СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

Объектная библиотека прикладного интерфейса

НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

РЕЛЭКС®

Товарные знаки

РЕЛЭКС™, ЛИНТЕР® являются товарными знаками, принадлежащими ЗАО НПП «Реляционные экспертные системы» (далее по тексту – компания РЕЛЭКС). Прочие названия и обозначения продуктов в документе являются товарными знаками их производителей, продавцов или разработчиков.

Интеллектуальная собственность

О документе

Материал, содержащийся в данном документе, прошел доскональную проверку, но компания РЕЛЭКС не гарантирует, что документ не содержит ошибок и пропусков, поэтому оставляет за собой право в любое время вносить в документ исправления и изменения, пересматривать и обновлять содержащуюся в нем информацию.

Контактные данные

394006, Россия, г. Воронеж, ул. Бахметьева, 2Б.

Тел./факс: (473) 2-711-711, 2-778-333.

e-mail: market@relex.ru.

Техническая поддержка

С целью повышения качества программного продукта ЛИНТЕР и предоставляемых услуг в компании РЕЛЭКС действует автоматизированная система учёта и обработки пользовательских рекламаций. Обо всех обнаруженных недостатках и ошибках в программном продукте и/или документации на него просим сообщать нам в раздел Поддержка на сайте ЛИНТЕР.

Содержание

Предисловие

Назначение документа

В документе приведено описание библиотеки LincppAPI, расширяющей функциональные возможности библиотеки LinAPI.

Документ предназначен для СУБД ЛИНТЕР СТАНДАРТ 6.0 сборка 17.48, далее по тексту СУБД ЛИНТЕР.

Для кого предназначен документ

Документ предназначен для лиц, разрабатывающих клиентские приложения на языке программирования C++ с использованием прикладного интерфейса для работы с ЛИНТЕР-сервером.

Назначение и условия применения

Назначение

Библиотека LincppAPI представляет собой объектную надстройку над библиотекой LinAPI. Важной особенностью данной библиотеки является возможность автоматического воспроизведения транзакции (с её начала), прерванной временным разрывом соединения с СУБД ЛИНТЕР. Это позволяет эффективно использовать библиотеку для разработки клиентских приложений, работающих, в частности, с системой резервирования СУБД ЛИНТЕР.

Условия применения

В системе резервирования СУБД ЛИНТЕР есть главный сервер, обслуживающий запросы клиентского приложения, и один или несколько резервных серверов, на которых дублируются все изменения в БД главного сервера. При отказе главного сервера его функции берет на себя один из резервных серверов, при этом переключение клиентского приложения на обслуживание новым главным сервером системы резервирования в большинстве случаев происходит незаметно (прозрачно) либо с небольшой задержкой, вызванной необходимостью повторения на новом главном сервере прерванной из-за отказа транзакции.

В редких случаях (отсутствие резервного сервера, готового выполнять функции главного сервера системы резервирования, либо недопустимая для приложения длительность перехода к новому главному серверу) обслуживание клиентского приложения будет прекращено.

При работе с системой резервирования с использованием функций библиотеки LinAPI клиентское приложение должно самостоятельно отслеживать отказ главного сервера, определять новый главный сервер, подсоединяться к нему и повторно выполнять прерванную транзакцию. При использовании библиотеки LincppAPI все эти действия выполняются средствами самой библиотеки – библиотека будет пытаться самостоятельно восстановить разорванное соединение с ЛИНТЕР-сервером в течение заданного промежутка времени и, в случае успешного соединения, повторит прерванную транзакцию.

Другой важной особенностью библиотеки LincppAPI является реализация в ней класса linDataSet, представляющего клиентскому приложению функциональные возможности, отсутствующие в библиотеке LinAPI. Класс linDataSet представляет собой размещенный полностью в оперативной памяти компьютера (кэшированный) набор данных, загруженный из БД СУБД ЛИНТЕР, что позволяет уменьшить число запросов к БД. Несложная обработка данных этого класса выполняется с помощью методов класса и не требует знания языка баз данных SQL (т. е. операторов INSERT, DELETE, UPDATE, конструкций WHERE,ORDER BY и др.). (При необходимости более сложной обработки данных, возможно, потребуется использование соответствующих SQL-операторов).

Класс linDataSet, расширяющий функциональность библиотеки LinAPI, можно использовать при разработке клиентских приложений, предназначенных как для работы в системе резервирования, так и для работы с независимыми ЛИНТЕР-серверами.

Характеристики библиотеки

Большинство методов библиотеки LincppAPI расширяет и покрывает функциональность библиотеки LinAPI, но ряд методов не имеет прототипов в LinAPI. В библиотеку LincppAPI не включены следующие функции из LinAPI:

- LINTER_KillChannel;
- LINTER_ServerInfo;
- LINTER_Cancel;
- LINTER_FetchBlobEx;
- LINTER_AddBlob2;
- LINTER_ExecControlQuery;
- LINTER_CreateCursor.

Пример программы, работающей с библиотекой LincppAPI, приведен в приложении.

Классы объектов

В библиотеке реализованы следующие классы объектов, предоставляемых пользователю (их объявление находится в файле lincppapi.h): linConnection, linCursor, linStatement и linDataSet.

Как правило, если метод класса возвращает значение типа L_LONG, то это значение содержит код завершения. Коды завершения библиотеки LincppAPI совпадают с кодами завершения библиотеки LinAPI:

1) LINAPI_ERROR_FOR_RE_CONNECT – информирует о том, что в течение заданного промежутка времени (тайм-аута) не удалось восстановить соединение с СУБД ЛИНТЕР, при этом библиотека LinAPI вернула специфический код завершения: (4000-5000), 1001, 1044, 1046, 1069, 6712, фиксирующий разрыв соединения с СУБД. В таком случае необходимо повторно инициировать соединение с сервером;
2) LINCPPAPI_ERROR – внутренняя ошибка библиотеки LincppAPI. При получении данного кода завершения необходимо детализировать причину ошибки с помощью вызовов библиотеки LincppAPI (GetErrorsCount/GetError для соответствующего класса);
3) LINAPI_SUCCESS – операция завершена успешно, данный код совпадает с кодом завершения библиотеки LinAPI.

Класс linConnection

При создании соединения открывается один канал связи с ЛИНТЕР-сервером. Создав соединение, клиентское приложение получает доступ к БД, с которой в данный момент работает ЛИНТЕР-сервер. Одно приложение может установить несколько соединений как с одним так и с разными ЛИНТЕР-серверами путём создания одного или нескольких объектов класса linConnection.

Конструктор класса

Синтаксис

```
linConnection();
```

Описание

Метод создает конструктор класса linConnection.

Прототип LinAPI

Отсутствует.

Деструктор класса

Синтаксис

```
~linConnection();
```

Описание

Метод создает деструктор класса linConnection.

Прототип LinAPI

Отсутствует.

Максимальное время ожидания перезапуска сервера

Синтаксис

```
void SetMaxTimeout (
  L_ULONG ulTimeout);        /* тайм-аут */
```

Описание

Метод позволяет изменять максимальное время ожидания запуска ЛИНТЕР-сервера после его останова (тайм-аут перезапуска). Величина тайм-аута задается в миллисекундах.

Если при выполнении какого-либо метода класса произошёл разрыв соединения с ЛИНТЕР-сервером, библиотека с периодичностью 0.5 секунды в течение заданного тайм-аута будет пытаться восстановить соединение. Если до истечения тайм-аута соединение удалось восстановить, прерванная транзакция воспроизведется вплоть

до разрыва соединения, в противном случае будет выдан соответствующий код завершения.

Значение тайм-аута по умолчанию 30000 мс (30 секунд).

Прототип LinAPI

Отсутствует.

Управление транзакциями после перезапуска сервера

Синтаксис

```
void IgnoreLinterErrorsDuringRestore (
  L_BOOL bIgnore);          /* режим транзакции */
```

Описание

Метод позволяет управлять поведением библиотеки при восстановлении транзакции после разрыва и соединения с ЛИНТЕР-сервером. Возможные значения bIgnore:

- L_TTRUE: игнорирование ошибок при воспроизведении транзакции. Это бывает полезно в случае, когда в рамках одной транзакции используются и DML, и DDL-запросы, например, создание таблицы и занесение в неё записей. В таком случае при восстановлении транзакции после разрыва соединения запрос на повторное создание таблицы будет завершен с ошибкой, т. к. DDL-запросы автоматически фиксируются в БД. В ряде случаев подобные ошибки желательно проигнорировать;
- L_TFALSE: запрет игнорирования ошибок при воспроизведении транзакции. Это необходимо, если в транзакции присутствуют только DML-запросы, т. к. наличие ошибок свидетельствует о том, что транзакция не приведёт к тем результатам, на которые рассчитывал пользователь.

Значение по умолчанию L_TTRUE.

Прототип LinAPI

Отсутствует.

Установить режим сохранения информации о вызове методов

Синтаксис

```
void SetUseConnQBuf(L_BOOL bSet);  /* режим сохранения информации
 */
```

Описание

Метод устанавливает режим сохранения информации о ранее вызванных для выполнения методах данного класса.

В любом транзакционном режиме, отличном от mAutocommit, библиотека LincppAPI осуществляет автоматическое сохранение информации о вызванных ранее методах с

тем, чтобы в случае разрыва и последующего восстановления соединения с ЛИНТЕР-сервером можно было повторить вызов этих методов.

В случае если bSet=true (режим по умолчанию), используется общий как для соединения (класс linConnection), так и для курсоров (класс linCursor) буфер текущей транзакции (в котором и хранится информация о вызываемых методах). Буфер текущей транзакции очищается при вызове методов linConnection::Commit и linConnection::RollBack.

В случае если bSet=false, буфер текущей транзакции создаётся в каждом из курсоров (класс linCursor) и очищается, соответственно, при вызове методов linCursor::CommitCursor и linCursor::RollBackCursor.

Прототип LinAPI

Отсутствует.

Получить режим сохранения информации о вызове методов

Синтаксис

```
L_BOOL GetUseConnQBuf();
```

Описание

Метод возвращает установленный режим сохранения информации о вызове методов.

Прототип LinAPI

Отсутствует.

Установить соединение с сервером

Синтаксис

Вариант 1.

```
L_LONG Connect(
  L_CHAR *,                 /* имя пользователя */
  L_SWORD,                  /* длина имени пользователя */
  L_CHAR *,                 /* пароль пользователя */
  L_SWORD,                  /* длина пароля пользователя */
  L_CHAR *,                 /* имя ЛИНТЕР-сервера */
  L_LONG,                   /* режим обработки транзакций */
  void *pAstFunction,
  void *pUserArg);
```

Вариант 2.

```
L_LONG ConnectCS(
  L_CHAR *,                 /* имя пользователя */
  L_SWORD,                  /* длина имени пользователя */
  L_CHAR *,                 /* пароль пользователя */
  L_SWORD,                  /* длина пароля пользователя */
```

```
  L_CHAR *,                 /* имя ЛИНТЕР-сервера */
  L_CHAR *,                 /* имя кодовой страницы */
  L_LONG,                   /* режим обработки транзакций */
  void *pAstFunction,
  void *pUserArg,
  linErrBuf *pErrBuf);      /* всегда NULL для внешнего вызова */
```

Описание

Метод ConnectCS так же, как метод Connect, открывает соединение с заданным режимом транзакции. Затем вызывает функцию LINTER_ConnectCS без последних двух аргументов в случае, если они имеют значение NULL, и функцию LINTER_AsyncConnectCS со всеми аргументами в противном случае.

Прототип LinAPI

LINTER_Connect, LINTER_ConnectCS, LINTER_AsyncConnectCS.

Переустановить соединение с сервером

Синтаксис

```
L_LONG ReConnect(
  L_CHAR *,                  /* имя пользователя */
  L_SWORD,                   /* длина имени пользователя */
  L_CHAR *,                  /* пароль пользователя */
  L_SWORD,                   /* длина пароля пользователя */
  L_CHAR *,                  /* имя ЛИНТЕР-сервера */
  L_CHAR *,                  /* имя кодовой страницы */
  L_LONG,                    /* режим обработки транзакций */
  linErrBuf *pErrBuf);       /* всегда NULL для внешнего вызова */
```

Описание

Метод ReConnect открывает заново ранее установленное соединение с ЛИНТЕР-сервером, возможно, уже с другими параметрами (новым пользователем, режимом транзакции, отличным от предыдущего режима, установленного вызовом метода Connect или ConnectCS).

Метод ReConnect сохраняет контекст предыдущего соединения, т. е., например, не надо заново открывать ранее открытые в соединении курсоры.

Аргументы метода аналогичны аргументам метода ConnectCS.

Прототип LinAPI

LINTER_Connect, LINTER_ConnectCS, LINTER_AsyncConnectCS.

Закрыть соединение с сервером

Синтаксис

```
L_LONG CloseConnect();
```


Описание

Метод разрывает текущее соединение с сервером и уничтожает все дочерние объекты классов linCursor и linStatement.

Прототип LinAPI

LINTER_CloseConnect.

Открыть курсор

Синтаксис

```
L_LONG OpenCursor(
  L_CHAR *,      /* имя курсора */
  L_SWORD,       /* длина имени курсора */
  linCursor **); /* указатель на созданный объект linCursor */
```

Описание

Метод создаёт объект класса linCursor и возвращает указатель на данный объект.

Прототип LinAPI

LINTER_OpenCursor.

Закрыть курсор

Синтаксис

```
L_LONG CloseCursor (linCursor *);  /* объект класса linCursor */
```

Описание

Метод закрывает переданный курсор. В зависимости от режима транзакции объект курсора при этом может (физически) как удалиться (mAutocommit), так и не удалиться (в других режимах транзакции – только помечается как удалённый для сохранения возможности повторения транзакции в случае разрыва соединения).

Прототип LinAPI

LINTER_CloseCursor.

Откатить транзакцию

Синтаксис

```
L_LONG RollBack (
  void *pAstFunction,       /* асинхронная функция */
  void *pUserArg);          /* пользовательские данные */
```

Описание

Метод очищает буфер текущей транзакции (в котором хранится последовательность функций, вызванных в рамках текущей транзакции, с их аргументами, необходимая для

воспроизведения транзакции в случае разрыва/восстановления соединения с сервером) и затем вызывает функцию LINTER_RollBack.

Прототип LinAPI

LINTER_RollBack.

Подтвердить транзакцию

Синтаксис

```
L_LONG Commit (
  void *pAstFunction,       /* асинхронная функция */
  void *pUserArg);          /* пользовательские данные */
```

Описание

Метод вызывает функцию LINTER_Commit и затем очищает буфер текущей транзакции.

Прототип LinAPI

LINTER_Commit.

Установить параметры соединения с сервером

Синтаксис

```
L_LONG SetConnectOption (
  L_SWORD,                  /* маска бит параметров соединения */
  void *,                   /* буфер значений параметров */
  L_LONG *);                /* длина буфера параметров */
```

Описание

Метод устанавливает требуемые параметры соединения.

Прототип LinAPI

LINTER_SetConnectOption.

Получить параметры соединения

Синтаксис

```
L_LONG GetConnectOption (
  L_SWORD,                  /* маска бит параметров соединения */
  void *,                   /* буфер значений параметров */
  L_LONG *);                /* длина буфера параметров */
```

Описание

Метод предоставляет требуемые параметры соединения.

Прототип LinAPI

LINTER_GetConnectOption.

Получить описание объекта БД

Синтаксис

```
L_LONG GetObjDesc (
  L_SWORD,   /* тип объекта */
  L_CHAR *,  /* имя объекта */
  L_SWORD,   /* длина имени объекта */
  void *,    /* буфер для размещения описания объекта */
  L_LONG *); /* длина буфера */
```

Описание

Метод предоставляет описание заданного объекта БД.

Прототип LinAPI

LINTER_GetObjDesc.

Проверить завершение операции по соединению

Синтаксис

```
L_LONG ConnectComplete (
  L_SWORD *,                /* флаг завершения */
  L_LONG *,                 /* код завершения LinAPI */
  L_LONG *,                 /* код завершения СУБД ЛИНТЕР*/
  L_LONG *);                /* код завершения ОС */
```

Описание

Метод проверяет завершение последней операции по соединению. Если операция завершена, то флаг завершения будет отличен от нуля и нулем в противном случае.

В случае если флаг завершения отличен от нуля, переменные `код завершения LinAPI`, `код завершения СУБД ЛИНТЕР` и `код завершения ОС` содержат коды завершения операции.

Прототип LinAPI

LINTER_ConnectComplete.

Проверить буфер кодов завершения

Синтаксис

```
L_ULONG GetErrorsCount();
```

Описание

Метод предоставляет информацию о количестве кодов завершения, сведения о которых сохранены в буфере кодов завершения библиотеки.

Метод вызывается в том случае, если один из предшествующих методов вернул код завершения, отличный от LINAPI_SUCCESS.

Прототип LinAPI

Отсутствует.

Очистить буфер кодов завершения

Синтаксис

```
void ClearErrors();
```

Описание

Метод очищает буфер ошибок. Он применяется перед вызовом практически всех функций библиотеки LinAPI внутри класса linConnection, т. к. предполагается, что если пользователь не извлёк ошибки из буфера после вызова предыдущего метода класса, то ему они не нужны.

Прототип LinAPI

Отсутствует.

Получить информацию о коде завершения

Синтаксис

```
L_LONG GetError (
  L_CHAR  *,  /* буфер для имени модуля, в котором произошла
 ошибка */
  L_ULONG *,  /* длина буфера для имени модуля */
  L_CHAR  *,  /* буфер для текста кода завершения */
  L_ULONG *,  /* длина буфера для текста кода завершения */
  L_LONG  *,  /* буфер для кода завершения LinAPI */
  L_LONG  *,  /* буфер для кода завершения СУБД ЛИНТЕР */
  L_LONG  *); /* буфер для кода завершения ОС */
```

Описание

Метод извлекает ошибку из буфера ошибок. Для аргументов метода с типом L_ULONG* требуется передать указатели на переменные, в которых хранятся размер буфера для получения имени модуля и размер буфера для получения текста сообщения соответственно. В эти же переменные метод запишет реальные размеры строк с именем модуля и текстом сообщения (в том случае, если буфер для строк окажется недостаточным, строки будут усечены).

Прототип LinAPI

Отсутствует.

Примеры

1) Пример функции обработки кода завершения:

```
static void processing_error(linConnection *pConnect,
                             L_LONG         ret_cod,
                             L_CHAR        *message)
```



```
{
  L_CHAR  sModuleName[64], sErrorText[1024];
  L_ULONG ulModuleNameLen = sizeof(sModuleName),
          ulErrorTextLen = sizeof(sErrorText);
  L_LONG  lLinApi, lLinter, lSystem, lRet;

  printf("Return code = %ld (%s)\n", ret_cod, message);
  while (pConnect->GetErrorsCount())
  {
    if ((lRet = pConnect->GetError(sModuleName,
                                   &ulModuleNameLen,
                                   sErrorText,
                                   &ulErrorTextLen,
                                   &lLinApi,
                                   &lLinter,
                                   &lSystem)) == LINAPI_SUCCESS)
      printf("%s: '%s'\n", sModuleName, sErrorText);
    else
      printf("GetError returned code #%ld\n", lRet);
  }
}
```

2) Пример вызова функции обработки кода завершения:

```
linConnection *pConnect = new linConnection;
L_LONG         lRet;

if ((lRet = pConnect->Connect("SYSTEM", 0, "MANAGER", 0, NULL,
 mAutocommit)) !=
 LINAPI_SUCCESS)
    processing_error(pConnect, lRet, "ERROR Connect");
```

Класс linCursor

Курсор – это средство выполнения запросов и/или операторов. При открытии курсора открывается канал связи с ЛИНТЕР-сервером.

Курсорный канал является дочерним (подчиненным) каналом по отношению к соединению, для которого открывается курсор.

Таким образом, под курсором понимается некоторая область данных, описывающая состояние выполнения запроса или оператора. С одним курсором могут быть связаны несколько SQL-операторов, но курсор будет содержать состояние того запроса или оператора, который был выполнен последним.

Пользователю недоступны конструктор и деструктор класса linCursor, т. к. пользователь должен создавать и уничтожать объекты этого класса с помощью методов OpenCursor/ CloseCursor класса linConnection.

Транслировать SQL-оператор

Синтаксис

```
L_LONG CreateStatement (
  L_CHAR *,          /* текст SQL-оператора */
  L_LONG,            /* длина текста SQL-оператора */
  linStatement **); /* указатель на созданный объект */
                     /* класса linStatement */
```

Описание

Транслирует SQL-оператор, создает объект класса linStatement и возвращает указатель на него.

Прототип LinAPI

LINTER_CreateStatement.

Закрыть SQL-оператор

Синтаксис

```
L_LONG FreeStatement (
  linStatement *); /* указатель на объект класса linStatement */
```

Описание

Метод закрывает переданный оператор (объект класса linStatement). В зависимости от режима транзакции объект оператора при этом может (физически) как удалиться (mAutocommit), так и не удалиться (в других режимах транзакции – только помечается как удалённый для сохранения возможности повторения транзакции в случае разрыва соединения).

Прототип LinAPI

LINTER_FreeStatement.

Получить порцию записей выборки

Синтаксис

```
L_LONG Fetch (
  L_SWORD,            /* направление перемещения по выборке */
  L_LONG,             /* номер записи выборки */
  L_LONG,             /* количество требуемых записей выборки */
  L_LONG *,           /* реальное количество предоставленных */
                      /* записей выборки */
  void *pAstFunction, /* асинхронная функция обработки ответа */
  void *pUserArg);    /* пользовательские данные */
```

Описание

Метод помещает в буфер ответа указанное количество записей выборки, начиная с заданного местоположения. Метод возвращает значение LINAPI_NO_DATA в случае достижения конца выборки.

Прототип LinAPI

LINTER_Fetch.

Выполнить SQL-оператор

Синтаксис

```
L_LONG ExecuteDirect (
  L_CHAR *,           /* текст SQL-оператора */
  L_LONG,             /* длина текста SQL-оператора */
  void *pAstFunction, /* асинхронная функция обработки ответа */
  void *pUserArg);    /* пользовательские данные */
```

Описание

Метод транслирует SQL-оператор и сразу его выполняет.

Прототип LinAPI

LINTER_ExecuteDirect.

Получить запись выборки

Синтаксис

```
L_LONG GetRowBuffer (
  void *,    /* буфер для размещения записи */
  L_LONG *); /* длина буфера для размещения записи */
```

Описание

Метод помещает в буфер ответа текущую запись выборки без преобразования.

Прототип LinAPI

LINTER_GetRowBuffer.

Блокировать текущую запись выборки

Синтаксис

```
L_LONG LockRow();
```

Описание

Метод блокирует текущую запись выборки.

Прототип LinAPI

LINTER_LockRow.

Разблокировать текущую запись выборки

Синтаксис

```
L_LONG UnlockRow();
```

Описание

Метод выполняет разблокирование текущей записи выборки.

Прототип LinAPI

LINTER_UnlockRow.

Очистить BLOB-данные первого столбца

Синтаксис

```
L_LONG ClearBlob (
  void *pAstFunction, /* асинхронная функция обработки ответа */
  void *pUserArg);    /* пользовательские данные */
```

Описание

Метод очищает первое BLOB-поле текущей записи выборки.

Прототип LinAPI

LINTER_ClearBlob.

Очистить BLOB-данные заданного столбца

Синтаксис

```
L_LONG PurgeBlob (
  L_SWORD,            /* номер столбца */
  void *pAstFunction, /* асинхронная функция обработки ответа */
  void *pUserArg);    /* пользовательские данные */
```

Описание

Метод очищает BLOB-поле заданного столбца текущей записи выборки.

Прототип LinAPI

LINTER_PurgeBlob.

Получить тип BLOB-данных первого BLOB-столбца

Синтаксис

```
L_LONG GetBlobType (
  L_WORD *);                /* возвращаемый тип BLOB-данных */
```

Описание

Метод возвращает тип BLOB-данных первого BLOB-столбца текущей записи выборки.

Прототип LinAPI

LINTER_GetBlobType.

Получить длину BLOB-данных

Синтаксис

```
L_LONG GetBlobLength (
  L_LONG *);  /* длина BLOB-данных в БД */
```

Описание

Метод возвращает длину BLOB-данных первого BLOB-столбца текущей записи выборки.

Прототип LinAPI

LINTER_GetBlobLength.

Добавить порцию BLOB-данных в первый BLOB-столбец

Синтаксис

```
L_LONG AppendBlob (
  L_WORD,             /* тип добавляемых BLOB-данных */
  void *,             /* адрес буфера с добавляемой порцией данных
 */
  L_LONG,             /* длина буфера с порцией данных */
  void *pAstFunction, /* асинхронная функция обработки ответа */
  void *pUserArg);    /* пользовательские данные */
```

Описание

Метод добавляет порцию данных в первое BLOB-поле текущей записи выборки.

Прототип LinAPI

LINTER_AppendBlob.

Добавить порцию BLOB-данных в заданный BLOB-столбец

Синтаксис

```
L_LONG AddBlob (
  L_SWORD,             /* номер столбца */
  L_LONG,              /* тип добавляемых BLOB-данных */
  void *,              /* адрес буфера с добавляемой порцией данных
 */
  L_LONG,              /* длина буфера с порцией данных */
  void *pAstFunction, /* асинхронная функция обработки ответа */
  void *pUserArg);     /* пользовательские данные */
```

Описание

Метод добавляет порцию данных в BLOB-поле заданного столбца текущей записи выборки.

Прототип LinAPI

LINTER_AddBlob.

Получить порцию BLOB-данных первого столбца

Синтаксис

```
L_LONG GetBlob (
  L_LONG,              /* начало требуемой порции данных */
  L_LONG *,            /* длина буфера для размещения порции данных
 */
  void *,              /* адрес буфера для порции данных */
  void *pAstFunction, /* асинхронная функция обработки ответа */
  void *pUserArg);     /* пользовательские данные */
```

Описание

Метод предоставляет требуемую порцию данных из первого BLOB-значения текущей записи выборки.

Прототип LinAPI

LINTER_GetBlob.

Получить порцию BLOB-данных заданного столбца

Синтаксис

```
L_LONG FetchBlob (
```



```
  L_SWORD,            /* номер столбца */
  L_LONG,             /* начало требуемой порции данных */
  L_LONG *,           /* длина буфера для размещения порции данных
 */
  void *,             /* адрес буфера для порции данных */
  void *pAstFunction, /* асинхронная функция обработки ответа */
  void *pUserArg);    /* пользовательские данные */
```

Описание

Метод предоставляет требуемую порцию данных из BLOB-значения заданного столбца текущей записи выборки.

Прототип LinAPI

LINTER_FetchBlob.

Установить тип BLOB-данных заданного столбца

Синтаксис

```
L_LONG SetBlobType (
  L_WORD );    /* тип BLOB-данных */
```

Описание

Метод изменяет тип первого BLOB-значения текущей записи.

Прототип LinAPI

LINTER_SetBlobType.

Получить характеристику курсора

Синтаксис

```
L_LONG GetCursorOption (
  L_SWORD,   /* тип характеристики курсора */
  L_SWORD,   /* номер столбца */
  void *,    /* буфер для размещения значения */
             /* характеристики курсора */
  L_LONG *); /* реальная длина значения характеристики */
```

Описание

Метод предоставляет значение требуемой характеристики заданного столбца текущей записи выборки. Аргумент «номер столбца» задается только для получения характеристики «описание столбца ответа».

Прототип LinAPI

LINTER_GetCursorOption.

Установить характеристику курсора

Синтаксис

```
L_LONG SetCursorOption (
  L_SWORD,    /* тип характеристики курсора */
  void *,     /* буфер для размещения значения */
              /* характеристики курсора */
  L_LONG *); /* реальная длина значения характеристики */
```

Описание

Метод устанавливает требуемую характеристику курсора.

Прототип LinAPI

LINTER_SetCursorOption.

Получить значение заданного столбца выборки

Синтаксис

```
L_LONG GetData(
  L_SWORD,    /* номер столбца */
  void *,     /* буфер для размещения значения */
  L_LONG,     /* длина буфера для размещения значения */
  L_SWORD,    /* возвращаемое значение типа данных столбца */
  L_SWORD,    /* возвращаемое значение точности данных столбца */
  L_SWORD,    /* возвращаемое значение масштаба данных столбца */
  L_LONG *); /* реальная длина значения столбца */
```

Описание

Метод предоставляет значение и его параметры заданного столбца текущей записи выборки.

Прототип LinAPI

LINTER_GetData.

Откатить транзакцию

Синтаксис

```
L_LONG RollBackCursor (
  void *pAstFunction, /* асинхронная функция */
  void *pUserArg);    /* пользовательские данные */
```

Описание

Метод очищает буфер текущей транзакции (в котором хранится информация о ранее вызванных в текущей транзакции методах с их аргументами, необходимая для воспроизведения транзакции в случае восстановления ранее разорванного соединения

с ЛИНТЕР-сервером) и затем вызывает функцию LINTER_RollBackCursor для отката выполненных в БД изменений.

Прототип LinAPI

LINTER_RollBackCursor.

Подтвердить транзакцию

Синтаксис

```
L_LONG CommitCursor (
  void *pAstFunction, /* асинхронная функция */
  void *pUserArg);    /* пользовательские данные */
```

Описание

Метод вызывает функцию LINTER_CommitCursor для фиксации выполненных в БД изменений и затем очищает буфер текущей транзакции.

Прототип LinAPI

LINTER_CommitCursor.

Получить указатель на объект «соединение»

Синтаксис

```
linConnection *GetConnection();
```

Описание

Метод возвращает переданный классу в конструкторе указатель на родительский объект linConnection.

Прототип LinAPI

Отсутствует.

Проверить завершение операции по курсору

Синтаксис

```
L_LONG CursorComplete (
  L_SWORD,                  /* флаг завершения */
  L_LONG *,                 /* код завершения LinAPI */
  L_LONG *,                 /* код завершения СУБД ЛИНТЕР */
  L_LONG *);                /* код завершения ОС */
```

Описание

Метод проверяет завершение последней операции по курсору.

Прототип LinAPI

LINTER_CursorComplete.

Проверить буфер кодов завершения

Синтаксис

```
L_ULONG GetErrorsCount();
```

Описание

Метод предоставляет информацию о количестве кодов завершения, сведения о которых сохранены в буфере кодов завершения библиотеки.

Метод вызывается в том случае, если один из предшествующих методов вернул код завершения, отличный от LINAPI_SUCCESS.

Прототип LinAPI

Отсутствует.

Очистить буфер кодов завершения

Синтаксис

```
void    ClearErrors();
```

Описание

Метод очищает буфер кодов завершения библиотеки. Он применяется перед вызовом практически всех функций библиотеки LinAPI внутри классов linCursor и linStatement (дочерних), т. к. предполагается, что если пользователь не извлёк коды завершения из буфера после вызова предыдущего метода класса, то они ему не понадобились.

Прототип LinAPI

Отсутствует.

Получить информацию о коде завершения

Синтаксис

```
L_LONG GetError (
  L_CHAR  *,  /* буфер для имени модуля, в котором произошла
 ошибка */
  L_ULONG *,  /* длина буфера для имени модуля */
  L_CHAR  *,  /* буфер для текста кода завершения */
  L_ULONG *,  /* длина буфера для текста кода завершения */
  L_LONG  *,  /* буфер для кода завершения LinAPI */
  L_LONG  *,  /* буфер для кода завершения СУБД ЛИНТЕР */
  L_LONG  *); /* буфер для кода завершения ОС */
```

Описание

Метод предоставляет информацию о коде завершения, сохраненном в буфере кодов завершения.

Для аргументов метода с типом L_ULONG* требуется передать указатели на переменные, в которых хранятся размер буфера для получения имени модуля и размер

буфера для получения текста сообщения соответственно. В этих же переменных метод возвращает реальные размеры указанных параметров. При недостаточном размере выделенных буферов информация усекается до размера буфера.

Прототип LinAPI

Отсутствует.

Класс linStatement

Оператор является результатом трансляции SQL-запроса. Операция трансляции проводится по каналу, открытому при создании соединения.

Запрос, поданный на трансляцию, может содержать параметры. Параметры можно рассматривать как переменные, значения которых можно изменять. Чтобы подставить в оператор конкретное значение параметра, необходимо произвести привязку буфера параметра, в который помещается значение, к оператору.

Создавать оператор имеет смысл тогда, когда предполагается его многократное выполнение или использование параметров.

Хотя в библиотеке LinAPI один и тот же оператор может быть связан с несколькими курсорами, относящимися к соединению, по которому был создан оператор, в библиотеке LincppAPI оператор всегда связан только с одним курсором.

Пользователю недоступен ни конструктор, ни деструктор класса linStatement, т.к. пользователь должен создавать и уничтожать объекты этого класса с помощью методов CreateStatement/FreeStatement класса linCursor.

Выполнить оператор

Синтаксис

```
L_LONG ExecuteStatement (
  L_LONG *,                /* номер выполняемого оператора */
  void *pAstFunction,      /* асинхронная функция обработки ответа
 */
  void *pUserArg);         /* пользовательские данные */
```

Описание

Метод выполняет заданный оператор, привязанный к текущему курсору.

Прототип LinAPI

LINTER_ExecuteStatement.

Установить характеристику оператора

Синтаксис

```
L_LONG SetStatementOption (
  L_SWORD,                 /* характеристика оператора */
  void *,                  /* буфер значения характеристики */
  L_LONG *);               /* длина значения характеристики */
```

Описание

Метод устанавливает заданную характеристику оператора.

Прототип LinAPI

LINTER_SetStatementOption.

Получить характеристику оператора

Синтаксис

```
L_LONG GetStatementOption (
  L_SWORD,                /* характеристика оператора */
  L_SWORD,                /* номер столбца */
  void *,                 /* буфер для значения характеристики */
  L_LONG *);              /* реальная длина значения
 характеристики */
```

Описание

Метод предоставляет значение заданной характеристики оператора. Аргумент «номер столбца» задается только для получения характеристики «описание столбца ответа».

Прототип LinAPI

LINTER_GetStatementOption.

Привязать параметр к оператору

Синтаксис

```
L_LONG BindParameter (
  L_SWORD,   /* номер привязываемого параметра */
  L_CHAR *,  /* имя привязываемого параметра */
  L_CHAR *,  /* признак NULL-значения */
  void *,    /* адрес значения привязываемого  параметра */
  L_LONG,    /* размерность массива привязываемых параметров */
  L_LONG,    /* размер одного элемента массива параметров */
  L_SWORD,   /* тип  привязываемого  параметра */
  L_LONG *); /* длина  привязываемого  параметра */
```

Описание

Метод привязывает буфер параметра к оператору. При выполнении оператора значение из этого буфера будет подставлено на место соответствующего параметра. Параметры могут быть как именованными, так и неименованными.

Прототип LinAPI

LINTER_BindParameter.

Присоединить столбец выборки к буферу

Синтаксис

```
L_LONG BindAnswer(
  L_SWORD,              /* номер столбца выборки */
  void *,               /* буфер для значений полей выборки */
  L_CHAR *,             /* признак NULL-значения поля выборки */
  L_LONG,               /* длина элемента буфера для одного
 ответа */
```

```
  L_SWORD,                /* тип данных поля выборки */
  L_LONG,                 /* длина поля выборки */
  L_SWORD,                /* точность значения поля выборки */
  L_SWORD,                /* масштаб значения поля выборки */
  L_LONG *);              /* реальная длина значения поля выборки
 */
```

Описание

Метод назначает буфер для размещения значений столбца выборки.

Прототип LinAPI

LINTER_BindAnswer.

Отсоединить буфер от столбца выборки

Синтаксис

```
L_LONG UnBindAnswer(
  L_SWORD); /* номер столбца выборки */
```

Описание

Метод отсоединяет ранее привязанный к столбцу выборки буфер.

Прототип LinAPI

LINTER_UnBindAnswer.

Получить указатель на объект «курсор»

Синтаксис

```
linCursor *GetCursor();
```

Описание

Метод возвращает переданный классу в конструкторе указатель на родительский объект linCursor.

Прототип LinAPI

Отсутствует.

Получить указатель на объект «соединение»

Синтаксис

```
linConnection *GetConnection();
```

Описание

Метод возвращает корневой родительский объект linConnection.

Прототип LinAPI

Отсутствует.

Класс linDataSet

Класс linDataSet представляет собой размещенный в оперативной памяти компьютера (кэшированный) набор данных, загруженный из БД СУБД ЛИНТЕР. Использование класса linDataSet позволяет уменьшить число запросов к БД, но ограничивает их возможности (для сложных запросов необходимо писать SQL-запросы на языке баз данных SQL).

Класс linDataSet реализует упрощенную реляционную модель БД со следующими возможностями:

- загрузку результата SQL-запроса выборки данных в оперативную память;
- обновление загруженной выборки данных путём добавления новых записей, удаления и модификации существующих (в случае, если SQL-запрос обновляемый);
- работу с BLOB-полями как в режиме предварительной загрузки всех значений BLOB-полей в память, так и в режиме индивидуальной работы с каждым из значений BLOB-полей;
- поиск значения на условия «равно», «не равно», «меньше», «больше», «меньше или равно», «больше или равно», «is NULL», «is NOT NULL» по одному или нескольким столбцам;
- сортировку значений по одному или нескольким столбцам;
- вычисление агрегатных функций (min, max, sum, count, avg) по загруженным столбцам;
- сохранение изменённых данных в БД либо в файле формата SQL (значения BLOB-полей сохраняются в виде отдельных файлов).

Большинство методов этого класса возвращают результат типа L_SWORD в виде одного или двух значений: LINAPI_ERROR или LINAPI_SUCCESS.

В случае если метод вернул значение LINAPI_ERROR, для получения диагностики следует вызвать методы linDataSet::GetErrorsCount, linDataSet::GetError.

Прототипы методов класса linDataSet в LinAPI-интерфейсе отсутствуют.

Конструктор класса

Синтаксис

```
linDataSet();
```

Описание

Метод создает конструктор класса linDataSet.

Деструктор класса

Синтаксис

```
~linDataSet();
```

Описание

Метод создает деструктор класса linDataSet.

Инициализировать класс

Синтаксис

```
L_SWORD Create (
  linConnection* pConnect,     /* указатель на объект */
                               /* класса linConnection */
  L_CHAR *SQLStr,              /* текст SQL-оператора */
  L_LONG lFlags = 0,           /* режим работы класса */
  L_LONG *plUniqCols = NULL,   /* номера столбцов выборки */
  L_LONG lUniqColsCount = 0); /* количество элементов */
                               /* в массиве plUniqCols */
```

Входные параметры

Параметр	Описание
pConnect	Предварительно созданный и проинициализированный объект класса linConnection
SQLStr	Оканчивающаяся двоичным нулем строка, содержащая SQL-запрос, предназначенный для выполнения выборки данных
lFlags	Режим работы класса: LDS_NLOAD_BLOB_BODY – при работе с BLOB-полями надо загружать в память только описатели (длина, тип BLOB-поля), а не значения BLOB-полей
plUniqCols	Массив номеров уникальных столбцов выборки (отсчет начинается с 1). Требование уникальности необходимо для обеспечения возможности обновления выборки – по этому/этим столбцам формируются запросы на удаление/модификацию записей выборки. По умолчанию, если это поле содержит NULL-указатель, считается, что уникальным является первый столбец в выборке
lUniqColsCount	Количество элементов в массиве plUniqCols

Выходные параметры

Отсутствуют.

Описание

Метод предназначен для начальной инициализации объекта класса linDataSet. Он открывает курсор и выполняет переданный SQL-запрос. Вся необходимая для работы класса информация (в том числе метаданные) о выборке заносится в оперативную память.

Деинициализировать класс

Синтаксис

```
L_SWORD Close();
```

Входные параметры

Отсутствуют.

Выходные параметры

Отсутствуют.

Описание

Метод выполняет деинициализацию объекта класса linDataSet. После деинициализации этот объект может быть повторно инициализирован с помощью вызова метода Create.

Получить указатель на объект «соединение»

Синтаксис

```
linConnection *GetConnection();
```

Входные параметры

Отсутствуют.

Выходные параметры

Отсутствуют.

Описание

Метод возвращает переданный классу методом linDataSet::Create указатель на объект linConnection.

Получить указатель на объект «курсор»

Синтаксис

```
linCursor *GetCursor();
```

Входные параметры

Отсутствуют.

Выходные параметры

Отсутствуют.

Описание

Метод возвращает указатель на объект linCursor (открытый с помощью указателя на объект linConnection, переданного классу методом linDataSet::Create).

Получить флаги класса

Синтаксис

```
L_LONG GetFlags();
```

Входные параметры

Отсутствуют.

Выходные параметры

Отсутствуют.

Описание

Метод возвращает переданные классу методом linDataSet::Create флаги.

Получить количество столбцов в выборке

Синтаксис

```
L_LONG GetColCount();
```

Входные параметры

Отсутствуют.

Выходные параметры

Отсутствуют.

Описание

Метод возвращает количество столбцов в выборке.

Получить количество строк в выборке

Синтаксис

```
L_LONG GetRowCount();
```

Входные параметры

Отсутствуют.

Выходные параметры

Отсутствуют.

Описание

Метод возвращает количество строк (записей) в выборке. После вызова linDataSet::Find вплоть до вызова linDataSet::FreeFind он возвращает количество записей, найденных по заданным условиям.

Проверить обновляемость выборки

Синтаксис

```
L_BOOL IsUpdatable();
```

Входные параметры

Отсутствуют.

Выходные параметры

Отсутствуют.

Описание

Метод возвращает информацию о том, является ли выборка обновляемой (true) или нет (false).

Получить описание столбца выборки

Синтаксис

```
t_ParamDesc *GetColInfo (
  L_LONG lColumn);          /* номер столбца */
```

Входные параметры

Параметр	Описание
lColumn	Порядковый номер столбца (отсчет начинается с 1)

Выходные параметры

Отсутствуют.

Описание

Метод возвращает информацию о столбце в структуре t_ParamDesc, описанной в linapi.h. В случае неудачи возвращается NULL-значение.

Присвоить значение полю выборки

Синтаксис

```
L_SWORD SetCellData (
  L_LONG lRow,              /* номер записи */
  L_LONG lColumn,           /* номер столбца */
  L_LONG lType,             /* тип данных */
  void *Data,               /* значение  данных */
  L_LONG DataLen,           /* длина значения */
  L_LONG lFlags = 0);       /* флаги */
```

Входные параметры

Параметр	Описание
lRow	Порядковый номер записи в выборке, в которой находится заполняемое поле (отсчет начинается с 1)
lColumn	Порядковый номер столбца в выборке, в котором находится заполняемое поле (нумерация начинается с 1)
lType	Тип данных, в котором передаётся значение для заполнения поля выборки
Data	Буфер, содержащий значение заполняемого поля выборки
DataLen	Длина значения
lFlags	Флаги, задающие атрибуты заполняемого поля выборки

Выходные параметры

Отсутствуют.

Описание

Метод устанавливает значение и атрибуты поля выборки.

Получить значение поля выборки

Синтаксис

```
L_SWORD GetCellData (
  L_LONG lRow,              /* номер записи */
  L_LONG lColumn,           /* номер столбца */
  L_LONG lType,             /* тип данных */
  void *Data,               /* значение данных */
  L_LONG *plDataLen,        /* длина значения */
  L_LONG *plFlags);         /* флаги */
```

Входные параметры

Параметр	Описание
lRow	Порядковый номер записи в выборке, в которой находится нужное поле (отсчет начинается с 1)
lColumn	Порядковый номер столбца в выборке, в котором находится нужное поле (нумерация начинается с 1)
lType	Тип данных, в котором требуется вернуть значение поля выборки
Data	Буфер для возвращаемого значения поля
plDataLen	Длина буфера
plFlags	Поле для возвращаемых атрибутов

Выходные параметры

Параметр	Описание
Data	Буфер со значением поля выборки
plDataLen	Реальная длина значения поля выборки
plFlags	Флаги, содержащие атрибуты поля выборки

Описание

Метод предоставляет значение и атрибуты поля выборки.

Если переданный буфер не позволяет разместить в нем значение поля, то возможны следующие варианты:

1) в поле plFlags будет выставлен флаг LDS_CD_SMALLBUFFER, а в поле plDataLen возвратится необходимый размер буфера;

2) в буфер будет скопирована часть значения (это связано со спецификой функции ConvertType библиотеки INTLIB);

3) помимо флага LDS_CD_SMALLBUFFER может быть возвращён индикатор NULL-значения LDS_CD_NULL, если поле содержит NULL-значение.

Получить длину значения поля выборки

Синтаксис

```
L_SWORD GetValueLength (
  L_LONG lRow,              /* номер записи */
  L_LONG lColumn,           /* номер столбца */
  L_LONG *plLength);        /* буфер для длины значения */
```

Входные параметры

Параметр	Описание
lRow	Порядковый номер записи в выборке, в которой находится нужное поле (отсчет начинается с 1)
lColumn	Порядковый номер столбца в выборке, в котором находится нужное поле (нумерация начинается с 1)
plLength	Буфер для возвращаемой длины значения

Выходные параметры

Параметр	Описание
plLength	Длина значения в буфере

Описание

Метод возвращает длину значения поля выборки (актуально для BLOB-полей, но может применяться и для остальных типов данных).

Установить тип BLOB-поля выборки

Синтаксис

```
L_SWORD SetBlobType (
  L_LONG lRow,              /* номер записи */
  L_LONG lColumn,           /* номер столбца */
  L_LONG lBlobType);        /* тип BLOB-данных */
```

Входные параметры

Параметр	Описание
lRow	Порядковый номер записи в выборке, в которой находится нужное поле (отсчет начинается с 1)
lColumn	Порядковый номер столбца в выборке, в котором находится нужное поле (нумерация начинается с 1)
lBlobType	Устанавливаемый тип BLOB-данных

Выходные параметры

Отсутствуют.

Описание

Метод устанавливает тип BLOB-данных в поле выборки.

Примечание
Изменить только тип BLOB-данных нельзя. Для сохранения изменений выборки необходимо также, чтобы было изменено и само значение BLOB-поля.

Получить тип BLOB-поля выборки

Синтаксис

```
L_SWORD GetBlobType (
  L_LONG lRow,              /* номер записи */
  L_LONG lColumn,           /* номер столбца */
  L_LONG *plBlobType);      /* буфер для типа BLOB-данных */
```

Входные параметры

Параметр	Описание
lRow	Порядковый номер записи в выборке, в которой находится нужное поле (отсчет начинается с 1)
lColumn	Порядковый номер столбца в выборке, в котором находится нужное поле (нумерация начинается с 1)
*plBlobType	Буфер для возвращаемого типа BLOB-данных

Выходные параметры

Параметр	Описание
plBlobType	Тип BLOB-данных

Описание

Метод возвращает тип данных BLOB-поля.

Добавить запись в выборку

Синтаксис

```
L_SWORD AddRow (
  L_LONG *plRow = NULL);    /* номер добавленной записи */
```

Входные параметры

Параметр	Описание
*plRow	Буфер для возвращаемого номера добавленной записи

Выходные параметры

Параметр	Описание
plRow	Номер добавленной записи

Описание

Метод добавляет новую запись в выборку. Все поля данной записи инициализируются NULL-значениями.

Если задан буфер plRow, то в него будет записан порядковый номер добавленной записи в выборке (нумерация начинается с 1). Новая запись всегда добавляется в конец выборки, даже если выборка перед добавлением записи была отсортирована. Т.е. для сохранения сортировки следует вызвать метод linDataSet::FreeSort и повторно выполнить сортировку.

Если предварительно был выполнен поиск с помощью метода linDataSet::Find, то добавленная запись не учитывается в результате поиска – для этого поиск потребуется выполнить повторно.

Удалить запись из выборки

Синтаксис

```
L_SWORD DeleteRow (
  L_LONG lRow);              /* номер удаляемой записи */
```

Входные параметры

Параметр	Описание
lRow	Порядковый номер удаляемой записи в выборке (нумерация начинается с 1)

Выходные параметры

Отсутствуют.

Описание

Метод удаляет запись из выборки.

Создать индекс для столбца выборки

Синтаксис

```
L_SWORD CreateIndex (
  L_LONG lColumn);           /* номер столбца */
```

Входные параметры

Параметр	Описание
lColumn	Порядковый номер столбца в выборке, для которого должен быть создан индекс (нумерация начинается с 1)

Выходные параметры

Отсутствуют.

Описание

Метод создаёт индекс (внутреннюю структуру класса) для заданного столбца выборки.

Удалить индекс столбца выборки

Синтаксис

```
L_SWORD DropIndex (
  L_LONG lColumn);          /* номер столбца */
```

Входные параметры

Параметр	Описание
lColumn	Порядковый номер столбца в выборке, у которого должен быть удален индекс (нумерация начинается с 1)

Выходные параметры

Отсутствуют.

Описание

Метод удаляет индекс (внутреннюю структуру класса) для заданного столбца выборки.

Сохранить выборку

Синтаксис

```
L_SWORD SaveChanges (
  L_CHAR *szFileName=NULL);  /* место сохранения */
```

Входные параметры

Параметр	Описание
szFileName	Спецификация (путь и имя) файла для сохранения выборки

Выходные параметры

Отсутствуют.

Описание

Метод сохраняет все изменённые данные выборки в БД или, если задан параметр szFileName, в указанном файле. В случае если соединение было открыто в транзакционном режиме, после выполнения последовательности SQL-запросов выполняется команда commit. Если при выполнении запросов возникают ошибки связи (например, разрывается соединение), соединение вновь пытается восстановиться и выполнить запросы в течение определённого интервала времени, т. е. всё происходит таким же образом, как для класса linConnection.

Параметр szFileName задаёт спецификацию файла, в котором будет сохранена последовательность SQL-запросов по изменению данных выборки. Должен быть указан полный путь к файлу, который открывается в режиме "wb", т.е. создаётся новый файл только для записи, при этом происходит перезапись любого существующего файла с тем же именем.

Изменённые значения BLOB-полей сохраняются в отдельных файлах в том же каталоге, что и файл с SQL-запросами. При этом среди SQL-запросов вставлены комментарии,

поясняющие, где и каким образом должно быть добавлено содержимое файлов с BLOB-значениями.

Пример

```
При выполнении метода linDataSet::SaveChanges ("c:\\sqldata.sql")
 в случае
 наличия модифицированных данных в выборке могут быть созданы
 файлы sqldata.sql
 и sqldata_1_6.blb (если выборка содержит BLOB-данные). Содержимое
 SQL-файла
 будет примерно следующим:

update "SYSTEM"."TEST" set "J"=222, "CH"=NULL, "VCH"='modified
 value1',
 "BL"=NULL where "I"=2;
! linCursor::PurgeBlob(4, NULL, NULL)
! linCursor::AddBlob(4, 10, 'c:\sqldata_1_6.blb', 4096, NULL,
 NULL)
update "SYSTEM"."TEST" set "BL"=NULL where "I"=3;
! linCursor::PurgeBlob(1, NULL, NULL)
```

Изменить условия поиска данных

Синтаксис

```
L_SWORD AddCondition (
  L_LONG lColumn,           /* номер столбца */
  L_LONG lType,             /* тип данных */
  void *Data,               /* условие поиска */
  L_LONG lDataLen,          /* длина значения условия поиска */
  L_SWORD Condition=fdEQ);  /* тип условия */
```

Входные параметры

Параметр	Описание
lColumn	Порядковый номер столбца в выборке (нумерация начинается с 1), для которого добавляется условие поиска
lType	Тип данных, в котором передано значение условия поиска
*Data	Буфер со значением условия поиска
lDataLen	Длина значения условия поиска
Condition	Тип условия поиска (таблица 1)

Таблица 1. Поддерживаемые условия поиска данных

Условие	Символьное обозначение, SQL	Описание
linDataSet:: fdEQ	column = value	Условие «равно»
linDataSet:: fdNEQ	column <> value	Условие «не равно»

Условие	Символьное обозначение, SQL	Описание
linDataSet:: fdGT	column > value	Условие «больше»
linDataSet:: fdLT	column < value	Условие «меньше»
linDataSet:: fdGE	column >= value	Условие «больше или равно»
linDataSet:: fdLE	column <= value	Условие «меньше или равно»
linDataSet:: fdNL	column is NULL	Условие на «NULL»
linDataSet:: fdNNL	column is not NULL	Условие на «не NULL»

Выходные параметры

Отсутствуют.

Описание

Метод задаёт условие поиска (если оно не было задано) или добавляет новое условие к ранее установленным.

Выполнить поиск данных

Синтаксис

```
L_SWORD Find (
  L_LONG lFindFrom=1);       /* начало поиска */
```

Входные параметры

Параметр	Описание
lFindFrom	Порядковый номер записи в текущей выборке, начиная с которого будет производиться поиск (нумерация с 1)

Выходные параметры

Отсутствуют.

Описание

Метод выполняет поиск отсортированных и несортированных данных в соответствии с ранее заданными условиями поиска. Если перед выполнением метода Find ранее также выполнился метод Find, но не был вызван метод FreeFind, то поиск осуществляется в уже найденных результатах.

После выполнения поиска результат поиска как бы подменяет собой текущую выборку, т.е. для того, чтобы извлечь результат поиска, следует вызывать методы GetRowCount, GetCellData.

Отменить условия поиска

Синтаксис

```
L_SWORD FreeFind();
```

Входные параметры

Отсутствуют.

Выходные параметры

Отсутствуют.

Описание

Метод очищает все условия поиска и результат поиска. После вызова данного метода выборка возвращается к тому состоянию, в котором она находилась до вызова метода Find.

Выполнить агрегатную функцию

Синтаксис

```
L_SWORD GetAggregate (
  L_LONG lColumn,           /* номер столбца */
  L_SWORD AggFunc,          /* тип функции */
  L_LONG lType,             /* тип данных */
  void *Data,               /* буфер для результата */
  L_LONG *plDataLen,        /* длина буфера */
  L_LONG *plFlags);         /* флаги */
```

Входные параметры

Параметр	Описание
`lColumn`	Порядковый номер столбца в выборке (нумерация начинается с 1), по которому вычисляется агрегатная функция
`AggFunc`	Тип агрегатной функции (таблица 2)
`lType`	Тип данных, в котором требуется получить результат
`*Data`	Буфер для результата работы функции
`*plDataLen`	Длина буфера
`*plFlags`	Возвращаемые атрибуты вычисленного функцией значения

Таблица 2. Типы поддерживаемых агрегатных функций

Агрегатная функция	Символьное обозначение, SQL	Описание
linDataSet::gaMin	min(column)	Функция возвращает наименьшее значение из заданного множества
linDataSet::gaMax	max(column)	Функция возвращает наибольшее значение из заданного множества
linDataSet::gaCount	count(column)	Функция возвращает число ячеек столбца без учёта NULL-значений
linDataSet::gaAvg	avg(column)	Функция возвращает среднее арифметическое значение набора числовых значений

Агрегатная функция	Символьное обозначение, SQL	Описание
linDataSet::gaSum	sum(column)	Функция возвращает сумму множества числовых значений

Выходные параметры

Параметр	Описание
`Data`	Результат работы функции
`plDataLen`	Длина вычисленного функцией значения
`plFlags`	Флаги атрибутов вычисленного функцией значения

Описание

Метод вычисляет агрегатную функцию для требуемого столбца. Если перед вызовом функции был выполнен поиск, агрегатная функция вычисляется для результата поиска.

Примечание
Функции linDataSet::gaAvg и linDataSet::gaSum применимы только к столбцам числового типа.

Задать условия сортировки данных

Синтаксис

```
L_SWORD AddSortColumn (
  L_LONG lColumn,               /* номер столбца */
  L_WORD SortDirection=sAsc);   /* тип сортировки */
```

Входные параметры

Параметр	Описание
`lColumn`	Порядковый номер столбца в выборке (нумерация начинается с 1), для которого добавляется условие сортировки
`SortDirection`	Тип сортировки (таблица 3)

Таблица 3. Поддерживаемые типы сортировок

Тип сортировки	Символьное обозначение, SQL	Описание
linDataSet:: sAsc	order by column asc	Сортировка по возрастанию
linDataSet:: sDesc	order by column desc	Сортировка по убыванию

Выходные параметры

Отсутствуют.

Описание

Метод добавляет условие сортировки для формирования порядка сортировки. При последовательных вызовах данного метода могут быть добавлены несколько столбцов для сортировки, в этом случае порядок сортировки аналогичен порядку для конструкции

ORDER BY языка SQL, т.е. основная сортировка проводится для первого добавленного столбца.

Выполнить сортировку данных

Синтаксис

```
L_SWORD Sort();
```

Входные параметры

Отсутствуют.

Выходные параметры

Отсутствуют.

Описание

Метод выполняет сортировку данных в выборке в соответствии с ранее сформированным порядком сортировки. Столбцы, по которым проводится сортировка, должны быть в обязательном порядке проиндексированы. Сортировка также действует на порядок найденных при поиске записей.

После выполнения сортировки её результат как бы подменяет собой текущую выборку, т.е. для того, чтобы извлечь результат сортировки, следует вызывать методы GetRowCount, GetCellData, как при работе с обычной выборкой.

NULL-значения при сортировке по возрастанию добавляются в конец выборки, при сортировке по убыванию – в начало.

Отменить условия и результат сортировки данных

Синтаксис

```
L_SWORD FreeSort();
```

Входные параметры

Отсутствуют.

Выходные параметры

Отсутствуют.

Описание

Метод очищает все заданные условия сортировки и сам результат сортировки. После вызова данного метода выборка возвращается к тому состоянию, в котором она находилась до вызова метода Sort.

Проверить буфер кодов завершения

Синтаксис

```
L_ULONG GetErrorsCount();
```


Описание

Метод возвращает количество ошибок, запомненных в буфере ошибок. Его имеет смысл вызывать, если один из прочих методов вернул код завершения, отличный от LINAPI_SUCCESS.

Очистить буфер кодов завершения

Синтаксис

```
void    ClearErrors();
```

Описание

Метод очищает буфер ошибок. Он применяется перед вызовом практически всех функций библиотеки LinAPI внутри классов linConnection, linCursor и linStatement, т. к. предполагается, что если пользователь не извлёк ошибки из буфера после вызова предыдущего метода класса, то ему они не нужны.

Получить информацию о коде завершения

Синтаксис

```
L_LONG GetError (
  L_CHAR  *,  /* буфер для имени модуля, в котором произошла
 ошибка */
  L_ULONG *,  /* длина буфера для имени модуля */
  L_CHAR  *,  /* буфер для текста кода завершения */
  L_ULONG *,  /* длина буфера для текста кода завершения */
  L_LONG  *,  /* буфер для кода завершения LinAPI */
  L_LONG  *,  /* буфер для кода завершения СУБД ЛИНТЕР */
  L_LONG  *); /* буфер для кода завершения ОС */
```

Описание

Метод извлекает ошибку из буфера ошибок. Для аргументов метода с типом L_ULONG* требуется передать указатели на переменные, в которых хранятся размер буфера для получения имени модуля и размер буфера для получения текста сообщения соответственно. В эти же переменные метод запишет реальные размеры строк с именем модуля и текстом сообщения (в том случае, если буфер для строк окажется недостаточным, строки будут усечены).

Управление транзакциями

В любом транзакционном режиме, отличном от mAutocommit, осуществляется автоматическое сохранение вызовов ряда методов, а затем, в случае разрыва и восстановления соединения, сохранённые методы вызываются заново. Буфер для сохранения вызова методов (и их параметров) привязан к соединению (в случае, если установлен режим сохранения вызова методов SetUseConnQBuf(L_TFALSE), то к курсору) и является двухуровневым. Первый уровень имеет размер 64 Кбайта. В случае если размер буфера окажется недостаточным, данные вытесняются во временный файл на диске.

В буфере сохраняются вызовы и их параметры для следующих методов:

- linConnection::SetConnectOption;
- linCursor::ExecuteDirect;
- linCursor::LockRow;
- linCursor::UnlockRow;
- linCursor::ClearBlob;
- linCursor::PurgeBlob;
- linCursor::AppendBlob;
- linCursor::AddBlob;
- linCursor::SetBlobType;
- linCursor::SetCursorOption;
- linCursor::Fetch;
- linCursor::CreateStatement;
- linCursor::CommitCursor;
- linCursor::RollBackCursor;
- linStatement::ExecuteStatement;
- linStatement::SetStatementOption;
- linStatement::BindParameter.

Особенности сохранения и восстановления метода linStatement::BindParameter:

1) сохранение выполняется непосредственно перед **каждым вызовом** метода linStatement::ExecuteStatement, даже если в реальности BindParameter вызывается однократно. Это связано с тем, что содержание одних и тех же буферов метода BindParameter перед каждым из вызовов ExecuteStatement может отличаться, в зависимости от того, что в них будет записано клиентским приложением;
2) при восстановлении транзакции буферы для BindParameter создаются заново, причём их освобождение происходит только в двух случаях: либо после вызова метода ExecuteStatement, либо при уничтожении объекта класса linStatement, к которому они привязаны.

Приложение

Пример работы с библиотекой LincppAPI

```
#include <stdio.h>
#include "lincppapi.h"

int main(void)
{
  L_LONG         lRet;     /* return code */
  L_CHAR         date[30]; /* date as a string */
  linCursor     *pCursor;
  linConnection *pConnect = new linConnection;

  if (!pConnect)
  {
    ptintf("No memory\n")
    exit(1);
  }

  if (lRet = pConnect->Connect("SYSTEM", 0, "MANAGER", 0, NULL,
 mAutocommit))
    processing_error(pConnect, lRet, "ERROR Connect");

 printf("Open cursor\n");
 if (lRet = pConnect->OpenCursor(NULL, 0, &pCursor))
   processing_error(pConnect, lRet, "Error open cursor");

 printf("ExecuteDirect: 'create or replace table TDATE ( D
 date )'\n");
 if (lRet = pCursor->ExecuteDirect("create or replace table TDATE
 (D date)", 0))
   processing_error(pConnect, lRet, "Error ExecuteDirect 1");

 printf("ExecuteDirect: 'insert into TDATE values(sysdate)'\n");
 if (lRet = pCursor->ExecuteDirect("insert into TDATE
 values(sysdate)", 0))
   processing_error(pConnect, lRet, "Error ExecuteDirect 2");

 printf("ExecuteDirect: 'select D from TDATE'\n");
 if (lRet = pCursor->ExecuteDirect("select D from TDATE", 0))
   processing_error(pConnect, lRet, "Error ExecuteDirect 3");

 printf("GetData (as tString)\n");
 if (lRet = pCursor->GetData(1,
                             date,     /* answer bufer */
                             30,       /* length of bufer */
```

```
                                tString,  /* type of the answer */
                                0, 0, NULL))
    processing_error(pConnect, lRet, "Error GetData");

  printf("    DATE : %s\n", date);

  printf(" ExecuteDirect: 'drop table TDATE'\n");
  if (lRet = pCursor->ExecuteDirect("drop table TDATE", 0))
    processing_error(pConnect, lRet, "Error ExecuteDirect 4",
 false);

  printf("CloseCursor\n");
  if (lRet = pConnect->CloseCursor(pCursor))
    processing_error(pConnect, lRet, "Error CloseCursor");

  printf("CloseConnect\n");
  if (lRet = pConnect->CloseConnect())
    processing_error(pConnect, lRet, "Error CloseConnect");

  delete pConnect;
}
```


Указатель методов