Archived:Using Symbian SQL API with SQL statements which return data

Archived: This article is **archived** because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Overview

The `RSqlStatement` class provides functionality to execute all types of SQL statements when using the SQL API. `RSqlStatement` encapsulates an SQL statement and the data that it returns. `RSqlStatement` statements can be executed with the methods `Exec()` and `Next()`. The method `Exec()` is used when the statement does not return any data and `Next()` when the statement returns data. This snippet gives some examples on how to use the `Next()` method.

Statements use so-called placeholders (:) in descriptors to mark positions where dynamic data is inserted at runtime. The user can bind the actual data to placeholders with different `Bind()` methods, such as `BindText()`. It is good practise to resolve the placeholder index with the `RSqlStatement::ParameterIndex()` method, and not use hardcoded values.

The method `ColumnSize()` helps when the size of returned data is unknown and the user wants to copy it to the allocated buffer. Data can be copied into the user-allocated buffer with `Column()` methods, such as `ColumnText()`.

This snippet can be self-signed.

MMP file

The following libraries are required:

```
LIBRARY euser.lib
LIBRARY sqldb.lib
```

The following capabilities are needed to test the example secure database:

```
CAPABILITY ReadUserData
CAPABILITY WriteUserData
```

Preconditions

Databases nonsecure.db and [UID3]secure.db need to be created before executing this code snippet. See Archived:Using SQL API for creating non-secure and secure databases on Symbian.

Source file

```
#include <e32base.h>
#include <SqlDb.h>

void NextStatementsL()
  {
  RSqlDatabase database;
  _LIT(KNonSecureDbName, "\\nonsecure.db");
```


(C) Copyright Nokia 2014. All rights reserved.

```
  _LIT(KSecureDbName, "[E80000AF]secure.db");

//== Retrieving numeric data
  TInt error = database.Open(KNonSecureDbName);

  if (error == KErrNone)
    {
    CleanupClosePushL(database);

    RSqlStatement sqlSelectFromMoviesTableStatement;
    _LIT(KSqlSelectFromMoviesTable, "SELECT * FROM MOVIES WHERE YEAR < :Val1");
    TInt ret = sqlSelectFromMoviesTableStatement.Prepare(database,
KSqlSelectFromMoviesTable);

    if(ret == KErrNone)
      {
      CleanupClosePushL(sqlSelectFromMoviesTableStatement);
      TInt paramIndex1 = sqlSelectFromMoviesTableStatement.ParameterIndex(_L(":Val1"));

      User::LeaveIfError(sqlSelectFromMoviesTableStatement.BindInt(paramIndex1, 2000));

      TInt columnIndex = sqlSelectFromMoviesTableStatement.ColumnIndex(_L("ID"));
      TInt err = KErrNone;

      while((err = sqlSelectFromMoviesTableStatement.Next()) == KSqlAtRow)
        {
        TInt id = sqlSelectFromMoviesTableStatement.ColumnInt(columnIndex);
        //Do something with id (movie released before year 2000 )
        }
      if(err == KSqlAtEnd)
        {
        //OK - no more records
        }
      else
        {
        //process the error
        }

      CleanupStack::PopAndDestroy(1); //sqlSelectFromMoviesTableStatement
      }
    else
      {
      //prepare sql statement failed
      }
    CleanupStack::PopAndDestroy(1); //database
    }
  else
    {
     //open database failed
    }

//== Retrieving text data

  error = database.Open(KSecureDbName);

  if (error == KErrNone)
    {
```

 (C) Copyright Nokia 2014. All rights reserved.

```
    CleanupClosePushL(database);

    RSqlStatement sqlSelectFromBooksTableStatement;
    _LIT(KSqlSelectFromBooksTable, "SELECT * FROM BOOKS");
    TInt ret = sqlSelectFromBooksTableStatement.Prepare(database,
KSqlSelectFromBooksTable);

    if(ret == KErrNone)
      {
      CleanupClosePushL(sqlSelectFromBooksTableStatement);

      TInt columnIndex1 = sqlSelectFromBooksTableStatement.ColumnIndex(_L("TITLE"));
      TInt columnIndex2 = sqlSelectFromBooksTableStatement.ColumnIndex(_L("AUTHOR"));

      TInt err = KErrNone;

      while((err = sqlSelectFromBooksTableStatement.Next()) == KSqlAtRow)
        {
         // access data with copying:
         RBuf title;
         TInt titleSize = sqlSelectFromBooksTableStatement.ColumnSize(columnIndex1);
         title.CreateL(titleSize);
         CleanupClosePushL(title);

User::LeaveIfError(sqlSelectFromBooksTableStatement.ColumnText(columnIndex1,title));
         //do something with the data...
         CleanupStack::PopAndDestroy(1); //title

         // ...or access data without copying:
         TPtrC author = sqlSelectFromBooksTableStatement.ColumnTextL(columnIndex2);
         //do something with the data...
        }
        if(err == KSqlAtEnd)
          {
          //OK - no more records
          }
        else
          {
          //process the error
          }
      CleanupStack::PopAndDestroy(1); //sqlSelectFromMoviesTableStatement
      }
    else
      {
      //prepare sql statement failed
      }
    CleanupStack::PopAndDestroy(1); //database
    }
  else
    {
    //open database failed
    }

  }
```

 (C) Copyright Nokia 2014. All rights reserved.

Postconditions

The `RSqlStatement` class is used to fetch data from the nonsecure.db and [UID3]secure.db databases. The first statement is used to retrieve numeric data and the second one is used to retrieve textual data.

See also

- Archived:Using SQL API for creating non-secure and secure databases on Symbian
- Archived:Using Symbian SQL API for attaching and detaching databases
- Archived:Using Symbian SQL API with SQL statements which do not return data
- Archived:Using Symbian SQL API with scalar queries
- Archived:Using Symbian SQL API with data streams

 (C) Copyright Nokia 2014. All rights reserved.