SQL Framework

© 2018 http://delphihtmlcomponents.com
# Table of Contents

Foreword 0

## Part I Introduction

1. Supported Databases ................................................................. 5
2. Supported DAC ................................................................. 5

## Part II Database schema

1. Loading schema ........................................................................... 7
2. Accessing schema objects ............................................................. 7
3. Creating SQL script ........................................................................ 8
4. Comparing schema objects ............................................................ 8
5. Creating triggers for autoincrement fields ..................................... 9
6. Schema objects descriptions .......................................................... 9
7. Using schema in multithread environment ...................................... 9
8. Schema serialization and deserialization ....................................... 9
9. TDMField class ........................................................................... 9
10. TDMTable class ....................................................................... 11

## Part III SQL Parsing

1. Database dialects ......................................................................... 13
2. Class hierarchy ........................................................................... 13
3. Query hierarchy .......................................................................... 14
4. Parsing sample ........................................................................... 14
5. Parsing errors and tolerant mode ................................................ 14
6. Templates .................................................................................. 14

## Part IV SQL formatting

1. TSQLFormatter class ................................................................. 15
2. Generating formatted SQL .......................................................... 16

## Part V SQL transforming

1. Transforming methods ............................................................... 17
2. Translating between dialects ........................................................ 17

## Part VI SQL context and code completion

1. TSQLContext class ................................................................. 18
2. Using SQL context ..................................................................... 19
3. TSQLHLEditor class .............................................................. 19
Part VII  How To

1 Get token at source position ................................................................. 21
2 Add table field to query columns ...................................................... 21
3 Add condition to Where ....................................................................... 21
4 Generate query for given table ............................................................ 22
5 Set row limit for query ......................................................................... 22

Index

0
1 Introduction

SQL framework is designed for simplifying access to database metadata and creating/editing SQL queries regardless of underlying database and data access components. It contains two parts

1. Database schema part
2. SQL queries part

Database schema part contains set of classes which represents database schema objects (tables, fields, sequences, etc.) and adapter classes for different databases and data access components. Following units are related to schema part:

- **DMSchema** - database objects classes and abstract classes for adapters.
- **DMFireDAC** - FireDAC provider.
- **DMUniDAC** - UniDAC provider.
- **DMUIB** - UIB provider.
- **DMFirebird** - Firebird adapter.
- **DMOracle** - Oracle adapter.
- **DMPostgres** - PostgreSQL adapter.
- **DMMYSQL** - MySQL adapter.
- **DMSQLServer** - Microsoft SQL adapter.

SQL part contains two units:

- **sqlparse** - SQL parser and transformer classes, SQL context class.
- **sqlhleditor** - example of SQL editor component based on JEDI JWideHLEditor

1.1 Supported Databases

- Firebird
- Oracle
- MySQL
- Postgres
- Microsoft SQL

1.2 Supported DAC

- **FireDAC** - unit DMFireDAC
- **UniDAC** - unit DMUniDAC
- **UIB** - unit DMUIB

Any other DAC can be used by implementing simple provider class descendant:
TDMProvider = class abstract
public type
TDMProviderType = (Oracle, MySQL, Postgres, MSSQL, Firebird);
function GetConnection: TObject; virtual; abstract;
public
constructor Create(const AConnection: string;
AProviderType: TDMProviderType;
const AConnectionOptions: string = ''); virtual; abstract;
procedure Connect; virtual; abstract;
function CreateQuery: TDMQuery; virtual; abstract;
procedure ExecuteScript(const AScript: string); virtual; abstract;
procedure CreateDatabase(const AName: string); virtual; abstract;
property Connection: TObject read GetConnection;
property ProviderType: TDMProviderType read FProviderType;
end;
2 Database schema

Database schema part represents database objects metadata and can be used for following purposes:

- Get list of database tables and its descriptions
- Get list of database views
- Get list of table or view fields with their types and descriptions
- Get list of table foreign keys and indexes
- Get list of database sequences
- Set table or field description
- Add new field into table
- Add new foreign key into table
- Add new index into table
- Add primary key into table
- Create new sequence
- Create new table with primary and foreign keys
- Get list of tables related to selected table
- Comparing tables metadata
- Comparing tables data
- Comparing schema metadata
- Creating trigger for autoincrement field simulation

2.1 Loading schema

Schema requires two objects - database adapter of TDMAdapter class which encapsulates specific features of a database and DAC provider of TDMProvider class for accessing DB server. Example of creating and loading DB schema:

```delphi
```

Adapter and Provider are destroyed automatically by schema object. When using schema loaded from XML (see Serialization and deserialization topic) provider parameter can be nil.

2.2 Accessing schema objects

Tables (via TDMSchema object)

- **Tables** List of all schema tables
- **FindTable** Find table object by name, return nil if table is not found
- **TableName** Find table object by name and raise exception if table is not found
- **FindTableByAlias** Find table object by default table alias
- **CreateTable** Create table in database by generating and executing SQL script

Sequences (via TDMSchema object)

- **Sequences** List of all sequences
- **SequenceByName** Return sequence by name and raise exception if sequence not found
- **FindSequence** Return sequence by name of nil when sequence is not found.
Fields (via TDMTable object)

Fields All table fields
FieldByName Return field by name or raise exception if field is not found
HasField Return true when field is found in table
FindField Find field by name, return nil if field is not found
AddField Add field into DB table by generating and executing SQL script

Foreign Keys (via TDMTable object)

ForeignKeys All table foreign keys
HasForeignKey Return true if table has foreign key to table T
AddForeignKey Add foreign key into DB table by generating and executing SQL script

Indexes

Indexes All table indexes
AddIndex Add new index into DB table by generating and executing SQL script
HasIndexOn Return true if table has index on field F (F is only field in index or first field)

2.3 Creating SQL script

SQL script for schema objects can be obtained via schema Adapter object. It has the following methods:

CreateUpdateScript Create script containing difference between two schemas
FieldSQL Script for single field
TableSQL Script for table and related objects (primary and foreign keys, indexes)
ForeignKeySQL Script for table foreign key
IndexSQL Script for table index
SequenceSQL Script for sequence
TableDiffSQL Script containing differences between two tables
TableDataDiffSQL Script containing differences between data in two tables (only inserted and deleted records using primary key)
TableDescriptionSQL Script for setting table description
FieldDescriptionSQL Script for setting field description
AutoIncrementTrigger Script for creating trigger for autoincrement field simulation

2.4 Comparing schema objects

Library has methods for creating SQL scripts containing differences between schema objects. Compared objects can belong to different schema with different database types.


- Schema.Adapter.TableDiffSQL: method for creating script containing difference in metadata between two tables
• **Schema.Adapter.TableDataDiffSQL**: method for creating script containing difference in table data between two tables (only inserted and deleted records)

## 2.5 Creating triggers for autoincrement fields

Use `Schema.Adapter.AutoIncrementTriggerSQL` to create SQL script for simulating autoincrement field in table.

```pascal
define AutoIncrementTriggerSQL(const T: TDMTable; const SQ: TDMSequence): string;
```

Trigger will fill primary key field with sequence value at insert when field is null.

## 2.6 Schema objects descriptions

Tables and Fields description (stored in database) can be read and modified via Description property. For sequences, description is read only and can be set only at sequence creation.

## 2.7 Using schema in multithread enviroment

When accessing schema and schema objects from different threads place all code that use schema objects between Schema.Acquire and Schema.Release calls.

## 2.8 Schema serialization and deserialization

Whole schema can be serialized to and deserialized from XML format using `TDMSchema.AsXML`:

```pascal
define prop AsXML: string
```

This can be used f.e. in following cases:

- Client application has no database connection (REST client)
- Current database should be compared with other database which is not accessible via network.
- Changes tracking

## 2.9 TDMField class

`TDMField` class represents table field metadata. It has the following members:

```pascal
function IsNumeric: boolean;
Check if field is numeric

function IsFloat: boolean;
Check if field is float.

function IsDateTime: boolean;
Check if field is date/time, date or time.

function IsText: boolean;
Check if field is text (varchar, memo)

function QuotedName: string;
Quoted field name in quotes

property Name: string;
Field name
```
property FullName: string
Field name with table name

property DataType: TFieldType
Field type

property Size: integer
Size for string and numeric fields

property Precision: integer read FPrecision;
Precision for numeric fields

property Scale: integer read FScale;
Scale for numeric fields

property Description: string
Field Description (from database)

property DefaultValue: string
Field default value

property Calculated: string
Expression for calculated fields

property Charset: string
Field charset

property Table: TDMTable
Reference to field table

property ForeignKey: TDMForeignKey
Reference to foreign key if field belongs to any.

property PrimaryKey: TDMIndex
Reference to primary key (if field is included in PK)

property IsPrimaryKey: boolean
Check if field is only primary key field

property IsReadonly: boolean
Check if field is read only

property NativeSQLType: string read FNativeSQLType write FNativeSQLType;
Native Field type (for source database)

property IsNotNull: boolean
Check if field is not null

property TableName: string read GetTableName;
Name of field table
2.10 TDMTable class

TDMTable class represents table or view metadata and has the following members:

function FieldByName(const FieldName: string): TDMField;
Find field by name. Raise exception when field is not found

function FindField(const FieldName: string): TDMField;
Find field by name. Do not raise exception when field is not found

function HasField(const FieldName: string): boolean;
Check if field exists in table

function HasForeignKeyTo(const T: TDMTable): boolean;
Check if table has foreign key to another table

function HasRelationWith(const T: TDMTable): boolean;
Check if one of the tables has foreign key to another.

function HasIndexOn(const F: TDMField): boolean;
Check if table has index on field

function AddField(const AName, ANativeType: string; ANotNull: boolean = false; const ADefault: string = ''): TDMField;
Add new field into table

procedure DeleteField(const AName: string);
Remove field from table

function AddForeignKey(const FKFields: array of TDMField;
const FKTable: TDMTable;
FKName: string = '';
ADeleteAction: TDMForeignKey.TFKAction = faNoAction;
AUpdateAction: TDMForeignKey.TFKAction = faNoAction): TDMForeignKey;
Add new foreign key into table

function AddIndex(const AIndexFields: array of TDMField; IndexName: string = ''): TDMIndex;
Add new index into table

procedure AddAutoincrementTrigger(const SQ: TDMSequence);
Add trigger for setting table primary key on inset using sequence value

function AsXML: string;
Table structure in XML format

function LikelyNameField: TDMField;
Return field which is most likely name field

property Fields: TDMFieldList
List of table fields

property Indexes: TDMIndexList
List of table indexes

property ForeignKeys: TDMFKList
List of table foreign keys
property Name: string
Table name

property FullName: string
Table name including schema name

property Description: string
Table description

property PrimaryKey: TDMIndex
Table primary key (if exists)

property Schema: TDMSchema
Table owner

property Alias: string
Table default alias for using in SQL queries (unique in schema)

property RelatedTables: TDMTableList
List of tables which has foreign keys to selected table of vice versa.

property Kind: TDMTableKind
Table type - regular table, view or stored procedure.
3 SQL Parsing

SQL parser parses an SQL select query and translates it into a hierarchy of Delphi classes. The generated hierarchy can be used for following purposes:

- Syntax checking
- Schema based query checking
- Query text formatting
- Getting list of used tables/fields/views/params
- Changing query columns, "where" conditions, "order by" columns.
- Adding new tables/columns
- Replacing tables/fields
- Translating between dialects

3.1 Database dialects

Supported database dialects:

- SQL92
- Oracle 9
- Oracle 12
- Firebird 2.0
- Firebird 3.0
- MySQL
- Microsoft SQL
- PostgreSQL

3.2 Class hierarchy

TSQLObject
  TSQLExpression
    TSQLOrderBy
  TSQLColumn
  TSQLTable
  TSQLTopRowLimit
  TSQLBottomRowLimit
  TSQLStatement
    TSQLCaseStatement
    TSQLCastStatement
    TSQLSelectStatement
    TSQLSelectQuery
    TSQLCTE

TSQLDialect
  TSQLDialectFireBird
    TSQLDialectFireBird3
  TSQLDialectOracle
    TSQLDialectOracle12
  TSQLDialectMSSQLServer
  TSQLDialectMySQL
  TSQLDialectPostgres

© 2018 http://delphihtmlcomponents.com
3.3 Query hierarchy

TSQLSelectQuery

[CTE: TSQLCTEStatements = list of TSQLSelectQuery]
Statements: TSQLSelectStatements = list of TSQLSelectStatement
[Order: TSQLOrderByList = list of TSQLOrderBy]
[BottomRowLimit: TSQLBottomRowLimit]

TSQLSelectStatement

[TopRowLimit: TSQLTopRowLimit]
Columns: TSQLColumns
Tables: TSQLTables = list of TSQLTable
[Where: TSQLExpression]
[Group: TSQLExpressions = list of TSQLExpression]
[Having: TSQLExpressions = list of TSQLExpression]

3.4 Parsing sample

var SQ: TSQLSelectQuery;
begin
  SQ := TSQLSelectQuery.Create(nil);
  try
    SQ.ParseString(Editor.Lines.Text, TSQLDialectOracle);
    ... 
  finally 
    SQ.Free 
  end;

3.5 Parsing errors and tolerant mode

In default parsing mode, exception is raised on first error in SQL script. Exception is of ESQLException class and has Line and SourcePos properties.
When query should be parsed to the end regardless of any errors, set TolerantMode property to true. In this mode only TSQLSelectQuery.OnError event is called but no exceptions raised.

3.6 Templates

Parser has support for mustache templates in SQL query. Templates has stTemplate token type and nkTemplate expression node kind.
Example: following query will be parsed without errors:

    select * from customers c where c.kind={{CUSTOMER_KIND}}
4 SQL formatting

Class hierarchy can be serialized back into query text. TSQLFormatter class is used for producing formatted SQL and has set of properties for adjusting produces text.

4.1 TSQLFormatter class

TSQLFormatter class has the following properties:

- **property** BlockIndent: integer
  Block indent size (spaces)

- **property** SpaceAfterComma: boolean
  Add space after comma symbol

- **property** AsBeforeFieldAlias: boolean
  Add "as" between field expression and field alias

- **property** AsBeforeTableAlias: boolean
  Add "as" between table expression and table alias

- **property** CaseReserved: TSQLFormatterCase
  Char case for reserved words

- **property** CaseTables: TSQLFormatterCase
  Char case for table names

- **property** CaseTableAliases: TSQLFormatterCase
  Char case for table aliases

- **property** CaseFields: TSQLFormatterCase
  Char case for field names

- **property** CaseFieldAliases: TSQLFormatterCase
  Char case for field aliases

- **property** CaseParams: TSQLFormatterCase
  Char case for parameters

- **property** CaseFunctions: TSQLFormatterCase
  Char case for functions

- **property** LineFeedSelect: TSQLFormatterLineFeeds
  Line feeds before and after SELECT word

- **property** LineFeedField: TSQLFormatterLineFeeds
  Line feeds before and after column

- **property** LineFeedFrom: TSQLFormatterLineFeeds
  Line feeds before and after FROM word

- **property** LineFeedTable: TSQLFormatterLineFeeds
  Line feeds before and after table in FROM section
property LineFeedJoin: TSQLFormatterLineFeeds
Line feeds before and after JOIN

property LineFeedWhere: TSQLFormatterLineFeeds read FLineFeedWhere write FLineFeedWhere default [slfBefore];
Line feeds before and after WHERE word

property LineFeedGroup: TSQLFormatterLineFeeds read FLineFeedGroup write FLineFeedGroup default [slfBefore];
Line feeds before and after GROUP word

4.2 Generating formatted SQL

var SQ: TSQLSelectQuery;
SF: TSQLFormatter;
begin
    SF := TSQLFormatter.Create(nil);
    try
        SQ := TSQLSelectQuery.Create(nil);
        try
            SQ.ParseString(Editor.Lines.Text, DefaultSQLDialect);
            SQ.CaretPosition := Editor.PosFromCaret(Editor.CaretX, Editor.CaretY) + 1;
            SQ.AsString(SF);
            Editor.Lines.Text := SF.AsString;
            Editor.SetFocus;
            Editor.CaretFromPos(SQ.CaretPosition - 1, X, Y);
            Editor.SetCaret(X, Y);
        finally
            SQ.Free;
        end;
    finally
        SF.Free;
    end;

In this sample caret position in Editor is preserved using TSQLSelectQuery.CaretPosition property
5 SQL transforming

5.1 Transforming methods

TSQLSelectQuery has the following methods for transforming:

- **procedure AddColumn(const TableName, FieldName: string);**
  Add field into column list. If table is not used in query it will be added into table list and join expression will be created.

- **procedure RemoveColumn(const TableName, FieldName: string);**
  Remove column containing field

- **procedure OrderByColumn(const TableName, FieldName: string; Desc: boolean = false);**
  Add field into Order By list

- **procedure ExpandAsterisk;**
  Replace table." column with all table columns

- **procedure ReplaceField(const SourceTableName, SourceFieldName, DestTableName, DestFieldName: string); override;**
  Replace all occurrences of table field with another field

- **procedure AddWhereCondition(const Condition: string; AndOperation: boolean = true); virtual;**
  Add condition to WHERE section of query first select statement or CTE

5.2 Translating between dialects

To translate query from one dialect to another, parse it using first dialect, change dialect property and then serialize. Example:

```pascal
var SQ: TSQLSelectQuery;
SF: TSQLFormatter;
begin
  SF := TSQLFormatter.Create(nil);
  try
    SQ := TSQLSelectQuery.Create(nil);
    try
      SQ.ParseString(Editor.Lines.Text, TSQLDialectFirebird);
      SQ.DialectClass := TSQLDialectOracle;
      SQ.AsString(SF);
      Editor.Lines.Text := SF.AsString;
    finally
      SQ.Free;
    end;
  finally
    SF.Free;
  end;
```
6 SQL context and code completion

TSQLContext class is used for creating context lists (code completion, etc.). It has several properties containing templates for different database objects - fields, tables, etc. (for template language description please refer to HTML Report Library manual) and methods for filling context list.

6.1 TSQLContext class

```pascal
procedure FillContext(const Query: TSQLSelectQuery; CaretPos: integer);
Fill items list with Query context at CarePos position

procedure AddTable(const T: TDMTable);
Add table into Items

procedure AddSequence(const S: TDMSequence);
Add sequence into Items

procedure AddTableAlias(const Alias, TableName: string);
Add table alias into Items

procedure AddField(const F: TDMField);
Add field without table alias into Items

procedure AddQueryField(const F: TDMField; const TableAlias: string = '');
Add field with table alias into Items

procedure AddReserved(const s: string);
Add reserved word into Items

procedure AddJoin(const FK: TDMForeignKey; ST: TSQLSelectStatement; LeftTable: TSQLTable);
Add join expression into Items

procedure AddColumn(const Name, Description: string; Index: integer);
Add column expression into Items

procedure AddFunction(const Name, Template: string);
Add function into Items

procedure AddFKTableValues(const Query: TSQLSelectQuery; const T: TDMTable; const FieldName: string);
Add values from foreign key table for given table and field.

property Items: TStringList
Completion items list

property TableTemplate: string
HTML template for Table

property ViewTemplate: string
HTML template for View

property FieldTemplate: string
HTML template for Field without table alias

property QueryFieldTemplate: string
```
6.2 Using SQL context

Example of filling SQL context list

```
Context := TSQLContext.Create(nil);
try
  SQ := TSQLSelectQuery.Create(nil);
  try
    SQ.TolerantMode := true;
    SQ.ParseString(Editor.Lines.Text);
    Context.FillContext(SQ, Editor.PosFromCaret(Editor.CaretX, Editor.CaretY) + 1);
  ....
  finally
    SQ.Free
  end
finally
  Context.Free
end
```

6.3 TSQLHLEditor class

TSQLHLEditorClass is a sample SQL editor with code completion implementation based on JVCL TJWideHLEditor component. It has the following members added:

```
procedure AddField(const AName: string);
Add column to query

procedure RemoveField(const AName: string);
Remove column from query
```
procedure OrderByField(const AName: string);
Add field into Order By list

property Query: TSQLSelectQuery
Editor Query object

property ErrorLine: integer read FErrorLine;
Error line (=1 if no errors found)

property ErrorMessage: string
Error message

property CompletionStyle: TStrings
CSS for completion list
7 How To

7.1 Get token at source position

Token at source position:

```pascal
var
  TokenIndex: integer;
  Query: TSQLSelectQuery;
  T: TSQLToken;
begin
  ...
  TokenIndex := Query.Tokenizer.GetTokenIndexAt(CaretPos);
  T := Query.Tokenizer.Tokens[TokenIndex];
  ...
```

Last non-space token at source position:

```pascal
var
  TokenIndex: integer;
  Query: TSQLSelectQuery;
  T: TSQLToken;
begin
  ...
  TokenIndex := Query.Tokenizer.GetNonSpaceTokenIndexAt(CaretPos);
  T := Query.Tokenizer.Tokens[TokenIndex];
  ...
```

7.2 Add table field to query columns

```pascal
var SQ: TSQLSelectQuery;
SF: TSQLFormatter;
begin
  SQ := TSQLSelectQuery.Create(nil);
  try
    SQ.ParseString(Editor.Lines.Text, SQLDialect);
    SQ.AddColumn(copy(FieldName, 1, pos('.', FieldName) - 1), copy(FieldName, pos('.', FieldName) + 1, MaxInt));
    SF := TSQLFormatter.Create(nil);
    try
      SQ.AsString(SF);
      Editor.Lines.Text := SF.AsString;
    finally
      SF.Free
    end;
  finally
    SQ.Free
  end;
```

7.3 Add condition to Where

```pascal
var SQ: TSQLSelectQuery;
SF: TSQLFormatter;
begin
  SQ := TSQLSelectQuery.Create(nil);
```
try
  SQ.ParseString(Editor.Lines.Text, SQLDialect);
  SQ.AddWhereCondition('order.price>0');
  SF := TSQLFormatter.Create(nil);
  try
    SQAsString(SF);
    Editor.Lines.Text := SF.AsString;
  finally
    SF.Free
  end;
finally
  SQ.Free
end;

7.4 Generate query for given table

Create query containing main table and all joined tables. Column list is generated using all non-FK columns from main table and TDMTable.LikelyNameColumn from joined tables.

SQ := TSQLSelectQuery.Create(nil);
try
  SQ.CreateQueryForTable('items', 1);
  SF := TSQLFormatter.Create(nil);
  try
    SQAsString(SF);
    Editor.Lines.Text := SF.AsString;
  finally
    SF.Free
  end;
finally
  SQ.Free
end;

7.5 Set row limit for query

var SQ: TSQLSelectQuery;
  SF: TSQLFormatter;
begin
  SQ := TSQLSelectQuery.Create(nil);
  try
    SQ.ParseSQL(Editor.Lines.Text, DefaultSQLDialect);
    if SQ.Statements[0].TopRowLimit = nil then
      SQ.Statements[0].TopRowLimit := TSQLTopRowLimit.Create(SQ.Statements[0]);
    SQ.Statements[0].TopRowLimit.ParseString('FIRST 100');
    SF := TSQLFormatter.Create(nil);
    try
      SQAsString(SF);
      Editor.Lines.Text := SF.AsString;
    finally
      SF.Free
    end;
  finally
    SQ.Free
  end;