)
from sample.sheet1
group by rollup(a, b)
order by 1,2 nulls last
```

5.4 Intersect

Countries (B) present in both segments (A)

```
select B from sample.sheet1 where A = 'Government'
intersect
select B from sample.sheet1 where A = 'Midmarket'
```

5.5 Subquery

Calc percent from max quantity

```
select c.name, o.orderId, o.date, p.name,
  round(100 * s.quantity / (select max(cast(quantity as float)) from sales_data.sales)
from
  sales_data.sales s
  join sales_data.orders o on o.orderid=s.orderid
  join sales_data.customers c on c.customerid=o.customerid
  join sales_data.products p on p.productid=s.productid
```

5.6 List/ListAgg

List of products by order

```
select o.orderId, o.date, list(p.name, ', ') products
from
  sales_data.sales s
  join sales_data.orders o on o.orderid=s.orderid
  join sales_data.products p on p.productid=s.productid
group by 1, 2
```

5.7 Contains (full text search)

Contains function allows to select records containing certain words in a text field.
Example:

```
select title, overview from movies.movie where Contains(overview, 'Leia Luke')
```

Result may be sorted by relevance

```
select title, overview from movies.movie where Contains(overview, 'Leia Luke')
order by relevance desc
```

Second parameter may contain AND and OR operators

```
select title, overview from movies.movie where Contains(overview, 'Han AND Leia OR Lu
```

To highlight found words use Highlight function

```
select title, highlight(overview, '<b>', '</b>') overview from movies.movie where Cont
```

5.8 SQL92 functions

```
CAST(expression as datatype)
CURRENT_DATE: DATE
CURRENT_TIME
CURRENT_TIMESTAMP: TIMESTAMP
SUM(NUMBER): NUMBER
MAX()
MIN()
AVG(NUMBER)
COUNT
COALESCE(expression, expression)
FLOAT(STRING [,SEPARATORS]): NUMBER
```

5.9 Firebird functions

Aggregate functions

LIST

Common functions

IIF
DECODE
NULLIF

Date functions

```
DATEADD(NUMBER DATE_PART to DATETIME): DATETIME
DATEDIFF(DATE_PART from DATETIME to DATETIME): NUMBER
DOW
EXTRACT(DATE_PART from DATETIME): DATETIME
```

String functions

```
SUBSTRING(VARCHAR from NUMBER to NUMBER): VARCHAR
ASCII_CHAR
CHAR_LENGTH
GEN_UUID
LPAD
LEFT
LOWER
OCTET_LENGTH
POSITION
REPLACE
REVERSE
RIGHT
RPAD
TRIM
UPPER
```

Mathematical functions

```
ABS
ACOS
ASIN
ATAN
ATAN2
BIN_AND
BIN_OR
BIN_SHL
BIN_SHR
BIN_XOR
CEILING
COS
COSH
COT
EXP
FLOOR
HASH
LN
```

```
LOG
LOG10
MAXVALUE
MINVALUE
MOD
PI
POWER
RAND
ROUND
SIGN
SIN
SINH
SQRT
TAN
TANH
TRUNC
```

5.10 Oracle functions

Aggregate functions

```
GROUPING(expression)
LISTAGG(EXPR [, DELIMITER]) WITHIN GROUP (ORDER BY FIELD)
```

Mathematical functions

```
ABS(NUMBER)
ACOS(NUMBER)
ASIN(NUMBER)
ATAN(NUMBER)
ATAN2(NUMBER)
BITAND(NUMBER, NUMBER)
CEIL(NUMBER)
COS(NUMBER)
COSH(NUMBER)
EXP(NUMBER)
FLOOR(NUMBER)
GREATEST(VALUE, [, VALUE])
GREATEST(VALUE, [, VALUE])
LN(NUMBER)
LOG(NUMBER, NUMBER)
MOD(NUMBER, NUMBER)
NANVL(NUMBER, NUMBER)
POWER(NUMBER, NUMBER)
REMINDER(NUMBER, NUMBER)
ROUND(NUMBER [, Digits])
SIGN(NUMBER)
SIN(NUMBER)
SINH(NUMBER)
SQRT(NUMBER)
TRUNC
TAN(NUMBER)
TANH(NUMBER)
```

Common functions

```
DECODE
WIDTH_BUCKET(EXPR, MIN_VALUE, MAX_VALUE, NUM_BUCKETS)
NULLIF
NVL
NVL2
```

String functions

```
CHR(NUMBER)
CONCAT(VARCHAR, VARCHAR)
INITCAP(VARCHAR)
INSTR(STRING, SUBSTRING [, POS])
LOWER(VARCHAR)
LPAD(EXPR, n [,BLANK])
LTRIM(EXP [, SET])
NLS_LOWER(EXPR, [,NLS_PARAM])
NLSSORT(EXPR, [,NLS_PARAM])
NLS_UPPER(EXPR, [,NLS_PARAM])
REGEXP_COUNT(EXPR, ,PATTERN[, POS, MATCH])
REGEXP_REPLACE(EXPR, [,PATTERN, REPLACE, POS, OCCURENCE, MATCH])
REGEXP_SUBSTR(EXPR, [,PATTERN, POS, OCCURENCE, MATCH])
REPLACE(EXPR, SEARCH [, REPLACE='']) 
RPAD(EXPR, n [,BLANK=' '])
RTRIM(EXP [, SET=' '])
SOUNDEX(VARCHAR)
SUBSTR(EXP, POS [,LEN])
SUBSTRB(BYTES, POS [,LEN])
SUBSTRC(UStr, POS [,LEN])
SUBSTR2(UCS2_STR, POS [,LEN])
SUBSTR4(UCS4_STR, POS [,LEN])
TO_CHAR(STRING, [FORMAT])
TO_NUMBER(STRING, [FORMAT])
TRANSLATE(EXP, FROM, TO)
TREAT(EXPR AS TYPE)
TRIM([LEADING|TRAILING|BOTH CHAR=' ' FROM] EXP)
UPPER(VARCHAR)
```

Date functions

```
TO_DATE(STRING, [FORMAT])
TO_TIMESTAMP(STRING, [FORMAT])
ADD_MONTHS(Date, Months)
EXTRACT(DATE_PART from DATETIME): DATETIME
LAST_DAY(Date)
MONTHS_BETWEEN(Date, Date)
SYSDATE
```

5.11 MS SQL functions

Aggregate functions

```
STRING_AGG
```

String functions

```
CHAR(NUMBER)
CHARINDEX(STRING, STRING, NUMBER)
CONCAT(STRING, STRING)
CONCAT_WS(STRING, STRING, STRING)
DATALENGTH: DATALENGTH(STRING)
FORMAT(STRING, STRING)
LEFT(STRING, NUMBER)
LEN(STRING)
LOWER(STRING)
LTRIM(EXP)
REPLACE(EXPR, SEARCH)
REVERSE(|)
RIGHT(STRING, NUMBER)
RTRIM(EXP)
SUBSTRING(VARCHAR, NUMBER, NUMBER): VARCHAR
STRING_SPLIT(Expression, Sep, Index)
```

Mathematical functions

```
ABS(NUMBER)
ACOS(NUMBER)
ASIN(NUMBER)
ATAN(NUMBER)
ATAN2(NUMBER)
CEIL(NUMBER)
COS(NUMBER)
COSH(NUMBER)
EXP(NUMBER)
FLOOR(NUMBER)
LN(NUMBER)
LOG(NUMBER, NUMBER)
POWER(NUMBER, NUMBER)
ROUND(NUMBER)
SIGN(NUMBER)
SIN(NUMBER)
SINH(NUMBER)
SQRT(NUMBER)
TAN(NUMBER)
TANH(NUMBER)
```

Date functions

```
DATEADD(NUMBER, DATE_PART, DATETIME): DATETIME
DATEDIFF(DATE_PART, DATETIME, DATETIME): NUMBER
DATEFROMPARTS(YEAR, MONTH, DAY): DATE
DATENAME(PART, DATE): NUMBER
DATEPART(PART, DATE): NUMBER
DAY(DATE)
MONTH(DATE): NUMBER
YEAR(DATE): NUMBER
```

6 Examples

6.1 JSON to objects and grid

Parse customers.json file, create list of TCustomer from this JSON, display list in grid.

```
type
  TCustomer = class
  public
    CustomerID: integer;
    Active: boolean;
    Name, Address, City, Country, Email: string;
    function OnSubscription(ADate: TDateTime): boolean;
  end;
  TCustList = TObjectList<TCustomer>;
...
  CL := TCustList.Create;
  Res := THtSQL.FromFolder(SelectedFolder, 'select * from customers', []).Unserialize
    function: TObject begin
      Result := TCustomer.Create;
      CL.Add(TCustomer(Result))
    end);

  THtSQL.FromObject(CL, 'select name, address, OnSubscription(CURRENT_DATE) from data
```

6.2 Tab delimited text to binary

Goal: parse tab delimited file containing encoding map and write result to binary file containing array of two integers.
File has header line. First column is integer number, other are hex numbers. Columns may have optional value after comma separator and may contain asterisk.

CID	GB	GB-EUC	GBpc-EUC	GBT	GBT-EUC	GBTpc-EUC	GBK-EUC	GBKp-EUC
0	*	*	*	*	*	*	*	*	*	*	*	*
1	*	*	20	*	*	20	*	20	20	0020	c2a0,2000a0,0020
2	*	*	21	*	*	21	*	21	21	0021	21 0021 00000021
571	2657	a6d7	a6d7	2657	a6d7	a6d7	a6d7	a6d7	a6d7	03c8	cf88 03c8 000003c8
572	2658	a6d8	a6d8	2658	a6d8	a6d8	a6d8	a6d8	a6d8	03c9	cf89 03c9 000003c9
573	2659,232cv	a6d9,a3acv	a6d9,a3acv	2659,232cv	a6d9,a3acv	a6d9,a3acv	a6d9,a3acv	a6d9,a3acv

We are using String_Split to extract first value and HextoDec to convert hex string to integer.

```
THtSQL.FromFolder(SelectedFolder, 'select cast(a as int),  cast(hextodec(string_split
 ' from cid where k != ''*'' and _row > 0)', []).ToBinaryFile('gbk.dat')
```

Index

- N -