Subject: Re: Access to S_* Structure of TABLE crash Application.
Posted by Sender Ghost on Thu, 17 May 2012 05:03:33 GMT
View Forum Message <> Reply to Message

Hello, Sergey.
sergeynikitin wrote on Thu, 17 May 2012 02:21call SQL[row.NAME] throw Assertion. What I'm wrong???

It throws NEVER assertion inside uppsrc/Sql/Sql.cpp file, because the S_* structure is not intended to be used like this.

To understand this, just preprocess reference/SQL_Sqlite3/simple.cpp file:
Toggle Spoiler

```
struct S_SIMPLE_TEST1
{
public:
 void Shrink();
 void Clear();
 static const char TableName[];
 static const SqlSet& ColumnSet();
 static SqlSet ColumnSet(const String& prefix);
 static SqlSet Of(SqlId table);
 void FieldLayoutRaw(FieldOperator& f, const String& prefix = String());
 void FieldLayout(FieldOperator& f);

 operator Fields()
 {
  return callback(this, &S_SIMPLE_TEST1::FieldLayout);
 }

 bool operator== (const S_SIMPLE_TEST1& x) const
 {
  return EqualFields(const_cast<S_SIMPLE_TEST1&>(*this),
const_cast<S_SIMPLE_TEST1&>(x));
 }

 bool operator!= (const S_SIMPLE_TEST1& x) const
 {
  return !EqualFields(const_cast<S_SIMPLE_TEST1&>(*this),
const_cast<S_SIMPLE_TEST1&>(x));
 }

 String ToString() const
 {
  return AsString((Fields) const_cast<S_SIMPLE_TEST1&>(*this));
 }
```

```
 S_SIMPLE_TEST1();

 enum { ID_WIDTH = 0, ID_PRECISION = 0 };
 int ID;
 enum { NAME_WIDTH = 0, NAME_PRECISION = 0 };
 String NAME;
 enum { LASTNAME_WIDTH = 0, LASTNAME_PRECISION = 0 };
 String LASTNAME;
 enum { BDATE_WIDTH = 0, BDATE_PRECISION = 0 };
 int BDATE;
};

extern SqlId SIMPLE_TEST1;
extern SqlId ID;
extern SqlId NAME;
extern SqlId LASTNAME;
extern SqlId BDATE;

static void SCHEMA_SIMPLE_TEST1(SqlSchema& schema)
{
 schema.Column("integer", "ID");
 schema.InlineAttribute("primary key");
 schema.Column("text", "NAME");
 schema.Column("text", "LASTNAME");
 schema.Column("integer", "BDATE");
}

void TABLE_SIMPLE_TEST1(SqlSchema& schema)
{
 schema.Table("SIMPLE_TEST1");
 SCHEMA_SIMPLE_TEST1(schema);
 schema.EndTable();
}

static void All_Tables(SqlSchema& schema)
{
 TABLE_SIMPLE_TEST1(schema);
}

SqlId SIMPLE_TEST1("SIMPLE_TEST1");
SqlId ID("ID");
SqlId NAME("NAME");
SqlId LASTNAME("LASTNAME");
SqlId BDATE("BDATE");

S_SIMPLE_TEST1::S_SIMPLE_TEST1()
{
 SqlSchemaInitClear(ID);
```

```
 SqlSchemaInitClear(NAME);
 SqlSchemaInitClear(LASTNAME);
 SqlSchemaInitClear(BDATE);
}

const char S_SIMPLE_TEST1::TableName[] = "SIMPLE_TEST1";
const SqlSet& S_SIMPLE_TEST1::ColumnSet()
{
 static SqlSet set = ColumnSet("");
 return set;
}

SqlSet S_SIMPLE_TEST1::Of(SqlId id)
{
 return ColumnSet(id.ToString() + '.');
}

SqlSet S_SIMPLE_TEST1::ColumnSet(const String& prefix)
{
 SqlSet set;
 td_scalar(set, prefix, "ID");
 td_scalar(set, prefix, "NAME");
 td_scalar(set, prefix, "LASTNAME");
 td_scalar(set, prefix, "BDATE");
 return set;
}

void S_SIMPLE_TEST1::Clear()
{
 SqlSchemaClear(ID);
 SqlSchemaClear(NAME);
 SqlSchemaClear(LASTNAME);
 SqlSchemaClear(BDATE);
}

void S_SIMPLE_TEST1::FieldLayout(FieldOperator& fo)
{
 fo.Table("SIMPLE_TEST1");
 FieldLayoutRaw(fo);
}

void S_SIMPLE_TEST1::FieldLayoutRaw(FieldOperator& fo, const String& prefix)
{
 fo(prefix + "ID", ID);
 fo(prefix + "NAME", NAME);
 fo(prefix + "LASTNAME", LASTNAME);
 fo(prefix + "BDATE", BDATE);
}
```

```
void SchDbInfoSIMPLE_TEST1()
{
 SchDbInfoColumn("ID");
 SchDbInfoPrimaryKey();
 SchDbInfoColumn("NAME");
 SchDbInfoColumn("LASTNAME");
 SchDbInfoColumn("BDATE");
}

struct SINS_SIMPLE_TEST1_
{
 SINS_SIMPLE_TEST1_();
} SINS_SIMPLE_TEST1__;

SINS_SIMPLE_TEST1_::SINS_SIMPLE_TEST1_()
{
 SchDbInfoTable("SIMPLE_TEST1");
 SchDbInfoSIMPLE_TEST1();
}
```

The row.ID, row.NAME, row.LASTNAME and row.BDATE are actual variables with int and String types, initialized to "empty" values. Therefore, when you call SQL[row.NAME], actually you call: SQL[SqlId(String(row.NAME))] -> SQL[SqlId("")] -> there are no columns with such a name -> NEVER assertion.
In case of SQL[row.ID]:
SQL[int(row.ID)] -> SQL[Null] -> ASSERT(NULL != current_stmt).

And this is easy to check with following source code:

```
RDUMP(IsNull(row.ID));
RDUMP(IsNull(row.NAME));
RDUMP(IsNull(row.LASTNAME));
RDUMP(IsNull(row.BDATE));
```

With following results:

```
IsNull(row.ID) = true
IsNull(row.NAME) = true
IsNull(row.LASTNAME) = true
IsNull(row.BDATE) = true
```