Subject: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Thu, 17 May 2012 00:21:31 GMT
View Forum Message <> Reply to Message

I try to access field by the expression:

row.NAME
row.ID

TABLE defenition: TABLE_(SIMPLE_TEST1)
  INT_   (ID) PRIMARY_KEY
  STRING_ (NAME, 200)
  STRING_ (LASTNAME, 200)
  INT_   (BDATE)
END_TABLE

call SQL[row.NAME] throw Assertion. What I'm wrong???

Testcase:
Package: /upp/reference/SQL_Sqlite3
File: simple.cpp

Fragment of modified code (lines 83-90):

```
 // Test selection:
 sql*Select(row).From(SIMPLE_TEST1);
 LOG(sql.ToString());
 while (sql.Fetch()) {
  LOG(Format("%d %s %s %d",sql[0],sql[1],sql[2],sql[3]));
  LOG(Format("%s %s %s %s",sql[ID],sql[NAME],sql[LASTNAME],sql[BDATE]));
//My added line vvvvvvvvvvvvv                    vvv       vvv       vvv            vvv
  LOG(Format("error line: %s %s %s
%s",sql[row.ID],sql[row.NAME],sql[row.LASTNAME],sql[row.BDATE]));
 }
```

Also Source of this example in attachment

PS
It looks like this problem come after entering SVO_VALUE.

PSPS
For temporary decision of this problem I use alternative scheme, like this:TABLE_(SIMPLE_TEST1)
  INT_   (row_ID) PRIMARY_KEY
  STRING_ (row_NAME, 200)
  STRING_ (row_LASTNAME, 200)

```
  INT_    (row_BDATE)
END_TABLE
```

File Attachments

1) SQL_Sqlite3.zip, downloaded 773 times

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

Subject: Re: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Thu, 17 May 2012 05:53:01 GMT
View Forum Message <> Reply to Message

Therefore it is logical error!

For example, there are two tables :

```
TABLE_(WORKER)
  INT_    (ID) PRIMARY_KEY
  STRING_  (NAME, 200)
  STRING_  (LASTNAME, 200)
  INT_    (PLANT_ID)
END_TABLE


TABLE_(PLANT)
  INT_    (ID) PRIMARY_KEY
  STRING_  (NAME, 200)
  STRING_  (ADDRESS, 200)
END_TABLE
```

I think, that S_* structure created to use in this situation, to decide this collision.

```
S_WORKER w;
S_PLANT p;

SQL * Select
(w.NAME,w.LASTNAME,p.NAME,p.ADDRESS).FROM(WORKER).LeftJoin(PLANT).On(w.PLANT
_ID==p.ID);

while SQL.Fetch() {
  Cout() << SQL[w.NAME] << SQL[w.LASTNAME] << SQL[p.NAME] << SQL[p.ADDRESS];
}
```

In other case S_-structure not help. I can write this code without S_struct more simple.

Subject: Re: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Thu, 17 May 2012 06:27:34 GMT
View Forum Message <> Reply to Message

In some toolkit I can write like:

```
TABLE_(WORKER) PREFIX(wrk)
  INT_    (ID) PRIMARY_KEY
  STRING_ (NAME, 200)
  STRING_ (LASTNAME, 200)
  INT_    (PLANT_ID)
END_TABLE


TABLE_(PLANT) PREFIX(pla)
  INT_    (ID) PRIMARY_KEY
  STRING_ (NAME, 200)
  STRING_ (ADDRESS, 200)
END_TABLE
```

And than I can use simple pla.NAME and wrk.NAME (or pla::NAME and wrk::NAME, or pla_NAME and wrk_NAME).

Now I must do like:

```
TABLE_(WORKER)
  INT_    (wrk_ID) PRIMARY_KEY
  STRING_ (wrk_NAME, 200)
  STRING_ (wrk_LASTNAME, 200)
  INT_    (wrk_PLANT_ID)
END_TABLE


TABLE_(PLANT) PREFIX(pla)
  INT_    (pla_ID) PRIMARY_KEY
  STRING_ (pla_NAME, 200)
  STRING_ (pla_ADDRESS, 200)
END_TABLE
```

Subject: Re: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Thu, 17 May 2012 06:28:52 GMT
View Forum Message <> Reply to Message

In some toolkit I can write like:

```
TABLE_(WORKER) PREFIX(wrk)
  INT_    (ID) PRIMARY_KEY
  STRING_ (NAME, 200)
  STRING_ (LASTNAME, 200)
```

```
  INT_    (PLANT_ID)
END_TABLE


TABLE_(PLANT) PREFIX(pla)
  INT_    (ID) PRIMARY_KEY
  STRING_  (NAME, 200)
  STRING_  (ADDRESS, 200)
END_TABLE
```

And than I can use simple pla.NAME and wrk.NAME (or pla::NAME and wrk::NAME, or pla_NAME and wrk_NAME).

Now I must do like:

```
TABLE_(WORKER)
  INT_    (wrk_ID) PRIMARY_KEY
  STRING_  (wrk_NAME, 200)
  STRING_  (wrk_LASTNAME, 200)
  INT_    (wrk_PLANT_ID)
END_TABLE


TABLE_(PLANT) PREFIX(pla)
  INT_    (pla_ID) PRIMARY_KEY
  STRING_  (pla_NAME, 200)
  STRING_  (pla_ADDRESS, 200)
END_TABLE
```

Subject: Re: Access to S_* Structure of TABLE crash Application.
Posted by Sender Ghost on Thu, 17 May 2012 07:59:52 GMT
View Forum Message <> Reply to Message

sergeynikitin wrote on Thu, 17 May 2012 07:53For example, there are two tables:

```
TABLE_(WORKER)
  INT_    (ID) PRIMARY_KEY
  STRING_  (NAME, 200)
  STRING_  (LASTNAME, 200)
  INT_    (PLANT_ID)
END_TABLE

TABLE_(PLANT)
  INT_    (ID) PRIMARY_KEY
  STRING_  (NAME, 200)
  STRING_  (ADDRESS, 200)
END_TABLE
```

Correct version of tables:

```
TABLE_(WORKER)
  INT_    (ID) PRIMARY_KEY
  STRING_  (NAME, 200)
  STRING_  (LASTNAME, 200)
  INT_    (PLANT_ID)
END_TABLE

TABLE_(PLANT)
  INT    (ID) PRIMARY_KEY
  STRING  (NAME, 200)
  STRING_ (ADDRESS, 200)
END_TABLE
```

The INT_(ID) macro creates new extern SqlId ID;. The same with other *_ macro versions. If you want to use the same SqlId, there are macros without '_' postfix.
Quote:
I think, that S_* structure created to use in this situation, to decide this collision.

```
S_WORKER w;
S_PLANT p;

SQL * Select
(w.NAME,w.LASTNAME,p.NAME,p.ADDRESS).FROM(WORKER).LeftJoin(PLANT).On(w.PLANT
_ID==p.ID);

while SQL.Fetch() {
  Cout() << SQL[w.NAME] << SQL[w.LASTNAME] << SQL[p.NAME] << SQL[p.ADDRESS];
}
```

Correct version of source code, might look like:

```
LOG("Inserting values:");
SQL * Insert(WORKER)(ID, 0)(NAME, "Joe")(LASTNAME, "Smith")(PLANT_ID, 0);
LOG(SQL.ToString());
SQL * Insert(WORKER)(ID, 1)(NAME, "Mike")(LASTNAME, "Smith")(PLANT_ID, 0);
LOG(SQL.ToString());
SQL * Insert(WORKER)(ID, 2)(NAME, "Jon")(LASTNAME, "Goober")(PLANT_ID, 1);
LOG(SQL.ToString());

SQL * Insert(PLANT)(ID, 0)(NAME, "First Plant")(ADDRESS, "First st.");
LOG(SQL.ToString());
SQL * Insert(PLANT)(ID, 1)(NAME, "Second Plant")(ADDRESS, "Second st.");
```

```
LOG(SQL.ToString());

LOG("Selecting values:");
SQL * Select(WORKER(NAME, LASTNAME), PLANT(NAME,
ADDRESS)).From(WORKER).LeftJoin(PLANT).On(WORKER(PLANT_ID) == PLANT(ID));
LOG(SQL.ToString());
while (SQL.Fetch()) {
 LOG("-----");
 DUMP(SQL[NAME]); DUMP(SQL[LASTNAME]);
 DUMP(SQL[2]); DUMP(SQL[3]);
}
```

With following output:

```
Inserting values:
insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (0, 'Joe', 'Smith', 0)
insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (1, 'Mike', 'Smith', 0)
insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (2, 'Jon', 'Goober', 1)
insert into PLANT(ID, NAME, ADDRESS) values (0, 'First Plant', 'First st.')
insert into PLANT(ID, NAME, ADDRESS) values (1, 'Second Plant', 'Second st.')
Selecting values:
select WORKER.NAME, WORKER.LASTNAME, PLANT.NAME, PLANT.ADDRESS from
WORKER left outer join PLANT on WORKER.PLANT_ID = PLANT.ID
-----
SQL[NAME] = Joe
SQL[LASTNAME] = Smith
SQL[2] = First Plant
SQL[3] = First st.
-----
SQL[NAME] = Mike
SQL[LASTNAME] = Smith
SQL[2] = First Plant
SQL[3] = First st.
-----
SQL[NAME] = Jon
SQL[LASTNAME] = Goober
SQL[2] = Second Plant
SQL[3] = Second st.
```

Subject: Re: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Thu, 17 May 2012 08:51:16 GMT
View Forum Message <> Reply to Message

OK! Please say from what table DUMP(SQL[NAME]) read data in read cycle (I say about monosemanticity and single meaning).

Sender Ghost wrote on Thu, 17 May 2012 11:59

```
SQL * Select(WORKER(NAME, LASTNAME), PLANT(NAME,
ADDRESS)).From(WORKER).LeftJoin(PLANT).On(WORKER(PLANT_ID) == PLANT(ID));
LOG(SQL.ToString());
while (SQL.Fetch()) {
 DUMP(SQL[NAME]); DUMP(SQL[LASTNAME]);
 DUMP(SQL[2]); DUMP(SQL[3]);
 LOG("-----");
}
```

Where went easily-readable code U++?

In this case it is better to write
WRK_NAME and PLA_NAME
(Exceptions are cases when not myself create the Database, and connected into the already existing DataBase.)
.

PS
By the way, sorry for the tone (if it seemed overpriced).
Nothing personal. I treat you with great respect.

Subject: Re: Access to S_* Structure of TABLE crash Application.
Posted by Sender Ghost on Thu, 17 May 2012 09:32:16 GMT
View Forum Message <> Reply to Message

sergeynikitin wrote on Thu, 17 May 2012 10:51
OK! Please say from what table DUMP(SQL[NAME]) read data in read cycle

I think, it depends from current select. Based on the logs, by the selection order:

SQL[0] from WORKER.NAME
SQL[1] from WORKER.LASTNAME
SQL[2] from PLANT.NAME
SQL[3] from PLANT.ADDRESS

Therefore, SQL[0] == SQL[NAME] from WORKER table here. It's like finding index by the SqlId name.
sergeynikitin wrote on Thu, 17 May 2012 10:51
Where went easily-readable code U++?

In this case it is better to write

WRK_NAME and PLA_NAME
(Exceptions are cases when not myself create the Database, and connected into the already existing DataBase.)
.

Agreed, this might help also. I just did it with my current understanding of the U++ source code.
sergeynikitin wrote on Thu, 17 May 2012 10:51
PS
By the way, sorry for the tone (if it seemed overpriced).
Nothing personal. I treat you with great respect.

No problem. Interesting topic.

Subject: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Thu, 17 May 2012 12:49:02 GMT
View Forum Message <> Reply to Message

PROPOSAL::New Table Architecture()
Maybe enclose TABLE into separate class, and NAME clashes automatically gone away? And then we can use this naming schema:
PLA::NAME and WRK::NAME.....

And then we will can extend the class with various access methods and control various aspect of access like link integrity and transactions for business logic.

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by Sender Ghost on Fri, 18 May 2012 18:35:15 GMT
View Forum Message <> Reply to Message

sergeynikitin wrote on Thu, 17 May 2012 14:49Maybe enclose TABLE into separate class, and NAME clashes automatically gone away?
No. It depends on how concrete database works. In case of SQLite, the U++ wrapper uses sqlite3_column_name function to assign column names. From the documentation of sqlite3_column_name function we see:
Column Names In A Result Set[...]
These routines return the name assigned to a particular column in the result set of a SELECT statement.
[...]
The name of a result column is the value of the "AS" clause for that column, if there is an AS clause. If there is no AS clause then the name of the column is unspecified and may change from one release of SQLite to the next.
[...]

Therefore, to resolve column name clashes the user need to specify "AS" clause for clashing column names.

The correct example will look like follows:

```
LOG("Inserting values:");
SQL * Insert(WORKER)(ID, 0)(NAME, "Joe")(LASTNAME, "Smith")(PLANT_ID, 0);
LOG(SQL.ToString());
SQL * Insert(WORKER)(ID, 1)(NAME, "Mike")(LASTNAME, "Smith")(PLANT_ID, 0);
LOG(SQL.ToString());
SQL * Insert(WORKER)(ID, 2)(NAME, "Jon")(LASTNAME, "Goober")(PLANT_ID, 1);
LOG(SQL.ToString());

SQL * Insert(PLANT)(ID, 0)(NAME, "First Plant")(ADDRESS, "First st.");
LOG(SQL.ToString());
SQL * Insert(PLANT)(ID, 1)(NAME, "Second Plant")(ADDRESS, "Second st.");
LOG(SQL.ToString());

LOG("Selecting values:");
SQLID(PLANT_NAME);
SQL * Select(WORKER(NAME, LASTNAME), PLANT(NAME).As(PLANT_NAME),
PLANT(ADDRESS)).From(WORKER).LeftJoin(PLANT).On(WORKER(PLANT_ID) ==
PLANT(ID));
LOG(SQL.ToString());
while (SQL.Fetch()) {
 LOG("-----");
 DUMP(SQL[NAME]); DUMP(SQL[LASTNAME]);
 DUMP(SQL[PLANT_NAME]); DUMP(SQL[ADDRESS]);
}
```

With following output:

```
Inserting values:
insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (0, 'Joe', 'Smith', 0)
insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (1, 'Mike', 'Smith', 0)
insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (2, 'Jon', 'Goober', 1)
insert into PLANT(ID, NAME, ADDRESS) values (0, 'First Plant', 'First st.')
insert into PLANT(ID, NAME, ADDRESS) values (1, 'Second Plant', 'Second st.')
Selecting values:
select WORKER.NAME, WORKER.LASTNAME, PLANT.NAME PLANT_NAME,
PLANT.ADDRESS from WORKER left outer join PLANT on WORKER.PLANT_ID = PLANT.ID
-----
SQL[NAME] = Joe
SQL[LASTNAME] = Smith
SQL[PLANT_NAME] = First Plant
SQL[ADDRESS] = First st.
-----
SQL[NAME] = Mike
SQL[LASTNAME] = Smith
SQL[PLANT_NAME] = First Plant
SQL[ADDRESS] = First st.
```

SQL[NAME] = Jon
SQL[LASTNAME] = Goober
SQL[PLANT_NAME] = Second Plant
SQL[ADDRESS] = Second st.

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Fri, 18 May 2012 19:27:43 GMT
View Forum Message <> Reply to Message

A colorful example of how we ourselves create artificial boundaries.

There are two types of SQL variables - variables tied to a table and dynamic variables created for ... or inside the query.

We just need to consider two cases.
1. The variables from the table.
2. The dynamic variables.

Otherwise, we are forced to write for each table, in which there is a field NAME or ID some artificial design.

Developing your idea, that the dynamic variables (fields) you can't make tied to the table, or give it another name, We just Need to let the variable is not tied to the table.

Given a STRING structure of SQL statements will be quite easy to achieve this.

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by Sender Ghost on Fri, 18 May 2012 20:27:55 GMT
View Forum Message <> Reply to Message

sergeynikitin wrote on Fri, 18 May 2012 21:27A colorful example of how we ourselves create artificial boundaries.
Yes, my conclusions based on current implementation. But I didn't say, that this is not possible in principle.
I showed real example about how to achieve this with current tools.
Quote:Otherwise, we are forced to write for each table, in which there is a field NAME or ID some artificial design.

I think, no. It's normal to write as follows:

SQL * Select(ID, NAME, LASTNAME, PLANT_ID).From(WORKER);

```
LOG(SQL.ToString());
while (SQL.Fetch()) {
 LOG("-----");
 DUMP(SQL[ID]);
 DUMP(SQL[NAME]);
 DUMP(SQL[LASTNAME]);
 DUMP(SQL[PLANT_ID]);
}

SQL * Select(ID, NAME, ADDRESS).From(PLANT);
LOG('\n' << SQL.ToString());
while (SQL.Fetch()) {
 LOG("-----");
 DUMP(SQL[ID]);
 DUMP(SQL[NAME]);
 DUMP(SQL[ADDRESS]);
}
```

With following output:

```
select ID, NAME, LASTNAME, PLANT_ID from WORKER
-----
SQL[ID] = 0
SQL[NAME] = Joe
SQL[LASTNAME] = Smith
SQL[PLANT_ID] = 0
-----
SQL[ID] = 1
SQL[NAME] = Mike
SQL[LASTNAME] = Smith
SQL[PLANT_ID] = 0
-----
SQL[ID] = 2
SQL[NAME] = Jon
SQL[LASTNAME] = Goober
SQL[PLANT_ID] = 1

select ID, NAME, ADDRESS from PLANT
-----
SQL[ID] = 0
SQL[NAME] = First Plant
SQL[ADDRESS] = First st.
-----
SQL[ID] = 1
SQL[NAME] = Second Plant
SQL[ADDRESS] = Second st.
```

For cases with column name clashes there is "AS" clause. And even without this, it still possible to

access result set through indexes.

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Fri, 18 May 2012 20:42:19 GMT
View Forum Message <> Reply to Message

Another example:
SQL * Select(WORKER(NAME, LASTNAME), PLANT(NAME).As(PLANT_NAME), PLANT(ADDRESS)).From(WORKER).LeftJoin(PLANT).On(WORKER(PLANT_ID) == PLANT(ID));

This expression generates the SQL statement:select WORKER.NAME, WORKER.LASTNAME, PLANT.NAME PLANT_NAME, PLANT.ADDRESS from WORKER left outer join PLANT on WORKER.PLANT_ID = PLANT.ID

Attention to PLANT.NAME and PLANT_NAME!!!
PLANT_NAME really not needed!!!.
We can get data from column PLANT.NAME by name PLANT.NAME or by name NAME.Of(PLANT).
I propose only more useful form is :
PLANT::NAME

If We enclose TABLE in separate class, We exclude more of name clashes (really absent in SQL).

This is not meant to prohibit the SQL column without the class prefix.
Should coexist both forms.

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Fri, 18 May 2012 21:22:00 GMT
View Forum Message <> Reply to Message

By the way, when We use images from iml, We go same way:
CtrlLib::PlusSign()

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by Sender Ghost on Fri, 18 May 2012 21:31:36 GMT
View Forum Message <> Reply to Message

sergeynikitin wrote on Fri, 18 May 2012 22:42Another example:
SQL * Select(WORKER(NAME, LASTNAME), PLANT(NAME).As(PLANT_NAME), PLANT(ADDRESS)).From(WORKER).LeftJoin(PLANT).On(WORKER(PLANT_ID) == PLANT(ID));

This expression generates the SQL statement:select WORKER.NAME, WORKER.LASTNAME, PLANT.NAME PLANT_NAME, PLANT.ADDRESS from WORKER left outer join PLANT on WORKER.PLANT_ID = PLANT.ID

Attention to PLANT.NAME and PLANT_NAME!!!

Yes. But, as I said about sqlite3_column_name function, it returns exactly:

NAME
LASTNAME
PLANT_NAME
ADDRESS

If it returns following columns:

WORKER.NAME
WORKER.LASTNAME
PLANT_NAME
PLANT.ADDRESS

It will be possible to access it with:

```
while (SQL.Fetch()) {
 DUMP(SQL["WORKER.NAME"]); DUMP(SQL["WORKER.LASTNAME"]);
 DUMP(SQL["PLANT_NAME"]); DUMP(SQL["PLANT.ADDRESS"]);
}
```

But it is not.

In case of:

```
SQL * Select(WORKER(NAME, LASTNAME), PLANT(NAME,
ADDRESS)).From(WORKER).LeftJoin(PLANT).On(WORKER(PLANT_ID) == PLANT(ID));

while (SQL.Fetch()) {
 DUMP(SQL["NAME"]); DUMP(SQL["LASTNAME"]);
 DUMP(SQL[2]); DUMP(SQL["ADDRESS"]);
}
```

It returns following columns:

NAME
LASTNAME
NAME
ADDRESS

Therefore, it is not possible to access second "NAME" column through simple search algorithm.

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Sat, 19 May 2012 00:02:06 GMT
View Forum Message <> Reply to Message

You're talking not about that. I was talking about something else (read my previous post). I do not understand how the subject is connected with the function of sqlite3_column_name. Talk about that (even in SQLite) you can execute the query

SELECT PLANT.NAME FROM PLANT
and it will be correctly understood by the SQL driver.
(By the way, - try make DUMP of NAME.Of(PLANT))

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by Sender Ghost on Sat, 19 May 2012 01:12:10 GMT
View Forum Message <> Reply to Message

sergeynikitin wrote on Sat, 19 May 2012 02:02Talk about that (even in SQLite) you can execute the query

SELECT PLANT.NAME FROM PLANT
and it will be correctly understood by the SQL driver.
(By the way, - try make DUMP of NAME.Of(PLANT))

```
SQL * Select(PLANT(NAME)).From(PLANT);
LOG(SQL.ToString());
while (SQL.Fetch()) {
 DUMP(SQL.GetColumnInfo(0).name); // returns "NAME"
 DUMP(NAME.Of(PLANT)); // returns PLANT\nNAME
 DUMP(SQL[NAME]); // returns correct value, associated with "NAME" column
 DUMP(SQL[0]); // returns previous value
 DUMP(SQL[NAME.Of(PLANT)]); // "Assertion failed" here
}
```

With following output:

```
select PLANT.NAME from PLANT
SQL.GetColumnInfo(0).name = NAME
NAME.Of(PLANT) = PLANT
NAME
SQL[NAME] = First Plant
SQL[0] = First Plant
Assertion failed in C:\upp\uppsrc\Sql\Sql.cpp, line 339
0
```

sergeynikitin wrote on Sat, 19 May 2012 02:02I do not understand how the subject is connected with the function of sqlite3_column_name

The subject is connected with incorrect using of S_* structure, which throws "Assertion failed", because of requested empty column. The sqlite3_column_name function is related to SQLite database and how U++ wrapper uses it to get result set. Therefore, I said, that it is dependent from concrete database.

I think, you are free to propose your changes, if you want. So, U++ developers, which responsible to SQL parts of U++, might analyse it and return the answer to you.

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by sergeynikitin on Sat, 19 May 2012 01:28:59 GMT
View Forum Message <> Reply to Message

Sender Ghost wrote on Sat, 19 May 2012 05:12
select PLANT.NAME from PLANT
SQL.GetColumnInfo(0).name = NAME
NAME.Of(PLANT) = PLANT
NAME
SQL[NAME] = First Plant
SQL[0] = First Plant
Assertion failed in C:\upp\uppsrc\Sql\Sql.cpp, line 339
0

Sad!! Very sad!
It turns out SQLite driver works with huge errors.

I confess, I have experimented on native sqlite3 tools.

From this it follows that it is the implementation of U++ provides such terrible mistakes?

Sad!! Very sad!

Interesting, how things are with other database engines?

Well, what is the salvation of the drowning - a handwork of drowning.....

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by Sender Ghost on Sat, 19 May 2012 03:47:47 GMT
View Forum Message <> Reply to Message

sergeynikitin wrote on Sat, 19 May 2012 03:28Sad!! Very sad!
It turns out SQLite driver works with huge errors.

I confess, I have experimented on native sqlite3 tools.

From this it follows that it is the implementation of U++ provides such terrible mistakes?

I think, no. After you wrote this, I tried to use SQLite command-line shell with following results:

```
sqlite3.exe
SQLite version 3.7.12 2012-05-14 01:41:23
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .read S_SQL_Sqlite3.sql

sqlite> .header on
sqlite> .mode column
sqlite> insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (0, 'Joe', 'Smith', 0);
sqlite> insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (1, 'Mike', 'Smith', 0);
sqlite> insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (2, 'Jon', 'Goober', 1);
sqlite> insert into PLANT(ID, NAME, ADDRESS) values (0, 'First Plant', 'First st.');
sqlite> insert into PLANT(ID, NAME, ADDRESS) values (1, 'Second Plant', 'Second st.');
sqlite> select WORKER.NAME, WORKER.LASTNAME, PLANT.NAME, PLANT.ADDRESS from
WORKER left outer join PLANT on WORKER.PLANT_ID = PLANT.ID;
NAME        LASTNAME    NAME         ADDRESS
----------  ----------  -----------  ----------
Joe         Smith       First Plant  First st.
Mike        Smith       First Plant  First st.
Jon         Goober      Second Plan  Second st.
sqlite> select WORKER.NAME, WORKER.LASTNAME, PLANT.NAME PLANT_NAME,
PLANT.ADDRESS from WORKER left outer join PLANT on WORKER.PLANT_ID = PLANT.ID;
NAME        LASTNAME    PLANT_NAME   ADDRESS
----------  ----------  -----------  ----------
Joe         Smith       First Plant  First st.
Mike        Smith       First Plant  First st.
Jon         Goober      Second Plan  Second st.
sqlite> .quit
```

Contents of "S_SQL_Sqlite3.sql" file, which generated U++ executable to create tables:

```
create table WORKER (
 ID integer primary key,
 NAME text,
 LASTNAME text,
 PLANT_ID integer
);

create table PLANT (
 ID integer primary key,
 NAME text,
 ADDRESS text
);
```

The U++ wrapper returns the same column names in my case.
sergeynikitin wrote on Sat, 19 May 2012 03:28Interesting, how things are with other database engines?
I don't have other database engines to check, for now.
sergeynikitin wrote on Sat, 19 May 2012 03:28Well, what is the salvation of the drowning - a handwork of drowning.....
No need to sinking :)
There are about three methods to solve such issue already:
- Accessing result set through indexes.
- Using "AS" clause for clashing column names.
- Using unique column names across database.

Edit:
I tested the same queries for MySQL database through U++ wrapper and MySQL Workbench. And it returns the same column names for two mentioned queries:

NAME LASTNAME NAME ADDRESS
NAME LASTNAME PLANT_NAME ADDRESS

The modified version of SQL_MySql reference you could find in the attachment.

File Attachments

1) SQL_MySql.zip, downloaded 508 times

Subject: Re: PROPOSAL: Access to S_* Structure of TABLE crash Application.
Posted by Sender Ghost on Sun, 20 May 2012 02:09:24 GMT
View Forum Message <> Reply to Message

Based on experience, when returned result set contains column names in the same order as constructed, I created following helper function and class:
Toggle Spoiler

```
NAMESPACE_UPP

template<> String AsString(const SqlSet& set) {
 StringBuffer text;
 const char *s = ~set;

 for(;;) {
  const char *b = s;

  while((byte)*s >= 32)
   s++;

  text.Cat(b, s);
  int c = *s;
```

```
  if (c == SQLC_AS) {
   text << ' ';
  }
  else
  if (c == SQLC_OF)
   text << '.';
  else
  if (c == SQLC_COMMA)
   text << ", ";

  s++;

  if (c == '\0' || c == SQLC_ID)
   break;
 }

 return text;
}

class SqlSetIndex {
private:
 VectorMap<String, int> list;
 inline void Parse(const SqlSet& set);
public:
 SqlSetIndex(const SqlSet& set)           { Parse(set); }
 void Clear()                             { list.Clear(); }
 int operator[](const SqlId& id) const    { return list.Get(~id); }
 int operator[](const String& id) const   { return list.Get(id); }
 SqlSetIndex& operator=(const SqlSet& set) { list.Clear(); Parse(set); return *this; }
 const VectorMap<String, int>& operator~() { return list; }
};

void SqlSetIndex::Parse(const SqlSet& set)
{
 int index = 0;
 StringBuffer text;
 bool isNext = true, isAs = false;
 const char *s = ~set;

 for(;;) {
  const char *b = s;

  while((byte)*s > 32 && (byte)*s != ',')
   s++;

  text.Cat(b, s);
  int c = *s;
```

```
  if (c == SQLC_OF) { // '.'
   text << *s;
   isNext = false;
  }
  else
  if (c == ' ')
   isNext = false;
  else
  if (c == SQLC_AS)
   isAs = true;

  if (isNext) {
   if (!isAs)
    list.GetAdd(text, index++);
   else {
    list.GetAdd(text, index);
    isAs = false;
   }

   text.Clear();
  }
  else
   isNext = true;

  s++;

  if (c == '\0' || c == SQLC_ID)
   break;
 }
}

END_UPP_NAMESPACE
```

- AsString function for SqlSet, which replaces special SQLC_* characters, almost the same as SqlCompile function does for full SQL statement.
- SqlSetIndex class to parse SqlSet to VectorMap<String, int> container with column names and their indexes.

Now, the part of example will look like follows:
Toggle Spoiler

```
LOG("Inserting values:");
SQL * Insert(WORKER)(ID, 0)(NAME, "Joe")(LASTNAME, "Smith")(PLANT_ID, 0);
LOG(SQL.ToString());
SQL * Insert(WORKER)(ID, 1)(NAME, "Mike")(LASTNAME, "Smith")(PLANT_ID, 0);
LOG(SQL.ToString());
SQL * Insert(WORKER)(ID, 2)(NAME, "Jon")(LASTNAME, "Goober")(PLANT_ID, 1);
```

```
LOG(SQL.ToString());

SQL * Insert(PLANT)(ID, 0)(NAME, "First Plant")(ADDRESS, "First st.");
LOG(SQL.ToString());
SQL * Insert(PLANT)(ID, 1)(NAME, "Second Plant")(ADDRESS, "Second st.");
LOG(SQL.ToString());

SQLID(PLANT_NAME); // Just to show parsing of "AS" clause here.
LOG('\n' << "Testing SqlSetIndex:");
SqlSet testing_set(WORKER(ID, NAME), ID.Of(WORKER), PLANT(NAME).As(PLANT_NAME),
PLANT(ID, NAME, ADDRESS));
DUMP(testing_set);
SqlSetIndex testing_list(testing_set);
DUMPM(~testing_list);

LOG('\n' << "Selecting values:");
SqlSet set(WORKER(NAME, LASTNAME), PLANT(NAME).As(PLANT_NAME),
PLANT(ADDRESS));
SqlSetIndex list(set);

SQL * Select(set).From(WORKER).LeftJoin(PLANT).On(WORKER(PLANT_ID) == PLANT(ID));
LOG(SQL.ToString());
while (SQL.Fetch()) {
 LOG("-----");
 DUMP(SQL[list[WORKER(NAME)]]);
 DUMP(SQL[list[WORKER(LASTNAME)]]);
 DUMP(SQL[list[PLANT(NAME)]]);
 //DUMP(SQL[list[PLANT_NAME]]); // The same as previous value
 DUMP(SQL[list[PLANT(ADDRESS)]]);
}
```

With following output:
Toggle Spoiler

```
Inserting values:
insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (0, 'Joe', 'Smith', 0)
insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (1, 'Mike', 'Smith', 0)
insert into WORKER(ID, NAME, LASTNAME, PLANT_ID) values (2, 'Jon', 'Goober', 1)
insert into PLANT(ID, NAME, ADDRESS) values (0, 'First Plant', 'First st.')
insert into PLANT(ID, NAME, ADDRESS) values (1, 'Second Plant', 'Second st.')

Testing SqlSetIndex:
testing_set = WORKER.ID, WORKER.NAME, WORKER.ID, PLANT.NAME PLANT_NAME,
PLANT.ID, PLANT.NAME, PLANT.ADDRESS
~testing_list:
 [0] = (WORKER
 ID) 0
```

[1] = (WORKER
NAME) 1
[2] = (PLANT
NAME) 3
[3] = (PLANT_NAME) 3
[4] = (PLANT
ID) 4
[5] = (PLANT
ADDRESS) 6

Selecting values:
select WORKER.NAME, WORKER.LASTNAME, PLANT.NAME PLANT_NAME,
PLANT.ADDRESS from WORKER left outer join PLANT on WORKER.PLANT_ID = PLANT.ID

SQL[list[WORKER(NAME)]] = Joe
SQL[list[WORKER(LASTNAME)]] = Smith
SQL[list[PLANT(NAME)]] = First Plant
SQL[list[PLANT(ADDRESS)]] = First st.

SQL[list[WORKER(NAME)]] = Mike
SQL[list[WORKER(LASTNAME)]] = Smith
SQL[list[PLANT(NAME)]] = First Plant
SQL[list[PLANT(ADDRESS)]] = First st.

SQL[list[WORKER(NAME)]] = Jon
SQL[list[WORKER(LASTNAME)]] = Goober
SQL[list[PLANT(NAME)]] = Second Plant
SQL[list[PLANT(ADDRESS)]] = Second st.

The full example with SqlSetIndex class you could find in the attachment.

Edit:
Fixed some typos in the text.

File Attachments

1) SqlSetIndex.zip, downloaded 565 times