Status of SQL and XML I/O

Sergey Linev,
GSI, Darmstadt, Germany

Outline

- Current development in SQL classes:
  - many improvements
  - new TSQLStatement class
  - new ODBC support
- TSQLFile class
  - functionality, table design
  - custom streamers support
- Improvements in XML I/O
- SQL I/O performance

SQL support in ROOT

- Since 7 years in ROOT there are classes:
  - TSQLServer – interface to SQL server
  - TSQLResult – result of SELECT queries
  - TSQLRow – access to single row in result
- Design was driven by text-based MySQL interface, not sufficient for real applications
- Implementations for:
  - MySQL, Oracle, PostgreSQL, SapDB

Improvements in SQL classes

- New methods of TSQLServer class:
  - Metainformation in DB-independent format:
    - GetTablesList() – list of tables
    - HasTable() – check if table exists
    - GetTableInfo() – provides information about table in TSQLTableInfo / TSQLColumnInfo objects
    - GetMaxIdentifierLength()
  - Transaction control:
    - StartTransaction(), Commit(), Rollback()
  - Error handling:
    - GetErrorCode(), GetErrorMsg(), EnableErrorOutput()
  - Exec() - for queries without results set
- Bug fixes and enhancement in Oracle classes
- Better support for different MySQL versions – now from 3.2.3 to 5.x

New TSQLStatement class

- Provides functionality to prepared SQL statements, supported by most of modern RDBMS
- Works with native data types: integer, double, date/time, string, null
- Introduces binary data support (BLOBs)
- Useful not only for SELECT, but also for INSERT queries
- Implemented for MySQL, Oracle and ODBC plugins
- Significant improvement in performance, especially for bulk operations, especially for Oracle (factor of 25 - 100)

ODBC support in ROOT

- ODBC – Open DataBase Connectivity
- Supported by most of RDBMS

- Implemented as new SQL plugin in ROOT
- Provides full functionality of TSQLServer class, including statements support
- Tested with MySQL and Oracle on Linux and Windows

New TSQLFile class

- Provides TFile interface to SQL database
- Uses only ROOT SQL classes
- Produces human-readable tables format
- Support most of TFile features:
  - schema evolution
  - subdirectory structure
  - custom streamers
  - the only exception – TTree class
- Tested with MySQL, Oracle, MyODBC, can be adjusted for other RDBMS

Short example with TBox class

```
class TBox: public TObject,
   public TAttLine, public TAttFill {
      Double_t fX1;
      Double_t fY1;
      Double_t fX2;
      Double_t fY2;
   ClassDef(TBox, 2);
};
```

```
{
   TSQLFile f("mysql://host.domain/test",
                "create","user","pass");
   for (int n=1;n<=10;n++) {
      TBox box(n,n*2,n*3,n*4);
      box.Write(Form("box%d",n));
   }
}
```

Creates tables:

- **TBox_ver2**
- **TObject_ver1**
- **TAttLine_ver1**
- **TAttFill_ver2**
- **KeysTable**
- **ObjectsTable**
- **Configurations**
- **IdsTable**

SELECT * FROM **TBox_ver2**

obj:id	TObject	TAttLine	TAttFill	fX1	fY1	fX2	fY2
1	1	1	1	1	2	3	4
2	1	1	1	2	4	6	8
3	1	1	1	3	6	9	12
4	1	1	1	4	8	12	16
5	1	1	1	5	10	15	20
6	1	1	1	6	12	18	24
7	1	1	1	7	14	21	28
8	1	1	1	8	16	24	32
9	1	1	1	9	18	27	36
10	1	1	1	10	20	30	40

TObject_ver1

obj:id	UniqueId	Bits	ProcessId
1	0	50331648	
2	0	50331648	
3	0	50331648	
4	0	50331648	
5	0	50331648	
6	0	50331648	
7	0	50331648	
8	0	50331648	
9	0	50331648	
10	0	50331648	

More complex example with TGraph

```
{
   TSQLFile f("mysql://host.domain/test",
                "update","user","pass");
   TGraph gr(10);
   for (int n=0;n<10;n++)
      gr.SetPoint(n, n+1, (n+1)*(n+1));
   gr.Write("gr");
}
```

TGraph_raw4

obj:id	raw:id	Field	Value
1	0	fX:Char_t	1
1	1	[0]:Double_t	1.000000
1	2	[1]:Double_t	2.000000
1	3	[2]:Double_t	3.000000
1	4	[3]:Double_t	4.000000
1	5	[4]:Double_t	5.000000
1	6	[5]:Double_t	6.000000
1	7	[6]:Double_t	7.000000
1	8	[7]:Double_t	8.000000
1	9	[8]:Double_t	9.000000
1	10	[9]:Double_t	10.000000
1	11	fY:Char_t	1
1	12	[0]:Double_t	1.000000
1	13	[1]:Double_t	4.000000
1	14	[2]:Double_t	9.000000
1	15	[3]:Double_t	16.000000
1	16	[4]:Double_t	25.000000
1	17	[5]:Double_t	36.000000
1	18	[6]:Double_t	49.000000
1	19	[7]:Double_t	64.000000
1	20	[8]:Double_t	81.000000
1	21	[9]:Double_t	100.000000

TGraph_ver4

parent classes

data members

obj:id	TNamed	TAttLine	TAttFill	TAttMarker	fNpoints	fX	fY	fFunctions	fHistogram	fMinimum	fMaximum
1	1	1	1	1	10	0	11	2	0	-1111	-1111

NULL pointer

TList_raw5

obj:id	raw:id	Field	Value
2	0	TList:Version	5
2	1	TObject:Version	1
2	2	UInt_t	0
2	3	UInt_t	50331648
2	4	UChar_t	0
2	5	Int_t	0

ObjectsTable

key:id	obj:id	Class	Version
10	1	TGraph	4
10	2	TList	5

External access to SQL tables

- Easy navigation with simple SELECT queries in any SQL browser
- One raw in table corresponds to one object
- Each column contains data of single class member
- Class name and version for each object can be found in *ObjectsTable*
- TSQLFile::MakeSelectQuery() produce SELECT statement, which aggregates data of object from different tables in one

Example with TBox class

Query, produced by f->MakeSelectQuery(TBox::Class()):

```
SELECT t1.`obj:id`, t2.UniqueId, t2.Bits, t2.ProcessId, t3.fLineColor, t3.fLineStyle,
t3.fLineWidth, t4.fFillColor, t4.fFillStyle, t1.fX1, t1.fY1, t1.fX2, t1.fY2
FROM TBox_ver2 AS t1 LEFT JOIN TObject_ver1 AS t2 USING(`obj:id`) LEFT JOIN
TAttLine_ver1 AS t3 USING(`obj:id`) LEFT JOIN TAttFill_ver1 AS t4 USING(`obj:id`)
```

	TObject			TAttLine			TAttFill		TBox			
obj:id	UniqueId	Bits	ProcessId	fLineColor	fLineStyle	fLineWidth	fFillColor	fFillStyle	fX1	fY1	fX2	fY2
1	0	50331648		1	1	1	19	1001	1	2	3	4
2	0	50331648		1	1	1	19	1001	2	4	6	8
3	0	50331648		1	1	1	19	1001	3	6	9	12
4	0	50331648		1	1	1	19	1001	4	8	12	16
5	0	50331648		1	1	1	19	1001	5	10	15	20
6	0	50331648		1	1	1	19	1001	6	12	18	24
7	0	50331648		1	1	1	19	1001	7	14	21	28
8	0	50331648		1	1	1	19	1001	8	16	24	32
9	0	50331648		1	1	1	19	1001	9	18	27	36
10	0	50331648		1	1	1	19	1001	10	20	30	40

Custom streamer support

- Important, while lot of ROOT and some user classes has custom streamers

- Two alternatives are possible:
  - produced by custom streamer data will be written without data member info into special tables like *TList_raw5* or *TGraph_raw4*
  - custom streamer can be instrumented in the way that object data will be stored in normal class table column-wise

- Even special case when custom streamer reads data written by standard I/O is supported (for instance, TStreamerElement, TStreamerInfo)

Example of custom streamer

Class definition:

```
class TTestClass : public TNamed {
  public:
    TTestClass(const char* name);
    virtual ~TTestClass();
    Int_t      fInt;
    Double_t fArr[3];
    TString    fStr;
  ClassDef(TTestClass, 1);
};
```

Entry in LinkDef.h:

```
...
#pragma link C++ class TTestClass-;
...
```

Custom streamer implementation:

```
void TTestClass::Streamer(TBuffer &b) {
  UInt_t R__s, R__c;
  if (b.IsReading()) {
    Version_t R__v = b.ReadVersion(&R__s, &R__c);
    TNamed::Streamer(b);
    b >> fInt;
    b.ReadFastArray(fArr, 3);
    fStr.Streamer(b);
    b.SetBufferOffset(R__s+R__c+sizeof(UInt_t));
  } else {
    UInt_t R__c = b.WriteVersion(TTestClass::Class(), kTRUE);
    TNamed::Streamer(b);
    b << fInt;
    b.WriteFastArray(fArr, 3);
    fStr.Streamer(b);
    b.SetByteCount(R__c, kTRUE);
  }
}
```

XML and SQL representation

Part of XML file:

```
<Object class="TTestClass">
 <Version v="1"/>
 <TNamed version="1">
  <TObject fUniqueID="0" fBits="3000000"/>
  <fName str="name2"/>
  <fTitle str="title1"/>
 </TNamed>
 <Int_t v="321"/>
 <Array>
  <Double_t v="11.110000"/>
  <Double_t v="22.220000"/>
  <Double_t v="33.330000"/>
 </Array>
 <UChar_t v="16"/>
 <CharStar v="&lt;&lt;string value&gt;&gt;"/>
</Object>
```

SELECT * FROM **TTestClass_raw1**

obj:id	raw:id	Field	Value
1	0	TTestClass:Version	1
1	1	TNamed:Version	1
1	2	ObjectInst	2
1	3	Int_t	321
1	4	[0]:Double_t	11.110000
1	5	[1]:Double_t	22.220000
1	6	[2]:Double_t	33.330000
1	7	UChar_t	16
1	8	CharStar	<<string value>>

Instrumented custom streamer

Instrumented custom streamer implementation:

```
void TTestClass::Streamer(TBuffer &b)
{
   UInt_t R__s, R__c;
   if (b.IsReading()) {
      Version_t R__v = b.ReadVersion(&R__s, &R__c);
      b.ClassBegin(TTestClass::Class(), R__v);
      b.ClassMember("TNamed");
      TNamed::Streamer(b);
      b.ClassMember("fInt","Int_t");
      b >> fInt;
      b.ClassMember("fArr","Double_t", 3);
      b.ReadFastArray(fArr, 3);
      b.ClassMember("fStr","TString");
      fStr.Streamer(b);
      b.ClassEnd(TTestClass::Class());
      b.SetBufferOffset(R__s+R__c+sizeof(UInt_t));
   } else {
      TTestClass::Class()->WriteBuffer(b,this);
   }
}
```

Meta-information about class itself and class members

In this case standard store procedure can be used

XML and SQL representation

Part of XML file:

```
<Object class="TTestClass">
  <Version v="1"/>
  <TNamed version="1">
    <TObject fUniqueID="0" fBits="3000000"/>
    <fName str="name2"/>
    <fTitle str="title1"/>
  </TNamed>
  <Int_t v="321"/>
  <Array>
    <Double_t v="11.110000"/>
    <Double_t v="22.220000"/>
    <Double_t v="33.330000"/>
  </Array>
  <UChar_t v="16"/>
  <CharStar v="&lt;&lt;string value&gt;&gt;"/>
</Object>
  </TTestClass>
</Object>
```

SELECT * FROM **TTestClass_ver1**

obj:id	TNamed:parent	fInt:Int_t	fArr[0]	fArr[1]	fArr[2]	fStr:str
1	1	321	11.11	22.22	33.33	<<string value>>

SELECT * FROM **TTestClass_raw1**

obj:id	raw:id	Field	Value
1	0	TTestClass:Version	1
1	1	TNamed:Version	1
1	2	ObjectInst	2
1	3	Int_t	321
1	4	[0]:Double_t	11.110000
1	5	[1]:Double_t	22.220000
1	6	[2]:Double_t	33.330000
1	7	UChar_t	16
1	8	CharStar	<<string value>>

Enhancements of XML I/O

- TXMLFile class since 2004 in ROOT
- Provides possibility to store in XML file arbitrary type of objects (beside TTree)

- New functionality of TXMLFile:
  - subdirectories structures
  - instrumented custom streamers
  - better support of TClonesArray
  - xml comments and style sheets

Converting objects to/from XML

- New static methods of TBufferXML class:
  - ConvertToXML() – store object data in from of XML structures
  - ConvertFromXML() – reconstruct object from XML structures
- Easy method to convert any single object into / from text representation
- Can be use to store any object in form of CINT macro

Converts any object into macro

```
void make_script(TObject* obj, const char* funcname)
{
  TString xmlbuf = TBufferXML::ConvertToXML(obj);

  xmlbuf.ReplaceAll("\"","\\\"");
  xmlbuf.ReplaceAll("\n","\";\n  sbuf+=\"");

  std::ofstream fs(TString(funcname)+".C");

  fs << obj->ClassName() << "* " << funcname
     << "()\n" "{\n" "  TString sbuf = \""
     << xmlbuf << "\";\n"
     << "  return (" << obj->ClassName() << "*)"
     << "TBufferXML::ConvertFromXML(sbuf);\n}\n";
}
```

Convert TBox class into macro

1. Generate macro:

```
...
root [0] TBox b(1, 2, 3, 4);
root [1] make_script(&b, "make_box");
...
```

2. Produced file make_box.C:

```
TBox* make_box()
{
 TString sbuf = "<Object class=\"TBox\">";
 sbuf+="  <TBox version=\"2\">";
 sbuf+="    <TObject fUniqueID=\"0\" fBits=\"3000000\"/>";
 sbuf+="    <TAttLine version=\"1\">";
 sbuf+="      <fLineColor v=\"1\"/>";
 sbuf+="      <fLineStyle v=\"1\"/>";
 sbuf+="      <fLineWidth v=\"1\"/>";
 sbuf+="    </TAttLine>";
 sbuf+="    <TAttFill version=\"1\">";
 sbuf+="      <fFillColor v=\"19\"/>";
 sbuf+="      <fFillStyle v=\"1001\"/>";
 sbuf+="    </TAttFill>";
 sbuf+="    <fX1 v=\"1.000000\"/>";
 sbuf+="    <fY1 v=\"2.000000\"/>";
 sbuf+="    <fX2 v=\"3.000000\"/>";
 sbuf+="    <fY2 v=\"4.000000\"/>";
 sbuf+="  </TBox>";
 sbuf+="</Object>";
 sbuf+="";
 return (TBox*) TBufferXML::ConvertFromXML(sbuf);
}
```

3. Reconstruct object:

```
...
root [0]   .L make_box.C
root [1]   b = make_box();
root [2]   b->Draw();
...
```

SQL I/O performance

- Two aspects:
  - Time and CPU usage on user host
  - Quality and number of SQL statements

```
// TFile f(“test.root”,”recreate”);
// TXMLFile f(“test.xml”,”recreate”);

TSQLFile f(“mysql://host.domain/test”,
           “recreate”,”user”,”pass”);
TClonesArray clones("TBox", 10000);
for(int n=0;n<10000;n++)
   new (clones[n]) TBox(n+1,n+2,n+3,n+4);
clones.Write("clones0", TObject::kSingleKey);

gBenchmark->Start(“Write");
clones.Write("clones",TObject::kSingleKey);
gBenchmark->Show(“Write");
```

```
// TFile f(“test.root”,”read”);
// TXMLFile f(“test.xml”,”read”);

TSQLFile f(“mysql://host.domain/test”,
           “read”,”user”,”pass”);
TClonesArray* clon = 0, *clon0 = 0;
f.GetObject("clones0", clon0);

gBenchmark->Start(“Read");
f.GetObject("clones", clon);
gBenchmark->Show(“Read");
```

Performance measurements

	Writing			Reading		
	CPU, s	Real, s	Queries / statements	CPU, s	Real, s	Queries / statements
Binary	0.12	0.12	-	0.04	0.04	-
XML	1.1	1.1	-	1.0	1.0	-
MySQL 5.0	5.9	8.1	95 / 0*	2.1	2.8	6 / 2
Oracle 10g	3.2	5.7	45 / 6	3.8	4.1	6 / 2
MyODBC 3.51**	8.7	30.1	45 / 6	5.7	6.5	6 / 2

* Only text queries are used for MySQL, while SQL statement works slower on MySQL compare to multiple INSERT query

** Same MySQL 5.0 server is used. Can be seen, that using statements while writing significantly increases time

To do:

- SQL performance improvements
  - avoid text conversions where it possible
  - use statements where it make sense
  - use BLOBs for custom data
- Adopt TSQLStatement in TTreeSQL class?
- Support of large (more 1Gb) XML files?
- Any missing features in XML / SQL classes?