СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

Интерфейс нижнего уровня

НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

РЕЛЭКС

Товарные знаки

РЕЛЭКС™, ЛИНТЕР® являются товарными знаками, принадлежащими АО 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: info@linter.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

Документ содержит описание команд взаимодействия клиентского приложения с СУБД ЛИНТЕР на уровне вызова функции (уровень CALL-интерфейса).

Документ предназначен для СУБД ЛИНТЕР СТАНДАРТ 6.0 сборка 20.3, далее по тексту СУБД ЛИНТЕР.

Для кого предназначен документ

Документ предназначен для программистов, разрабатывающих клиентские приложения на основе СУБД ЛИНТЕР с использованием языка программирования C/C++.

Необходимые предварительные знания

Для работы с интерфейсом нижнего уровня необходимо знать:

- язык программирования C/C++;
- язык баз данных SQL.

Дополнительные документы

- Библиотеки специальных типов данных
- Справочник кодов завершения
- Справочник по SQL
- Запуск и останов СУБД ЛИНТЕР в среде ОС Windows
- Запуск и останов СУБД ЛИНТЕР в среде ОС Linux, Unix
- Системные таблицы и представления
- Создание и конфигурирование базы данных

Назначение и условия применения программы

Назначение программы

Интерфейс нижнего уровня СУБД ЛИНТЕР является инструментальным средством разработки клиентских приложений базы данных (БД) ЛИНТЕР с использованием языков программирования C/C++. Гибкий и мощный набор команд интерфейса позволяет разработчику приложения получать доступ ко всем возможностям СУБД ЛИНТЕР и адаптировать приложение для оптимальной производительности. Интерфейс нижнего уровня является основой всех остальных программных интерфейсов СУБД ЛИНТЕР. Приложение, разработанное на базе стандартного языка программирования C/C++ и использующее интерфейс нижнего уровня, будет переносимым на все операционные системы, в которых функционирует СУБД ЛИНТЕР.

Условия применения

Интерфейс нижнего уровня между клиентским приложением и ядром СУБД ЛИНТЕР обеспечивается с помощью вызова функции `inter`. Функция реализована для ОС типа Windows в библиотеке `inter325` или `inter64` (в зависимости от установленной разрядности СУБД ЛИНТЕР) или для ОС типа Linux, ЗОСРВ Нейтрино в библиотеке `intlib`. Обычно в дистрибутиве поставляются библиотеки для разных компиляторов и разных типов сборки, что позволяет использовать интерфейс так, как необходимо программе.

Для подключения интерфейса нижнего уровня необходимо включить в пользовательскую программу заголовочный файл `inter.h`, расположенный в подкаталоге `intlib` установочного каталога СУБД ЛИНТЕР.

Для сборки приложений, использующих интерфейсы СУБД ЛИНТЕР, должен быть определен ряд макросов, определяющих окружение сборки – операционную систему, особенности процессора, версию ЛИНТЕР. В Linux, ЗОСРВ Нейтрино все эти макросы собраны в файле `Definition`, который может быть включен в `makefile` сборки приложения. Все необходимые макросы собраны в переменных `$(SYSTEMS)`, `$(OS)`, `$(PACKING)`, `$(VERSION)`.

В ОС Windows для сборки необходимо определить макросы `INTER_MSWINDOWS`, `WIN32`, `_VER_MAX=600`. Список кодов ОС и макросов приведён в приложениях 1, 2 соответственно.

Примечание
Для сборки приложений рекомендуется использовать в качестве примера имеющиеся файлы сборки `makefile` из подкаталога `/samples` установочного каталога СУБД ЛИНТЕР.

В программе может потребоваться список символических имён кодов завершения СУБД ЛИНТЕР и её программного интерфейса. Этот список находится в файле `errors.h` в подкаталоге `intlib`.

При использовании в пользовательской программе специальных типов данных (десятичные числа с фиксированной точкой, тип данных «дата-время» и длинные целые числа), неподдерживаемых стандартным компилятором C/C++, необходимо

включить в текст программы заголовочные файлы соответствующих библиотек (см. документ «Библиотеки специальных типов данных», разделы «Библиотека Decimals», «Библиотека Tick», «Библиотека Int64» и подключить к программе в процессе сборки соответствующие библиотеки.

В дистрибутив СУБД ЛИНТЕР входят исходный и заголовочный файлы интерфейса нижнего уровня `intlib.c` и `intlib.h`, которые можно использовать для полного контроля за приложением в устройствах специального назначения. Модуль `intlib.c` может быть оттранслирован пользователем самостоятельно (макросы трансляции приведены в приложении 2).

Примечание

За ошибки, связанные с пользовательскими правками функции `inter`, разработчики СУБД ЛИНТЕР ответственности не несут.

Характеристики программы

Объемные характеристики

Максимальные длины обрабатываемого СУБД ЛИНТЕР SQL-запроса и возвращаемой выборки данных равняются 64 Кбайт.

Буфер обмена данными между клиентским приложением и СУБД ЛИНТЕР, поддерживаемый интерфейсом нижнего уровня, равен 8192 байтам. В него включаются управляющие и информационные блоки. Так как средний суммарный размер управляющих блоков интерфейса составляет примерно 108 байт, то для информационных блоков (текст посылаемого SQL-запроса или возвращаемая запись выборки данных) предоставляется примерно 8000 байт памяти. Если длина запроса превышает допустимый размер информационного блока, то разбивка на порции посылаемого запроса и сборка выборки данных из возвращаемых порций выполняются автоматически.

Примечание
Среди команд собственно интерфейса нижнего уровня отсутствуют команды, возвращающие выборку данных длиной более 8 Кбайт. Длинную выборку данных могут возвращать только SQL-запросы к БД.

Контроль выполнения

Интерфейс нижнего уровня осуществляет выявление и диагностику всех ошибочных ситуаций, которые могут быть обнаружены на его уровне (недопустимые значения или несовместимость переданных параметров, отсутствие обязательных параметров и т.п.), и при обнаружении ошибки прекращает дальнейшую обработку команды, возвращая соответствующий код завершения.

Доступ к базе данных

Взаимодействие клиентского приложения и ядра СУБД ЛИНТЕР осуществляется через функцию `inter`, которая имеет следующий формат вызова:

```
inter(CBL, VarBuf, OpBuf, CondBuf, RowBuf);
```

где:

- `CBL` – адрес блока управления запросом (контрольного блока);
- `VarBuf` – адрес буфера параметров команды;
- `OpBuf` – адрес буфера SQL-запроса;
- `CondBuf` – адрес подпрограммы асинхронной обработки;
- `RowBuf` – адрес буфера записи (выборки данных).

Описание параметров вызова приведено в разделе «Входные данные».

Функция `inter` осуществляет инициализацию интерфейса, передачу данных серверу и прием от него результата.

Примечание
Понятие *запрос* в интерфейсе нижнего уровня объединяет в себе команду ядра СУБД ЛИНТЕР и ее входные (выходные) параметры.

Деинициализация интерфейса выполняется функцией UninitLinterClient():

```
void UninitLinterClient(void);
```

При деинициализации освобождаются все ресурсы, используемые интерфейсом.

Примечания

1. В ОС Linux после системного вызова функции fork необходимо вызвать функцию UninitLinterClient в дочернем процессе.
2. Для ОС Linux с ядрами версии меньше 2.6 функцию UninitLinterClient() необходимо вызывать из той же нити, из которой первый раз успешно вызвалась функция inter.

Входные данные

Блок управления запросом CBL

Блок управления запросом `CBL` (контрольный блок интерфейса) содержит управляющую информацию, необходимую СУБД ЛИНТЕР для выполнения запроса. Контрольный блок является **обязательным** параметром функции `inter`.

```
typedef TCBL
{
    L_LONG  CodErr;
    L_WORD  Prior;
    L_WORD  NumChan;
    L_CHAR  UserName[4];
    L_CHAR  Command[4];
    L_CHAR  Node[8];
    L_LONG  RowId;
    L_LONG  RowCount;
    L_LONG  PrzExe;
    L_LONG  SysErr;
    L_WORD  LnBufRow;
    L_WORD  CharSet;
};
```

В процессе взаимодействия клиентского приложения и СУБД ЛИНТЕР значения некоторых полей контрольного блока устанавливаются либо только клиентским приложением, либо только СУБД ЛИНТЕР (системные поля), а некоторые поля – поочередно приложением и СУБД ЛИНТЕР.

Описание полей контрольного блока приведено в таблице 1. Типы полей соответствуют типам данных, приведенным в приложении 3, знак √ указывает на то, что поле заполняется соответствующим процессом.

Таблица 1. Поля блока управления запросом CBL

Имя поля	Описание	Кто заполняет	
		приложение	СУБД
`CodErr`	Код завершения запроса к СУБД ЛИНТЕР		√
`Prior`	Приоритет канала	√	
`NumChan`	Номер канала	√	√
`UserName`	Резерв		
`Command`	Команда СУБД ЛИНТЕР	√	
`Node`	Имя ЛИНТЕР-сервера	√	
`RowId`	Значение специфично для каждой команды	√	√
`RowCount`	Число найденных записей выборки данных		√

Имя поля	Описание	Кто заполняет	
		приложение	СУБД
PrzExe	Флаги выполнения запроса	√	
SysErr	Код состояния операционной системы		√
LnBufRow	Длина буфера записи (выборки данных)	√	√
CharSet	Идентификатор кодовой страницы	√	

Структура контрольного блока оформлена в виде заголовочного файла inter.h (для C/C++). Обращаться к полям контрольного блока в пользовательских C/C++ программах можно после включения в исходный текст заголовочного файла (с помощью директивы #include "inter.h").

Поля контрольного блока

CodErr

Код завершения запроса к СУБД ЛИНТЕР. Устанавливается СУБД ЛИНТЕР. В это поле заносятся коды завершения, сформированные самим интерфейсом нижнего уровня (при неправильном вызове функции inter), и результаты обработки команды СУБД ЛИНТЕР. Поле CodErr дублируется значением, возвращаемым непосредственно функцией. Так, например, если адрес контрольного блока при вызове функции был передан неверно, то такую ошибку можно обнаружить только через анализ кода завершения самой функции inter. Список возможных кодов завершения приведен в документе «Справочник кодов завершения», подраздел «Коды завершения при коммуникационных операциях (1000-1011)».

Prior

Приоритет канала; может иметь значения от 0 (самый низкий приоритет) до 249 (высший приоритет). Используется для изменения очередности обработки запросов ядром СУБД. В большинстве приложений рекомендуется задавать нулевой приоритет, так как в этом случае ядро выделяет поочередно одинаковый квант времени для всех запросов, и обработка запросов выполняется в псевдопараллельном режиме. Более высокий приоритет канала следует устанавливать в приложениях, предъявляющих повышенные требования к временным характеристикам.

NumChan

Номер канала СУБД, по которому выполняется обработка запроса. При создании (открытии) канала СУБД возвращает в этом поле номер, назначенный созданному каналу. Если пользовательская программа открыла несколько каналов и использует только один контрольный блок, то при каждом обращении к СУБД в этом поле должен явно задаваться номер канала, по которому происходит обработка команды СУБД. При этом содержимое остальных полей контрольного блока зависит от подаваемого запроса: если запрос является продолжением обработки последнего выполненного по данному каналу запроса, то контрольный блок должен содержать контекст последнего запроса, если предполагается выполнение нового запроса, контрольный блок должен содержать контекст нового запроса. В обоих случаях поля, значения которых использовались для открытия канала, не должны изменяться. Если же открыт один канал или несколько, но каждый из них имеет собственный контрольный блок, то это поле можно не формировать (оно остается неизменным после открытия канала в течение всего сеанса работы с СУБД).

UserName

Имя пользователя. Зарезервировано для будущего использования.

`Command`

Имя команды интерфейса нижнего уровня.

`Node`

Имя ЛИНТЕР-сервера (см. таблицу 1). Если поле содержит пробелы или двоичные нули, предполагается работа с сервером по умолчанию.

`RowId`

Внутренний системный номер записи, которая была последней обработана (найдена/добавлена/изменена/удалена) в данном канале. Заполняется СУБД ЛИНТЕР только при выполнении запроса на обработку данных. В некоторых случаях устанавливается приложением для прямого доступа к этой записи.

`RowCount`

Количество реально обработанных (найденных, добавленных, удаленных или измененных) записей при выполнении запроса. Заполняется СУБД ЛИНТЕР после нормального завершения обработки запроса.

`PrzExe`

Указание о режиме обработки команды интерфейса нижнего уровня. Режим устанавливается пользовательской программой в контексте команды СУБД ЛИНТЕР. Конкретные значения этого поля приведены в таблице 2.

`SysErr`

Код ошибки, переданный СУБД операционной/сетевой средой при обработке запроса. В это поле передаются, в основном, коды завершения файловых операций и операций обмена данными ОС. Значение `SysErr` используется для уточнения и детализации кода завершения, сформированного ядром СУБД ЛИНТЕР. Так, например, код завершения СУБД ЛИНТЕР `Ошибка создания файла` может быть вызван разными причинами: отсутствием свободного места на устройстве, неактивным состоянием устройства и т.п. Код завершения, помещаемый в поле `SysErr` для однотипных ошибочных ситуаций, в различных операционных средах может иметь разное значение, поэтому для его анализа следует руководствоваться справочными документами соответствующей операционной системы.

`LnBufRow`

Длина буфера записи выборки данных (`RowBuf`) при обработке команды. Для входных параметров – это предполагаемая или максимально возможная длина записи выборки данных, для выходных – реальная (фактически полученная) длина записи выборки данных. В первом случае поле заполняется пользовательской программой, во втором – СУБД ЛИНТЕР. Если реальная длина записи выборки данных окажется больше, чем длина буфера выборки данных, выполнение SELECT-запроса будет прервано с выдачей соответствующего кода завершения (кроме случая использования `RowBuf` в команде `OPEN`). Если буфер приложения может вместить запись выборки данных, это поле будет содержать реальную длину выбранной записи. Так как значение в `LnBufRow` изменяется после каждой переданной записи, то при переходе к другим SELECT-запросам нужно вновь инициализировать это поле.

`CharSet`

Номер кодовой страницы (для внутреннего использования СУБД ЛИНТЕР).

Таблица 2. Возможные значения поля PrzExe контрольного блока

Значение PrzExe	Макроопределение в intlib2.h	Спецификация
0x00000000	M_BINARY	Передать выборку данных на SELECT-запрос в формате без спецификаций полей записи
0x00000003	M_SPEC	Передать выборку данных на SELECT-запрос в формате со спецификациями полей записи
0x00000100	M_OPTIMISTIC	Устанавливает режим обработки транзакций OPTIMISTIC **Примечание** Режим OPTIMISTIC устарел. Применять не рекомендуется.
0x00000200	M_SHARE	Не используется
0x00000400	M_EXCLUSIVE	Устанавливает режим обработки транзакций EXCLUSIVE (PESSIMISTIC)
0x00001000	Q_ASYNC	Устанавливает режим асинхронного взаимодействия с СУБД ЛИНТЕР
0x00002000	Q_ASYNCDONE	Устанавливается интерфейсом нижнего уровня по факту завершения обращения к СУБД ЛИНТЕР при работе в асинхронном режиме
0x00400000	Q_USE_UTF8	Транслировать SQL-запросы в кодировке UTF8 независимо от установленной клиентской кодировки
0x00800000	Q_USE_ADO	Информировать о наличии заблокированных записей при выборке данных (см. команды GETM, GETN, GETP). Зарезервировано для будущего применения
0x40000000	Q_KRBREQ	Идентификация и аутентификация пользователя должна выполняться по Kerberos-протоколу

Если поле `PrzExe` не заполнено, устанавливаются следующие параметры канала:

- режим обработки транзакций AUTOCOMMIT;
- синхронный режим выполнения запросов;
- формат выборки данных не содержит спецификации полей записи;
- режим приема/передачи данных: по умолчанию.

Режим приема/передачи данных для СУБД ЛИНТЕР по умолчанию выбирается следующим образом:

- если по каналу подан SQL-запрос **set names** (кодировка соединения по умолчанию) (см. документ «Справочник по SQL», пункт «Кодировка соединения по умолчанию»),

то используется заданная в запросе кодовая страница (при условии, что она загружена в БД);

- иначе, если задана переменная окружения `LINTER_CP`, то используется указанная в этой переменной кодовая страница (при условии, что она загружена в БД);
- иначе используется кодировка по умолчанию, которая автоматически определяется СУБД ЛИНТЕР в соответствии с правилами, диктуемыми ОС, в которой функционирует СУБД;
- если имя кодовой страницы (заданное любым из перечисленных выше способов) неверное, то будет использована кодовая страница `DEFAULT` (только первые 127 символов).

Буфер запроса OpBuf

Буфер запроса `OpBuf` предназначен для записи полного текста SQL-запроса в символьном виде. Он используется с теми командами интерфейса нижнего уровня, для выполнения которых требуется спецификация запроса обработки.

Буфер параметров команды VarBuf

Буфер `VarBuf` предназначен для передачи дополнительных параметров команды (например, параметров регистрации пользователя в команде `OPEN`). Правила заполнения этого буфера приведены при описании соответствующих команд.

Адрес процедуры асинхронной обработки запроса CondBuf

Параметр `CondBuf` задает адрес пользовательской процедуры завершения обработки асинхронного запроса. Параметр имеет смысл только в том случае, если поданный на выполнение запрос должен обрабатываться ядром СУБД асинхронно с клиентским приложением.

Выходные данные

Блок управления запросом CBL

Структура контрольного блока описана в разделе «Входные данные».

Буфер записи (буфер выборки данных) RowBuf

Буфер `RowBuf` предназначен для размещения в нем записей (выборки данных), получаемых при обработке SELECT-запроса. Записи в `RowBuf` могут передаваться в двоичном формате или в специфицированном. Вид представления записей устанавливается в поле `PrzExe` контрольного блока: значение M_BINARY (M_SPEC) задает представление выборки данных без спецификаций (со спецификациями) полей записи.

Использование значения M_BINARY (без спецификаций полей записи) предполагает знание структуры хранимой в БД информации. Такой формат выборки данных называется *неспецифицированным* (таблицы 3 и 4). В этом случае буфер выборки данных можно описать как структуру (с жестким типом), которая будет заполнена реальной информацией после завершения обработки запроса.

Примечание
При использовании для выборки данных структур данных необходимо помнить, что СУБД ЛИНТЕР подготавливает записи выборки данных в упакованном виде, т.е. без выравнивания полей, каждое следующее поле начинается с байта, непосредственно следующего за концом предыдущего поля.

При задании значения M_SPEC (со спецификациями полей записи) в буфер выборки данных данные помещаются вместе с их спецификацией – указанием типа данных и длины (таблица 5). Этот формат представления удобно использовать в тех случаях, когда приложение определяет структуру данных и генерирует запрос на их выборку в процессе выполнения.

Такой формат выборки данных называется *специфицированным*.

Специфицированный формат выборки данных удобен в том случае, когда запросы прикладной программы разнообразны (а, возможно, генерируются), и трудно отследить состав полей буфера выборки данных.

Таблица 3. Неспецифицированный формат типов данных в выборке данных

Тип данных в БД	Тип данных в C/C++	Возвращаемое в буфере выборки данных значение
CHAR(N)	`char data[N]`	Последовательность ASCII-символов, дополненная справа пробелами до ширины поля
VARCHAR(N)	`struct { L_WORD len; char data[N]; }`	Первые 2 байта – длина значения, далее последовательность ASCII-символов
BYTE(N)	`char data[N]`	Последовательность байтов, дополненная справа двоичными нулями до ширины поля
VARBYTE(N)	`struct { L_WORD`	Первые 2 байта – длина значения, далее – последовательность байтов

Тип данных в БД	Тип данных в C/C++	Возвращаемое в буфере выборки данных значение
	`len; char data[N]; }`	
NCHAR(N)	`L_WORD data[N]`	Последовательность шестнадцатеричных значений UNICODE-символов, дополненная справа пробелами до ширины поля
NCHAR VARYING(N)	`struct { L_WORD len; L_WORD data[N]; }`	Первые 2 байта – длина значения, далее последовательность шестнадцатеричных значений UNICODE-символов. Длина равна 2*N, где N – число символов UNICODE
DECIMAL, NUMERIC	Прямого соответствия нет	16 байт (см. файл decimals.h и документ «Библиотеки специальных типов данных», раздел «Библиотека Decimals»)
BIGINT	В 64-разрядных ОС long long/__int64	8-байтовое знаковое целое (от -9 223 372 036 854 775 808 до +9 223 372 036 854 775 807) (см. L_DLONG в файле int64.h)
INT	В 32-разрядных ОС long, в некоторых 64-разрядных ОС – int	4-байтовое знаковое целое (от -2 147 483 648 до +2 147 483 647) (см. L_LONG в файле lintypes.h)
SMALLINT	short	2-байтовое знаковое целое (от -32 768 до +32 767) (см. L_WORD в файле lintypes.h)
REAL	float	4-байтовое число с плавающей точкой
DOUBLE PRECISION	double	8-байтовое число с плавающей точкой
DATE	Прямого соответствия нет	16 байт, соответствует типу данных DECIMAL СУБД ЛИНТЕР (см. файл decimals.h и документ «Библиотеки специальных типов данных», раздел «Библиотека Decimals»)
BOOLEAN	`char[1]`	1 байт
BLOB	Прямого соответствия нет	Описатель BLOB-данных (24 байта) (структура BLOB_ATR)
EXTFILE	`struct { L_LONG filterid; L_BYTE indextime[6]; char filename[512]; }`	Описание внешнего файла (522 байта)

Таблица 4. Неспецифицированный формат псевдозначений в выборке данных

Тип данных в БД	Тип данных в СУБД ЛИНТЕР	Возвращаемое в буфере выборки данных значение
Псевдостолбцы		

Тип данных в БД	Тип данных в СУБД ЛИНТЕР	Возвращаемое в буфере выборки данных значение
USER	DT_CHAR	CHAR(MAX_ID_LEN)
SYSDATE	DT_DATE	DATE
LAST_ROWID	DT_INTEGER	INTEGER
ROWNUM	DT_INTEGER	INTEGER
ROWID	DT_INTEGER	INTEGER
ROWTIME	DT_DATE	DATE
LAST_AUTOINC	DT_INTEGER	INTEGER
LEVEL	DT_INTEGER	INTEGER
LINTER_NAME_LENGTH	DT_INTEGER	INTEGER
TRIGGER_INFO_SIZE	DT_INTEGER	INTEGER
AUD_OBJ_NAME_LEN	DT_INTEGER	INTEGER
PROC_INFO_SIZE	DT_INTEGER	INTEGER
PROC_PAR_NAME_LEN	DT_INTEGER	INTEGER
Литералы		
Строковый	DT_CHAR	CHAR
Байтовый	DT_BYTE	BYTE
UNICODE	DT_NCHAR	NCHAR
Вещественный	DT_REAL	DOUBLE
С фиксированной точкой	DT_REAL, DT_DECIMAL	DECIMAL (если помещается), иначе DOUBLE

Размер буфера для представления NULL-значения СУБД ЛИНТЕР равен длине непустого значения соответствующего типа, а содержимое этого буфера может быть любым. Для идентификации NULL-значений используется дополнительный массив – маска NULL-значений.

Таблица 5. Специфицированный формат выборки данных

Поле	Тип поля	Содержание
Number_Of_Field	L_WORD	Число полей в записи буфера выборки данных
Len_Field1	L_WORD	Длина 1-го поля в записи буфера выборки данных
Type_Field1	L_BYTE	Тип 1-го поля в записи буфера выборки данных
Precision_Field1	L_BYTE	Точность 1-го поля в записи буфера выборки данных
Scale_Field1	L_BYTE	Масштаб 1-го поля в записи буфера выборки данных
Reserv_Field1	L_BYTE	Резерв
CharSet1	L_WORD	Номер кодовой страницы

Поле	Тип поля	Содержание
Len_Field2	L_WORD	Длина 2-го поля в записи буфера выборки данных
Type_Field2	L_BYTE	Тип 2-го поля в записи буфера выборки данных
Precision_Field2	L_BYTE	Точность 2-го поля в записи буфера выборки данных
Scale_Field2	L_BYTE	Масштаб 2-го поля в записи буфера выборки данных
Reserv_Field2	L_BYTE	Резерв
CharSet2	L_WORD	Номер кодовой страницы
...	...	...
Len_FieldN	L_WORD	Длина N-го поля в записи буфера выборки данных
Type_FieldN	L_BYTE	Тип N-го поля в записи буфера выборки данных
Precision_FieldN	L_BYTE	Точность N-го поля в записи буфера выборки данных
Scale_FieldN	L_BYTE	Масштаб N-го поля в записи буфера выборки данных
Reserv_FieldN	L_BYTE	Резерв
CharSetN	L_WORD	Номер кодовой страницы
Field № 1	Тип 1-го поля в записи буфера выборки данных	Значение 1-го поля в записи буфера выборки данных
Field № 2	Тип 2-го поля в записи буфера выборки данных	Значение 2-го поля в записи буфера выборки данных
...	...	...
Field № N	Тип N-го поля в записи буфера выборки данных	Значение N-го поля в записи буфера выборки данных

Первый элемент буфера выборки данных содержит число полей буфера выборки данных (`Number_Of_Field`). Далее следуют описатели полей буфера выборки данных, в каждом из которых элемент `Type_Field` указывает тип данных поля, а элемент `Len_FieldN` – длину соответствующего поля. Вслед за описателями полей размещаются значения полей в том виде, в каком они представлены в БД. Тип данных полей выборки данных (значение полей Type_Field1,… Type_FieldN) указывается с помощью байтовых типов данных интерфейса нижнего уровня (приложение 4, inter.h).

Для получения записи выборки данных в специфицированном формате нужно присвоить флагу выполнения запроса (`PrzExe`) контрольного блока значение M_SPEC.

Специфицированную запись можно разобрать по отдельным полям последовательно или выборочно.

В приложении 5 приведен пример работы со специфицированным форматом.

Команды СУБД ЛИНТЕР

Команды СУБД ЛИНТЕР по функциональному назначению можно разделить на следующие группы:

1) коммуникационные команды;
2) команды работы со словарем СУБД ЛИНТЕР;
3) команды обработки данных, включающие:
   - команды для работы с поисковыми запросами;
   - команды работы с BLOB-данными;
   - команды обработки непоисковых запросов;
   - команды пакетной обработки данных;
4) команды управления доступом к данным;
5) команды управления транзакциями;
6) команды мониторинга БД.

В данном документе описание команд интерфейса нижнего уровня дается в следующей последовательности:

Название рубрики	Краткое описание
Назначение команды	Краткое функциональное описание команды.
Параметры вызова	Список параметров функции `inter` для описываемой команды. Обозначение параметров соответствует приведенным в разделе «Доступ к базе данных». При описании вызова некоторые параметры вызова будут обозначены словом NULL. Так обозначаются неиспользуемые при выполнении команды параметры, которые могут иметь произвольные значения, однако рекомендуемое значение все же NULL. Если в неиспользуемом параметре вместо NULL-значения указан реальный адрес, то при выполнении команды он будет проигнорирован. Необязательный параметр (как правило, это адрес пользовательской функции завершения асинхронной обработки запроса) представлен в квадратных скобках.
Входные параметры	Подробное описание входных параметров (задаваемых полей контрольного блока, структуры буферов оператора и др.).
Выходные параметры	Подробное описание выходных параметров (возвращаемых полей контрольного блока, структуры буфера выборки данных, буфера и др.).
Описание	Приводятся результаты выполнения команды, а также ограничения, накладываемые СУБД ЛИНТЕР на параметры и условия выполнения команды.
Коды завершения	Приводятся возможные коды завершения, возвращаемые интерфейсом нижнего уровня или СУБД ЛИНТЕР и относящиеся непосредственно к команде. Коды завершения, вырабатываемые при обработке SQL-запросов, выполняемых в данной команде, не описываются. Для подробной информации о них следует обратиться к документу «Справочник кодов завершения».

Название рубрики	Краткое описание
Пример формирования команды	В качестве примера, иллюстрирующего задание входных параметров и анализ результатов выполнения команды, приводится текст функции, реализующей выполнение данной команды из библиотеки, поставляемой в составе дистрибутива СУБД ЛИНТЕР (приложение 6). Примеры функций даны, как правило, для синхронного режима обработки запросов.
Пример использования команды	Приводится текст несложной C/C++ программы, иллюстрирующей использование описываемой команды. Представленная программа является, как правило, полностью законченной разработкой и может, при необходимости, быть скопирована и выполнена либо использована (с необходимыми изменениями) в разрабатываемых приложениях.

Полный список команд интерфейса нижнего уровня приведен в приложении 7.

Коммуникационные команды

Коммуникационные команды предназначены для установления и прекращения логической связи между клиентским приложением и СУБД ЛИНТЕР. Прежде чем начать информационный обмен с БД клиентское приложение должно открыть *канал связи* (соединение). При этом приложение должно указать имя ЛИНТЕР-сервера, с которым устанавливается соединение, а также регистрационные данные для проверки санкционированного доступа (имя пользователя и пароль, под которыми приложение будет зарегистрировано как пользователь БД).

Через соединение могут быть открыты подчиненные каналы, называемые *курсорами*.

Механизм соединений и курсоров используется СУБД ЛИНТЕР для управления транзакциями. Изменения данных в БД, прошедшие по некоторому курсору, могут фиксироваться (откатываться) независимо от работы по другим курсорам соединения, однако фиксация/откат по соединению вызовет фиксацию/откат по всем подчиненным курсорам.

Для открытия курсора приложению уже не нужно указывать регистрационные данные – курсоры наследуют эту информацию от соединения, которому они подчинены.

Открыть канал (соединение)

Назначение

Команда `OPEN` используется для установления логической связи между клиентским приложением и СУБД ЛИНТЕР.

Параметры вызова

```
inter(CBL, VarBuf, [OpBuf], [CondBuf], [RowBuf]);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;

- буфер параметров команды `VarBuf`;
- буфер SQL-запросов `OpBuf`;
- буфер для получения параметров БД `RowBuf`

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`Prior`	Приоритет канала
`Command`	`"OPEN"`
`Node`	Имя ЛИНТЕР-сервера
`PrzExe`	Режим обработки запросов
`LnBufRow`	Длина буфера `RowBuf` (если `RowBuf` задан)

Примечание

Если поле `Node` содержит пробелы, то предполагается, что устанавливается соединение с СУБД на **локальной машине** или с сервером по умолчанию. Поле `Node` имеет существенное значение, ибо номера открытых на разных серверах каналов могут совпадать. Это правило справедливо для всех команд интерфейса нижнего уровня.

Буфер параметров команды `VarBuf` должен содержать имя и пароль зарегистрированного в БД пользователя, под чьим именем клиентское приложение будет взаимодействовать с БД.

Буфер SQL-запросов `OpBuf` может содержать имя устанавливаемой для данного канала кодовой страницы, которая должна быть известна СУБД. Список доступных кодовых страниц находится в системной таблице `LINTER_SYSTEM_USER.$$$CHARSET`.

При этом:

1) при указании способа CP_AS_NUM (значение 0) соответствие кодовой страницы ищется в столбце WIN_CODE таблицы LINTER_SYSTEM_USER.$$$CHARSET;
2) при указании способа CP_AS_STR (значение 1) кодовая страница ищется среди
   - предопределенных кодовых страниц ("UTF-8","UCS2");
   - кодовых страниц в системной таблице LINTER_SYSTEM_USER.$$$CHARSET (столбец NAME);
   - синонимов кодовых страниц в системной таблице LINTER_SYSTEM_USER.$$$CSALIAS.
3) в случае если кодовая страница не найдена ни среди предопределенных значений, ни в таблицах LINTER_SYSTEM_USER.$$$CHARSET, LINTER_SYSTEM_USER.$$$CSALIAS, то ядром СУБД используется кодовая страница с именем DEFAULT (или встроенная ASCII 7 кодировка, если нет DEFAULT);
4) при ошибке в имени кодировки ядро СУБД ЛИНТЕР использует входные данные в кодировке с именем DEFAULT из таблицы $$$CHARSET (о чём сообщает флагами в структуре OPEN_DESC возвращаемой при открытии соединения).

Если параметр `OpBuf` содержит неверное имя кодировки, то будет использована кодовая страница по умолчанию для БД, заданная через переменную окружения `LINTER_CP`, а если содержит пустую строку или пропущен (NULL), то будет использована кодовая страница по умолчанию интерфейсом нижнего уровня.

Заданная в команде OPEN кодовая страница имеет более высокий приоритет по сравнению с переменной окружения LINTER_CP.

При указании имени пользователя, имени кодовой страницы и пароля необходимо соблюдать следующие правила:

- имя и пароль, разделенные символом «косая черта» (/), нужно указывать одной строкой;
- в имени/пароле могут быть использованы только алфавитно-цифровые символы;
- имя/пароль могут быть взяты в кавычки ("). В этом случае символы имени/пароля используются в том виде, в каком они представлены (в нижнем/верхнем регистре, в русской/латинской транскрипции);
- если кавычки отсутствуют, то все буквы имени/пароля будут приведены к верхнему регистру;
- если имя/пароль содержат кавычки, то все кавычки должны быть удвоены;
- символьная строка в VarBuf должна заканчиваться двоичным нулем.

Пример.

Пусть имя и пароль пользователя имеют следующие значения:

Имя пользователя	Пароль
SYSTEM	MANAGER8
Sys"	abD

Ниже приведено несколько иллюстраций употребления имени и пароля при обращении к СУБД:

Пример ввода имени и пароля	Восприятие СУБД ЛИНТЕР	Комментарий
System/Manager	SYSTEM +MANAGER8	Правильно
"system"/"manager"	system+manager	Ошибка доступа, т.к. нет перевода к верхнему регистру
"SYSTEM/ MANAGER8"	SYSTEM/ MANAGER8	Ошибка доступа, т.к. будет трактоваться как одно слово
"Sys"""/"abD"	Sys"+abD	Правильно
"Sys""/abD	Sys"/abD…	Ошибка доступа, т.к. будет трактоваться как одно слово без закрывающей кавычки
Sys"/abD	SYS"+ABD	Будет зафиксирована ошибка доступа, т.к. не закрыта кавычка

Примечание
Если не указано имя и/или пароль, то по умолчанию имя равно MAX_ID_LEN пробелам, пароль – 18 пробелам.

Выходные данные

Выходными данными является контрольный блок CBL и буфер RowBuf (если задан).

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
NumChan	Номер канала
SysErr	Код состояния ОС

Если задан аргумент RowBuf, то в буфере возвращается структура данных OPEN_DESC, содержащая описание параметров БД, с которой установлено соединение:

```
typedef struct {
    L_LONG VerMajor;
    L_LONG VerMinor;
    L_LONG VerBuild;
    L_BYTE Flags;
    L_BYTE Reserv;
    L_WORD MaxRecSize;
    L_CHAR BaseName[18];
    L_CHAR SysLog;
    L_CHAR Sync;
    L_CHAR Log;
    L_CHAR Os;
    L_WORD DefCharSet;
    L_WORD UseCharSet;
    L_WORD Reserved;
    L_CHAR UseCharSetName[MAX_ID_LEN];
} OPEN_DESC;
```

Описание полей структуры OPEN_DESC приведено в таблице 6.

Объем выдаваемой информации определяется длиной буфера RowBuf, которая устанавливается в поле LnBufRow контрольного блока CBL.

Если произошла ошибка при поиске запрашиваемой кодировки, и кодировка была изменена, то в поле Flags структуры (таблица 6) устанавливается флаг 0x01.

Таблица 6. Описание полей структуры OPEN_DESC

Поле структуры	Описание
VerMajor	Старший номер версии СУБД ЛИНТЕР, для которой построена БД
VerMinor	Младший номер версии СУБД ЛИНТЕР, для которой построена БД
VerBuild	Номер сборки версии СУБД ЛИНТЕР
Flags	Флаги БД (таблица 7)
DBFlags	Атрибуты БД
MaxRecSize	Предельная длина записи в таблице БД
BaseName[18]	Имя БД
SysLog	Признак активности системного журнала: 0 – журнал не ведется; 1 – журнал ведется.

Поле структуры	Описание
Sync	Признак синхронизации ввода/вывода (устанавливается при запуске СУБД ключом [NO]SYNC): 0 – синхронизация отменена; 1 – синхронизация установлена. **Примечание** Только для Windows-платформы.
Log	Признак ведения файла-протокола (устанавливается при запуске СУБД ключом [NO]LOG): 0 – протокол не ведется; 1 – протокол ведется.
Os	Идентификатор операционной системы (приложение 2)
DefCharSet	Идентификатор кодовой страницы по умолчанию для БД, заданный запросом set database default character set
UseCharSet	Идентификатор установленной кодовой страницы канала
Reserved	Резерв
UseCharSetName	Имя установленной кодовой страницы канала

Таблица 7. Флаги БД

Флаг	Значение	Описание
OPENDESC_CSET_CHANGED	0x01	Установленная кодовая страница не найдена. Используется кодовая страница по умолчанию.
OPENDESC_PASS_EXTERNAL	0x02	Пароль пользователю был задан администратором СУБД, и пользователь должен его сменить (при первом соединении с БД с помощью команды **ALTER USER IDENTIFIED BY**). До того, как произойдет смена пароля, доступ к БД будет запрещен.
OPENDESC_PASSLIFEEND	0x04	Флаг говорит о том, что «время жизни» у пароля истекло, и пользователь должен его сменить с помощью команды **ALTER USER IDENTIFIED BY**. До смены пароля все другие запросы будут заблокированы (доступ запрещен).

Описание

В контрольном блоке (поле NumChan) устанавливается номер открытого канала.

Для идентификации и аутентификации пользователя с использованием Kerberos-сервера необходимо:

- перед запуском клиентского приложения получить Kerberos-тикет автоматически при входе в ОС либо с использованием специального программного обеспечения (см. документы «Запуск и останов СУБД ЛИНТЕР в среде ОС Windows», приложение 1 «Пример настройки СУБД ЛИНТЕР для идентификации и аутентификации по Kerberos-протоколу» и «Запуск и останов СУБД ЛИНТЕР в среде ОС Linux,

Unix», приложение 1 «Пример настройки СУБД ЛИНТЕР для идентификации и аутентификации по Kerberos-протоколу»);

- использовать пустое имя пользователя или задать режим идентификации и аутентификации по Kerberos-протоколу установкой глобального флага ICR_FORCE_KRB (с помощью переменной окружения LINTER_KRB или функции интерфейса нижнего уровня inter_control) либо для всех соединений клиентского приложения, либо только для соединения, выполняемого по данной команде OPEN.

Пример идентификации и аутентификации по Kerberos-протоколу приведен в приложении 8.

Примечание
Асинхронное выполнение OPEN с идентификацией и аутентификацией по Kerberos-протоколу не поддерживается.

Коды завершения

Код	Описание
NOFREEKAN	Нет свободных каналов для организации соединения
ERRPASSWORD	Зарегистрированному пользователю в данный момент по каким-либо причинам запрещен доступ к БД (например, работа вне установленного времени расписания)
Invalid_User_Name	Указано имя незарегистрированного в БД пользователя
Invalid_User_Passwd	Указан неправильный пароль зарегистрированного пользователя

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterOPEN(TCBL * pCBL, L_CHAR * Name_Pass, L_CHAR * Node,
 L_WORD Prior, L_LONG PrzExe)
  {
  memcpy(pCBL->Command, "OPEN", 4);
  if (strlen(Node) > MAX_NODE_LEN)
    {
    return SQLLONGID;
    }
  memset(pCBL->Node, 0, MAX_NODE_LEN);
  memcpy(pCBL->Node, Node, strlen(Node));
  pCBL->PrzExe = PrzExe;
  pCBL->Prior = Prior;
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, Name_Pass, NULL, NULL, NULL);
  return pCBL->CodErr;
  }
```

Примеры использования команды

1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exopen()
#endif
  {
  TCBL    CBLconnect;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");

  printf("End Example\n");

  return 0;
  }
```

2) В приложении 9 приведен пример работы с двумя ядрами СУБД.

Создать подчиненный канал

Назначение

Команда `OCUR` предназначена для создания подчиненного канала между клиентским приложением и СУБД ЛИНТЕР.

Параметры вызова

```
inter(CBL, NULL, NULL, {CondBuf }, NULL);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
Prior	Приоритет подчиненного канала
NumChan	Номер главного канала
Command	"OCUR"
Node	Имя ЛИНТЕР-сервера
PrzExe	Режим обработки запросов

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
NumChan	Номер канала
SysErr	Код состояния ОС

Описание

В контрольном блоке (поле NumChan) устанавливается номер открытого подчиненного канала (курсора).

Команда OCUR должна использоваться только после успешного создания главного канала.

Приоритет подчиненного канала может быть выше главного канала.

Различные подчиненные каналы (курсоры) могут иметь отличающиеся друг от друга и от главного канала режимы обработки запросов.

Коды завершения

Код	Описание
NOFREEKAN	Нет свободных каналов для создания подчиненного канала (исчерпаны ресурсы СУБД ЛИНТЕР)

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterOCUR(TCBL * pCBL, L_WORD NumChan, L_WORD Prior,
 L_LONG PrzExe)
  {
  memset(pCBL, 0, sizeof(TCBL));
```



```
  memcpy(pCBL->Command, "OCUR", 4);
  pCBL->NumChan = NumChan;
  pCBL->PrzExe = PrzExe;
  pCBL->Prior = Prior;
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, NULL, NULL, NULL, NULL);
  return pCBL->CodErr;
  }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exocur()
#endif
  {
  TCBL    CBLconnect,
          CBL1;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");

  Err = LinterOCUR(&CBL1, CBLconnect.NumChan, Priority, PrzExe);
  if (Err != NORMAL)
    PrintError(&CBL1);
  printf("Opening Channel\n");

  printf("End Example\n");
```

```
return 0;
}
```

Установить параметры курсора

Назначение

По любому каналу (главному или подчиненному) можно выдать SELECT-запрос. Полученная в результате выполнения запроса выборка данных называется курсором. Если приложение открыло несколько каналов и выдало несколько SELECT-запросов, будет организовано соответствующее количество курсоров. Чтобы иметь возможность работать с курсором в любом открытом канале (а не только в том, где курсор был создан), ему следует присвоить имя. Если имя курсору не установлено, он считается неименованным. Имя курсора используется в SQL-запросах, работающих с текущим положением курсора (операторы **update/delete … current of [<курсор>] cursor**). Если в этих запросах имя курсора явно не задается, считается, что они относятся к курсору, созданному в канале, по которому подается на выполнение SQL-запрос.

Для создания или изменения имени курсора предназначена команда `SETO`.

Дополнительно команда `SETO` используется для административного контроля за исполнением SQL-запроса:

- изменение приоритета запроса;
- приостановка/продолжение выполнения запроса.

Параметры вызова

```
inter(CBL, NULL, NULL, NULL, [RowBuf]);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;
- буфер выборки данных `RowBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`NumChan`	Номер канала, по которому выполняется команда
`Prior`	Устанавливаемый приоритет
`Command`	`"SETO"`
`Node`	Имя ЛИНТЕР-сервера
`LnBufRow`	Длина имени курсора
`RowId`	Код операции
`RowCount`	Номер канала, для которого выполняется команда

Буфер `RowBuf` должен содержать символьную строку длиной не более MAX_ID_LEN символов, задающую имя курсора.

Коды операций задаются в поле `RowId` (символические имена описаны в `inter.h`):

Код	Значение
crName	Установить имя курсора
crPrior	Установить приоритет канала
crQueryPrior	Установить приоритет запроса
crCancel	Прервать выполнение запроса
crPause	Приостановить выполнение запроса
crContinue	Продолжить выполнение запроса
crMode	Изменить режим канала

В таблице 8 приведены значения полей блока CBL в зависимости от кода операции.

Таблица 8. Значения полей блока CBL

Код операции	Содержание полей блока CBL		
	Prior	NumChan	RowCount
crName		Номер канала, по которому выполняется команда SETO	
crPrior	Новый приоритет	--//--	Номер канала, для которого изменяется приоритет
crQueryPrior	Новый приоритет запроса	--//--	
crCancel		--//--	
crPause		--//--	Номер канала, где должен быть приостановлен текущий запрос
crContinue		--//--	Номер канала, где должен быть продолжен текущий запрос
crMode		--//--	

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
SysErr	Код состояния ОС

Описание

Присвоение имени курсору необязательно делать после выполнения SELECT-запроса, оно может быть установлено заранее для любого открытого канала. Если в канале было последовательно задано несколько SELECT-запросов и для данного канала было задано имя курсора, то каждая выборка (курсор) будет иметь одно и то же имя.

Для выполнения операций crCancel, crPause, crContinue по каналам других пользователей необходимы привилегии администратора СУБД.

Коды завершения

Код	Описание
DupCurName	Дубликат имени курсора
ERRFALSEOPER	Неверный код операции
ERRFALSEPRIOR	Неверное значение приоритета
ERRFALSECHSTATE	Неверное состояние канала. Выдается при попытке продолжить выполнение запроса по каналу, который не был приостановлен
QueryCanceled	Запрос прерван. Возвращается для канала, запрос которого был прерван

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterSETO(TCBL *pCBL, L_CHAR *NameCur)
   {
   pCBL->RowId = crName;
   memcpy(pCBL->Command, "SETO", 4);
   if ((pCBL->LnBufRow = strlen(NameCur)) > MAX_ID_LEN)

     {
     return SQLLONGID;
     }
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, NameCur);
   return pCBL->CodErr;
   }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exseto()
#endif
```



```
{
    TCBL    CBLconnect,
            CBL1;
    L_CHAR  Name_Pass[]="SYSTEM/MANAGER8";
    L_CHAR  Node[]="    ";
    L_CHAR  NameCur[]="Cursor1";
    L_WORD  Priority=0;
    L_LONG PrzExe=M_EXCLUSIVE | Q_ENCODE | M_BINARY;
    L_LONG Err;
memset(&CBLconnect,0,sizeof(TCBL));
Err=LinterOPEN(&CBLconnect, Name_Pass, Node, Priority, PrzExe);
    if (Err != NORMAL)
        PrintError(&CBLconnect);
    printf("Connect to RDBMS Linter\n");

Err=LinterOCUR(&CBL1, CBLconnect.NumChan, Priority, PrzExe);
    if (Err != NORMAL)
        PrintError(&CBL1);
    printf("Open Channel\n");

Err=LinterSETO(&CBL1, NameCur);
if (Err != NORMAL)
    PrintError(&CBL1);
    printf("Set Name for Channel\n");
    printf("End Example\n");
    return 0;
}
```

Закрыть канал

Назначение

Команда `CLOS` предназначена для освобождения главного или подчиненного канала связи (соединения или курсора) между приложением и ядром СУБД ЛИНТЕР.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], NULL);
```

Входные данные

Входными данными является контрольный блок `CBL`.

В нем должны быть заполнены поля:

Имя поля	Значение
`NumChan`	Номер закрываемого канала
`Command`	`"CLOS"`
`Node`	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
SysErr	Код состояния ОС

Описание

Команда CLOS делает канал неактивным и освобождает все связанные с ним ресурсы. Закрытый канал может быть использован другим приложением либо повторно тем же самым приложением.

Курсоры могут закрываться независимо друг от друга.

При закрытии главного канала СУБД ЛИНТЕР автоматически закрывает все подчиненные каналы (курсоры).

Право на закрытие канала имеет только то приложение, которое открыло этот канал.

Если в момент закрытия канала в нем имеется незавершенная транзакция, то она будет завершена командой **COMT** автоматически.

Коды завершения

Код	Описание
DupCurName	Дубликат имени курсора

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"


L_LONG LinterCLOS(TCBL * pCBL)
  {
  memcpy(pCBL->Command, "CLOS", 4);
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, NULL, NULL, NULL, NULL);
  return pCBL->CodErr;

  }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
```



```
#include <string.h>
#include "inter.h"
#include "exlib.h"

#ifndef WINCE
int    main()
#else
int    exclos()
#endif
  {
  TCBL    CBLconnect;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");

  /* ... */

  Err = LinterCLOS(&CBLconnect);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Close Channel\n");

  printf("End Example\n");

  return 0;
  }
```

Аварийное закрытие канала

Назначение

Команда `KILL` предназначена для аварийного закрытия канала связи между клиентским приложением и ядром СУБД ЛИНТЕР и освобождения всех связанных с ним ресурсов. Она может использоваться в канальном и неканальном варианте.

Команда `KILL` может быть использована для того, чтобы снять с исполнения запрос, обработка которого затянулась на неоправданно длительное время, или его выполнение мешает прохождению более насущных запросов и т.п. При этом приложению, ожидающему выполнение этого запроса, вернется соответствующий код завершения,

так что запрос придется либо снова повторить, либо вообще отказаться от его выполнения.

Параметры вызова

- канальный вариант: `inter(CBL, NULL, [OpBuf], [CondBuf], NULL);`
- неканальный вариант: `inter(CBL, VarBuf, [OpBuf], [CondBuf], NULL).`

Входные данные

Канальный вариант команды

Входными данными являются:

- контрольный блок `CBL`;
- буфер SQL-запросов `OpBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`NumChan`	Номер канала, по которому подается команда `KILL`
`Command`	`"KILL"`
`Node`	Имя ЛИНТЕР-сервера
`RowId`	Номер аварийно закрываемого канала

Неканальный вариант команды

Входными данными являются:

- контрольный блок `CBL`;
- буфер параметров `VarBuf`;
- буфер SQL-запросов `OpBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`Command`	`"KILL"`
`Node`	Имя ЛИНТЕР-сервера
`RowId`	Номер аварийно закрываемого канала

Буфер параметров команды `VarBuf` должен содержать имя и пароль пользователя с привилегиями администратора БД. Способы задания имени и пароля и механизм идентификации и аутентификации по Kerberos-протоколу описаны в команде OPEN.

Буфер SQL-запросов OpBuf может содержать имя устанавливаемой для данного канала кодовой страницы, которая должна быть известна СУБД (см. описание алгоритма выборка кодовой страницы в команде OPEN).

Выходные данные

Выходными данными для обоих вариантов команды является контрольный блок `CBL`.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
SysErr	Код состояния ОС

Описание

Для использования канального варианта команды KILL приложение должно иметь открытый канал связи с СУБД ЛИНТЕР.

В канальном варианте команды номер аварийно закрываемого канала и номер канала, по которому подается команда KILL, не должны совпадать (то есть при использовании канального варианта приложение сможет аварийно закрыть все свои каналы, кроме какого-нибудь одного).

Чтобы приложение имело возможность аварийно закрыть канал (или несколько каналов), оно должно открыть для этой цели дополнительный канал; все остальные в общем случае должны работать в асинхронном режиме.

Если в канальном варианте команда KILL на аварийное закрытие главного канала подается по одному из его подчиненных каналов, то аварийно закрываются и главный канал, и подчиненные.

Неканальный вариант команды может использовать любое приложение, зарегистрированное в БД с правами администратора БД. Для получения информации об открытых в ядре СУБД каналах необходимо запросить интересующую информацию из виртуальной системной таблицы $$$CHAN с помощью запроса:

```
select * from $$$CHAN where STATUS <> '';
```

Подчиненные каналы (курсоры) могут аварийно закрываться независимо друг от друга.

При аварийном закрытии главного канала автоматически закрываются и все подчиненные ему каналы (курсоры).

Все незавершенные транзакции (в главном и подчиненных каналах) при аварийном закрытии канала откатываются (**ROLLBACK**).

При неканальном использовании команды буфер параметров должен содержать имя и пароль пользователя аналогично команде OPEN.

Примечание
Асинхронное выполнение неканального варианта KILL с идентификацией и аутентификацией по Kerberos-протоколу не поддерживается.

Коды завершения

Код	Описание
ERRPASSWORD	Аварийное закрытие «чужого» канала
Invalid_User_Name	Указано имя незарегистрированного в БД пользователя

Код	Описание
Invalid_User_Passwd	Указан неправильный пароль зарегистрированного пользователя

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterKILL(TCBL * pCBL, L_WORD NumChan)
  {
  memcpy(pCBL->Command, "KILL", 4);
  pCBL->RowId = NumChan;
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, NULL, NULL, NULL, NULL);
  return pCBL->CodErr;
  }
```

Пример использования канального варианта команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#ifndef WINCE
int    main()
#else
int    exkill()
#endif
  {
  TCBL    CBLconnect,
          CBL1;
  L_CHAR  Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR  Node[] = "        ";
  L_WORD  Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");
```



```
  Err = LinterOCUR(&CBL1, CBLconnect.NumChan, Priority, PrzExe);
  if (Err != NORMAL)
    PrintError(&CBL1);
  printf("Open Channel\n");

  Err = LinterKILL(&CBLconnect, CBL1.NumChan);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Kill Channel\n");
  printf("End Example\n");
  return 0;
  }
```

Пример использования неканального варианта команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#ifndef WINCE
int main()
#else
int exkill2()
#endif
{
  TCBL CBLconnect, cblKiller;
  L_CHAR Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR Node[] = " ";
  L_WORD Priority = 0;
  L_LONG PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG Err;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");

  memset(&cblKiller, 0, sizeof(cblKiller));
  memcpy(cblKiller.Command, "KILL", 4);
  cblKiller.RowId = CBLconnect.NumChan;
  inter(&cblKiller, Name_Pass, NULL, NULL, NULL);
  if (cblKiller.CodErr != NORMAL)
    PrintError(&cblKiller);
```

```
  printf("Kill Channel\n");
  Err = LinterCLOS(&CBLconnect);
  if (Err != 1069) /* here should be 'wrong channel number' error
 */
    PrintError(&CBLconnect);
  printf("End Example\n");
  return 0;
}
```

Сбросить изменения на диск

Назначение

Изменения, производимые клиентским приложением, не сразу переносятся в файлы БД, т.к. СУБД ЛИНТЕР выполняет буферизацию ввода/вывода с целью оптимизации обработки запросов. Команда `SNAP` заставляет ядро СУБД ЛИНТЕР произвести запись накопленных изменений, не дожидаясь заполнения буфера вывода.

Эта команда позволяет регулировать загрузку буферов ввода/вывода, снижать их заполнение обновленными страницами и элементами очередей.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], NULL);
```

Входные данные

Входными данными является контрольный блок `CBL`.

В нем должны быть заполнены поля:

Имя поля	Значение
`Command`	`"SNAP"`
`Node`	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными является контрольный блок `CBL`.

В нем будут возвращены:

Имя поля	Значение
`CodErr`	Код завершения запроса к СУБД ЛИНТЕР
`SysErr`	Код состояния ОС

Описание

Команду `SNAP` можно подавать в любом месте приложения.

Указание номера открытого канала не требуется (это неканальная команда). Номер канала в контрольном блоке можно не заполнять.

Идентификации пользователя не требуется, эту команду может подать любой пользователь.

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"


L_LONG LinterSNAP(L_CHAR * Node)
  {
  TCBL    CBL;

  memset(&CBL, 0, sizeof(TCBL));
  memcpy(CBL.Command, "SNAP", 4);
  if (strlen(Node) > MAX_NODE_LEN)
    {
    return SQLLONGID;
    }
  memset(CBL.Node, 0, MAX_NODE_LEN);
  memcpy(CBL.Node, Node, strlen(Node));
  CBL.PrzExe &= ~Q_ASYNC;
  inter(&CBL, NULL, NULL, NULL, NULL);
  return CBL.CodErr;
  }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exsnap()
#endif
  {
  TCBL    CBL;
  L_CHAR    Node[] = "        ";
  L_LONG  Err;

  /* ... */

  memset(&CBL,0,sizeof(TCBL));
  Err = LinterSNAP(Node);
  if (Err != NORMAL)
```

```
    PrintError(&CBL);
  printf("SNAP\n");

  /* ... */

  printf("End Example\n");

  return 0;
  }
```

Завершить работу СУБД

Назначение

Команда `SHUT` предназначена для завершения работы СУБД ЛИНТЕР. Команда может выполняться в канальном и неканальном режимах.

Параметры вызова

- канальный вариант: `inter(CBL, NULL, [OpBuf], [CondBuf], NULL);`
- неканальный вариант: `inter(CBL, VarBuf, [OpBuf], [CondBuf], NULL).`

Входные данные

Канальный вариант команды

Входными данными являются:

- контрольный блок `CBL`;
- буфер SQL-запросов `OpBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`Command`	`"SHUT"`
`Node`	Имя ЛИНТЕР-сервера
`NumChan`	Номер канала, по которому должна быть выполнена команда SHUT
`RowId`	Указание на способ обработки незавершенных транзакций: -1 – выполнить откат (rollback)

Неканальный вариант команды

Входными данными являются:

- контрольный блок `CBL`;
- буфер параметров `VarBuf`;
- буфер SQL-запросов `OpBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
Command	"SHUT"
Node	Имя ЛИНТЕР-сервера
RowId	Указание на способ обработки незавершенных транзакций: -1 – выполнить откат (rollback)

Буфер параметров команды VarBuf должен содержать имя и пароль пользователя с привилегиями администратора БД. Способы задания имени и пароля и механизм идентификации и аутентификации по Kerberos-протоколу описаны в команде OPEN.

Буфер SQL-запросов OpBuf может содержать имя устанавливаемой для данного канала кодовой страницы, которая должна быть известна СУБД (см. описание алгоритма выборка кодовой страницы в команде OPEN).

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
SysErr	Код состояния ОС

Описание

Работа СУБД ЛИНТЕР завершается при условии, что в момент подачи команды SHUT в ядре нет открытых каналов.

Код завершения команды SHUT возвращается только после закрытия всех файлов БД и СУБД.

Примечание
Асинхронное выполнение неканального варианта SHUT с идентификацией и аутентификацией по Kerberos-протоколу не поддерживается.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
NOPRIVSHUT	В СУБД есть активные пользователи
ERRPASSWORD	Отсутствие прав (привилегий): • зарегистрированному пользователю в данный момент по каким-либо причинам запрещен доступ к БД (например, работа вне установленного времени расписания); • недостаточно привилегий для выполнения команды (необходимы привилегии администратора БД)
Invalid_User_Name	Указано имя незарегистрированного в БД пользователя
Invalid_User_Passwd	Указан неправильный пароль зарегистрированного пользователя

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterSHUT(TCBL * pCBL, L_CHAR * Name_Pass, L_CHAR * Node)
  {
  memcpy(pCBL->Command, "SHUT", 4);
  if (strlen(Node) > MAX_NODE_LEN)
    {
    return SQLLONGID;
    }
  memset(pCBL->Node, 0, MAX_NODE_LEN);
  memcpy(pCBL->Node, Node, strlen(Node));
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, Name_Pass, NULL, NULL, NULL);
  return pCBL->CodErr;
  }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exshut()
#endif
  {
  TCBL    CBLconnect;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");
```

```
#if _VER_MAX >= 500
  Err = LinterSHUT(&CBLconnect, Name_Pass, Node);
#else
  Err = LinterSHUT(&CBLconnect, Node);
#endif
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("SHUT RDBMS Linter\n");

  printf("End Example\n");

  return 0;
  }
```

Команды работы со словарем системы

Эта группа команд предназначена для получения справочной информации об объектах БД и для изменения структуры БД. В группу входят следующие команды:

- FREL – дать описание таблицы;
- FATR – дать описание столбца таблицы;
- FUSR – дать идентификатор пользователя;
- DESC – дать описание БД;
- запрос на изменение структур БД (специальное имя команды отсутствует).

Примечание
При выполнении этих команд значение поля PrzExe в контрольном блоке не играет роли, выборка данных всегда выдается в не специфицированном формате.

Дать описание таблицы

Назначение

Команда FREL предназначена для получения системной информации о хранящемся в БД объекте (таблице, представлении, синониме).

Справочная информация содержит следующие сведения:

- системный номер объекта;
- идентификатор владельца;
- имя объекта;
- идентификатор узла;
- маска доступа к записям объекта для пользователей категории **PUBLIC**;
- тип объекта (таблица, представление, синоним);
- количество атрибутов в объекте;

- количество ключей в объекте;
- процент заполнения страницы таблицы;
- количество ключей в составном ключе;
- число внешних (ссылочных) ключей в объекте;
- уровень доступа на чтение;
- уровень доступа на запись;
- дата создания объекта;
- время жизни объекта в БД;
- номер страницы целостности;
- идентификатор атрибута первичного ключа;
- количество ссылок на таблицу;
- описание аудита;
- признак наличия INSERT-триггера;
- признак наличия DELETE-триггера;
- признак наличия UPDATE-триггера;
- максимальный номер RowId;
- число занятых RowId;
- текущее количество записей в объекте;
- длина записи в неупакованном виде;
- количество экстентов области индексов;
- количество экстентов области данных;
- количество экстентов области BLOB-данных;
- размер страницы файла индексов в блоках;
- размер страницы файла области данных в блоках;
- размер страницы файла области BLOB-данных в блоках;
- номер первой страницы конвертера;
- номер BLOB-столбца в схеме таблицы;
- процент заполнения BLOB-страницы;
- описание экстента области индексов;
- описание экстента области данных;
- описание экстента области BLOB-данных.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;

- буфер описания параметров VarBuf.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"FREL"
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера
PrzExe	Режим обработки запроса

Буфер описания параметров команды VarBuf должен содержать информацию об интересующей таблице в следующем виде:

```
struct FREL_IN
  {
  L_LONG        Owner;
  L_CHAR        TblName[MAX_ID_LEN];
  };
```

Примечания

1. Значение «идентификатор владельца» можно также получить при помощи команды FUSR интерфейса нижнего уровня или выбрать из системной таблицы $$$USR (при наличии права на SELECT-запросы к этой таблице).
2. Если длина имени объекта меньше MAX_ID_LEN символов, то поле TblName должно быть дополнено справа пробелами до MAX_ID_LEN символов.

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
LnBufRow	Фактическая длина буфера выборки данных
RowId	Системный номер записи таблицы $$$SYSRL, в которой содержится описание объекта
SysErr	Код состояния ОС

Информация в буфере выборки данных RowBuf будет представлена в следующем виде:

```
struct FREL_OUT
  {
  L_LONG        Table;
```

```
  L_LONG        Owner;
  L_CHAR        TblName[MAX_ID_LEN];
#if _VER_MAX >= 600
  L_BYTE        Desc[288];
#elif _VER_MAX >= 570
  L_BYTE        Desc[284];
#elif _VER_MAX >= 500
  L_BYTE        Desc[244];
#else
  L_BYTE        Desc[162];
#endif
  };
```

Примечания

1. Длина поля `Desc` в буфере выборки данных зависит от используемой версии СУБД ЛИНТЕР. Значение макросу _VER_MAX присваивается в командной строке транслятора C/C++ (раздел «Условия применения»).
2. Поле `Desc` является копией столбца $$$S14 из системной таблицы `$$$SYSRL`.
3. Структура столбца $$$S14 описана в документе «Системные таблицы и представления», подраздел «$$$SYSRL».

Описание

При выполнении команды СУБД сначала просматривает очередь таблиц, размещенную в оперативной памяти ядра, в поисках описания указанного объекта. Если описание объекта в очереди не найдено, поиск продолжается в системной таблице `$$$SYSRL`. Из этого алгоритма следует, что в зависимости от параметров запуска ядра СУБД и способа получения описания объекта результаты, в общем случае, могут быть различными.

Если ядро СУБД запущено с ключом `SYNC`, то результаты выполнения команды `FREL` и SELECT-запроса из таблицы `$$$SYSRL` будут совпадать.

Если ядро СУБД запущено с ключом `NOSYNC`, то результаты выполнения команды `FREL` и SELECT-запроса из таблицы `$$$SYSRL` могут отличаться между собой, а именно, полученная с помощью команды `FREL` информация является «самой новой» (в отличие от полученной SELECT-запросом из таблицы `$$$SYSRL`). То есть, если объект был только что создан, информация о нем записана в очереди таблиц, расположенной в оперативной памяти ядра, и еще не сброшена в файлы БД, то по команде `FREL` описание объекта будет получено, а по SELECT-запросу из таблицы `$$$SYSRL` – нет. Если в таком режиме запуска ядра для получения системной информации об объекте предполагается использовать вместо команды `FREL` SELECT-запрос, то для исключения рассогласования данных перед его подачей надо выполнить команду `SNAP`.

Коды завершения

Код	Описание
`NORMAL`	Нормальное завершение
`SMALLBUFKOR`	Недостаточный размер буфера выборки данных
`EORR`	Указанный объект не найден

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterFREL(TCBL * pCBL, FREL_IN * pIn, FREL_OUT * pOut)
  {
  memcpy(pCBL->Command, "FREL", 4);
  pCBL->LnBufRow = sizeof(FREL_OUT);
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, pIn, NULL, NULL, pOut);
  return pCBL->CodErr;
  }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exfrel()
#endif
  {
  TCBL    CBLconnect;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;
  L_LONG  ID;
  L_CHAR    User[] = "SYSTEM";
  L_CHAR    Table[] = "PERSON";
  FREL_IN In;
  FREL_OUT Out;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");
```

```
Err = LinterFUSR(&CBLconnect, User, &ID);
if (Err != NORMAL)
  PrintError(&CBLconnect);

In.Owner = ID;
memset(In.TblName, ' ', MAX_ID_LEN);
strncpy(In.TblName, Table, strlen(Table));

Err = LinterFREL(&CBLconnect, &In, &Out);
if (Err != NORMAL)
  PrintError(&CBLconnect);
printf("FREL:\n");
printf("\tOut.Table=%ld\n\tOwner=%ld\n", Out.Table, Out.Owner);

printf("End Example\n");

return 0;
}
```

Дать описание столбца таблицы

Назначение

Команда FATR используется для получения системной информации о столбце таблицы.

Справочная информация содержит следующие сведения:

- тип данных столбца;
- точность;
- масштаб;
- длина столбца в байтах;
- признак ключа;
- номер файла области индексов для индекса по столбцу;
- признак ссылающегося столбца;
- номер байта, с которого начинается значение столбца в неупакованной записи;
- наличие операции удаления по ссылочной целостности;
- наличие операции обновления по ссылочной целостности;
- порядковый номер столбца в составном ключе;
- группа доступа;
- уровни доступа;
- начальное значение для автоинкремента;
- значение по умолчанию;
- количество ссылок;

- идентификатор внешней таблицы;
- флаги столбца;
- номер первой страницы индекса верхнего уровня;
- номер последней страницы индекса верхнего уровня.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;
- буфер описания параметров `VarBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`NumChan`	Номер канала
`Command`	`"FATR"`
`LnBufRow`	Длина буфера выборки данных
`Node`	Имя ЛИНТЕР-сервера
`PrzExe`	Режим обработки запроса

Буфер описания параметров `VarBuf` должен содержать информацию об интересующем столбце в следующем виде:

```
struct FATR_IN
  {
  L_LONG        Table;
  L_CHAR        ColName[MAX_ID_LEN];
      };
```

Примечания

1. Значение `Table` можно получить при помощи команды FREL интерфейса нижнего уровня или выбрать из системной таблицы `$$$SYSRL` (при наличии права на SELECT-запросы к этой таблице).
2. Если длина имени столбца меньше MAX_ID_LEN символов, то поле `ColName` должно быть дополнено справа пробелами до MAX_ID_LEN символов.

Выходные данные

Выходными данными являются:

- контрольный блок `CBL`;
- буфер выборки данных `RowBuf`.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Информация в буфере выборки данных RowBuf будет представлена в следующем виде:

```
struct FATR_OUT
  {
  L_LONG        Table;
  L_WORD        Column;
  L_CHAR        ColName[MAX_ID_LEN];
#if _VER_MAX >= 570
  L_BYTE        Desc[104];
#elif _VER_MAX >= 500
  L_BYTE        Desc[96];
#else
  L_BYTE        Desc[64];
#endif
  };
```

Примечания

1. Длина поля Desc в буфере выборки данных зависит от версии СУБД ЛИНТЕР. Значение макросу _VER_MAX присваивается в командной строке транслятора C/C++ (раздел «Условия применения»).
2. Поле Desc является копией столбца $$$S24 из системной таблицы $$$ATTRI.
3. Структура столбца $$$S24 описана в документе «Системные таблицы и представления», подраздел «$$$ATTRI».

Описание

По этой команде сначала просматривается очередь столбцов в оперативной памяти ядра СУБД для поиска описания указанного столбца. Если нужного столбца не найдено, поиск продолжается в системной таблице $$$ATTRI. В остальном данная команда подобна команде FREL.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Указанный столбец не найден

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"
```



```
L_LONG LinterFATR(TCBL * pCBL, FATR_IN * pIn, FATR_OUT * pOut)
  {
  memcpy(pCBL->Command, "FATR", 4);
  pCBL->LnBufRow = sizeof(FATR_OUT);
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, pIn, NULL, NULL, pOut);
  return pCBL->CodErr;
  }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exfatr()
#endif
  {
  TCBL    CBLconnect;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;
  L_LONG  ID;
  L_CHAR    User[] = "SYSTEM";
  L_CHAR    Table[] = "PERSON";
  L_CHAR    ColName[] = "NAME";
  FREL_IN FREL_In;
  FREL_OUT FREL_Out;
  FATR_IN FATR_In;
  FATR_OUT FATR_Out;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");
```

```
  Err = LinterFUSR(&CBLconnect, User, &ID);
  if (Err != NORMAL)
    PrintError(&CBLconnect);

  FREL_In.Owner = ID;
  memset(FREL_In.TblName, ' ', MAX_ID_LEN);
  strncpy(FREL_In.TblName, Table, strlen(Table));

  Err = LinterFREL(&CBLconnect, &FREL_In, &FREL_Out);
  if (Err != NORMAL)
    PrintError(&CBLconnect);

  FATR_In.Table = FREL_Out.Table;
  memset(FATR_In.ColName, ' ', MAX_ID_LEN);
  strncpy(FATR_In.ColName, ColName, strlen(ColName));

  Err = LinterFATR(&CBLconnect, &FATR_In, &FATR_Out);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("FATR:\n");
  printf("\tTable=%ld\n\tColumn=%ld\n\tColName=%.18s\n",
         FATR_Out.Table,
         FATR_Out.Column,
         FATR_Out.ColName);

  printf("End Example\n");

  return 0;
  }
```

Дать идентификатор пользователя

Назначение

Команда `FUSR` предназначена для получения идентификатора указанного пользователя.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;
- буфер описания параметров `VarBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"FUSR"
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Буфер описания параметров VarBuf должен содержать имя интересующего пользователя. В отличие от употребления имени пользователя при его идентификации (например, в команде OPEN), в данной команде оно не проходит синтаксического разбора, его символы воспринимаются в том виде, в котором заданы, без изменения регистра. Поэтому в конкретном случае любые двойные кавычки (если они используются в имени пользователя) представляют самих себя и не являются элементом синтаксической конструкции. Например, приведенные ниже имена пользователей (как они представлены в VarBuf) являются различными именами:

Иванов, ИВАНОВ, "ИВАНОВ", Smit, SMIT, "Smit"

Примечание
Если длина имени пользователя меньше MAX_ID_LEN символов, то имя пользователя должно быть дополнено справа пробелами до MAX_ID_LEN символов.

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Информация в буфере выборки данных RowBuf будет представлена в следующем виде:

```
L_LONG User_Id; /* Идентификатор пользователя */
```

Описание

По этой команде СУБД сначала просматривает очередь пользователей в оперативной памяти ядра для поиска указанного имени. Если нужного пользователя не найдено, поиск продолжается в таблице пользователей $$$USR.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Указанный пользователь не найден

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"


L_LONG LinterFUSR(TCBL * pCBL, L_CHAR * User, L_LONG * pUser_ID)
  {
  memcpy(pCBL->Command, "FUSR", 4);
  pCBL->LnBufRow = sizeof(L_LONG);
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, User, NULL, NULL, pUser_ID);
  return pCBL->CodErr;
  }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exfusr()
#endif
  {
  TCBL    CBLconnect;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;
  L_LONG  ID;
  L_CHAR    User[] = "SYSTEM";

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");

  Err = LinterFUSR(&CBLconnect, User, &ID);
```

```
if (Err != NORMAL)
  PrintError(&CBLconnect);
printf("FUSR:\n");
printf("User=%s,ID=%ld\n", User, ID);

printf("End Example\n");

return 0;
}
```

Дать идентификатор узла сети

Назначение

Команда FNOD предназначена для получения идентификатора заданного узла сети.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок CBL;
- буфер описания параметров VarBuf.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"FNOD"
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Буфер описания параметров VarBuf должен быть строкой длиной не более 8 символов и содержать имя интересующего узла сети.

Примечание
Если строка имени узла в VarBuf меньше 8 символов, она должна дополняться пробелами до 8 символов либо заканчиваться двоичным нулем.

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Информация в буфере выборки данных RowBuf будет представлена в следующем виде:

```
L_LONG Node_Id; /* Идентификатор узла */
```

Описание

Для выполнения команды используется системная таблица SERVERS.

Примечание
Вместо команды FNOD можно воспользоваться SELECT-запросом из системной таблицы SERVERS (при наличии прав на чтение данных из нее).

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Указанный узел не найден

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterFNOD(TCBL * pCBL, L_CHAR * Node, L_LONG * Out)
    {
memcpy(pCBL->Command, "FNOD", 4);
pCBL->LnBufRow = sizeof(L_LONG);
pCBL->PrzExe &= ~Q_ASYNC;
inter(pCBL, Node, NULL, NULL, Out);
return pCBL->CodErr;
}
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
```



```
int    main()
#else
int    exfnod()
#endif
  {
  TCBL    CBLconnect;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_CHAR    NodeFind[] = "Linter";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;
  L_LONG  Out;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");

  Err = LinterFNOD(&CBLconnect, NodeFind, &Out);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("FNOD, Out=%ld\n", Out);

  printf("End Example\n");

  return 0;
  }
```

Дать параметры курсора

Назначение

Команда `FCUR` предназначена для получения информации о заданном курсоре.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;
- буфер описания параметров `VarBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"FCUR"
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Буфер описания параметров VarBuf должен содержать имя курсора (не больше MAX_ID_LEN символов).

Примечание
Строка имени курсора в VarBuf должна заканчиваться двоичным нулем.

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Информация в буфере выборки данных RowBuf будет представлена в следующем виде:

```
struct FCUR_OUT
  {
  L_WORD       Channel;
  L_BYTE       Updatable;
  L_BYTE       Process;
  L_LONG       AnsCnt;
  };
```

В полях буфера выборки данных будет содержаться:

Имя поля	Значение
Channel	Номер канала, соответствующий заданному курсору
Updatable	Признак обновляемости курсора: 1 – курсор обновляемый, 0 – не обновляемый
Process	Идентификатор текущего процесса, выполняемого в канале, соответствующем курсору (см. приложение 11)
AnsCnt	Количество записей в заданном курсоре, если текущим процессом является выполнение SELECT-запроса, в противном случае значение не определено

Описание

Команда FCUR канальная. Она может быть подана только по главному родительскому каналу или по любому подканалу, подчиненному этому главному.

Если в качестве имени курсора задана пустая строка (нулевой длины или заполненная пробелами), то FCUR возвращает описание того канала, по которому она подана.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Указанное имя курсора не найдено или команда была подана по недопустимому каналу
ERRTRANSLSTR	Ошибка перекодирования имени курсора

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"
L_LONG LinterFCUR(TCBL * pCBL, L_CHAR * NameCur, FCUR_OUT * pOut)
  {
  memcpy(pCBL->Command, "FCUR", 4);
  pCBL->LnBufRow = sizeof(FCUR_OUT);
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, NameCur, NULL, NULL, pOut);
  return pCBL->CodErr;
  }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exfcur()
#endif
  {
  TCBL    CBLconnect,
          CBL1;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
```

```
  L_CHAR    NameCur[] = "Cursor1";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;
  FCUR_OUT FCUR_Out;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");

  Err = LinterOCUR(&CBL1, CBLconnect.NumChan, Priority, PrzExe);
  if (Err != NORMAL)
    PrintError(&CBL1);
  printf("Open Channel\n");

  Err = LinterSETO(&CBL1, NameCur);
  if (Err != NORMAL)
    PrintError(&CBL1);
  printf("Set Name for Channel\n");

  Err = LinterFCUR(&CBLconnect, NameCur, &FCUR_Out);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("FCUR:\n");
  printf("\tChannel=%d\n\tUpdatable=%hd\n\tProcess=%hd\n\tAnsCnt=
%ld\n",
          FCUR_Out.Channel,
          FCUR_Out.Updatable,
          FCUR_Out.Process,
          FCUR_Out.AnsCnt);

  printf("End Example\n");

  return 0;
  }
```

Дать описание БД

Назначение

Команда `DESC` предназначена для получения системных параметров БД.

Параметры вызова

```
inter(CBL, NULL, [OpBuf], [CondBuf], RowBuf);
```


Входные данные

Входными данными являются:

- контрольный блок CBL;
- буфер SQL-запросов OpBuf.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
Command	"DESC"
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Буфер SQL-запросов OpBuf может содержать имя устанавливаемой для данного канала кодовой страницы, которая должна быть известна СУБД (см. описание алгоритма выборка кодовой страницы в команде OPEN).

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Информация в буфере выборки данных RowBuf будет представлена в следующем виде:

```
struct DB_DESC
  {
  L_LONG VerMajor;         /* Старший номер версии ЛИНТЕР, */
                           /* для которой построена БД */
  L_LONG VerMinor;         /* Младший номер версии ЛИНТЕР, */
                           /* для которой построена БД */
  L_LONG VerBuild;         /* Номер сборки версии ЛИНТЕР */
  L_LONG SortPoolSize;     /* Размер пула (в страницах) */
                           /* подсистемы сортировки */
  L_LONG KernelPoolSize;   /* Размер пула (в страницах) */
                           /* ядра системы */
  L_LONG FileQueueSize;    /* Размер очереди файлов */
  L_LONG UserQueueSize;    /* Размер очереди пользователей */
  L_LONG TableQueueSize;   /* Размер очереди таблиц */
  L_LONG ColumnQueueSize;  /* Размер очереди столбцов */
  L_LONG ChannelQueueSize;/* Размер очереди каналов */
```

```
  L_LONG SnapTimeout;      /* Период времени между операциями */
                           /* полного Snap */
  L_LONG KillTimeout;      /* Тайм-аут опроса существования клиента
*/
  L_WORD NumOfSort;        /* Количество файлов сортировки */
  L_BYTE Flags;            /* Атрибуты БД (таблица 9) */
  L_BYTE Flags2;
  L_LONG LReserv2;         /* Зарезервировано */
  L_WORD SQLUsrCacheSize; /* Размер кэша пользователей SQL*/
  L_WORD SQLTabCacheSize; /* Размер кэша таблиц SQL*/
  L_WORD SQLColCacheSize; /* Размер кэша столбцов SQL*/
  L_WORD SQLPrcCacheSize; /* Размер кэша хранимых процедур SQL*/
  L_WORD SQLChsCacheSize; /* Размер кэша кодировок SQL*/
  L_WORD MaxRecSize;       /* Предельная длина записи в таблице БД
*/
  L_CHAR BaseName[18];     /* Имя БД */
  L_CHAR SysLog;           /* Признак активности журнала транзакций
*/
  L_CHAR Sync;             /* Признак синхронизации ввода/вывода */
  L_CHAR Log;              /* Признак ведения файла-протокола */
  L_CHAR Os;               /* Идентификатор операционной системы */
  L_WORD CharSet;          /* Идентификатор кодовой страницы */
                           /* данной БД */
  };
```

Примечание
Идентификаторы (коды) операционных систем приведены в приложении 1.

Таблица 9. Атрибуты БД

Обозначение	Значение атрибута	Описание
KERNEL_INVBYTEORD	0x01	Порядок байт клиента и сервера не совпадают (будет выполняться автоматическое преобразование типов данных)
KERNEL_DEMOLIC	0x02	СУБД ЛИНТЕР работает на условиях демонстрационной лицензии
KERNEL_DEMOLICEXP	0x04	Срок лицензии истек. СУБД будет работать еще 14 дней после завершения срока лицензии
KERNEL_STANDARD_MODE	0x10	СУБД ЛИНТЕР работает в стандартном режиме
KERNEL_GEOPREFIX_MODE	0x20	СУБД поддерживает геометрический тип данных

Обозначение	Значение атрибута	Описание
KERNEL_READONLY_MODE	0x40	СУБД ЛИНТЕР работает в режиме «только чтение»
KERNEL_QUANT_TIME	0x80	СУБД ЛИНТЕР работает в режиме квантования (см. SQL-команду **SET DATABASE QUANT**)

Описание

Команда неканальная, и ее выполнение разрешено любому пользователю БД.

Клиентскому приложению возвращается столько первых полей структуры DB_DESC, сколько целиком умещается в указанный им размер буфера.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterDESC(TCBL * pCBL, DB_DESC * pOut)
  {
  memcpy(pCBL->Command, "DESC", 4);
  pCBL->LnBufRow = sizeof(DB_DESC);
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, NULL, NULL, NULL, pOut);
  return pCBL->CodErr;
  }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#ifndef WINCE
int    main()
#else
int    exdesc()
#endif
  {
  TCBL    CBLconnect;
```

```
 L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
 L_CHAR    Node[] = "        ";
 L_WORD    Priority = 0;
 L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
 L_LONG  Err;
 DB_DESC desc;

 memset(&CBLconnect,0,sizeof(TCBL));
 Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
PrzExe);
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 printf("Connect to RDBMS Linter\n");

 Err = LinterDESC(&CBLconnect, &desc);
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 printf("DESC :\n");
 printf("\tVerMajor=%ld\n", desc.VerMajor);
 printf("\tVerMinor=%ld\n", desc.VerMinor);
 printf("\tVerBuild=%ld\n", desc.VerBuild);
 printf("\tSortPoolSize=%ld\n", desc.SortPoolSize);
 printf("\tKernelPoolSize=%ld\n", desc.KernelPoolSize);
 printf("\tFileQueueSize=%ld\n", desc.FileQueueSize);
 printf("\tUserQueueSize=%ld\n", desc.UserQueueSize);
 printf("\tTableQueueSize=%ld\n", desc.TableQueueSize);
 printf("\tColumnQueueSize=%ld\n", desc.ColumnQueueSize);
 printf("\tChannelQueueSize=%ld\n", desc.ChannelQueueSize);
 printf("\tSnapTimeout=%ld\n", desc.SnapTimeout);
 printf("\tKillTimeout=%ld\n", desc.KillTimeout);
 printf("\tBaseName=%.18s\n", desc.BaseName);
 printf("\tSysLog=%hd\n", desc.SysLog);
 printf("\tSync=%hd\n", desc.Sync);
 printf("\tLog=%hd\n", desc.Log);
 printf("\tOs=%ld\n", (L_LONG ) desc.Os);
 printf("\n");

 printf("End Example\n");

 return 0;
 }
```

Команды обработки данных

Команды манипулирования данными позволяют выполнять следующие операции:

- поиск данных в БД;

- добавление данных в таблицу;
- модификацию данных в таблице;
- удаление данных из таблицы (по условию или всех).

Для выполнения этих операций в СУБД ЛИНТЕР предусмотрено две команды: `SLCT` и " "(имя из четырех пробелов). По управляющему блоку СУБД отличает поисковые запросы от всех остальных, дополнительные различия извлекаются из текста SQL-запроса, находящегося в буфере оператора.

Поиск и выборка данных

Назначение

Команда `SLCT` позволяет приложению послать к СУБД поисковый запрос и получить в ответ найденную в БД информацию.

Параметры вызова

```
inter(CBL, VarBuf, OpBuf, [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;
- буфер оператора `OpBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`NumChan`	Номер канала
`Command`	`"SLCT"`
`LnBufRow`	Длина буфера выборки данных
`Node`	Имя ЛИНТЕР-сервера
`PrzExe`	Режим обработки запроса

Буфер оператора `OpBuf` должен содержать текст SELECT-запроса.

Выходные данные

Выходными данными являются:

- контрольный блок `CBL`;
- буфер выборки данных RowBuf;
- буфер параметров `VarBuf`.

В контрольном блоке будут возвращены:

Имя поля	Значение
`CodErr`	Код завершения запроса к СУБД ЛИНТЕР
`RowId`	Системный номер первой выбранной записи
`RowCount`	Число найденных записей
`LnBufRow`	Фактическая длина выбранной записи выборки данных

Имя поля	Значение
SysErr	Код состояния ОС

Примечание
Значение поля RowId однозначно определяет системный номер выбранной записи только в случае выборки из одной таблицы. В случае запроса, который выбирает записи из нескольких множеств, значение RowId неоднозначно и будет установлено в 0.

Буфер выборки данных RowBuf будет содержать первую запись выборки данных.

В буфере VarBuf возвращается маска NULL-значений, представленная в следующем виде:

```
L_WORD K1;       /* Количество записей в порции выборки данных */
L_WORD K2;       /* Количество полей в записи выборки данных */
L_BYTE [K1*K2]; /* По одному байту на каждое поле */
                 /* записи выборки данных */
/* Нулевое значение байта соответствует определенному */
/* значению поля, значение 1 – неопределенному (NULL-значение) */
```

Описание

При обработке SELECT-запроса СУБД ЛИНТЕР полностью выбирает из БД все найденные записи, однако в приложение передается только первая найденная запись. Дальнейшая работа с записями осуществляется с помощью команд:

- GETA – дать информацию о структуре записи выборки данных;
- GETF – дать первую запись выборки данных;
- GETL – дать последнюю запись выборки данных;
- GETN – дать следующую запись выборки данных;
- GETP – дать предыдущую запись выборки данных;
- GETS – дать указанную запись выборки данных;
- GETM – дать несколько записей за одно обращение.

Примечание
СУБД ЛИНТЕР «не знает» реальной границы программных буферов приложения и при размещении записи в буфере выборки данных руководствуется только полученным значением LnBufRow. Поэтому, если заданное значение LnBufRow больше фактической длины буфера выборки данных, то часть записи может быть размещена за пределами буфера выборки данных, запортив при этом данные и/или программный код приложения.

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Искомые данные не найдены
Целочисленное значение	Коды завершения, вызванные синтаксическими или семантическими ошибками поданного SELECT-запроса (см. документ «Справочник кодов завершения», подраздел «Коды завершения выборки данных (1-3)»)

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterSLCT(TCBL * pCBL, L_LONG PrzExe, L_CHAR * Statement,
 void *RowBuf, L_WORD RowBufLen, void *NullBuf)
  {
  memcpy(pCBL->Command, "SLCT", 4);
  pCBL->PrzExe = PrzExe;
  pCBL->LnBufRow = RowBufLen;
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, NullBuf, Statement, NULL, RowBuf);
  return pCBL->CodErr;
  }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exslct()
#endif
  {
  struct TRowBuf
    {
    L_CHAR    Name[20];
    L_CHAR    FirstName[15];
    L_CHAR    Sex;
    };
  typedef struct TRowBuf TRowBuf;
  TCBL    CBLconnect;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;
  L_CHAR    Query[] = "select NAME,FIRSTNAM,SEX from PERSON;";
  TRowBuf RowBuf;
```

```
 memset(&CBLconnect,0,sizeof(TCBL));
 Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
PrzExe);
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 printf("Connect to RDBMS Linter\n");

 Err = LinterSLCT(&CBLconnect, PrzExe, Query, &RowBuf,
sizeof(TRowBuf), NULL);
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 printf("First Selected Row:\n");
 printf("%.20s %.15s %c\n", RowBuf.Name, RowBuf.FirstName,
RowBuf.Sex);

 printf("End Example\n");

 return 0;
 }
```

Дать информацию о структуре записи выборки данных

Назначение

Команда GETA возвращает информацию о полях записей выборки данных.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"GETA"
RowId	Порядковый номер поля записи выборки данных, начиная с которого выдавать структуру записи; отсчет порядковых номеров начинается с 0
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными являются:

- контрольный блок CBL;

- буфер выборки данных RowBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
RowCount	Количество полей в буфере выборки данных. Устанавливается даже в случае, когда в буфере выборки нет места для информации об одном поле
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Информация в буфере выборки данных RowBuf будет представлена в виде массива, каждый элемент которого имеет следующую структуру:

```
struct PACKED_ATTR GETA_OUT
   {
   L_CHAR User[MAX_ID_LEN];    /* Имя пользователя */
   L_CHAR Table[MAX_ID_LEN];   /* Имя таблицы */
   L_CHAR Column[MAX_ID_LEN]; /* Имя столбца */
#if _VER_MAX >= 500
   L_WORD Length;              /* Длина столбца */
   L_BYTE Type;                /* Тип данных столбца */
#else
   L_BYTE Type;                /* Тип данных столбца */
   L_BYTE Length;
#endif
   L_BYTE Precision;           /* Точность (для числовых данных */
                               /* типа Numeric) */
   L_BYTE Scale;               /* Масштаб (для данных типа Numeric)
 */
#if _VER_MAX >=500
   L_BYTE Reserve;
#endif
#if _VER_MAX >=600
   L_WORD CharSet;             /* Идентификатор кодовой страницы */
#endif
   };
```

Коды возвращаемых в поле Type типов данных приведены в приложении 3.

Максимальный размер массива равен количеству полей в выборке данных. Если количество полей велико, то возможно получение информации обо всех столбцах порциями. Размер порции описания столбцов, полученной за один раз, ограничивается длиной буфера и максимальным размером сообщения. Реальное количество структур в порции можно вычислить по формуле: Количество_Полей = LnBufRow/ sizeof(GETA_OUT).

Если количество описаний полей в порции меньше количества полей в выборке данных, то для получения следующей порции информации о полях записи необходимо повторно выдать команду GETA, установив в поле RowId соответствующее значение.

Примечание
Для получения характеристик одного конкретного поля буфера выборки данных надо в `LnBufRow` задать длину sizeof(GETA_OUT), а в `RowId` – порядковый номер интересующего поля.

Описание

Если выборка данных была выполнена из представления, то имя схемы и имя таблицы будут содержать имя схемы и имя представления.

Если в SELECT-запросе имя таблицы или представления задано через псевдоним, то в буфере выборки данных в качестве имени таблицы (представления) будет помещено имя псевдонима.

Если поле буфера выборки данных является функцией или константой, то именем такого поля будут пробелы или псевдоним (если он задан).

В качестве типа данных полей, являющихся функциями, возвращается результирующий тип функции.

Типом данных константных полей является тип данных константы.

Коды завершения

Код	Описание
`NORMAL`	Нормальное завершение
`SMALLBUFKOR`	Недостаточный размер буфера выборки данных (невозможно поместить информацию даже об одном поле выборки данных) **Примечание** Если `LnBufRow` == 0, то возвращается количество полей выборки RowCount и ошибочный код завершения не устанавливается
`EORR`	Искомые данные не найдены (указан порядковый номер несуществующего поля записи)
`ERRSEQCOM`	Неправильная последовательность команд (команда `GETA` подана вне контекста SELECT-запроса)

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterGETA(TCBL * pCBL, L_WORD N, void *Out, L_WORD OutLen)
  {
  memcpy(pCBL->Command, "GETA", 4);
  pCBL->RowId = N;                /* порядковый номер столбца */
  pCBL->LnBufRow = OutLen;
  pCBL->PrzExe &= ~Q_ASYNC;
  inter(pCBL, NULL, NULL, NULL, Out);
```



```
  return pCBL->CodErr;
  }
```

Примеры использования команды

1) Библиотечная функция получения количества полей в записи выборки данных.

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterAnsColNum(TCBL * pCBL, L_WORD * N)
  {
  memcpy(pCBL->Command, "GETA", 4);
  pCBL->PrzExe &= ~Q_ASYNC;
  pCBL->LnBufRow = 0;
  inter(pCBL, NULL, NULL, NULL, NULL);
  *N = (L_WORD) pCBL->RowCount;
  return pCBL->CodErr;
  }
```

2) Пример использования функции.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"


#ifndef WINCE
int    main()
#else
int    exgeta()
#endif
  {
  TCBL    CBLconnect;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;
  L_CHAR    Query[] = "select NAME,FIRSTNAM,SEX from PERSON;";
  GETA_OUT *answer;
  L_WORD    NColumn;
  L_WORD    i;


  memset(&CBLconnect,0,sizeof(TCBL));
```

```
 Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
PrzExe);
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 printf("Connect to RDBMS Linter\n");

 Err = LinterSLCT(&CBLconnect, PrzExe, Query, NULL, L_MAXWORD,
NULL);
 if (Err != NORMAL)
   PrintError(&CBLconnect);

 Err = LinterAnsColNum(&CBLconnect, &NColumn);
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 printf("GETA (1), Answer Hold %d column\n", NColumn);

 answer = (GETA_OUT *) calloc(NColumn, sizeof(GETA_OUT));
 if (answer == NULL)
   {
   printf("Error: not memory\n");
   exit(1);
   }

 Err = LinterGETA(&CBLconnect, 0, answer, (L_WORD) (NColumn
*sizeof(GETA_OUT)));
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 printf("GETA (2):\n");
 for (i = 0; i < NColumn; i++)
   {
   printf("Answer Column %d\n", i + 1);
   printf("\tUser=%.18s\n", answer[i].User);
   printf("\tTable=%.18s\n", answer[i].Table);
   printf("\tColumn=%.18s\n", answer[i].Column);
   printf("\tLength=%d\n", answer[i].Length);
   printf("\tType=%hd\n", answer[i].Type);
   printf("\tPrecision=%hd\n", answer[i].Precision);
   printf("\tScale=%hd\n\n", answer[i].Scale);
   }
 free(answer);

 printf("End Example\n");

 return 0;
 }
```


Дать первую запись выборки данных

Назначение

Команда GETF возвращает первую запись выборки данных.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"GETF"
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf;
- буфер параметров VarBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
RowId	Системный номер выбранной записи
RowCount	Без изменений
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Буфер выборки данных RowBuf будет содержать первую запись выборки данных.

В буфере VarBuf возвращается маска NULL-значений.

Описание

Команда выдает первую запись SELECT-запроса. Выданная запись становится текущей записью в канале.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Искомые данные не найдены (выборка данных пуста)

Код	Описание
NOKOR	Запись недоступна (удалена или заблокирована по другому каналу)

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterGETF(TCBL *pCBL, void *RowBuf, L_WORD RowBufLen, void
 *NullBuf)
   {
   memcpy(pCBL->Command, "GETF", 4);
   pCBL->LnBufRow=RowBufLen;
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NullBuf, NULL, NULL, RowBuf);
   return pCBL->CodErr;
   }
```

Пример использования команды

Дать последнюю запись выборки данных

Назначение

Команда GETL возвращает последнюю запись выборки данных.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"GETL"
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf;
- буфер параметров VarBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
RowId	Системный номер выбранной записи
RowCount	Без изменений
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Буфер выборки данных RowBuf будет содержать последнюю запись выборки данных.

В буфере VarBuf возвращается маска NULL-значений.

Описание

Команда выдает последнюю запись SELECT-запроса. Выданная запись становится текущей записью в канале.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Искомые данные не найдены (выборка данных пуста)
ERRSEQCOM	Неправильная последовательность команд (команда GETL подана вне контекста SELECT-запроса)
NOKOR	Запись недоступна (удалена или заблокирована по другому каналу)

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterGETL(TCBL *pCBL, void *RowBuf, L_WORD RowBufLen, void
 *NullBuf)
   {
   memcpy(pCBL->Command, "GETL", 4);
   pCBL->LnBufRow=RowBufLen;
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NullBuf, NULL, NULL, RowBuf);
   return pCBL->CodErr;
   }
```

Пример использования команды

Дать следующую запись выборки данных

Назначение

Команда GETN возвращает следующую (после текущей)запись выборки данных.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"GETN"
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf;
- буфер параметров VarBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
RowId	Системный номер выбранной записи
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Буфер выборки данных RowBuf будет содержать следующую (после текущей) запись выборки данных.

В буфере VarBuf возвращается маска NULL-значений.

Описание

Команда выдает следующую запись выборки данных после текущей записи выборки. Текущей записью является последняя запись, реально выданная по данному каналу.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Искомые данные не найдены (выборка данных пуста, или запрашивается следующая запись после последней в выборке)
ERRSEQCOM	Неправильная последовательность команд (команда GETN подана вне контекста SELECT-запроса)

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterGETN(TCBL *pCBL, void *RowBuf, L_WORD RowBufLen, void
 *NullBuf)
   {
   memcpy(pCBL->Command, "GETN", 4);
   pCBL->LnBufRow=RowBufLen;
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NullBuf, NULL, NULL, RowBuf);
   return pCBL->CodErr;
   }
```

[Пример использования команды]

Дать предыдущую запись выборки данных

Назначение

Команда GETP возвращает предыдущую (по отношению к текущей) запись выборки данных.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"GETP"
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf;
- буфер параметров VarBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР

Имя поля	Значение
RowId	Системный номер выбранной записи
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Буфер выборки данных RowBuf будет содержать предыдущую (по отношению к текущей) запись выборки данных.

В буфере VarBuf возвращается маска NULL-значений.

Описание

Команда выдает запись выборки данных, предшествующую текущей записи выборки. Текущей записью является последняя запись, реально выданная по данному каналу.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Искомые данные не найдены (выборка данных пуста, или запрашивается предыдущая запись перед первой записью выборки)
ERRSEQCOM	Неправильная последовательность команд (команда GETP подана вне контекста SELECT-запроса)

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterGETP(TCBL *pCBL, void *RowBuf, L_WORD RowBufLen, void
 *NullBuf)
   {
   memcpy(pCBL->Command, "GETP", 4);
   pCBL->LnBufRow=RowBufLen;
   inter(pCBL, NullBuf, NULL, NULL, RowBuf);
   return pCBL->CodErr;
   }
```

Пример использования команды

Дать указанную запись выборки данных

Назначение

Команда GETS возвращает указанную запись выборки данных.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"GETS"
RowId	Порядковый номер требуемой записи выборки данных
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf;
- буфер параметров VarBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
RowId	Системный номер выбранной записи
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Буфер выборки данных RowBuf будет содержать указанную запись выборки данных.

В буфере VarBuf возвращается маска NULL-значений.

Описание

Команда выдает указанную запись выборки данных, которая становится текущей.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Искомые данные не найдены (выборка данных пуста, или запрашивается несуществующая запись)
ERRSEQCOM	Неправильная последовательность команд (команда GETS подана вне контекста SELECT-запроса)
NOKOR	Запись недоступна (удалена или заблокирована по другому каналу)

Пример формирования команды

```
#include <string.h>
```

```
#include <stdlib.h>
#include "inter.h"

L_LONG LinterGETS(TCBL *pCBL, L_LONG N, void *RowBuf, L_WORD
 RowBufLen,
   void *NullBuf)
   {
   memcpy(pCBL->Command, "GETS", 4);
   pCBL->LnBufRow=RowBufLen;
   pCBL->RowId=N;         /* номер записи */
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NullBuf, NULL, NULL, RowBuf);
   return pCBL->CodErr;
   }
```

Пример использования команды

Дать группу записей выборки данных

Назначение

Команда GETM возвращает записи выборки данных в виде порции из заданного количества записей.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"GETM"
RowCount	Количество требуемых записей
RowId	Начальный номер требуемой порции записей
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Примечания

1. Если RowId=0, то первой записью в порции будет следующая запись после текущей записи выборки.
2. Если RowCount=0, то в порции будет возвращено максимально возможное количество записей.

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf;
- буфер параметров VarBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
RowId	Системный номер последней записи в полученном пакете данных
LnBufRow	Фактическая длина буфера выборки данных
RowCount	Реальное число выбранных записей
SysErr	Код состояния ОС

Буфер выборки данных RowBuf будет содержать порцию записей выборки данных.

В буфере VarBuf возвращается маска NULL-значений.

Описание

После выполнения GETM в буфер RowBuf считывается очередная порция записей выборки данных, курсор позиционируется на последней выбранной по GETM записи.

Количество записей, выбираемых в данной порции, ограничивается количеством данных в выборке, размером буфера для сохранения порции LnBufRow, количеством запрашиваемых данных RowCount (если != 0) и максимальным размером данных, передаваемых интерфейсом (64 Кбайт минус накладные расходы). Реальное число выбранных записей возвращается в поле RowCount.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Искомые данные не найдены (выборка данных пуста, или задан неправильный номер начальной записи)
ERRSEQCOM	Неправильная последовательность команд (команда GETM подана вне контекста SELECT-запроса)

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterGETM(TCBL *pCBL, L_LONG Start, L_LONG N,
    void *RowBuf, L_WORD RowBufLen, void *NullBuf)
   {
   memcpy(pCBL->Command, "GETM", 4);
```

```
   pCBL->LnBufRow=RowBufLen;
   pCBL->RowId=Start; /* start record, if RowId=0 then next
 *record */
   pCBL->RowCount=N;  /* количество записей */
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NullBuf, NULL, NULL, RowBuf);
   return pCBL->CodErr;
   }
```

Пример использования команд для работы с выборкой данных

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#ifndef WINCE
int    main()
#else
int    exgetx()
#endif
    {
struct TRowBuf
   {
        L_CHAR  Name[20];
L_CHAR  FirstName[15];
L_CHAR  Sex;
};
    typedef struct TRowBuf TRowBuf;

TCBL  CBLconnect;
L_CHAR  Name_Pass[]="SYSTEM/MANAGER8";
L_CHAR  Node[]="    ";
L_WORD  Priority=0;
L_LONG PrzExe=M_EXCLUSIVE | Q_ENCODE | M_BINARY;
L_LONG Err;
L_CHAR  Query[]="select NAME,FIRSTNAM, SEX from PERSON;";
    L_WORD  i;
TRowBuf RowBuf;

#define M 5
    TRowBuf MRowBuf[M];

memset(&CBLconnect,0,sizeof(TCBL));
Err=LinterOPEN(&CBLconnect,Name_Pass,Node, Priority,PrzExe);
    if (Err != NORMAL)
```



```
        PrintError(&CBLconnect);
    printf("Connect to RDBMS Linter\n");

Err =LinterSLCT(&CBLconnect,PrzExe,Query,&RowBuf, sizeof(TRowBuf),
 NULL);
    if (Err != NORMAL)
        PrintError(&CBLconnect);
    printf("Select\n");
Err=LinterGETF(&CBLconnect, &RowBuf, sizeof(TRowBuf), NULL);
    if (Err != NORMAL)
        PrintError(&CBLconnect);
    printf("GETF:\n");
printf("%.20s %.15s %c\n\n", RowBuf.Name, RowBuf.FirstName,
 RowBuf.Sex);
    Err=LinterGETN(&CBLconnect, &RowBuf, sizeof(TRowBuf), NULL);
    if (Err != NORMAL)
        PrintError(&CBLconnect);
    printf("GETN:\n");
printf("%.20s %.15s %c\n\n", RowBuf.Name, RowBuf.FirstName,
 RowBuf.Sex);
    Err=LinterGETL(&CBLconnect, &RowBuf, sizeof(TRowBuf), NULL);
    if (Err != NORMAL)
        PrintError(&CBLconnect);
    printf("GETL:\n");
printf("%.20s %.15s %c\n\n", RowBuf.Name, RowBuf.FirstName,
 RowBuf.Sex);

Err=LinterGETS(&CBLconnect, 5, &RowBuf, sizeof(TRowBuf), NULL);
    if (Err != NORMAL)
        PrintError(&CBLconnect);
    printf("GETS:\n");
printf("%.20s %.15s %c\n\n", RowBuf.Name, RowBuf.FirstName,
 RowBuf.Sex);

Err =LinterGETM(&CBLconnect,1,M,&MRowBuf, sizeof(TRowBuf)*M,
 NULL);
    if (Err != NORMAL)
        PrintError(&CBLconnect);
    printf("GETM:\n");
for (i=0; i < M; i++)
        printf("%.20s %.15s %c\n",
            MRowBuf[i].Name,
MRowBuf[i].FirstName,
MRowBuf[i].Sex);
    printf("\n");
```

```
printf("End Example\n");

return 0;
}
```

Пакетное добавление данных

Назначение

Команда PUTM предназначена для добавления в таблицу или обновляемое представление БД группы записей (пакета). Добавление выполняется в тот объект БД, который указан в предварительно выданной SQL-команде **START APPEND**.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок CBL;
- буфер записи RowBuf.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"PUTM"
LnBufRow	Размер пакета в байтах
Node	Имя ЛИНТЕР-сервера

Пакет данных передается в буфере RowBuf.

Структура пакета данных:

1) количество записей в пакете (2 байта);
2) первая запись пакета;
3) вторая запись пакета;
4) …
5) последняя запись пакета.

Структура записи пакета данных:

1) длина первого поля записи (2 байта), следом – значение поля;
2) длина второго поля записи, следом – значение поля;
3) длина N-го поля записи, следом – значение поля;
4) длина последнего поля записи, следом – значение поля.

Значение поля для VAR типов (VARCHAR, VARBYTE, NVARCHAR) является комбинацией длины поля и его значения. То есть в пакете для этих полей должно стоять 2 длины по 2 байта подряд. Одно из них определяет общую длину поля, включая длину поля VAR типа, вторая длина – на 2 байта меньше и определяет только длину данных.

Примечания

1. Если для поля типа VARCHAR указано преобразования типа BYTE, реально это поле передается ядру СУБД ЛИНТЕР не в байтовом формате, а в символьном, поэтому в этом случае должен быть сформирован префикс длины для значения полей типа VAR.
2. Особенности формирования пакета данных в зависимости от его формата описаны в конструкции START APPEND в документе «Справочник по SQL», подраздел «Пакетное добавление».
3. Для представления NULL-значений и значений по умолчанию указывается длина -1 и -2 соответственно. Значение поля в пакете при этом должно отсутствовать.

Выходные данные

Выходными данными является контрольный блок `CBL`.

В нем будут возвращены:

Имя поля	Значение
`CodErr`	Код завершения запроса к СУБД ЛИНТЕР
`RowCount`	Фактическое количество добавленных записей
`SysErr`	Код состояния ОС

Описание

Максимальный размер пакета равен 8000 байт (если ядро СУБД ЛИНТЕР и клиентское приложение собраны без макроса LARGE_EXCHANGE) и 64000 байт, если они собраны с макросом LARGE_EXCHANGE (по крайней мере с этим макросом должен быть транслирован файл `intlib.c`, входящий в состав клиентского приложения либо непосредственно, либо через какую-то библиотеку). Все библиотеки, входящие в состав последних версий СУБД ЛИНТЕР, также собраны с макросом LARGE_EXCHANGE.

В следующих версиях СУБД способ задания размера буфера `PUTM` с помощью макроса может измениться.

Если неизвестно, каким способом задается размер буфера `PUTM`, то фактический размер определяется эмпирически.

Если в процессе загрузки пакета записей обнаружится ошибка добавления некоторой записи, то загрузка оставшейся части пакета прекращается. Поле `RowCount` в блоке `CBL` будет содержать фактическое число загруженных записей.

Пакетное добавление может выполняться в обновляемое представление.

При выполнении `PUTM` блокировка записей выполняется в соответствии с установленным режимом обработки транзакций. Если число заблокированных записей превысит 1024, то будет заблокирована вся таблица.

Команда выполняется только в контексте пакетного режима, поэтому перед началом пакетного добавления должен быть выполнен SQL-запрос **START APPEND INTO…**, окончание пакетного режима объявляется SQL-запросом **END APPEND**.

В пакетном режиме, кроме команды `PUTM`, разрешены команды COMT, RBAC, CLOS, KILL и SQL-команда **END APPEND**. Все другие команды запрещены.

Примечание
При пакетной вставке данных триггеры, настроенные на вставку данных, срабатывать не будут.

Код	Описание
NORMAL	Нормальное завершение
ERRSEQCOM	Неправильная последовательность команд (команда PUTM подана вне контекста запроса START APPEND)
BADPACKET	Неправильный пакет данных

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterPUTM(TCBL *pCBL, void *RowBuf, L_WORD RowBufLen)
   {
   memcpy(pCBL->Command, "PUTM", 4);
   pCBL->LnBufRow=RowBufLen;
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, RowBuf);
   return pCBL->CodErr;
   }
```

Примеры использования команды

1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#define MAX_ROWS 50

#ifndef WINCE
int    main()
#else
int    exputm()
#endif
   {
   TCBL  CBLconnect;
   L_CHAR  Name_Pass[]="SYSTEM/MANAGER8";
   L_CHAR  Node[]="    ";
   L_WORD  Priority=0;
   L_LONG PrzExe=M_EXCLUSIVE | Q_ENCODE | M_BINARY;
```

```
  L_LONG Err;
  L_BYTE  PackBuf[4096];
  L_WORD  *Len;
  L_LONG i;
  L_WORD  size;

  memset(&CBLconnect,0,sizeof(TCBL));
  Err=LinterOPEN(&CBLconnect, Name_Pass, Node, Priority, PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");

  LinterNotSelect(&CBLconnect, "drop table TT;");
  LinterNotSelect(&CBLconnect, "create table TT(CC int);");


  Len=(L_WORD *) (PackBuf+2);
  size=sizeof(L_WORD);;
  for (i=0; i < MAX_ROWS; i++)
     {
     *Len=sizeof(L_LONG);
     memcpy(Len+1, &i, sizeof(L_LONG));
     Len += 1+sizeof(L_LONG)/sizeof(L_WORD);
     size += sizeof(L_LONG)+sizeof(L_WORD);;
     }
  *((L_WORD *) PackBuf)=(L_WORD) i;

  if(LinterNotSelect(&CBLconnect, "START APPEND INTO TT BYTE
(CC);") != NORMAL)
    PrintError(&CBLconnect);

  Err=LinterPUTM(&CBLconnect, PackBuf, size);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("PUTM\n");

  if (LinterNotSelect(&CBLconnect, "END APPEND INTO TT;") !=
NORMAL)
    PrintError(&CBLconnect);

  printf("End Example\n");

  return 0;
  }
```

2) В приложении 10 приведены примеры пакетной обработки данных.

Выполнение SQL-запроса

Назначение

Выполнение любого SQL-запроса СУБД ЛИНТЕР.

Примечание
Для выполнения поисковых SQL-запросов можно вместо данной команды использовать специальную команду SLCT.

Параметры вызова

```
inter(CBL, NULL, OpBuf, [CondBuf], NULL);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;
- буфер оператора `OpBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`NumChan`	Номер канала
`Command`	`"4 пробела"`
`Node`	Имя ЛИНТЕР-сервера

Буфер оператора `OpBuf` должен содержать символьный текст SQL-оператора в кодировке канала, по которому подается запрос. Текст SQL-запроса должен заканчиваться двоичным нулем.

Выходные данные

Выходными данными является контрольный блок `CBL`.

В нем будут возвращены:

Имя поля	Значение
`CodErr`	Код завершения запроса к СУБД ЛИНТЕР
`RowId`	Зависит от выполняемого SQL-запроса (таблица 10)
`RowCount`	Зависит от выполняемого SQL-запроса (таблица 10)
`SysErr`	Код состояния ОС

Таблица 10. Возвращаемое значение поля RowId для SQL-запросов

SQL-запрос	Значение поля RowId
BACKUP DATABASE …	`RowId` записи процесса архивирования в системной таблице `$$$INKERNBACK` (системный идентификатор процесса архивирования)

SQL-запрос	Значение поля RowId
BACKUP STOP ALL	Количество прерванных процессов архивирования
WAIT EVENT … GET EVENT …	Битовая маска наступивших (для WAIT) или установленных (для GET) событий, указанных в запросе. Первое указанное в запросе событие соответствует нулевому биту маски, второе – первому биту и т.д. Для WAIT EVENT значение 1 бита маски соответствует наступившему событию, значение 0 – ожидаемому. Для GET EVENT значение 1 бита маски означает, что событие установлено (активно). Значение лишних битов маски равно 0. Если <*выражение-событие*> в SQL-запросе содержит более 32 событий, то информация о 33-м и последующих событиях клиентскому приложению недоступна
SELECT …	`RowId` первой выбранной записи, если SELECT-запрос выполнялся из одной таблицы или обновляемого представления. `RowCount` – количество выбранных записей
INSERT … DELETE … UPDATE …	`RowId` последней реально обработанной записи. `RowCount` – количество реально обработанных записей

Описание

Команда должна посылаться по открытому каналу.

Если по каналу предварительно был выполнен SQL-запрос **SET NAMES** (см. документ «Справочник по SQL», пункт «Кодировка соединения по умолчанию»), то по умолчанию символьные столбцы в создаваемых таблицах (по запросу **CREATE TABLE …**) будут иметь кодировку канала.

Обработка SELECT-запроса с помощью данной команды и команды `SLCT` выполняется идентично, за исключением того, что содержимое полей блока `CBL` в файле журнала протоколирования (`LINTER.LOG`) будет различным.

Коды завершения

Возвращается код завершения, сформированный ядром СУБД ЛИНТЕР при обработке полученного SQL-оператора.

Возможные коды завершения приведены в документе «Справочник кодов завершения», подраздел «Коды завершения при обработке данных (8100-8888)».

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"
L_LONG LinterNotSelect(TCBL *pCBL, L_CHAR *Statement)
   {
   memset(pCBL->Command, ' ', 4);
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, Statement, NULL, NULL);
   return pCBL->CodErr;
```

```
}
```

[Пример использования команды]

Команды работы с BLOB-данными

Команды этой группы предназначены для работы с бесформатной байтовой информацией большого объема (BLOB-данными).

BLOB-функции применяются к текущей записи из предшествующего запроса выборки (SELECT-запроса) или к текущей добавленной записи (INSERT-запрос), поэтому в этих функциях параметр «номер BLOB-столбца» относится не к исходной таблице, а к *номеру столбца в предшествовавшем SELECT/INSERT запросе*.

Дать порцию BLOB-данных

Назначение

Команда `GBLB` предназначена для чтения заданной порции BLOB-данных.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными является контрольный блок `CBL`.

В нем должны быть заполнены поля:

Имя поля	Значение
`NumChan`	Номер канала
`Command`	`"GBLB"`
`RowId`	Начало требуемой порции BLOB-данных
`LnBufRow`	Длина буфера выборки данных (размер требуемой порции данных в байтах)
`Node`	Имя ЛИНТЕР-сервера
`RowCount`	Порядковый номер BLOB-столбца

Выходные данные

Выходными данными являются:

- контрольный блок `CBL`;
- буфер выборки данных `RowBuf`.

В контрольном блоке будут возвращены:

Имя поля	Значение
`CodErr`	Код завершения запроса к СУБД ЛИНТЕР

Имя поля	Значение
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Описание

Максимальный размер порции BLOB-значения – 16*4048 байт.

Команда GBLB читает порцию BLOB-данных из **указанного по номеру** столбца **текущей** записи канала. Если номер BLOB-столбца не задан, то поведение команды не определено.

Примечания

1. Номер столбца должен ссылаться на BLOB-столбец. Нумерация столбцов записи начинается с 1 и является абсолютной, то есть нумеруются столбцы всех типов данных, а не только BLOB. Это касается всех BLOB-команд.
2. Команда GOBJ, используемая для выборки порции BLOB-данных из записи с одним BLOB-столбцом (т.е. без указания номера BLOB-столбца), устарела и не рекомендуется для применения.

Текущей записью в канале является последняя выбранная, добавленная или измененная запись. Таким образом, команда GBLB имеет смысл только после выполнения следующих запросов:

- SELECT;
- INSERT;
- UPDATE.

Началом первой порции BLOB-значения считается 1.

Если при считывании получена неполная порция (считывание остатка), то поле LnBufRow в контрольном блоке будет равно реальному числу принятых байтов.

Команда GBLB не изменяет текущее положение записи в канале.

Примечание

Не допускается смешивать в канале обработку BLOB-команд данной таблицы и работу с иными таблицами. Это сбивает текущие настройки интерфейса нижнего уровня. Если **одновременно** с обработкой BLOB-данных какой-либо таблицы необходима работа с другой (другими) таблицами, то для этого следует использовать отдельный канал. Замечание касается всех BLOB-команд.

Для получения информации о типе BLOB-данных (текст, графика, музыка и т.п.) следует выполнить SELECT-запрос на чтение BLOB-столбца. При выполнении этого запроса в буфере RowBuf данные будут представлены в следующем виде:

```
struct BLOB_ATR
   {
   L_LONG Size;
   L_LONG Adr_First_Pag;
```

```
   L_LONG Adr_Last_Pag;
   L_BYTE NmrFil;
   L_BYTE Pad[1];
   L_BYTE modTime[6];
   L_LONG TypeObj;
   };
```

Поле modTime содержит время последнего обновления BLOB-значения. Сравнивается с временем последнего обновления фразового индекса в случае, когда нужно определить, следует ли использовать это BLOB-значение при очередном обновлении фразового индекса или нет.

Если BLOB-столбец имеет не NULL-значение, то в полях буфера выборки данных будет содержаться:

Имя поля	Значение
Size	Размер BLOB-данных
Adr_First_Pag	Номер первой страницы BLOB-файла (в единицах 4 Кбайт)
Adr_Last_Pag	Номер последней страницы BLOB-файла (в единицах 4 Кбайт)
NmrFil	Логический номер BLOB-файла (значение от 1 до 63)
TypeObj	Тип BLOB-данных (задается пользователем при загрузке этих данных; это значение СУБД ЛИНТЕР не контролирует)
Pad	Резерв
modTime	Дата/время последнего обновления BLOB-значения

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
EORR	Начало порции превышает размер BLOB-данных
ERRSEQCOM	Неправильная последовательность команд (команда GBLB подана вне контекста текущего запроса канала)
COLNOTBLOB	Заданный столбец не является BLOB-столбцом
ERRVALRANGE	Неправильный номер столбца

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterGBLB(TCBL *pCBL, L_LONG Position, void *Out, L_WORD
 OutLen,
    L_WORD numCol)
   {
   memcpy(pCBL->Command, "GBLB", 4);
```



```
pCBL->LnBufRow=OutLen;
pCBL->RowId=Position;
pCBL->RowCount = numCol;
pCBL->PrzExe &= ~Q_ASYNC;
inter(pCBL, NULL, NULL, NULL, Out);
return pCBL->CodErr;
}
```

Пример использования команды

Добавить порцию BLOB-данных

Назначение

Команда ABLB предназначена для добавления заданной порции BLOB-данных.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок CBL;
- буфер записи RowBuf.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"ABLB"
RowId	Тип добавляемых данных
LnBufRow	Длина буфера записи (размер добавляемой порции данных в байтах)
Node	Имя ЛИНТЕР-сервера
RowCount	Порядковый номер BLOB-столбца

Буфер записи RowBuf должен содержать добавляемую порцию BLOB-данных.

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР

Имя поля	Значение
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Описание

Максимальный размер порции BLOB-значения – 16*4048 байт.

Максимальный размер BLOB-значения – 2 Гбайт.

Добавляемые данные помещаются в конец BLOB-значений.

При добавлении BLOB-данных СУБД ЛИНТЕР не контролирует тип добавляемых данных, поэтому, если добавляемые порции содержали разные типы BLOB-значения, то они все равно будут добавлены к BLOB-значению; окончательный тип BLOB-значения устанавливается по последней добавленной порции.

Команда ABLB добавляет порцию BLOB-данных к **текущему** BLOB-значению (BLOB-значению текущей записи выборки данных) в **указанном** BLOB-столбце записи.

Если номер BLOB-столбца не задан, то поведение команды не определено.

Примечание
Команда AOBJ, используемая для добавления порции BLOB-данных в запись с одним BLOB-столбцом (т.е. без указания номера BLOB-столбца), устарела и не рекомендуется для применения.

Текущей записью в канале является последняя выбранная, добавленная или измененная запись. Таким образом, команда ABLB имеет смысл только после выполнения следующих запросов:

- SELECT;
- INSERT;
- UPDATE.

Запросы INSERT и UPDATE должны быть обновляемыми.

Команда ABLB не изменяет текущее положение записи в канале.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
ERRPARTBL	Размер добавляемой порции больше 16*4048 байтов
EORR	Превышен допустимый размер данных BLOB-столбца
ERRSEQCOM	Неправильная последовательность команд (команда ABLB подана вне контекста текущего запроса канала)
COLNOTBLOB	Заданный столбец не является BLOB-столбцом
ERRVALRANGE	Неправильный номер столбца

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterABLB(TCBL *pCBL, void *Out, L_WORD OutLen, L_WORD
 numCol)
   {
   memcpy(pCBL->Command, "ABLB", 4);
   pCBL->LnBufRow=OutLen;
   pCBL->RowCount = numCol;
   pCBL->RowId = 1;
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, Out);
   return pCBL->CodErr;
   }
```

[Пример использования команды]

Удалить BLOB-данные

Назначение

Команда CBLB предназначена для удаления всех данных BLOB-столбца.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], NULL);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"CBLB"
Node	Имя ЛИНТЕР-сервера
RowCount	Порядковый номер BLOB-столбца

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР

Имя поля	Значение
SysErr	Код состояния ОС

Описание

Команда CBLB удаляет BLOB-данные из **указанного** BLOB-столбца **текущей** записи канала. Если номер BLOB-столбца не задан, то поведение команды не определено.

Примечание
Команда COBJ, используемая для удаления BLOB-данных в записи с одним BLOB-столбцом (т.е. без указания номера BLOB-столбца), устарела и не рекомендуется для применения.

Текущей записью в канале является последняя выбранная, добавленная или измененная запись. Таким образом, команда CBLB имеет смысл только после выполнения следующих запросов:

- SELECT;
- INSERT;
- UPDATE.

Запросы INSERT и UPDATE должны быть обновляемыми.

Команда CBLB удаляет BLOB-данные полностью. После выполнения операции длина данных становится равной 0, тип BLOB-данных не изменяется.

Команда CBLB не изменяет текущее положение записи в канале.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
ERRSEQCOM	Неправильная последовательность команд (команда CBLB подана вне контекста текущего запроса канала)
COLNOTBLOB	Заданный столбец не является BLOB-столбцом
ERRVALRANGE	Неправильный номер столбца

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"
L_LONG LinterCBLB(TCBL *pCBL)
   {
   memcpy(pCBL->Command, "CBLB", 4);
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, NULL);
   return pCBL->CodErr;
   }
```

Пример использования запросов обработки BLOB-данных

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#ifndef WINCE
int    main()
#else
int    exblob()
#endif
   {
   TCBL  CBLconnect;
   L_CHAR  Name_Pass[] = "SYSTEM/MANAGER8";
   L_CHAR  Node[] = "        ";
   L_WORD  Priority = 0;
   L_LONG PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
   L_LONG Err;
   L_CHAR  Buf[] = "Part of blob Part of blob Part of blob";

   memset(&CBLconnect,0,sizeof(TCBL));
   Err=LinterOPEN(&CBLconnect, Name_Pass, Node, Priority, PrzExe);
   if (Err != NORMAL)
     PrintError(&CBLconnect);
   printf("Connect to RDBMS Linter\n");

   if (LinterNotSelect (&CBLconnect, "create table OBJ(OBJ
 blob);") != NORMAL)
     PrintError(&CBLconnect);
   if (LinterNotSelect(&CBLconnect, "INSERT INTO OBJ
 VALUES(NULL);") != NORMAL)
     PrintError(&CBLconnect);

   Err=LinterABLB (&CBLconnect, Buf, (L_WORD) strlen(Buf),1);
   if (Err != NORMAL)
     PrintError(&CBLconnect);
   printf("ABLB\n");

   Err=LinterSLCT (&CBLconnect, PrzExe, "select * from OBJ;", Buf,
 sizeof(Buf), NULL);
   if (Err != NORMAL)
     PrintError(&CBLconnect);
   printf ("seleted BLOB length=%ld\n", *(L_LONG  *) Buf);

   Err=LinterGBLB (&CBLconnect, 1, Buf, sizeof(Buf) - 1);
```

```
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("GBLB:\n");
  printf("\t%s\n", Buf);

  Err=LinterGBLB (&CBLconnect);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  printf("GBLB\n");

  if (LinterNotSelect (&CBLconnect, "drop table OBJ;") != NORMAL)
    PrintError(&CBLconnect);

  printf("End Example\n");

  return 0;
  }
```

Команды управления доступом к данным

Указанные ниже команды блокирования/разблокирования таблиц и их записей устарели и не рекомендуются для использования (приведены для понимания работы клиентских приложений с ранними версиями СУБД ЛИНТЕР). Для управления доступа к данным следует использовать соответствующие режимы транзакций и/или запросы типа **SELECT FOR UPDATE**.

Блокировать таблицу

Назначение

Команда `LREL` предназначена для явной монопольной блокировки таблицы или обновляемого представления.

Параметры вызова

```
inter(CBL, VarBuf, NULL, [CondBuf], NULL);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;
- буфер параметров команды `VarBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`NumChan`	Номер канала
`Command`	`"LREL"`
`Node`	Имя ЛИНТЕР-сервера

Буфер описания параметров команды `VarBuf` должен содержать информацию о блокируемом объекте в следующем виде:

```
struct FREL_IN
   {
   L_LONG Owner;                  /*Идентификатор владельца*/
   L_CHAR TblName[MAX_ID_LEN];    /*Имя блокируемого объекта*/
   };
```

Примечания

1. Значение `Owner` можно получить при помощи команды FUSR интерфейса нижнего уровня или выбрать из системной таблицы `$$$USR` (при наличии права на SELECT-запросы к этой таблице).
2. Если длина имени объекта меньше MAX_ID_LEN символов, то поле `TblName` должно быть дополнено справа пробелами до MAX_ID_LEN символов.

Выходные данные

Выходными данными является контрольный блок `CBL`.

В нем будут возвращены:

Имя поля	Значение
`CodErr`	Код завершения запроса к СУБД ЛИНТЕР
`SysErr`	Код состояния ОС

Описание

Максимальное суммарное количество блокировок в СУБД ЛИНТЕР (по всем активным каналам) равно 100. Если один и тот же объект блокируется несколько раз, то увеличивается только счетчик блокировок данного объекта, но общее количество блокировок не увеличивается.

Все обновляемые представления, созданные из блокированной таблицы, также блокируются.

Примечание

Идеология СУБД ЛИНТЕР подразумевает, что запросы подаются не только от имени пользователя, но и от имени приложения, и даже от имени нити приложения. Поэтому внутри нити, блокировавшей таблицу, сама блокировка ощущаться не будет, однако для других приложений и даже нитей того же приложения таблица станет блокированной.

При завершении транзакции (**COMMIT** или **ROLLBACK**) установленная блокировка сбрасывается.

При выполнении по каналу оператора определения данных (типа **CREATE**…) установленная в нем блокировка сбрасывается.

При нормальном или аварийном закрытии канала установленная в нем блокировка сбрасывается.

При завершении работы ядра СУБД все блокировки сбрасываются.

При выполнении SELECT-запроса с блокированной таблицей в режиме WAIT (действует по умолчанию) клиентское приложение переходит в состояние ожидания разблокирования таблицы, в режиме NOWAIT – выдается код завершения, информирующий о блокировке таблицы.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
EORR	Блокируемый объект не найден
RELLOCKED	Блокируемый объект уже заблокирован другим приложением или другой нитью приложения

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterLREL(TCBL *pCBL, L_CHAR *User, L_CHAR *Table)
   {
   FREL_IN Tbl;
   L_LONG User_ID;
   L_LONG Err;

   Err=LinterFUSR(pCBL, User, &User_ID);
   if (Err == NORMAL)
     {
     L_LONG len;

     Tbl.Owner=User_ID;
     len=strlen(Table);
     if (len > MAX_ID_LEN)
       {
       return SQLLONGID;
       }
     memset(Tbl.TblName, ' ', MAX_ID_LEN);
     memcpy(Tbl.TblName, Table, len);
     }
   else
     return Err;

   memcpy(pCBL->Command, "LREL", 4);
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, &Tbl, NULL, NULL, NULL);
   return pCBL->CodErr;
```



```
}
```

[Пример использования команды]

Разблокировать таблицу

Назначение

Команда UREL предназначена для отмены всех ранее установленных в канале блокировок таблиц.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], NULL);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"UREL"
Node	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
SysErr	Код состояния ОС

Описание

Команда UREL сбрасывает **все** установленные в команде блокировки таблиц.

Если канал не содержит блокированного объекта, то команда UREL игнорируется (в этом случае код завершения команды будет NORMAL, как и при нормальном завершении).

При завершении транзакции (**COMMIT** или **ROLLBACK**) установленная блокировка сбрасывается.

При выполнении по каналу оператора определения данных (типа **CREATE**…) установленная в нем блокировка сбрасывается.

При нормальном или аварийном закрытии канала все установленные в нем блокировки сбрасываются.

При завершении работы ядра СУБД все блокировки сбрасываются.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterUREL(TCBL *pCBL)
   {
   memcpy(pCBL->Command, "UREL", 4);
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, NULL);
   return pCBL->CodErr;
   }
```

[Пример использования команды]

Блокировать запись

Назначение

Команда LROW блокирует записи одной или нескольких таблиц или обновляемого представления.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], NULL);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"LROW"
Node	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
SysErr	Код состояния ОС

Описание

Команда LROW блокирует **текущую** запись таблицы (обновляемого представления). Текущей записью в канале является последняя выбранная или измененная запись некоторой таблицы либо обновляемого представления. Таким образом, команда LROW имеет смысл только в контексте следующих запросов:

- SELECT;
- SELECT FOR UPDATE;
- UPDATE.

Примечание
При невыполнении данных условий поведение СУБД не определено.

При выполнении LROW вне указанного контекста она игнорируется (в этом случае код завершения команды будет NORMAL, как и при нормальном завершении).

Выполнение LROW не приводит к изменению положения текущей записи таблицы (представления).

При задании модификатора FOR UPDATE все записи обновляемой таблицы, вошедшие в выборку, сразу же блокируются от изменения параллельно работающими пользователями БД (каналами СУБД). Такие записи можно читать (для них задается внутренняя блокировка). На каждую запись может быть наложена только одна блокировка FOR UPDATE. Если число записей в выборке данных превышает 1000, то в этом случае блокируется вся таблица. Блокировка действует до обработки запроса COMMIT/ROLLBACK.

По одному каналу можно явно блокировать только одну запись; при блокировании другой записи предыдущая блокировка автоматически сбрасывается.

При завершении транзакции (**COMMIT** или **ROLLBACK**) установленные блокировки сбрасываются.

При выполнении по каналу оператора определения данных установленные в нем блокировки сбрасываются.

Идеология СУБД ЛИНТЕР подразумевает, что запросы подаются не только от имени пользователя, но и от имени приложения, и даже нити приложения. Поэтому внутри нити, блокировавшей запись, сама блокировка ощущаться не будет. Однако для других приложений и даже для других нитей того же приложения запись станет блокированной.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
Row_Locked	Запись блокирована другим приложением или другой нитью приложения

Пример формирования команды

```
#include <string.h>
```

```
#include <stdlib.h>
#include "inter.h"
L_LONG LinterLROW(TCBL *pCBL)
   {
   memcpy(pCBL->Command, "LROW", 4);
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, NULL);
   return pCBL->CodErr;
   }
```

Пример использования команды

Разблокировать запись

Назначение

Команда UROW разблокирует ранее заблокированную запись.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], NULL);
```

Входные данные

Входными данными является контрольный блок CBL.

В нем должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"UROW"
Node	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
SysErr	Код состояния ОС

Описание

Команда пытается разблокировать последнюю выбранную по каналу (по которому подана сама команда) запись.

Разблокировать можно только ранее блокированную по данному каналу запись. При отсутствии в канале блокированных записей UROW игнорируется (в этом случае код завершения команды будет NORMAL, как и при нормальном завершении).

Между командами LROW и UROW для обновления заблокированной записи можно использовать только одну команду **UPDATE OF CURRENT**.

Разблокирование записи не приводит к изменению текущей записи канала – текущей остается та запись, которая была до выполнения UROW.

При завершении транзакции (**COMMIT** или **ROLLBACK**) установленные блокировки сбрасываются.

При выполнении по каналу оператора определения данных (типа **CREATE**…) установленные в нем блокировки сбрасываются.

При нормальном или аварийном закрытии канала блокировка записи сбрасывается.

При завершении работы ядра СУБД все блокировки сбрасываются.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterUROW(TCBL *pCBL)
   {
   memcpy(pCBL->Command, "UROW", 4);
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, NULL);
   return pCBL->CodErr;
   }
```

Пример использования команды

Команды управления транзакциями

Фиксировать изменения

Назначение

Команда COMT предназначена для сохранения (фиксации) в БД всех изменений, сделанных приложением во время текущей транзакции.

Команда COMT эквивалентна SQL-запросу COMMIT; (см. документ «Справочник по SQL», пункт «Сохранение изменений»).

Параметры вызова

```
inter(CBL, {VarBuf | NULL}, NULL, [CondBuf], NULL);
```

Входные данные

Входными данными являются:

- контрольный блок CBL;
- буфер параметров VarBuf (для транзакции с контрольными точками). В противном случае значение параметра должно быть NULL.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"COMT"
Node	Имя ЛИНТЕР-сервера

Буфер параметров VarBuf используется при обработке команды COMT для транзакции с контрольными точками и должен содержать информацию о месте в транзакции, с которого необходимо выполнить фиксацию изменений в БД:

```
typedef struct
{
   L_BYTE    SavePoint;
   L_BYTE    Named;
   L_CHAR    Name[MAX_ID_LEN];
}  SAVE_POINT;
```

Описание полей структуры SAVE_POINT приведено в таблице 11.

Таблица 11. Описание полей структуры данных SAVE_POINT

Поле	Описание
SavePoint	Режим обработки контрольных точек: 0 – игнорировать контрольные точки (если они есть); 1 – учитывать наличие контрольных точек.
Named	Тип контрольной точки: 0 – неименованная; 1 – именованная.
Name	Имя (регистрозависимое) контрольной точки (в случае именованной точки). Должно быть заполнено пробелами до длины MAX_ID_LEN.

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
SysErr	Код состояния ОС

Описание

После выполнения команды COMT текущая транзакция завершается с сохранением изменений в БД.

Изменения, проводимые приложением в таблицах БД, становятся доступны всем пользователям (клиентским приложениям), имеющим доступ к таблицам, задействованным в данной транзакции.

Если клиентское приложение имеет главный и подчиненные каналы и команда COMT подается по главному каналу, то `COMT` автоматически выполняется и для всех транзакций в подчиненных каналах.

Если у канала есть подчиненные каналы, то подача `COMT` по главному каналу должна выполняться после завершения всех запросов в подчиненных каналах, в противном случае поведение СУБД ЛИНТЕР не специфицировано.

При наличии в канале выборки данных и подачи команды COMT по этому каналу выборка данных будет закрыта только при запуске ядра СУБД в режиме /COMPATIBILITY=STANDARD.

Правила выполнения команды (таблица 12):

- если VarBuf == NULL или поле SavePoint == 0, то команда выполняется без учета контрольных точек;
- если поле SavePoint == 1, то фиксация изменений выполняется до контрольной точки;
- если Named == 0, то фиксация изменений выполняется до последней контрольной точки;
- если Named == 1, то фиксация изменений выполняется до контрольной точки, имя которой задано в поле Name.

Контрольная точка, соответствующая началу транзакции, должна ставиться сразу после команды `COMT/RBAC`, а не до неё.

Например, если необходимо обращаться к контрольной точке SP3:

```
Err=LinterCOMT(&CBLconnect);
Err=LinterNotSelect(&CBLconnect, "set savepoint SP3;");
```

При нижеследующем порядке установки контрольной точки она будет недоступна:

```
Err=LinterNotSelect(&CBLconnect, "set savepoint SP3;");
Err=LinterCOMT(&CBLconnect);
```

При возникновении ошибки приложение должно откатывать всю транзакцию. После выполнения команды `COMT` до некоторой контрольной точки все более ранние контрольные точки автоматически удаляются.

Пример

```
      «действия 1»

SET SAVEPOINT SP1

      «действия 2»

COMT (SP1)
```

«действия 1» зафиксированы в БД, контрольная точка SP1 удалена,

«действия 2» пока не зафиксированы в БД, но если подать COMT –

тоже зафиксируются в БД, транзакцию можно продолжать.

Таблица 12. Алгоритм фиксации изменений при наличии контрольных точек

Значение поля			Результат
SavePoint	Named	Name	
0	не важно	не важно	Фиксация до точки SP3
1	0	не важно	Фиксация до точки SP3
1	1	SP3	Фиксация до точки SP3
1	1	"	Ошибка
1	1	SP2	Ошибки нет, но COMMIT не выполняется

Коды завершения

Код	Описание
NORMAL	Нормальное завершение (изменения зафиксированы в БД)
ILLTRANS	Неправильная транзакция. Эта ситуация означает, что все или часть данных, измененные приложением в текущей транзакции, были уже изменены некоторой другой транзакцией, и эти действия зафиксированы в БД. Для текущей транзакции выполняется откат

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterCOMT(TCBL *pCBL)
   {
   memcpy(pCBL->Command, "COMT", 4);
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, NULL);
   return pCBL->CodErr;
   }
```

Пример использования команды

Откатить изменения

Назначение

Команда RBAC предназначена для отмены (отката) всех изменений, выполненных текущей транзакцией.

Примечание
Команда RBAC эквивалентна SQL-запросу ROLLBACK; (см. документ «Справочник по SQL», пункт «Отмена изменений»).

Параметры вызова

```
inter(CBL, {VarBuf | NULL}, NULL, [CondBuf], NULL);
```

Входные данные

Входными данными являются:

- контрольный блок CBL;
- буфер параметров VarBuf (для транзакции с контрольными точками). В противном случае значение параметра должно быть NULL.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
NumChan	Номер канала
Command	"RBAC"
Node	Имя ЛИНТЕР-сервера

Буфер параметров VarBuf используется при обработке команды RBAC для транзакции с контрольными точками и должен содержать информацию о месте в транзакции, с которого необходимо выполнить откат транзакции (см. структуру SAVE_POINT в описании команды COMT).

Выходные данные

Выходными данными является контрольный блок CBL.

В нем будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
SysErr	Код состояния ОС

Описание

После выполнения команды RBAC текущая транзакция завершается без сохранения изменений.

Если клиентское приложение имеет главный и подчиненные каналы и команда RBAC подается по главному каналу, то RBAC автоматически выполняется и для всех транзакций в подчиненных каналах.

Если перед аварийным закрытием канала (KILL) в нем имеется незавершенная транзакция, то по умолчанию она завершается командой RBAC.

Если в транзакции были установлены контрольные точки (SAVEPOINT) (см. документ «Справочник по SQL», пункт «Создание точки сохранения»), и фиксация изменений в них не производилась, то по команде RBAC выполняется откат всей транзакции.

Если в транзакции были установлены контрольные точки (например, SP1, SP2, SP3), и в некоторых из них выполнялась фиксация изменений (т.е. устанавливалось новое начало текущей транзакции, например, SP3), то команда RBAC выполняется в соответствии с алгоритмом, приведенным в таблице 13.

После выполнения команды RBAC до некоторой контрольной точки автоматически удаляются все более поздние контрольные точки.

При наличии в канале выборки данных и подачи команды RBAC по этому каналу выборка данных будет закрыта только в случае запуска ядра СУБД в режиме /COMPATIBILITY=STANDARD.

Пример

```
      «действия 1»
…
SET SAVEPOINT SP1
      «действия 2»
RBAC (SP1)
```

«действия 2» были отменены в БД (и уже не могут быть восстановлены),

«действия 1» пока не зафиксированы в БД, но если подать команду COMT

– будут зафиксированы в БД и транзакцию можно продолжать.

Таблица 13. Алгоритм отката изменений при наличии контрольных точек

Значение поля			Результат
SavePoint	**Named**	**Name**	
0	не важно	не важно	Откат до точки SP3
1	0	не важно	Откат до точки SP3
1	1	SP3	Откат до точки SP3
1	1	"	Ошибка
1	1	SP2	Ошибки нет, но ROLLBACK не выполняется

Коды завершения

Код	Описание
NORMAL	Нормальное завершение (выполнен откат изменений)
ERRMODE	Использование команды в недопустимом режиме. RBAC можно использовать в любом режиме, кроме **AUTOCOMMIT**

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"
L_LONG LinterRBAC(TCBL *pCBL)
   {
   memcpy(pCBL->Command, "RBAC", 4);
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, NULL);
   return pCBL->CodErr;
   }
```

Пример использования команды

В примере показано использование:

- команд обработки непоисковых запросов;
- команд управления транзакциями;
- команд управления доступом к данным.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#ifndef WINCE
int    main()
#else
int    exnosel()
#endif
   {
   TCBL  CBLconnect;
   L_CHAR  Name_Pass[]="SYSTEM/MANAGER8";
   L_CHAR  Node[]="    ";
   L_WORD  Priority=0;
   L_LONG PrzExe=M_EXCLUSIVE | Q_ENCODE | M_BINARY;
   L_LONG Err;

   memset(&CBLconnect,0,sizeof(TCBL));
   Err=LinterOPEN(&CBLconnect, Name_Pass, Node, Priority, PrzExe);
   if (Err != NORMAL)
     PrintError(&CBLconnect);
   printf("Connect to RDBMS Linter\n");

   Err=LinterNotSelect(&CBLconnect,
        "insert into PERSON(PERSONID)
   values(10000);");
   if (Err != NORMAL)
     PrintError(&CBLconnect);
   printf("insert into PERSON\n");

   Err=LinterRBAC(&CBLconnect);
   if (Err != NORMAL)
     PrintError(&CBLconnect);
   printf("RollBack\n");

   Err=LinterNotSelect(&CBLconnect,
        "insert into PERSON(PERSONID)
   values(10000);");
```

```
if (Err != NORMAL)
  PrintError(&CBLconnect);
printf("insert into PERSON\n");

Err=LinterNotSelect(&CBLconnect,
     "delete from PERSON where
PERSONID=10000;");
if (Err != NORMAL)
PrintError(&CBLconnect);
printf("delete from PERSON\n");

Err=LinterCOMT(&CBLconnect);
if (Err != NORMAL)
  PrintError(&CBLconnect);
printf("Commit\n");

Err=LinterLREL(&CBLconnect, "SYSTEM", "PERSON");
if (Err != NORMAL)
  PrintError(&CBLconnect);
printf("LREL\n");

Err=LinterUREL(&CBLconnect);
if (Err != NORMAL)
  PrintError(&CBLconnect);
printf("UREL\n");

Err=LinterLROW(&CBLconnect);
if (Err != NORMAL)
  PrintError(&CBLconnect);
printf("LROW\n");

Err=LinterUROW(&CBLconnect);
if (Err != NORMAL)
  PrintError(&CBLconnect);
printf("UROW\n");

printf("End Example\n");

return 0;
}
```

Команды мониторинга СУБД

Используя команды интерфейса нижнего уровня, приложение может не только описывать и изменять структуры баз данных, манипулировать данными (находить, добавлять, менять, удалять), но и получать сведения о процессах, проходящих в СУБД. Эта информация нужна для оптимальной настройки функционирования ядра СУБД.

Одной из важных задач администратора СУБД является настройка производительности СУБД с учетом особенностей взаимодействующих с ней прикладных систем. Производительность СУБД зависит от многих факторов: от используемой сервером СУБД операционной системы, от выделенных системных ресурсов, от параметров настройки ядра СУБД, от структуры запросов клиентских приложений к БД (например, преимущественное использование простых запросов, претранслированных, хранимых процедур, триггеров) и др. Все эти факторы оказывают разное влияние на возможность изменения производительности СУБД. Так, после того, как операционная система для сервера СУБД выбрана, производительность СУБД будет зависеть от других факторов и, в первую очередь, от параметров настройки ядра СУБД. Единственным способом узнать, насколько эффективно функционирует сервер СУБД, является отслеживание его операций, т.е. мониторинг состояния и производительности СУБД в целом. Мониторинг может выполняться с помощью специальных программ, при разработке которых рекомендуется использовать команды интерфейса нижнего уровня СУБД ЛИНТЕР, специально предназначенные для этих целей. Имеющийся набор команд позволяет собрать статистику о том, как работает прикладная система и как СУБД ЛИНТЕР обрабатывает ее запросы. Это особенно важно для многокомпонентных приложений, работающих в многозадачной среде, когда запросы одного из компонентов приложения могут задерживать обработку запросов других подсистем. При этом общая производительность прикладной системы снижается. Собрав и изучив статистику, администратор СУБД может выполнить тонкую настройку производительности системы.

Для мониторинга СУБД используется следующий набор команд интерфейса нижнего уровня:

- DIRR – дать элемент очереди таблиц;
- DIRA – дать элемент очереди столбцов;
- DIRF – дать элемент очереди файлов.

Каждая из этих команд предоставляет информацию об основных очередях ядра СУБД ЛИНТЕР. Свопинг этих очередей оказывает большое влияние на скорость обработки потока запросов.

Команды мониторинга СУБД являются `неканальными` командами, т.е. могут подаваться до установления клиентским приложением соединения с каким-либо ЛИНТЕР-сервером.

Информация о какой именно СУБД будет предоставлена зависит от используемого ЛИНТЕР-сервера. В случае удаленного ЛИНТЕР-сервера его местоположение прописывается в сетевом драйвере клиента. В случае локальной СУБД используется ЛИНТЕР-сервер по умолчанию.

Общая информация об очередях

Информация о размерах очередей располагается в первой строке системной таблицы `$$$SYSRL`. Здесь же хранится и другая системная информация (имя БД, имена устройств рабочих файлов БД и т.п.).

Эту информацию можно получить (при наличии доступа на чтение к таблице `$$$SYSRL`) по следующему запросу:

```
SELECT GETSTR($$$s14,0,18), /* Имя БД */
GETWORD($$$s14,18),      /* Размер очереди таблиц */
GETWORD($$$s14,20),      /* Размер очереди столбцов */
```

```
GETWORD($$$s14,22),       /* Размер очереди файлов */
GETWORD($$$s14,24),       /* Размер очереди каналов */
GETLONG($$$s14,34),       /* Квант обработки при работе по RowId */
GETLONG($$$s14,38),       /* Квант обработки при работе по индексам
 */
GETSTR($$$s14,26,4),      /* Лог. имя устройства файла векторов */
GETSTR($$$s14,30,4),      /* Лог. имя устройства рабочего файла */
GETSTR($$$s14,46,4),      /* Лог. имя устройства файла сортировки
 */
GETSTR($$$s14,50,4)       /* Лог. имя устройства файла сист.
 журнала */
FROM $$$sysrl WHERE ROWID=1;
```

Дать элемент очереди таблиц

Назначение

Команда DIRR предназначена для получения информации об указанном элементе очереди таблиц.

Параметры вызова

```
inter(CBL, NULL, [OpBuf], [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок CBL;
- буфер SQL-запросов OpBuf.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
Command	"DIRR"
RowId	Порядковый номер элемента очереди таблиц (отсчет начинается с 1)
LnBufRow	Длина буфера выборки данных
Node	Имя ЛИНТЕР-сервера

Буфер SQL-запросов OpBuf может содержать имя устанавливаемой для данного канала кодовой страницы, которая должна быть известна БД, то есть:

- находиться в системной таблице LINTER_SYSTEM_USER.$$$CHARSET;
- или быть описана как синоним в таблице LINTER_SYSTEM_USER.$$$CSALIAS;
- или быть предопределённым значением UTF-8 или UCS2.

Выходные данные

Выходными данными являются:

- контрольный блок CBL;

- буфер выборки данных RowBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
LnBufRow	Фактическая длина буфера выборки данных (sizeof(DIRR_OUT))
SysErr	Код состояния ОС

В буфере выборки данных RowBuf информация будет представлена в следующем виде:

```
struct DIRR_OUT
   {
   L_LONG Owner;                /* Идентификатор владельца таблицы
 */
   L_CHAR TblName[MAX_ID_LEN]; /* Имя таблицы */
   L_WORD UpdFlag;              /* Признак корректировки таблицы */
   };
```

В поле UpdFlag возвращается значение 1, если в момент выдачи команды DIRR измененные данные соответствующей таблицы еще не были перенесены из системного пула в БД. После того как данные будут изменены в БД, этот признак сбрасывается. Поэтому для получения достоверной информации о частоте изменения данных периодичность опроса очереди таблиц должна быть не больше периодичности обновления информации в БД.

С помощью команды DIRR может быть получена следующая информация:

- о частоте обращения к таблицам БД (по частоте присутствия данного элемента в очереди таблиц);
- об интенсивности обращения к таблицам (чем меньше порядковый номер элемента очереди таблиц, тем чаще происходит обращение к данной таблице);
- о частоте корректировки данных в таблицах (путем суммирования значений поля UpdFlag);
- о влиянии установленного размера очереди таблиц на производительность СУБД (по частоте вытеснения и повторного включения элементов в очередь).

Описание

Первые три элемента очереди таблиц всегда используются для системных таблиц.

Новый элемент записывается на первое свободное место в очереди, а в случае отсутствия такового на место самого «старого» элемента очереди. Если при этом замещаемый элемент предназначен к записи, то он записывается, и только после этого его место занимает новый элемент. После этого очередь перестраивается таким образом, что новый элемент становится первым.

СУБД ЛИНТЕР ведет очереди, не меняя положения элемента очереди в оперативной памяти. Поэтому, если при двух обращениях к элементу с тем же самым порядковым номером были возвращены разные данные, то между двумя этими обращениями СУБД

осуществила запись этого элемента очереди на диск, а на освободившееся место была записана новая информация.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Элемент очереди с заданным порядковым номером не найден

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterDIRR(TCBL *pCBL, L_LONG N, DIRR_OUT *Out)
   {
   memcpy(pCBL->Command, "DIRR", 4);
   pCBL->LnBufRow=sizeof(DIRR_OUT);
   pCBL->RowId=N;
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, Out);
   return pCBL->CodErr;
   }
```

Дать элемент очереди столбцов

Назначение

Команда DIRA предназначена для получения информации об указанном элементе очереди столбцов.

Параметры вызова

```
inter(CBL, NULL, [OpBuf], [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок CBL;
- буфер SQL-запросов OpBuf.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
Command	"DIRA"
RowId	Порядковый номер элемента очереди столбцов (отсчет начинается с 1)
LnBufRow	Длина буфера выборки данных

Имя поля	Значение
Node	Имя ЛИНТЕР-сервера

Буфер SQL-запросов OpBuf может содержать имя устанавливаемой для данного канала кодовой страницы, которая должна быть известна БД, то есть:

- находиться в системной таблице LINTER_SYSTEM_USER.$$$CHARSET;
- или быть описана как синоним в таблице LINTER_SYSTEM_USER.$$$CSALIAS;
- или быть предопределённым значением UTF-8 или UCS2.

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения запроса к СУБД ЛИНТЕР
LnBufRow	Фактическая длина буфера выборки данных (sizeof(DIRA_OUT))
SysErr	Код состояния ОС

В буфере выборки данных RowBuf информация будет представлена в следующем виде:

```
struct DIRA_OUT
   {
   L_LONG Owner;               /* Идентификатор владельца таблицы
 */
   L_CHAR TblName[MAX_ID_LEN]; /* Имя таблицы */
   L_CHAR ColName[MAX_ID_LEN]; /* Имя столбца */
   L_BYTE UpdFlag;             /* Признак корректировки столбца */
   L_BYTE Reserve[3];          /* Резерв */
   };
```

В полях буфера выборки данных будет содержаться:

Имя поля	Значение
Owner	Идентификатор владельца таблицы, столбец которой находится в указанном элементе очереди столбцов
TblName	Имя таблицы, содержащей столбец элемента
ColName	Имя столбца, находящегося в указанном элементе
UpdFlag	Возвращается значение 1, если в момент выдачи команды DIRA измененные данные соответствующего столбца еще не были перенесены из системного пула в БД. После того как данные будут изменены в БД, этот признак сбрасывается. Поэтому для получения достоверной информации о частоте изменения данных столбца периодичность опроса очереди столбцов должна быть не больше периодичности обновления информации в БД

С помощью команды DIRA может быть получена следующая информация:

- о частоте обращения к столбцу данной таблицы (по частоте присутствия данного элемента в очереди таблиц);
- об интенсивности обращения к столбцу таблицы (чем меньше порядковый номер элемента очереди столбцов, тем чаще происходит обращение к данному столбцу таблицы);
- о частоте корректировки данных в столбце таблицы (путем суммирования значений поля UpdFlag);
- о влиянии установленного размера очереди столбцов на производительность (по частоте вытеснения и повторного включения элементов в очередь).

Первые 13 элементов очереди столбцов заняты столбцами системных таблиц в порядке, приведенном в таблице 14.

Таблица 14. Первые 13 элементов очереди столбцов

№ элемента	Имя столбца	Имя таблицы
1	$$$S11	$$$SYSRL
2	$$$S12	$$$SYSRL
3	$$$S13	$$$SYSRL
4	$$$S14	$$$SYSRL
5	$$$S21	$$$ATTRI
6	$$$S22	$$$ATTRI
7	$$$S23	$$$ATTRI
8	$$$S24	$$$ATTRI
9	$$$S31	$$$USR
10	$$$S32	$$$USR
11	$$$S33	$$$USR
12	$$$S34	$$$USR
13	$$$S35	$$$USR

Описание

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Элемент очереди с заданным порядковым номером не найден

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"
```

```
L_LONG LinterDIRA(TCBL *pCBL, L_LONG N, DIRA_OUT *Out)
   {
   memcpy(pCBL->Command, "DIRA", 4);
   pCBL->LnBufRow=sizeof(DIRA_OUT);
   pCBL->RowId=N;
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, Out);
   return pCBL->CodErr;
   }
```

Дать элемент очереди файлов

Назначение

Команда DIRF предназначена для получения информации об указанном элементе очереди дескрипторов файлов.

Параметры вызова

```
inter(CBL, NULL, [Opbuf], [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок CBL;
- буфер SQL-запросов OpBuf.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
Command	"DIRF"
RowId	Порядковый номер элемента очереди файлов (отсчет начинается с 1)
LnBufRow	Длина буфера выборки данных (sizeof(DIRF_OUT))
Node	Имя ЛИНТЕР-сервера

Буфер SQL-запросов OpBuf может содержать имя устанавливаемой для данного канала кодовой страницы, которая должна быть известна БД, то есть:

- находиться в системной таблице LINTER_SYSTEM_USER.$$$CHARSET;
- или быть описана как синоним в таблице LINTER_SYSTEM_USER.$$$CSALIAS;
- или быть предопределённым значением UTF-8 или UCS2.

Выходные данные

Выходными данными являются:

- контрольный блок CBL;

- буфер выборки данных `RowBuf`.

В контрольном блоке будут возвращены:

Имя поля	Значение
`CodErr`	Код завершения запроса к СУБД ЛИНТЕР
`LnBufRow`	Фактическая длина буфера выборки данных
`SysErr`	Код состояния ОС

В буфере выборки данных `RowBuf` информация будет представлена в следующем виде:

```
struct DIRF_OUT
   {
   L_LONG Owner;                /* Идентификатор владельца таблицы
 */
   L_CHAR TblName[MAX_ID_LEN]; /* Имя таблицы */
   L_BYTE Type;                 /* Тип файла */
   L_BYTE Extent;               /* Номер файла */
   L_LONG State;                /* Признак состояния */
   };
```

В полях буфера выборки данных будет содержаться:

Имя поля	Значение
`Owner`	Идентификатор владельца таблицы, дескриптор файла которой находится в указанном элементе очереди дескрипторов файлов
`TblName`	Имя таблицы, дескриптор файла которой находится в указанном элементе очереди дескрипторов файлов
`Type`	Тип файла (таблица 15)
`Extent`	Логический номер файла данной таблицы. Это поле может принимать значение от 1 до 63, в зависимости от количества файлов, объявленных при создании таблицы
`State`	Текущее состояние элемента очереди: 1 – элемент занят дескриптором файла; 0 – элемент свободен.

Команда `DIRF` может быть использована для определения оптимального размера очереди дескрипторов файлов. Дескриптор файла, включенный в очередь, вытесняется из нее только в одном случае – отсутствие свободных элементов при включении в очередь нового элемента. Поэтому в процессе мониторинга БД можно выявить следующие ситуации:

1) часть элементов очереди файлов никогда не используется. Это говорит о том, что размер очереди слишком велик и его без ущерба для производительности БД можно уменьшить (примерно на количество неиспользуемых элементов);
2) в очереди файлов происходит интенсивное вытеснение и добавление элементов. Это говорит о недостаточном размере очереди файлов.

Изменяя размер очереди файлов (насколько позволят системные ресурсы) и выполняя мониторинг БД, можно добиться оптимального размера – когда элементы из очереди

не вытесняются, и в то же время количество неиспользуемых элементов очереди минимально.

Таблица 15. Коды файлов СУБД ЛИНТЕР

Обозначение	Значение	Файл
FT_INDEX	0	Файл индексов
FT_DATA	1	Файл данных
FT_SYSWBV	2	Рабочий файл бит-векторов
FT_SYSWRK	3	Рабочий файл
FT_SYSSRT	4	Рабочий файл сортировки
FT_SYSLOG	5	Системный журнал
FT_BLOB	7	BLOB-файл

Первые 10 элементов очереди файлов заняты дескрипторами системных файлов в порядке, приведенном в таблице 16.

Таблица 16. Первые 10 элементов очереди файлов

№ элемента	Имя файла	Комментарий
1	1.01	Файл индексов таблицы `$$$SYSRL`
2	1.11	Файл данных таблицы `$$$SYSRL`
3	2.01	Файл индексов таблицы `$$$ATTRI`
4	2.11	Файл данных таблицы `$$$ATTRI`
5	3.01	Файл индексов таблицы `$$$USR`
6	3.11	Файл данных таблицы `$$$USR`
7	1.41	Рабочий файл
8	1.31	Рабочий файл бит-векторов
9	1.51	Рабочий файл сортировки
10	Резерв	Файлы системного журнала

Описание

СУБД ЛИНТЕР ведет очереди, не меняя положения элемента очереди в оперативной памяти. Поэтому, если при двух обращениях к элементу с тем же самым порядковым номером были возвращены разные данные, то между двумя этими обращениями СУБД осуществила запись элемента очереди на диск, а на освободившееся место была записана новая информация.

Для файлов с кодами типа 2,3,4 (таблица 15) поля `Owner` и `TblName` будут заполнены нулем и пробелами соответственно.

Информация о файлах с кодом типа 5 (таблица 15) по команде `DIRF` не выдается.

Коды завершения

Код	Описание
`NORMAL`	Нормальное завершение

Код	Описание
SMALLBUFKOR	Недостаточный размер буфера выборки данных
EORR	Элемент очереди с заданным порядковым номером не найден

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterDIRF(TCBL *pCBL, L_LONG N, DIRF_OUT *Out)
   {
   memcpy(pCBL->Command, "DIRF", 4);
   pCBL->LnBufRow=sizeof(DIRF_OUT);
   pCBL->RowId=N;
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, Out);
   return pCBL->CodErr;
   }
```

Управление интерфейсом нижнего уровня

Средства интерфейса нижнего уровня предоставляют возможность управлять некоторыми параметрами самого интерфейса.

Назначение

Получить/Установить/Изменить параметры интерфейса.

Параметры вызова

```
inter_control(NULL, IC_code, OpBuf, LenOpBuf);
```

Входные данные

Входными данными являются:

- код операции управления интерфейсом `IC_code` (значение типа integer);
- буфер входных/выходных данных `OpBuf`;
- длина буфера входных/выходных данных `LenOpBuf`.

Параметр `IC_code` должен содержать код операции управления интерфейсом (значение типа integer, таблица 17). Определения кодов операций содержатся в файле `inter.h`.

Таблица 17. Коды операций управления интерфейсом

Код операции	Значение	Выполняемая операция
ICR_ENABLE_MSG	0	Разрешить/запретить выдачу диагностических сообщений на консоль пользовательского приложения
ICR_GET_CP	1	Получить описание кодовой страницы, используемой интерфейсом по умолчанию
ICR_SET_CP	2	Установить для интерфейса кодовую страницу по умолчанию
ICR_ADD_NODE	3	Зарезервировано
ICR_DEL_NODE	4	Зарезервировано
ICR_USE_OEM_CP	5	Только для ОС Windows. При попытке автоматического определения кодовой страницы работать в OEM-кодировке (кодовая страница CP866)
ICR_USE_UTF8	6	Передавать тексты SQL-запросов в коде UTF8
ICR_REREAD_ENV	7	Получить новое значение переменной окружения `LINTER_CP`
ICR_SET_STACK_SIZE	8	Размер стека процедуры асинхронной обработки запроса
ICR_FORCE_KRB	9	Инициировать идентификацию и аутентификацию с помощью Kerberos-сервера

Значения входных данных, в зависимости от типа операции, приведены в таблице 18.

Таблица 18. Значения входных данных операций управления интерфейсом

IC_code	Значение в OpBuf	LenOpBuf
ICR_ENABLE_MSG	0 – разрешение 1 – запрет	1
ICR_GET_CP	Структура данных типа `LCODEPAGE`	Размер структуры данных типа `LCODEPAGE`
ICR_SET_CP	Числовое значение кодировки, которое должно соответствовать полю WIN_CODE системной таблицы $$$CHARSET	CP_AS_NUM (значение 0)
	Строковое значение кодировки, которое должно соответствовать полю NAME системной таблицы $$$CHARSET	CP_AS_STR (значение 1)
ICR_ADD_NODE	–	Зарезервировано
ICR_DEL_NODE	–	Зарезервировано
ICR_REREAD_ENV	Значение переменной окружения `LINTER_CP`	Длина значения `LINTER_CP`
ICR_SET_STACK_SIZE	–	Размер стека процедуры асинхронной обработки запроса
ICR_FORCE_KRB	Глобальный флаг идентификации и аутентификации по Kerberos-протоколу (см. «Описание»)	1

Примечание
Если значение аргумента LenOpBuf равно 0xFFFF (т.е. -1), то независимо от значения аргумента OpBuf произойдет сброс установленного значения кодовой страницы по умолчанию. Следующая команда, использующая кодовую страницу (например, OPEN) произведет установку кодовой страницы по умолчанию в соответствии с параметрами ОС.

Структура LCODEPAGE:

```
typedef struct
{
   L_WORD cp_type;
   L_WORD cp_len;
   L_WORD pad;
   union
   {
   L_WORD n_cp;
   L_CHAR s_cp[MAX_ID_LEN];
   } cp;
} LCODEPAGE;
```

Описание полей структуры `LCODEPAGE`:

Имя поля	Значение
`cp_type`	Способ указания кодовой страницы:

Имя поля	Значение
	CP_AS_NUM (значение 0) – по номеру (866,1251 и т.п.); CP_AS_STR (значение 1) – по имени ('CP866', 'KOI8-R' т.п.). Определения CP_AS_NUM, CP_AS_STR находятся в файле `inter.h`
`cp_len`	Длина имени кодировки (если тип кодировки CP_AS_STR)
`pad`	Выравнивание
`n_cp`	Номер кодировки (в случае CP_AS_NUM)
`s_cp`	Имя кодировки (в случае CP_AS_STR)

Примечание
Список доступных кодовых страниц находится в таблице `$$$CHARSET`.

Выходные данные

Выходными данными является буфер входных/выходных данных `OpBuf` (для операции ICR_GET_CP).

Описание

В случае если кодировка по умолчанию для интерфейса не была установлена с помощью операции `ICR_SET_CP`, то для её определения используется переменная окружения `LINTER_CP`. Если переменная `LINTER_CP` не установлена, то интерфейс пытается сделать кодировкой по умолчанию текущую кодировку ОС. Если такой кодировки нет, то фиксируется код завершения NOCSETQUE, и устанавливается специфическая для каждой ОС встроенная в интерфейс кодировка по умолчанию.

Например, для ОС Windows, имеющей две кодировки (ANSI и OEM), автоматически устанавливается кодировка OEM, если этот выбор задан с помощью операции `ICR_USE_OEM_CP`, в противном случае используется ANSI.

Флаг `ICR_FORCE_KRB` задает безусловный режим идентификации и аутентификации приложения по Kerberos-протоколу.

Допустимые значения параметра команды `ICR_FORCE_KRB`:

- 1 – установить глобальный флаг идентификации и аутентификации по Kerberos-протоколу. Это означает, что аутентификация всех последующих соединений с СУБД ЛИНТЕР в данном клиентском приложении будет выполняться с помощью Kerberos-протокола;
- -1 – необходимость идентификации и аутентификации по Kerberos-протоколу проверяется по следующему алгоритму:
  - при первом после запуска клиентского приложения выполнении любой команды нижнего интерфейса проверяется наличие переменной окружения `LINTER_KRB`;
  - если переменная окружения `LINTER_KRB` определена, то устанавливается глобальный флаг идентификации и аутентификации по Kerberos-протоколу;
  - если переменная окружения `LINTER_KRB` не определена, то глобальный флаг идентификации и аутентификации по Kerberos-протоколу сбрасывается.

Таким образом, можно управлять режимом идентификации и аутентификации клиентского приложения в целом без изменения исходного кода. Для поддержки клиентским приложением идентификации и аутентификации по Kerberos-протоколу

необходимо только перекомпоновать программу с соответствующей версией библиотеки нижнего уровня (объектного файла) `intlib`.

Коды завершения

Код	Описание
`0`	Нормальное завершение
`1`	Ошибка выполнения

Работа с хранимыми процедурами и триггерами

Создание (модификация) хранимой процедуры (триггера)

Назначение

Создание в БД новой хранимой процедуры (триггера) или модификация текста существующей хранимой процедуры (триггера).

Параметры вызова

```
inter(CBL, NULL, OpBuf, [CondBuf], NULL);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;
- буфер оператора `OpBuf`.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`NumChan`	Номер канала
`Command`	`4 пробела`
`Node`	Имя ЛИНТЕР-сервера

Буфер оператора `OpBuf` должен содержать SQL-выражение создания или модификации хранимой процедуры либо триггера.

Выходные данные

Выходными данными является контрольный блок `CBL`.

В нем будут возвращены:

Имя поля	Значение
`CodErr`	Код завершения запроса к СУБД ЛИНТЕР
`SysErr`	Код состояния ОС

Описание

Для выполнения команды в БД должны быть соответствующие системные таблицы (см. документ «Создание и конфигурирование базы данных», пункт «Создание БД»).

Исходный текст процедуры сохраняется в БД перед каждой трансляцией процедуры (исходный код триггера записывается только при отсутствии ошибок трансляции),

оттранслированный код – только при отсутствии ошибок трансляции, поэтому при первой трансляции процедуры (триггера) необходимо использовать SQL-оператор **CREATE PROCEDURE**…, при всех последующих трансляциях – **ALTER PROCEDURE**….

Поле `NumChan` блока `CBL` должно ссылаться на открытый канал.

Максимальная длина тела процедуры (триггера) 64 Кбайта.

Если процедура (триггер) оттранслирована без ошибок и в БД нет процедуры (триггера) с аналогичным именем, то она сохраняется в БД.

Владельцем созданной процедуры (триггера) является пользователь, создавший процедуру (триггер).

Право на модификацию процедуры имеет только ее владелец.

Если в момент модификации процедуры она используется некоторыми другими приложениями, то может возникнуть рассогласование кода оттранслированной процедуры (часть кода была выполнена из старой версии, а часть кода будет выполнена уже из новой версии процедуры).

В любой момент времени может транслироваться только одна процедура (триггер), поэтому, если одновременно поступило несколько команд на трансляцию, то они выполнятся в порядке очередности.

Если в момент трансляции процедуры (триггера) осуществляется аварийное закрытие канала (команда `KILL`), то результаты трансляции будут непредсказуемы.

Коды завершения

Код	Описание
`NORMAL`	Нормальное завершение
`Proc_Translation_Error`	Ошибка трансляции процедуры (триггера)
`Proc_Table_Is_Absent`	Таблица процедур `$$$PROC` отсутствует в БД
`Trigger_Table_Is_Absent`	Таблица триггеров `$$$TRIG` отсутствует в БД
`ERRPASSWORD`	Нарушение привилегий
`NOTSP`	Транслятор хранимых процедур не активен или в данный момент занят трансляцией другой процедуры

Пример работы с хранимой процедурой

См. приложение 12.

Выполнение хранимой процедуры

Назначение

Выполнение предварительно оттранслированной и сохраненной в БД пользовательской процедуры.

Параметры вызова

```
inter(CBL, NULL, OpBuf, [CondBuf], RowBuf);
```

Входные данные

Входными данными являются:

- контрольный блок `CBL`;
- буфер оператора OpBuf.

В контрольном блоке должны быть заполнены поля:

Имя поля	Значение
`NumChan`	Номер канала
`Command`	`4 пробела`
`LnBufRow`	Длина буфера выборки данных
`Node`	Имя ЛИНТЕР-сервера

Буфер оператора `OpBuf` должен содержать SQL-выражение исполнения процедуры:

```
EXECUTE <имя процедуры>(<значения параметров>);
```

<значения параметров> представляют собой совокупность допустимых в SQL выражений, разделенных запятой. Некоторые из этих выражений могут отсутствовать (помечаются запятой). В случае пропуска фактических параметров и/или непривязке значений к параметрам формальные параметры получают значения по умолчанию (если для них заданы значения по умолчанию), либо NULL-значения в противном случае по умолчанию так же, как при вызове хранимой процедуры изнутри другой хранимой процедуры.

Для передачи логических значений используются целые числа (0 интерпретируется как `FALSE`, 1 – как `TRUE`) или символьные константы `'true'` и `'false'` (в любом регистре).

Выходные данные

Выходными данными являются:

- контрольный блок `CBL`;
- буфер выборки данных `RowBuf`.

В контрольном блоке будут возвращены:

Имя поля	Значение
`CodErr`	Код завершения запроса к СУБД ЛИНТЕР
`SysErr`	Код состояния ОС

Результат выполнения процедуры возвращается в буфере `RowBuf` в двоичном представлении в виде специальной структуры, которая содержит последовательно:

1) описатель возвращаемого значение (Result);

2) количество выходных параметров (2 байта);
3) описатели выходных параметров (если процедура создана с отладочной информацией, возвращаются также имена параметров);
4) буфер возвращаемого значения и значений выходных параметров.

Описатель возвращаемого значения и описатель выходного параметра имеют следующую структуру:

```
struct ARGPROC_OUT
   {
   L_BYTE Flags;      /* Флаги значения */
   L_BYTE Reserv;     /* Зарезервировано */
#if _VER_MAX >= 550
   L_WORD Expr;       /* Идентификатор выражения */
#endif
   L_WORD Value;      /* Смещение значения в буфере */
   P_TYPE Type;       /* Тип значения */
#if _VER_MAX >= 550
   L_WORD Reserv2;    /* Зарезервировано */
#endif
   };
```

Описатель возвращаемого значения (выходного параметра) содержит следующие поля:

Имя поля	Значение
Flags	Битовая маска атрибутов параметра: • PARM_FLAG_ISNULL – признак NULL-значения; • PARM_FLAG_ISCURSOR – признак курсора; • PARM_FLAG_NAMESENT – признак наличия имени параметра в буфере выборки данных `RowBuf`.
Expr	Зарезервировано для будущего использования.
Value	Позиция (смещение) данного возвращаемого значения в двоичном буфере выборки данных относительно его начала. Представление самого значения имеет следующий формат: • если для значения установлен признак наличия имени параметра, то по указанному смещению сначала находится имя: два байта длины и соответствующее количество символов. Далее следует само значение. Если признак наличия имени параметра не установлен, значение находится сразу по указанному смещению; • для типов данных CHAR и BYTE хранится соответствующее количество символов согласно описанию типа; • для целочисленных типов данных SMALLINT и INT хранятся два или четыре байта значения соответственно; • для вещественных чисел REAL и DOUBLE – 4 и 8 байт соответственно; • значения типов данных NUMERIC и DATE представляются в формате DECIMAL СУБД ЛИНТЕР – по 16 байт;

Имя поля	Значение
	• значение типа CURSOR содержит всю необходимую информацию для заполнения контрольного блока CBL, используемого для навигации по выборке с помощью соответствующих команд интерфейса нижнего уровня. Информация выдается в следующем виде:

```
struct CR_INFO
 {
 L_WORD NumChan;  /* Номер канала */
 L_WORD LnBufRow; /* Размер одной записи в байтах */
 L_LONG RowId;    /* Текущий ROWID в выборке */
 L_LONG RowCount; /* Количество записей в выборке */
 L_LONG CodErr;   /* Текущий код завершения */
 };
```

Type — Описатель типа данных параметра, который имеет следующую структуру:

```
struct P_TYPE
 {
 L_WORD length;  /* Длина типа данных */
 L_BYTE ntyp;    /* Код типа данных */
 L_BYTE prcs;    /* Точность */
 L_BYTE scale;   /* Масштаб */
 L_BYTE reserv;  /* Резерв */
 #if _VER_MAX >= 600
 L_WORD charset; /* Идентификатор кодовой стр-цы */
 #endif
 };
```

Описание

Выполнение хранимых процедур осуществляется последовательно в режиме квантования времени.

Допускается рекурсивный вызов процедуры. Глубина рекурсии ограничивается предоставленными ядру СУБД ЛИНТЕР ресурсами (размерами дисковой и оперативной памяти).

В рамках одной транзакции первая запущенная на выполнение хранимая процедура открывает для своей работы дочерний канал. Все последующие процедуры, вызываемые в рамках той же транзакции, переиспользуют этот канал, не открывая новых. Поэтому команды **COMMIT/ROLLBACK** в процедуре влияют не только на изменения, сделанные данной процедурой, но и на все изменения, осуществленные всеми вызванными ранее процедурами. Чтобы избежать этого, процедура должна использовать контрольные точки **SAVEPOINT**: установить точку сохранения в начале транзакции и подавать **COMMIT/ROLLBACK** до нее.

Примечание

Поскольку триггеры выполняются точно так же, как процедуры, все сказанное верно и для них. Кроме того, в контексте триггеров это дает еще один полезный эффект, поскольку все изменения совершаются триггерами по одному общему каналу, в случае нарушения

логики работы, обнаруженного триггером, он может подать **ROLLBACK**, откатывающий изменения всех вызванных ранее триггеров и обеспечивающий целостность в рамках запроса.

Поскольку триггерные транзакции выполняются точно так же, как и процедурные, то все сказанное о процедурной транзакции верно и для триггерных. Кроме того, в контексте триггеров такой механизм обработки транзакций дает полезный эффект: поскольку все изменения совершаются триггерами по одному общему каналу, то в случае нарушения логики работы, обнаруженной триггером, он может подать команду **ROLLBACK**, откатывающую изменения всех вызванных ранее триггеров и обеспечивающую целостность БД в рамках текущего запроса обработки данных.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных
Proc_Raised_Exception	Процедура завершилась с исключением. В этом случае поле SysErr контрольного блока будет содержать код исключения (возможно положительное или отрицательное значение). Положительное значение формируется ядром СУБД ЛИНТЕР при обработке SQL-запросов, входящих в тело процедуры. Отрицательное значение формируется системой исполнения хранимых процедур (приложение 13)
Proc_Not_Translated	Процедура не была оттранслирована (оттранслирована с ошибкой)
Proc_No_Memory	Недостаточно памяти для исполнения процедуры
Proc_Bad_Code	Ошибка в оттранслированном коде процедуры
Proc_Inv_Version	Неправильная версия оттранслированного кода процедуры (необходимо выполнить SQL-оператор **ALTER PROCEDURE**)
ERRPASSWORD	Нарушение привилегий

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterExec(TCBL *pCBL, L_CHAR *ExecStat, void *Buf, L_WORD
 LenBuf)

   {
   memcpy(pCBL->Command, "  ", 4);
   pCBL->LnBufRow=LenBuf;
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, ExecStat, NULL, Buf);
   return pCBL->CodErr;
```



```
}
```

Пример использования команды

См. приложение 14.

Асинхронная обработка запросов

Средства интерфейса нижнего уровня обеспечивают возможность асинхронной обработки запросов СУБД ЛИНТЕР. Для этого при обращении к СУБД необходимо установить бит Q_ASYNC (признак асинхронного выполнения запроса) в поле `PrzExe` контрольного блока обработки запроса `CBL` и, если необходимо, адрес процедуры завершения асинхронной обработки, которой СУБД передаст результаты выполнения асинхронного запроса. После отправки асинхронного запроса к СУБД управление сразу возвращается клиентскому приложению, которое может продолжать свою работу. После выполнения ядром СУБД асинхронного запроса (а это, в зависимости от сложности запроса, может быть длительный процесс) работа клиентского приложения прерывается, управление передается пользовательской процедуре завершения обработки асинхронного запроса (если она была задана) и после ее завершения вновь возвращается клиентскому приложению.

Пользовательская процедура завершения асинхронной обработки должна иметь следующий прототип: `void Proc_Name(TCBL *CBL, void*RowBuf, void *VarBuf);`

В параметрах `CBL` (адрес контрольного блока запроса), `RowBuf` (адрес буфера для приема выборки данных) и `VarBuf` (адрес буфера для приема маски NULL-значений выборки данных) будут переданы результаты выполнения асинхронного запроса. Если какой-либо из параметров (это может быть либо `VarBuf`, либо `RowBuf`) не является выходным параметром асинхронного запроса, вместо его адреса рекомендуется передать NULL-значение.

Внутри пользовательской процедуры завершения асинхронной обработки можно также использовать асинхронные команды, однако нельзя использовать синхронные обращения к СУБД.

В многонитевых (многопотоковых) приложениях процедура асинхронной обработки выполняется в отдельной, специально созданной для этих целей, нити (потоке). В некоторых случаях для процедур асинхронной обработки может потребоваться увеличить размер стека. Для этого до первого вызова функции inter (или любой функции интерфейса) должна быть вызвана функция `inter_control`, в которой код операции `IC_code` должен быть установлен в `ICR_SET_STACK_SIZE`, а длина буфера данных `LenOpBuf` должна содержать нужный размер стека. Минимальный размер стека должен составлять MAX_CLIENT_BUFSIZE + 8192 байт. В случае необходимости это значение должно быть увеличено на размер данных в стеке процедуры асинхронной обработки запроса.

Примечания

1. Процедура асинхронной обработки для многонитевых (многопотоковых) приложений выполняется в специально созданной для этого нити (потоке). Интерфейс для ОС Windows всегда многонитевый. ОС Linux, ЗОСРВ Нейтрино может использовать как многонитевый (многопотоковый) интерфейс, так и однопотоковый. В последнем случае процедура асинхронной обработки выполняется из обработчика сигнала.
2. Для удобства пользователей в файле `inter.h`, входящем в состав дистрибутива СУБД ЛИНТЕР, определен прототип функции USR_PROC: `typedef void (*USR_PROC)(TCBL *, HPVOID, HPVOID);`

Клиентское приложение может синхронизировать свое выполнение с любым асинхронным запросом. Для этого необходимо вызвать функцию inter_wait_single: `L_BOOL inter_wait_single(TCBL *CBL, LONGINT Timeout);`

где:

- `CBL` – контрольный блок, использованный для подачи асинхронного запроса (для различных асинхронных запросов должны использоваться разные `CBL`);
- `Timeout` – предельное время ожидания завершения асинхронного запроса. Зарезервировано для будущего использования.

Если к моменту вызова функции inter_wait_single выполнение запроса (совместно с пользовательской функцией завершения обработки асинхронного запроса) уже завершилось, inter_wait_single вернет управление клиентскому приложению. По факту выполнения асинхронного запроса в поле `PrzExe` контрольного блока выставляется бит Q_ASYNCDONE.

Функция возвращает:

- L_TTRUE, если вызов был с правильным CBL (т.е. с тем CBL, с которым выполнялся соответствующий ей вызов inter);
- L_TFALSE, если вызов был с ошибочным CBL.

Примечание
Если функция `inter` использует асинхронную обработку, то соответствующая ей функция `inter_wait_single` должна ссылаться на тот же самый контрольный блок, что и функция `inter`.

Примеры формирования асинхронного запроса

1)

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

void  LinterASYNC_OPEN(TCBL *pCBL, L_CHAR *Name_Pass,
                       L_CHAR *Node, L_WORD Prior,
                       L_LONG PrzExe, USR_PROC func)
   {
   memcpy(pCBL->Command, "OPEN", 4);
   if (strlen(Node) > MAX_NODE_LEN)
     {
     pCBL->CodErr=SQLLONGID;
     func(pCBL, NULL, NULL);
     return;
     }
   memset(pCBL->Node, 0, MAX_NODE_LEN);
   memcpy(pCBL->Node, Node, strlen(Node));
   pCBL->PrzExe=PrzExe | Q_ASYNC;
   pCBL->Prior=Prior;
   inter(pCBL, Name_Pass, NULL, (void *) func, NULL);
```

```
   }
```

2)

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

void  LinterASYNC_SLCT(TCBL *pCBL, L_LONG PrzExe,
                       L_CHAR *Statement, void *RowBuf,
                       L_WORD RowBufLen, void *VarBuf, USR_PROC
 func)
   {
   memcpy(pCBL->Command, "SLCT", 4);
   pCBL->PrzExe=PrzExe | Q_ASYNC;
   pCBL->LnBufRow=RowBufLen;
   inter (pCBL, VarBuf, Statement, (void *) func, RowBuf);
   }
```

Примеры асинхронной обработки запроса

1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#ifndef WINCE
int    main()
#else
int    aexopen()
#endif
   {
   TCBL  CBLconnect;
   L_CHAR  Name_Pass[] = "SYSTEM/MANAGER8";
   L_CHAR  Node[] = "        ";
   L_WORD  Priority = 0;
   L_LONG PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;

   memset(&CBLconnect,0,sizeof(TCBL));
   LinterASYNC_OPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe, WorkError);
   printf("Connecting to RDBMS Linter ... ");
   fflush(stdout);

   /* ... */
```



```
   inter_wait_single(&CBLconnect, -1);
   printf("Ok\n");

   printf("End Example\n");

   return 0;
   }
```

2)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#ifndef WINCE
int    main()
#else
int    aexslct()
#endif
   {
   struct TRowBuf
     {
     L_CHAR  Name[20];
     L_CHAR  FirstName[15];
     L_CHAR  Sex;
     };
   typedef struct TRowBuf TRowBuf;
   TCBL  CBLconnect;
   L_CHAR Name_Pass[] = "SYSTEM/MANAGER8";
   L_CHAR Node[] = "        ";
   L_WORD Priority = 0;
   L_LONG PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
   L_LONG Err;
   L_CHAR  Query[] = "select NAME,FIRSTNAM, SEX from PERSON;";
   TRowBuf RowBuf;

   memset(&CBLconnect,0,sizeof(TCBL));
   Err=LinterOPEN(&CBLconnect, Name_Pass, Node, Priority, PrzExe);
   if (Err != NORMAL)
     PrintError(&CBLconnect);
   printf("Connect to RDBMS Linter\n");

   LinterASYNC_SLCT(&CBLconnect, PrzExe, Query, &RowBuf,
           sizeof(TRowBuf), NULL, WorkError);
   printf("Selecting ... ");
```

```
   fflush(stdout);

   /* ... */

   inter_wait_single(&CBLconnect, -1);
   printf("Ok\n");
   printf("First Selected Row:\n");
   printf("%.20s %.15s %c\n", RowBuf.Name, RowBuf.FirstName,
 RowBuf.Sex);

   printf("End Example\n");

   return 0;
   }
```

Анализ и обработка кодов завершения

Коды завершения команд

Корректно написанные приложения обязаны проверять код завершения каждого поданного ими запроса. При нормальном завершении запроса гарантируется, что приложение получило правильные (в контексте поданной команды) данные и дальнейший процесс обработки информации имеет смысл.

Если СУБД не может выполнить команду, то в приложение возвращается код завершения, указывающий на возможную причину ошибки. Проведя анализ кода завершения, можно выявить причину отказа в выполнении запроса и, в зависимости от ситуации, либо повторить запрос с правильными значениями параметров, либо прекратить работу приложения.

Интерфейс нижнего уровня предоставляет клиентскому приложению, в общем случае, два кода завершения – код завершения, сформированный ядром СУБД ЛИНТЕР или непосредственно самим интерфейсом нижнего уровня (возвращается в поле `CodErr` контрольного блока) и, в некоторых случаях, дополнительный код завершения, формируемый операционной системой при выполнении запросов ядром СУБД (возвращается в поле `SysErr` контрольного блока), и детализирующий код завершения ядра СУБД. Например, если в поле `CodErr` возвращен код завершения `ERROPFIL` («Ошибка открытия файла»), то в поле `SysErr` будет возвращено более подробное описание ошибки (например, файл с заданным именем не найден, не разрешен совместный доступ к файлу, нельзя открыть файл для заданной операции и т.п.). При этом в разных операционных системах в случае идентичных ошибочных ситуаций код завершения, возвращаемый в поле `CodErr`, будет всегда одинаков, в то время как значение, возвращаемое в поле `SysErr`, в общем случае, будет отличаться.

Обработка кодов завершения

Дать описание кода завершения СУБД

Назначение

Команда `GETE` возвращает текстовое описание кода завершения операции СУБД ЛИНТЕР.

Параметры вызова

```
inter(CBL, NULL, NULL, [CondBuf], RowBuf);
```

Входные данные

Входными данными является контрольный блок `CBL`.

В нем должны быть заполнены поля:

Имя поля	Значение
`Command`	`"GETE"`
`RowId`	Числовое значение кода завершения СУБД
`LnBufRow`	Длина буфера выборки данных

Имя поля	Значение
Node	Имя ЛИНТЕР-сервера

Выходные данные

Выходными данными являются:

- контрольный блок CBL;
- буфер выборки данных RowBuf.

В контрольном блоке будут возвращены:

Имя поля	Значение
CodErr	Код завершения операции СУБД ЛИНТЕР
LnBufRow	Фактическая длина буфера выборки данных
SysErr	Код состояния ОС

Текстовое описание кода завершения возвращается в буфере выборки данных RowBuf.

Для кодов завершения СУБД, относящихся к созданию (модификации) триггеров и хранимых процедур, вместо текста выдается другая информация (пункт «Обработка ошибок трансляции хранимых процедур (триггеров)»).

Описание

При выполнении команды GETE используется таблица ERRORS, содержащая текстовые описания кодов завершения. Если эта таблица отсутствует в БД, или задан несуществующий числовой код завершения, то в RowBuf возвращается следующий текст: Linter error NNNN

где NNNN – указанный в команде GETE числовой код завершения.

Примечание
Для создания таблицы ERRORS, в случае ее отсутствия в БД, необходимо выполнить командный файл errors.sql и **затем** загрузить в созданную таблицу данные из файла errors.lod. Указанные файлы входят в состав дистрибутива СУБД ЛИНТЕР.

Коды завершения

Код	Описание
NORMAL	Нормальное завершение
SMALLBUFKOR	Недостаточный размер буфера выборки данных

Пример формирования команды

```
#include <string.h>
#include <stdlib.h>
#include "inter.h"

L_LONG LinterGETE(TCBL *pCBL, L_LONG CodErr, void *Str, L_WORD
 StrLen)
```

```
   {
   memcpy(pCBL->Command, "GETE", 4);
   pCBL->LnBufRow=StrLen;
   pCBL->RowId=CodErr;
   pCBL->PrzExe &= ~Q_ASYNC;
   inter(pCBL, NULL, NULL, NULL, Str);
   *((char*)Str + ((pCBL->LnBufRow>StrLen)?(StrLen-1):(pCBL-
>LnBufRow))) = 0;
   return pCBL->CodErr;
   }
```

Пример использования команды

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#ifndef WINCE
int    main()
#else
int    exgete()
#endif
   {
   TCBL  CBLconnect;
   L_CHAR Name_Pass[] = "SYSTEM/MANAGER8";
   L_CHAR Node[] = "        ";
   L_WORD Priority = 0;
   L_LONG PrzExe = M_EXCLUSIVE | M_BINARY;
   L_LONG Err;
   L_CHAR  Str[256];
   L_CHAR  Query[] = "select ERROR NAME,FIRSTNAM,SEX from
 PERSON;";
   /* ошибка */

   memset(&CBLconnect,0,sizeof(TCBL));
   Err=LinterOPEN(&CBLconnect, Name_Pass, Node, Priority, PrzExe);
   if (Err != NORMAL)
     PrintError(&CBLconnect);
   printf("Connect to RDBMS Linter\n");

   LinterSLCT(&CBLconnect, PrzExe, Query, NULL, L_MAXWORD, NULL);
   Err=LinterGETE(&CBLconnect, CBLconnect.CodErr, Str, 255);
   if (Err != NORMAL)
     PrintError(&CBLconnect);
   printf("GETE: %s\n", Str);
```

```
printf("End Example\n");

return 0;
}
```

Обработка ошибок трансляции хранимых процедур (триггеров)

Для получения подробного описания ошибок трансляции следует подать команду GETE **по тому же каналу, по которому подавался запрос на трансляцию** (**CREATE PROCEDURE** или **ALTER PROCEDURE**), передавая ей в качестве входного параметра код завершения Proc_Translation_Error («ошибка трансляции хранимой процедуры (триггера)»). В этом случае в буфере RowBuf вместо текстового описания кода завершения возвращается информация следующего вида:

- количество ошибок трансляции (2 байта);
- соответствующее количество описателей ошибок трансляции.

Описатель ошибки содержит заголовок и, в зависимости от типа ошибки, дополнительные параметры. Заголовок представляет собой структуру типа ERRS_DESCR, которая описана в заголовочном файле inter.h.

Заголовок описателя ошибки содержит следующие поля:

Имя поля	Значение
CodErr	Код конкретной ошибки трансляции (например, неизвестный оператор процедурного языка); расшифровка (текст) этого кода может быть получена путем подачи команды GETE с указанием ей в качестве анализируемого кода завершения значение CodErr +10000
Row	Номер строки исходного текста процедуры (триггера), в которой обнаружена синтаксическая ошибка. Если значение поля Row равно -1, это говорит о том, что ошибка произошла не в какой-либо конкретной позиции, а относится ко всему тексту (например, «Слишком много ошибок»)
Position	Номер позиции в строке Row, где обнаружена ошибка. Для поля Row со значением -1 содержимое данного поля не определено
ArgType	Характеристика диагностического сообщения. Если значение ArgType равно PROC_ERR_NOARG, то описатель состоит из одного заголовка. В противном случае кроме кода ошибки возвращается и дополнительный параметр, который располагается непосредственно за заголовком. Если значение ArgType равно PROC_ERR_ARGSHORT, то в качестве параметра передается двухбайтное целое число, а расшифровка ошибки, полученная с помощью команды GETE, представляет собой форматную строку для стандартной функции sprintf языка программирования C, содержащую символы '%d', вместо которых подставляется параметр. Если значение ArgType равно PROC_ERR_ARGSTRING, то в качестве параметра передается строка, а расшифровка ошибки, полученная с помощью команды GETE, представляет собой форматную строку для стандартной функции sprintf языка программирования C, содержащую символы '%s', вместо которых подставляется

Имя поля	Значение
	параметр. Параметр-строка передается как 2 байта длины и далее соответствующее количество символов

Пример типовой последовательности анализа ошибок трансляции процедур на языке C/C++ приведен в приложении 12.

Коды завершения интерфейса нижнего уровня

В таблице 19 приведены коды завершения, возвращаемые непосредственно программными средствами интерфейса нижнего уровня. Полный перечень возможных кодов завершения приведен в документе «Справочник кодов завершения». Мнемонические имена кодов завершения интерфейса содержатся в файле `lincodes.h`.

Таблица 19. Коды завершения интерфейса нижнего уровня СУБД ЛИНТЕР

Мнемоническое имя	Числовое значение	Причина ошибки	Комментарий
`NOCOMMAND`	1000	Неизвестная команда	Указанная в вызове функции `inter` команда не предусмотрена в этой версии СУБД ЛИНТЕР
`ERROPENQUE`	1001	Ошибка сервиса обмена	Ошибка установки соединения с ядром СУБД ЛИНТЕР или клиентским сетевым драйвером. СУБД ЛИНТЕР или клиентский сетевой драйвер не запущены
`NOVS`	1002	Ошибка присоединения сервиса обмена	Как правило, ошибка выявляется операционной системой, поэтому следует посмотреть код возврата операционной среды `SysErr` и обратиться к документации по операционной системе
`ERRWRITEMSG`	1003	Ошибка передачи сообщения	См. комментарий к коду 1002
`ERRREADMSG`	1004	Ошибка приема сообщения	См. комментарий к коду 1002
`NOENDOFOPER`	1006	Не определен конец оператора	Текст в буфере оператора `OpBuf` не закончен символом «точка с запятой» (;)
`NoMemoryForAsyncQuery`	1043	Нет памяти для асинхронного процесса	Недостаточно оперативной памяти для запуска асинхронной обработки запроса из-за ограниченных ресурсов ядра СУБД ЛИНТЕР или из-за большого количества выполняемых асинхронных процессов
`NULLPOINTER`	1057	Нулевой указатель	Нулевой адрес контрольного блока или буфера, используемого в качестве обязательного параметра
`CALLFROMHANDLER`	1085	Синхронный вызов	Выполнен синхронный вызов из асинхронного обработчика

Приложение 1
Коды операционных систем

Мнемонические обозначения кодов ОС содержатся в файле `inter.h`.

Обозначение	Код	Операционная система
OS_UNKNOWN	0	Неизвестная операционная система
OS_UNIX	1	Неизвестная система типа UNIX
OS_UNIXWARE	2	ОС UNIXWARE
OS_USIX	3	ОС USIX
OS_FREEBSD	4	ОС семейства BSD (OpenBSD, FreeBSD, NetBSD и т.п.)
OS_LINUX	5	ОС семейства LINUX
OS_UNIXSCO	6	ОС SCO OpenServer
OS_SINIX	7	ОС SINIX
OS_MSVS	8	ОС МСВС (может возвращаться OS_LINUX)
OS_MSVT	9	ОС МСВТ (может возвращаться OS_LINUX)
OS_MSDOS	10	ОС семейства DOS (PCDOS, MSDOS, DRDOS и т.п.)
OS_WIN3XX	11	ОС Windows 3.x или Windows for Workgroups 3.x
OS_WIN95	12	ОС Windows 9x (95,98,Me)
OS_WINNT	13	ОС семейства Windows NT
OS_OS9	14	ОС семейства OS9
OS_OS9000	15	ОС OS9000
OS_VAXVMS	16	ОС VMS (на Alpha, VAX или другой платформе)
OS_NETWARE	17	ОС Netware
OS_IBMOS2	18	ОС OS/2
OS_QNX	19	ОС QNX, ЗОСРВ «Нейтрино»
OS_DIGITAL	20	ОС DigitalUNIX (CompaqUNIX, True64)
OS_SUNSPARC	21	ОС SOLARIS на платформе SPARC
OS_SUNI386	22	ОС SOLARIS на платформе x86
OS_AIX	23	ОС AIX
OS_SGI	24	ОС SGI
OS_VXWORKS	25	ОС VxWorks (или ОС2000)
OS_WINCE	26	ОС семейства WindowsCE
OS_ANDROID	27	ОС Android
OS_IOS	28	ОС iOS

Приложение 2
Определения препроцессора для intlib.c

Определение		Комментарий
основное	**дополнительное**	
INTER_MSDOS	MSDOS	MS-DOS для реального режима
INTER_DPMI	MSDOS	MS-DOS для защищенного режима
INTER_OS2		IBM OS/2
INTER_MSWINDOWS	WIN32, WINDOWS_NT	ОС Windows NT, 2000, XP
INTER_MSWINDOWS	WIN32	ОС Windows 9x, ME
INTER_MSWINDOWS	WIN32, WINCE, WINDOWS_NT	ОС Windows CE
INTER_MSWINDOWS	WIN16, WIN3XX	ОС Windows 3.x
OS9000		Microware OS9000
OS9000	OS9	Microware OS9
VMS	VAX, _VMS_	VMS или OpenVMS
QNX		QNX, ЗОСРВ «Нейтрино»
QNX	QNX6	QNX6, ЗОСРВ «Нейтрино»
VXWORKS		VxWorks
OS2000	VXWORKS	ОС2000
INTER_NLM		Novell NetWare
UNIX	BSD	ОС семейства BSD FreeBSD, OpenBSD, BSDI и т.п.
UNIX	LINUX	Linux
UNIX	UNIXSCO,SCO	SCO OpenServer
UNIX	SINIX	SINIX
UNIX	UNIXWARE	Novell UnixWare, SCO UnixWare
UNIX	USIX	USIX
UNIX	DIGITAL_UNIX	Digital UNIX, Compaq UNIX, True64
UNIX	SUN	Solaris
UNIX	AIX	AIX
UNIX	SGI	IRIX
UNIX	MSVS, LINUX	MCBC
UNIX	MSVT, LINUX	MCBT

Кроме макросов, определяющих ОС, существуют дополнительные макросы, определяющие особенности ОС.

Имя макроса	Необходимость применения
__64BIT_ARCH__	В случае 64-битной компиляции (адрес – 8 байт)
SUPP_LONGLONG	В случае если ОС поддерживает тип данных long long
__UNSIG_CHAR__	В случае если тип данных char по умолчанию беззнаковый
DATA_MSBF	На MSBF архитектурах (например, SPARC)
LONG_ALIGN	В случае если архитектура требует выровненный доступ к данным (SPARC, ARM и т.п.)
SOCKETS	При работе через Unix Domain Sockets (для всех ОС UNIX)
LIN_PTHREAD	В ОС Linux, ЗОСРВ Нейтрино для сборки многонитевой версии (posix threads)
SVR42, SVR4	Во всех ОС UNIX

Пример

Трансляция для UNIX BSD.

```
$ cc -c -DUNIX -DFREEBSD -D_VER_MAX=600 intlib.c
```

Приложение 3
Типы данных интерфейса нижнего уровня

Приведенные ниже типы данных (заголовочный файл `lintypes.h` в подкаталоге `intlib` установочного каталога СУБД ЛИНТЕР) рекомендуется использовать при разработке мобильных Си-приложений.

Тип данных интерфейса	Характеристика данных
L_LONG	Знаковый 4-х байтовый
L_ULONG	Беззнаковый 4-х байтовый
L_SWORD	Знаковый 2-х байтовый
L_WORD	Беззнаковый 2-х байтовый
L_DLONG	Знаковый 8-и байтовый
L_UDLONG	Беззнаковый 8-и байтовый
L_BYTE	Беззнаковый байт
L_SBYTE	Знаковый байт
L_CHAR	Строка однобайтовых символов
L_UNICHAR	Строка двухбайтовых (UNICODE) символов
L_REAL	Вещественное число одинарной точности
L_DOUBLE	Вещественное число двойной точности
L_BOOL	Беззнаковый байт
L_DECIMAL	Массив из 16 байт. Используется для хранения значений типа DATE

Приложение 4

Соответствие типов данных СУБД ЛИНТЕР и интерфейса нижнего уровня

Тип данных СУБД ЛИНТЕР	Тип данных интерфейса
char	DT_CHAR
smallint	DT_INTEGER[1)]
int	DT_INTEGER
bigint	DT_INTEGER
real	DT_REAL[2)]
double	DT_REAL
date	DT_DATE
numeric	DT_DECIMAL
byte	DT_BYTE
blob	DT_BLOB
varchar	DT_VARCHAR
varbyte	DT_VARBYTE
boolean	DT_BOOL
nchar	DT_NCHAR
nchar varying, nvarchar	DT_NVARCHAR
extfile	DT_EXTFILE

[1)] Тип данных smallint, int и bigint имеют совпадающий код 2. Для детализации этих типов данных следует анализировать длину их данных: у smallint она равна 2 байтам, у int – 4 байтам, у bigint – 8.

[2)] Тип данных real и double имеют совпадающий код 3. Для детализации этих типов данных следует анализировать длину их данных: у real она равна 4 байтам, у double – 8 байтам.

Приложение 5
Пример разбора спецификации выборки данных

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "decimals.h"
#include "inter.h"
#include "tick.h"

#ifdef WIN32
#include <float.h>
#define isnan(f)   _isnan(f)
#define isinf(f) (!_finite(f))
#endif

#include "exlib.h"

void PrintError(TCBL*pCBL);
void PrintResult(L_BYTE*Buf, L_BYTE*VarBuf);
void Print_value(L_CHAR*text, P_TYPE Type, L_BYTE*Buf);

char Query_Create_Table[]="create or replace table TEST (C1
 char(10),"
  "C2 int,C3 real,C4 date,C5 decimal,C6 byte(10),C7 smallint,C8
 double,"
  "C9 bigint,C10 varchar(10),C11 varbyte(10),C12 boolean,C13
 nchar(10),"
  "C14 nchar varying(10), C15 BLOB, C16 extfile);";

char Query_Create_TabNan[]="create or replace table TESTnan ("
  "C1 int,C2 real,C3 double);";

char Query_Insert_Table1[]="insert into test values
 ('azzabyybcc',12,-12.345670,"
  "to_date('1998-12-01','YYYY-MM-DD'), 12.34,"
  "hex('FFFEFDFCFBFAF9F8F7F6'),30000,30.01,182834584,"
  "'azzabyybcc',hex('FFFEFDFCFBFAF9F8F7F6'),"
  "true,n'абвгдеж',n'abcdefg', NULL, extfile('1.txt'));";

char Query_Insert_Table[]="insert into test values "
  "(NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,
  NULL,NULL,NULL,NULL,NULL);";

char Query_InsNaN_Table[]="insert into test(c3,c8) "
```

```
"values ( cast HEXTORAW('0000807F00000000') as real,"
"cast HEXTORAW('000000000000F07F0000000000000000') as double);";

char Query_InsNaN1_Table[]="insert into test(c3,c8) "
"values ( cast HEXTORAW('000080FF00000000') as real,"
"cast HEXTORAW('000000000000F0FF0000000000000000') as DOUBLE);";

char Query_InsNaN2_Table[]="insert into test(c3,c8) "
"values ( cast HEXTORAW('0000C0FF00000000') as real,"
"cast HEXTORAW('000000000000F8FF0000000000000000') as DOUBLE);";

void PrintResult(L_BYTE*Buf,L_BYTE*VarBuf)
{
  L_WORD *pCntArg;
  P_TYPE Type;
  L_WORD i;
  L_CHAR str[100];
  L_BYTE *ptr;
  L_BYTE *ptr1;
  L_BYTE *ptr2;

  pCntArg = (L_WORD *) Buf;

  ptr1 = Buf+sizeof(L_WORD);
  ptr = ptr1+*pCntArg*sizeof(P_TYPE);

  VarBuf += 2*sizeof(WORD);

  for (i = 0; i < *pCntArg; ++i)
  {
    if ((L_WORD)VarBuf[i])
      sprintf(str, "field %d:<NULL>\n", i + 1);
    else
      sprintf(str, "field %d:\n", i + 1);

    Type = *(P_TYPE*)ptr1;
    ptr2 = ptr;

    if ( Type.ntyp == DT_VARCHAR ||
         Type.ntyp == DT_VARBYTE ||
         Type.ntyp == DT_NVARCHAR )
      ptr2 += 2;

    Print_value(str, Type, ptr2);
    ptr += Type.length;
    if ( Type.ntyp == DT_VARCHAR ||
```

```
         Type.ntyp == DT_VARBYTE ||
         Type.ntyp == DT_NVARCHAR )
      ptr += 2;
    ptr1 += sizeof(P_TYPE);
  }
}

void Print_value(L_CHAR*text, P_TYPE Type, L_BYTE*ptr)
{
  L_WORD i;
  L_CHAR StrBuf[4000];
  L_WORD SmallBuf;
  L_LONG IntBuf;
  L_REAL RealBuf;
  L_DOUBLE DblBuf;
  L_DECIMAL DecBuf;
  L_DOUBLE NumBuf;

  printf(text);
  printf("typ = %d, len = %d ", (L_WORD)Type.ntyp,
 (L_WORD)Type.length);
  switch (Type.ntyp)
  {
    case DT_CHAR:
    case DT_VARCHAR:
      memcpy(StrBuf, ptr, Type.length);
      StrBuf[Type.length] = 0;
      printf("value = '%s'",StrBuf);
    break;
    case DT_INTEGER:
      if (Type.length == sizeof(L_WORD))
      {
         memcpy(&SmallBuf, ptr, sizeof(L_WORD));
         printf("value = %d", SmallBuf);
      }
      else
      {
         memcpy(&IntBuf, ptr, sizeof(L_LONG));
         printf("value = %ld", IntBuf);
      }
    break;
    case DT_REAL:
      if (Type.length == sizeof(L_REAL))
      {
         memcpy(&RealBuf, ptr, sizeof(L_REAL));
         if (isnan(RealBuf))
```

```
        printf("value = %4s", RealBuf > 0 ? "nan" : "-nan");
      else
        if (isinf(RealBuf))
          printf("value = %4s", RealBuf > 0 ? "inf" : "-inf");
        else
          printf("value = %g", RealBuf);
  }
  else
  {
     memcpy(&DblBuf, ptr, sizeof(L_DOUBLE));
     if (isnan(DblBuf))
       printf("value = %4s", DblBuf > 0 ? "nan" : "-nan");
     else
       if (isinf(DblBuf))
         printf("value = %4s", DblBuf > 0 ? "inf" : "-inf");
       else
         printf("value = %g", DblBuf);
  }
break;
case DT_DATE:
  memcpy(DecBuf, ptr, sizeof(L_DECIMAL));
  TICKTOSTRF(DecBuf, "value = dd.mm.yyyy:hh:mi:ss", StrBuf);
  printf(StrBuf);
break;
case DT_DECIMAL:
  memcpy(DecBuf, ptr, sizeof(L_DECIMAL));
  DecToDbl(DecBuf, &NumBuf);
  printf("value = %g", NumBuf);
break;
case DT_BYTE:
case DT_VARBYTE:
case DT_NCHAR:
case DT_NVARCHAR:
  printf("value = ' ");
  for (i = 0; i < Type.length; ++i)
    printf("%2x ", (L_WORD) ptr[i]);
  printf(" ' ");
break;
case DT_BLOB:
  printf("@:BLOB:@");
break;
case DT_EXTFILE:
  printf("@:EXTFILE:@");
break;
case DT_BOOL:
{
```

```
      byte qqq = 0;
      memcpy(&qqq, ptr, 1);
      if (qqq)
        printf("value = <TRUE> ");
      else
        printf("value = <FALSE> ");
     }
     break;
  }
  printf("\n");
}

void main()
{
  TCBL CBLconnect;
  L_CHAR Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR Node[] = " ";
  L_WORD Priority = 0;
  L_LONG PrzExe = M_EXCLUSIVE | Q_ENCODE | M_SPEC;
  L_LONG Err;
  L_CHAR Query[] = "select * from test;";
  L_CHAR QueryNaN[] = "select c3,c8 from test where c3 is nan or
 c8 is nan;";
  L_BYTE Buf[4096];
  L_BYTE VarBuf[20];

  Err=LinterOPEN(&CBLconnect, Name_Pass, Node,Priority, PrzExe);

  if (Err != NORMAL)
    PrintError(&CBLconnect);

  printf("Connect to RDBMS Linter\n");

  Err= LinterNotSelect(&CBLconnect, Query_Create_Table);
  if (Err != NORMAL)
    PrintError(&CBLconnect);

  Err= LinterNotSelect(&CBLconnect, Query_Insert_Table);
  if (Err != NORMAL)
    PrintError(&CBLconnect);

  Err= LinterNotSelect(&CBLconnect, Query_Insert_Table1);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
```

```
 Err=LinterSLCT(&CBLconnect, PrzExe, Query, Buf,sizeof(Buf),
VarBuf);
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 else
 {
   printf("First Selected Row:\n");
   PrintResult(Buf,VarBuf);
 }

 Err=LinterGETL(&CBLconnect, Buf,sizeof(Buf), VarBuf);
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 else
 {
   printf("\nLast Selected Row:\n");
   PrintResult(Buf,VarBuf);
 }

 Err= LinterNotSelect(&CBLconnect, Query_InsNaN_Table);
 if (Err != NORMAL)
   PrintError(&CBLconnect);

 Err= LinterNotSelect(&CBLconnect, Query_InsNaN1_Table);
 if (Err != NORMAL)
   PrintError(&CBLconnect);

 Err= LinterNotSelect(&CBLconnect, Query_InsNaN2_Table);
 if (Err != NORMAL)
   PrintError(&CBLconnect);

 Err=LinterSLCT(&CBLconnect, PrzExe, QueryNaN, Buf,sizeof(Buf),
VarBuf);
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 else
 {
   printf("\nFirst Selected NaN values:\n");
   PrintResult(Buf,VarBuf);
 }

 Err=LinterGETN(&CBLconnect, Buf,sizeof(Buf), VarBuf);
 if (Err != NORMAL)
   PrintError(&CBLconnect);
 else
 {
```

```
    printf("Next Selected NaN values:\n");
    PrintResult(Buf,VarBuf);
  }

  Err=LinterGETL(&CBLconnect, Buf,sizeof(Buf), VarBuf);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  else
  {
    printf("Last Selected NaN values:\n");
    PrintResult(Buf,VarBuf);
  }

  printf("End Example\n");
}
```

Результаты выполнения примера.

```
Connect to RDBMS Linter
First Selected Row:
field 1:<NULL>
typ = 1, len = 10 value = '          '
field 2:<NULL>
typ = 2, len = 4 value = 0
field 3:<NULL>
typ = 3, len = 4 value = 0
field 4:<NULL>
typ = 4, len = 16 VALUE = 00.00.0000:00:00:00
field 5:<NULL>
typ = 5, len = 16 value = 0
field 6:<NULL>
typ = 6, len = 10 value = '  0  0  0  0  0  0  0  0  0  0  '
field 7:<NULL>
typ = 2, len = 2 value = 0
field 8:<NULL>
typ = 3, len = 8 value = 0
field 9:<NULL>
typ = 2, len = 8 value = 0
field 10:<NULL>
typ = 8, len = 10 value = ''
field 11:<NULL>
typ = 9, len = 10 value = '  0  0  0  0  0  0  0  0  0  0  '
field 12:<NULL>
typ = 10, len = 1 value = <FALSE>
field 13:<NULL>
typ = 11, len = 20 value = ' 20  0 20  0 20  0 20  0 20  0 20  0
 20  0 20  0 20  0 20  0  '
```

```
field 14:<NULL>
typ = 12, len = 20 value = '  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  '
field 15:
typ = 7, len = 24 @:BLOB:@
field 16:<NULL>
typ = 13, len = 522 @:EXTFILE:@

Last Selected Row:
field 1:
typ = 1, len = 10 value = 'azzabyybcc'
field 2:
typ = 2, len = 4 value = 12
field 3:
typ = 3, len = 4 value = -12.3457
field 4:
typ = 4, len = 16 VALUE = 01.12.1998:00:00:00
field 5:
typ = 5, len = 16 value = 12.34
field 6:
typ = 6, len = 10 value = ' ff fe fd fc fb fa f9 f8 f7 f6  '
field 7:
typ = 2, len = 2 value = 30000
field 8:
typ = 3, len = 8 value = 30.01
field 9:
typ = 2, len = 8 value = 182834584
field 10:
typ = 8, len = 10 value = 'azzabyybcc'
field 11:
typ = 9, len = 10 value = ' ff fe fd fc fb fa f9 f8 f7 f6  '
field 12:
typ = 10, len = 1 value = <TRUE>
field 13:
typ = 11, len = 20 value = ' 30  4 31  4 32  4 33  4 34  4 35  4
 36  4 20  0 20  0 20  0  '
field 14:
typ = 12, len = 20 value = ' 61  0 62  0 63  0 64  0 65  0 66  0
 67  0  0  0  0  0  0  0  '
field 15:
typ = 7, len = 24 @:BLOB:@
field 16:
typ = 13, len = 522 @:EXTFILE:@

First Selected NaN values:
field 1:
```

```
typ = 3, len = 4 value =  inf
field 2:
typ = 3, len = 8 value =  inf
Next Selected NaN values:
field 1:
typ = 3, len = 4 value = -inf
field 2:
typ = 3, len = 8 value = -inf
Last Selected NaN values:
field 1:
typ = 3, len = 4 value = -nan
field 2:
typ = 3, len = 8 value = -nan
End Example
```

Приложение 6
Состав примеров интерфейса нижнего уровня

Ниже приведен перечень примеров использования команд интерфейса нижнего уровня: библиотека `exlib.c` содержит тексты функций, выполняющих вызов соответствующих команд интерфейса; файлы примеров – примеры использования этих функций.

Все указанные файлы находятся в подкаталоге `/samples/call` дистрибутива или установочного каталога СУБД ЛИНТЕР.

Команда	Функция библиотеки exlib.c	Файл примера
AOBJ	LinterAOBJ, LinterASYNC_AOBJ	exblob.c, aexblob.c
CLOS	LinterCLOS, LinterASYNC_CLOS	exclos.c, aexclos.c
COBJ	LinterCOBJ, LinterASYNC_COBJ	exblob.c, aexblob.c
COMT	LinterCOMT, LinterASYNC_COMT	exnosel.c, aexnosel.c
DESC	LinterDESC, LinterASYNC_DESC	exdesc.c, aexdesc.c
DIRA	LinterDIRA, LinterASYNC_DIRA	exdirx.c, aexdirx.c
DIRF	LinterDIRF, LinterASYNC_DIRF	exdirx.c, aexdirx.c
DIRR	LinterDIRR, LinterASYNC_DIRR	exdirx.c, aexdirx.c
FATR	LinterFATR, LinterASYNC_FATR	exfatr.c, aexfatr.c
FCUR	LinterFCUR, LinterASYNC_FCUR	exfcur.c, aexfcur.c
FNOD	LinterFNOD, LinterASYNC_FNOD	exfnod.c, aexfnod.c
FREL	LinterFREL, LinterASYNC_FREL	exfatr.c, aexfatr.c, exfrel.c, aexfrel.c
FUSR	LinterFUSR, LinterASYNC_FUSR	exfatr.c, aexfatr.c, exfrel.c, aexfrel.c, exfusr.c, aexfusr.c
GETA	LinterGETA, LinterASYNC_GETA	exgeta.c, aexgeta.c
GETE	LinterGETE, LinterASYNC_GETE	exgete.c, aexgete.c
GETF	LinterGETF, LinterASYNC_GETF	exgetx.c, aexgetx.c
GETL	LinterGETL, LinterASYNC_GETL	exgetx.c, aexgetx.c

Команда	Функция библиотеки exlib.c	Файл примера
GETN	LinterGETN, LinterASYNC_GETN	exgetx.c, aexgetx.c
GETP	LinterGETP, LinterASYNC_GETP	exgetx.c, aexgetx.c
GETS	LinterGETS, LinterASYNC_GETS	exgetx.c, aexgetx.c
GOBJ	LinterGOBJ, LinterASYNC_GOBJ	exblob.c, aexblob.c
KILL	LinterKILL, LinterASYNC_KILL	aexdesc.c, aexkill.c
LREL	LinterLREL, LinterASYNC_LREL	exnosel.c, aexnosel.c
LROW	LinterLROW, LinterASYNC_LROW	exnosel.c, aexnosel.c
OCUR	LinterOCUR, LinterASYNC_OCUR	exfcur.c, aexfcur.c, exkill.c, aexkill.c, exocur.c, aexocur.c, exseto.c, aexseto.c
OPEN	LinterOPEN, LinterASYNC_OPEN	Во всех примерах канальных команд
PUTM	LinterPUTM, LinterASYNC_PUTM	exputm.c, aexputm.c
RBAC	LinterRBAC, LinterASYNC_RBAC	exnosel.c, exnull.c, aexnosel.c
SETO	LinterSETO, LinterASYNC_SETO	exfcur.c, aexfcur.c, exfseto.c, aexseto.c
SHUT	LinterSHUT, LinterASYNC_SHUT	exshut.c, aexshut.c
SLCT	LinterSLCT, LinterASYNC_SLCT	exblob.c, aexblob.c, exgeta.c, aexgeta.c, exgete.c, aexgete.c, exgetx.c, aexgetx.c, exslct.c, aexslct.c
SNAP	LinterSNAP, LinterASYNC_SNAP	exdesc.c, aexdesc.c, exsnap.c, aexsnap.c
UREL	LinterUREL, LinterASYNC_UREL	exnosel.c, aexnosel.c
UROW	LinterUROW, LinterASYNC_UROW	exnosel.c, aexnosel.c
пробелы	LinterNotSelect, LinterASYNC_NotSelect	exblob.c, exputm.c, excreat.c, exnull.c, exnosel.c
	LinterExec, LinterASYNC_Exec[1)]	exexec.c
	LinterAnsColNum, LinterASYNC_AnsColNum[2)]	exgeta.c

Команда	Функция библиотеки exlib.c	Файл примера

[1)]Выполнение хранимых процедур

[2)]Подсчет числа полей в записи выборки данных

Приложение 7
Список команд интерфейса нижнего уровня

Команда	Канальность[1)]	Описание
AOBJ[2)], ABLB	√	Добавить порцию BLOB-данных
CLOS	√	Закрыть и освободить канал СУБД ЛИНТЕР
COBJ[2)], CBLB	√	Удалить BLOB-данные
COMT	√	Фиксировать изменения, сделанные транзакцией
DESC		Дать описание БД
DIRA		Дать элемент очереди столбцов
DIRF		Дать элемент очереди файлов
DIRR		Дать элемент очереди таблиц
FATR	√	Дать описание столбца
FCUR	√	Дать описание курсора
FNOD	√	Дать описание узла сети
FREL	√	Дать описание таблицы
FUSR	√	Дать описание пользователя
GETA	√	Дать информацию о структуре записи выборки данных
GETE	√	Дать описание кода завершения СУБД
GETF	√	Дать первую запись выборки данных
GETL	√	Дать последнюю запись выборки данных
GETM	√	Дать несколько записей за одно обращение
GETN	√	Дать следующую запись выборки данных
GETP	√	Дать предыдущую запись выборки данных
GETS	√	Дать указанным запись выборки данных
GOBJ[2)], GBLB	√	Дать следующую порцию BLOB-данных
KILL	√[3)]	Удалить канал (соединение либо курсор)
LREL	√	Блокировать таблицу для монопольного использования каналом
LROW	√	Блокировать запись для монопольного использования каналом
OCUR	√	Открыть подчиненный канал (курсор) для связи с СУБД ЛИНТЕР
OPEN		Открыть главный канал (соединение) для связи с СУБД ЛИНТЕР
PUTM	√	Добавить порцию (пакет) записей в таблицу
RBAC	√	Откатить все изменения, сделанные транзакцией
SETO	√	Установить имя курсора

Команда	Канальность[1)]	Описание
SHUT		Записать все измененные элементы очередей на диск и завершить работу СУБД
SLCT	√	Поиск и выборка данных
SNAP		Сбросить все измененные элементы очередей на диск
UREL	√	Разблокировать заблокированную ранее таблицу
UROW	√	Разблокировать заблокированную ранее запись таблицы
4 пробела	√	Выполнить непоисковый SQL-запрос

[1)]Знаком √ отмечены канальные команды.

[2)]Команда устарела и не рекомендуется для применения.

[3)]Возможен неканальный вариант команды.

Приложение 8
Пример идентификации и аутентификации по Kerberos-протоколу

В примере демонстрируется открытие соединения в клиентском приложении с СУБД ЛИНТЕР с идентификацией и аутентификацией по Kerberos-протоколу.

Предполагается, что Kerberos-сервер корректно настроен, ядро СУБД ЛИНТЕР активно и на компьютере с клиентским приложением получен Kerberos-тикет для пользователя "user" (т.е. пользователь с таким именем создан в БД аутентификации Kerberos-сервера).

Пользователь с именем "invalid_user" в БД ЛИНТЕР не существует, он используется для демонстрации того, что введенные регистрационные данные при идентификации и аутентификации по Kerberos-протоколу игнорируются.

```
#include <stdio.h>
#include <stdlib.h>
#include "inter.h"
int main()
{
  TCBL CBL = {0};
  L_CHAR Node[] = "        ";
  L_CHAR name[] = "invalid_user/invalid_password";
  L_CHAR forceKrb = 1;
  memset( CBL.Node, 0, MAX_NODE_LEN );
  memcpy( CBL.Node, Node, strlen(Node) );
  memcpy( CBL.Command, "OPEN", 4);
  CBL.PrzExe = 0;
  inter( &CBL, name, NULL, NULL, NULL );
  if ( CBL.CodErr )
    printf( "1.Error %d\n", CBL.CodErr );
 /* Код завершения 1025, т.к. такого пользователя */
 /* в БД не существует */
  inter_control( NULL, ICR_FORCE_KRB, &forceKrb, 1 );
  inter( &CBL, name, NULL, NULL, NULL );
  if ( !CBL.CodErr )
    printf( "2.Success\n");
 /* Нет ошибки, т.к. аутентифицировался пользователь */
 /* user по Kerberos-протоколу */
  return 0;
}
```

После запуска будет выведено

```
1.Error 1025
2.Success
```

Приложение 9
Пример работы с двумя ядрами СУБД

```
#include "inter.h"
#if defined(VXWORKS)
#include "vxstart.h"
#endif

static void PrintErrorAndCloseChannel(TCBL * pCBL);
#if defined(VXWORKS)
 MainStart(c_select, 1024 * 32, UninitLinterClient)
#else
 int main()
#endif

    {
char name[] = "SYSTEM/MANAGER8";
int Count;
TCBL Cbl;
printf("\n*** Table b must be created in one database ***\n");
memset(&Cbl, 0, sizeof(TCBL));
printf("Open channel to default mbx\n");
Cbl.PrzExe &= ~(M_OPTIMISTIC | M_SHARE | M_EXCLUSIVE); /*
 Autocommit */
memcpy(Cbl.Command, "OPEN", 4);
inter(&Cbl, name, NULL, NULL, NULL);
    if if (Cbl.CodErr)
        {PrintErrorAndCloseChannel(&Cbl);
return 1;
}
    Cbl.LnBufRow = sizeof(Count);
memcpy(Cbl.Command, "SLCT", 4);
inter(&Cbl, NULL, "SELECT count(*) from b;", NULL, &Count);
    If (Cbl.CodErr)
        { PrintErrorAndCloseChannel(&Cbl);}
    printf("Close channel\n");
memcpy(Cbl.Command, "CLOS", 4);
inter(&Cbl, NULL, NULL, NULL, NULL);


setenv("LINTER_MBX", "1254", 1);
inter_control(&Cbl, ICR_REREAD_ENV, NULL, 0);
memset(&Cbl, 0, sizeof(TCBL));
printf("Open channel to mbx 1254 \n");
```

```
Cbl.PrzExe &= ~(M_OPTIMISTIC | M_SHARE | M_EXCLUSIVE); /*
 Autocommit */
memcpy(Cbl.Command, "OPEN", 4);
inter(&Cbl, name, NULL, NULL, NULL);
    if (Cbl.CodErr)
        { PrintErrorAndCloseChannel(&Cbl);
return 1;
}
    Cbl.LnBufRow = sizeof(Count);
memcpy(Cbl.Command, "SLCT", 4);
inter(&Cbl, NULL, "SELECT count(*) from b;", NULL, &Count);
    if (Cbl.CodErr)
        { PrintErrorAndCloseChannel(&Cbl); }
    printf("Close channel\n");
memcpy(Cbl.Command, "CLOS", 4);
inter(&Cbl, NULL, NULL, NULL, NULL);
printf("done\n");
printf("\nEnd of work.\n");
return 0;
}
void PrintErrorAndCloseChannel(TCBL * pCBL)
    {
printf("\n");
if (strncmp(pCBL->Node, " ", MAX_NODE_LEN) != 0 && *pCBL->Node !=
 0)
        printf("* Linter server <%.8s>\n", pCBL->Node);
        printf("* Channel N %d\n", pCBL->NumChan);
printf("* Command [%.4s]\n", pCBL->Command);
printf("* Linter error: %ld\n", pCBL->CodErr);
    if (pCBL->CodErr >= 2000 && pCBL->CodErr < 3000)
        fprintf(stderr, " Line %hd position %hd\n", ((L_WORD *)
 (&pCBL->SysErr))[0], ((L_WORD *) (&pCBL->SysErr))[1]);
    else if (pCBL->SysErr)
        printf("* System error: %ld\n", pCBL->SysErr);
        memcpy(pCBL->Command, "CLOS", 4);
    /* Close channel */
inter(pCBL, NULL, NULL, NULL, NULL);
}
```


Приложение 10
Примеры пакетной обработки данных

Байтовый формат пакета

Заполнение пакета

```
#include "decimals.h"
#include "inter.h"
#include "tick.h"
#include "exlib.h"

static const L_LONG Byte_Const = 0x010203ff;
static const L_LONG Varbyte_Const = 0xff0302;
static const L_CHAR Char_Const[] = "char";
static const L_CHAR Varchar_Const[] = "vchar";
static const wchar_t NVarchar_Const[] = L"nvarchar";
static const wchar_t NChar_Const[]  = L"nchar";
static const L_DOUBLE Double_Const = 30.01;
static const L_REAL Real_Const = -12.34567f;
static const L_BOOL Bool_Const = L_TTRUE;
static const L_CHAR Extfile_Const[] = "EXTFILE";
static const L_SWORD Smallint_Const = 2;
static const L_LONG  Int_Const = 4;

// Заполняет буфер простейшим значением в зависимости от типа в
 байтовом виде:
L_BYTE *MakeBYTEValue(int type, int Len, L_BYTE* buf)
{
  switch (type)
  {
    case DT_CHAR:     return memcpy(buf, Char_Const,
 sizeof(Char_Const));
    case DT_VARCHAR:  return memcpy(buf, Varchar_Const,
 sizeof(Varchar_Const));
    case DT_BYTE:     return memcpy(buf, &Byte_Const,
 sizeof(Byte_Const));
    case DT_VARBYTE:  return memcpy(buf, &Varbyte_Const,
 sizeof(Varbyte_Const));
    case DT_NCHAR:    return memcpy(buf, NChar_Const,
 sizeof(NChar_Const));
    case DT_NVARCHAR: return memcpy(buf, NVarchar_Const,
 sizeof(NVarchar_Const));
    case DT_DATE:
      {
```

```
        DECIMAL dateVal;
        STRTOTICK("03.03.2003:16:29:00.00", dateVal);
        return memcpy(buf, &dateVal, Len);
      }
    case DT_INTEGER:
      if (Len == sizeof(L_SWORD))
        return memcpy(buf, &Smallint_Const, Len);
      if (Len == sizeof(L_LONG))
        return memcpy(buf, &Int_Const, Len);
      return NULL; // Будет обозначать NULL-значение
    case DT_REAL:
      if (Len == sizeof(L_REAL))
        return memcpy(buf, &Real_Const, Len);
      return memcpy(buf, &Double_Const, Len);
    case DT_DECIMAL:
      {
        DECIMAL decVal;
        DblToDec(12.34, decVal);
        return memcpy(buf, &decVal, Len);
      }
    case DT_BOOL:     return memcpy(buf, &Bool_Const, Len);
    case DT_EXTFILE:  return memcpy(buf, Extfile_Const,
 sizeof(Extfile_Const));
    default: return NULL;
  } // switch
} // MakeBYTEValue
```

Обработка пакета

```
#include <stdio.h>
#include <stdlib.h>
#include "decimals.h"
#include "inter.h"
#include "tick.h"
#include "exlib.h"

extern L_BYTE *MakeBYTEValue(int, int, L_BYTE*); // Генерирует
 байтовый вид
 значения

#define MAX_BUF_LEN 2048
#define FILTER_DESC_LEN 10

L_BYTE *fillByteBuffer(L_WORD Len, L_WORD typ, L_BYTE *outAdr,
 L_WORD *outSize)
{
  L_SWORD totalLength;
```

```
 L_WORD curLen = Len;
 L_BYTE curValue[32] = {0}; // Буфер под наше самое длинное
значение
 L_BYTE *pcurValue = MakeBYTEValue(typ, Len, curValue);

 if (pcurValue == NULL) // Указывает, что это NULL-значение
 {
   totalLength = -1; // При NULL-значении пишем только длину,
равную -1
   memcpy(outAdr, &totalLength, sizeof(totalLength));
   outAdr += sizeof(totalLength);
   *outSize += sizeof(totalLength);
   return outAdr;
 }

 if (typ == DT_VARCHAR || typ == DT_VARBYTE || typ ==
DT_NVARCHAR)
 {
   if (typ == DT_NVARCHAR)
     curLen = wcslen((wchar_t *)pcurValue) * sizeof(wchar_t);
   else
     curLen = strlen((const char*)pcurValue);

   totalLength = curLen + sizeof(L_WORD); // Длина с учетом поля
длины
   memcpy(outAdr, &totalLength, sizeof(totalLength));
   outAdr += sizeof(totalLength);
   *outSize += sizeof(totalLength);
 }

 if (typ == DT_EXTFILE)
 {
   curLen = strlen((const char*)pcurValue);

   totalLength = FILTER_DESC_LEN + 512;
   memcpy(outAdr, &totalLength, sizeof(totalLength));
   outAdr += sizeof(totalLength);
   *outSize += sizeof(totalLength);

   memset(outAdr, 0, totalLength);
   outAdr += FILTER_DESC_LEN;
   *outSize += FILTER_DESC_LEN;

   memcpy(outAdr, pcurValue, curLen);
   outAdr += 512;
   *outSize += 512;
```

```
  }
  else
  {
    memcpy(outAdr, &curLen, sizeof(curLen));
    outAdr += sizeof(curLen);
    *outSize += sizeof(curLen);

    memcpy(outAdr, pcurValue, curLen);
    outAdr += curLen;
    *outSize += curLen;
  }
  return outAdr;
} // fillByteBuffer

int main()
{
  TCBL CBLconnect = {0};
  L_CHAR Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR Node[] = "    ";
  L_WORD Priority = 0;
  L_LONG PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG Err = NORMAL;
  L_CHAR Query[] = "select * from test;";

  L_LONG i;
  L_WORD curSize = sizeof(L_WORD);
  L_BYTE Buf[MAX_BUF_LEN];

  L_WORD recCnt = 2; // Число записей в одной порции пакета
  L_WORD ColumnCount; // Число столбцов в выборке данных
  GETA_OUT* ansDesc = NULL; // Описатель столбцов выборки данных
  L_BYTE PackBuf[MAX_BUF_LEN] = {0}; // Буфер для размещения
 добавляемой порции
 пакета
  L_BYTE *CurAdr = PackBuf + sizeof(recCnt);

  *((L_WORD *)PackBuf) = recCnt; // Первым пишем число записей в
 порции пакета

  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect); /* Если ошибка, то всегда exit(1)! */
  printf("Connect to RDBMS Linter\n");
```

```
 // Создадим таблицу с использованием всех типов данных СУБД
ЛИНТЕР:
 Err = LinterNotSelect(
   &CBLconnect,
   "create or replace table TEST(INT_COLUMN int,"
                     "SMALLINT_COLUMN smallint,"
                     "BIGINT_COLUMN bigint,"
                     "CHAR_COLUMN char(5),"
                     "VARCHAR_COLUMN varchar(10),"
                     "BYTE_COLUMN byte(4),"
                     "VARBYTE_COLUMN varbyte(5),"
                     "NCHAR_COLUMN nchar(5),"
                     "NCHAR_VARYING_COLUMN nchar varying(10),"
                     "DATE_COLUMN date,"
                     "REAL_COLUMN real,"
                     "DOUBLE_COLUMN Double,"
                     "DEC_COLUMN decimal,"
                     "BOOL_COLUMN boolean,"
                     "EXTFILE_COLUMN extfile);");
 if (Err != NORMAL)
   PrintError(&CBLconnect);

 Err = LinterSLCT(&CBLconnect, PrzExe, Query, Buf, sizeof(Buf),
NULL);
 Err = LinterAnsColNum(&CBLconnect, &ColumnCount);
 if (Err != NORMAL)
   PrintError(&CBLconnect);

 ansDesc = (GETA_OUT*)calloc(ColumnCount, sizeof(GETA_OUT));
 if (ansDesc == NULL)
 {
   printf("Error: not memory\n"); exit(1);
 }

 // Получаем полные описания столбцов, но нам потребуется только
тип и длина:
 Err = LinterGETA(&CBLconnect, 0, ansDesc,
ColumnCount*sizeof(GETA_OUT));
 if (Err != NORMAL)
   PrintError(&CBLconnect);

 for (i = 0; i < ColumnCount; i++)
   CurAdr = fillByteBuffer(ansDesc[i].Length, ansDesc[i].Type,
CurAdr,
&curSize);
 for (i = 0; i < ColumnCount; i++)
```

```
    CurAdr = fillByteBuffer(ansDesc[i].Length, ansDesc[i].Type,
 CurAdr,
 &curSize);

  printf("PUTM AS BYTE");
  Err = LinterNotSelect(
    &CBLconnect,
    "start append into TEST byte(INT_COLUMN, SMALLINT_COLUMN,
 BIGINT_COLUMN, "
                                "CHAR_COLUMN, VARCHAR_COLUMN, "
                                "BYTE_COLUMN, VARBYTE_COLUMN, "
                                "NCHAR_COLUMN,
 NCHAR_VARYING_COLUMN, "
                                "DATE_COLUMN, "
                                "REAL_COLUMN, DOUBLE_COLUMN,
 DEC_COLUMN, "
                                "BOOL_COLUMN, "
                                "EXTFILE_COLUMN);");
  if (Err != NORMAL)
    PrintError(&CBLconnect);

  Err = LinterPUTM(&CBLconnect, PackBuf, curSize);
  if (Err != NORMAL)
    PrintError(&CBLconnect);

  if (LinterNotSelect(&CBLconnect, "end append into TEST;") !=
 NORMAL)
    PrintError(&CBLconnect);
  printf(" - OK\n");
  free(ansDesc);
  Err = LinterCLOS(&CBLconnect);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  return 0;
} // Main
```

Символьный формат пакета

Заполнение пакета

```
#include <stdio.h>
#include "decimals.h"
#include "inter.h"
#include "tick.h"
#include "exlib.h"

static const L_LONG Byte_Const = 0x010203ff;
```



```
static const L_LONG Varbyte_Const = 0xff0302;

// Заполняет буфер простейшим значением в символьном виде  в
 зависимости от типа данных:
L_BYTE* MakeCHARValue(int type, int Len, L_BYTE* buf)
{
  switch (type)
  {
    case DT_CHAR:
      return (L_BYTE*)strcpy((char*)buf, "char");
    case DT_VARCHAR:
      return (L_BYTE*)strcpy((char*)buf, "vchar");
    case DT_BYTE:
      return (L_BYTE*)memcpy(buf, & Byte_Const,
 sizeof(Byte_Const));
    case DT_VARBYTE:
      return (L_BYTE*)memcpy(buf, & Varbyte_Const,
 sizeof(Varbyte_Const));
    case DT_NCHAR:
      return (L_BYTE*)wcscpy((wchar_t*)buf, L"nchar");
    case DT_NVARCHAR:
      return (L_BYTE*)wcscpy((wchar_t*)buf, L"nvarchar");
    case DT_DATE:
      return (L_BYTE*)strcpy((char*)buf, "03.03.2003");
    case DT_INTEGER:
      if (Len == sizeof(L_SWORD))
        return (L_BYTE*)strcpy((char*)buf, "2");
      if (Len == sizeof(L_LONG))
        return (L_BYTE*)strcpy((char*)buf, "4");
      return NULL; // Будет обозначать NULL-значение
    case DT_REAL:
      if (Len == sizeof(L_REAL))
        return (L_BYTE*)strcpy((char*)buf, "-12.34567");
      return (L_BYTE*)strcpy((char*)buf, "30.01");
    case DT_DECIMAL:
      return (L_BYTE*)strcpy((char*)buf, "12.34");
    case DT_BOOL:
      return (L_BYTE*)strcpy((char*)buf, "TRUE");
    case DT_EXTFILE:
      return (L_BYTE*)strcpy((char*)buf, "EXTFILE");
    default:
      return NULL;
  } // switch
} // MakeCHARValue
```

Обработка пакета

```
#include <stdio.h>
#include <stdlib.h>
#include "decimals.h"
#include "inter.h"
#include "tick.h"
#include "exlib.h"

extern L_BYTE* MakeCHARValue(int, int, L_BYTE*); // Генерирует
 значение

#define MAX_BUF_LEN 2048

L_BYTE *fillCharBuffer(L_WORD Len, L_WORD typ, L_BYTE *outAdr,
 L_WORD *outSize)
{
  L_WORD curLen = Len;
  L_BYTE curValue[64] = {0};  // Буфер под самое длинное значение
  L_BYTE *pcurValue = MakeCHARValue(typ, Len, curValue);

  if (pcurValue == NULL) // Указывает, что это NULL-значение
  {
    curLen = -1; // При NULL-значении  пишем только длину, равную
 -1
    memcpy(outAdr, &curLen, sizeof(curLen));
    outAdr += sizeof(curLen);
    *outSize += sizeof(curLen);
    return outAdr;
  }

  if (typ == DT_VARCHAR || typ == DT_VARBYTE || typ ==
 DT_NVARCHAR)
  {
    if (typ == DT_NVARCHAR)
      curLen = wcslen((wchar_t *)pcurValue) * sizeof(wchar_t);
    else
      curLen = strlen((const char*)pcurValue);
  }

  if (typ == DT_REAL || typ == DT_DECIMAL || typ == DT_DATE ||
      typ == DT_BOOL || typ == DT_EXTFILE)
    curLen = strlen((const char*)pcurValue);

  memcpy(outAdr, &curLen, sizeof(curLen));
  outAdr += sizeof(curLen);
```



```
  *outSize += sizeof(curLen);

  memcpy(outAdr, pcurValue, curLen);
  outAdr += curLen;
  *outSize += curLen;
  return outAdr;
} // fillCharBuffer

int main()
{
  TCBL CBLconnect = {0};
  L_CHAR Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR Node[] = "    ";
  L_WORD Priority = 0;
  L_LONG PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG Err = NORMAL;
  L_CHAR Query[] = "select * from test;";

  L_LONG i;
  L_WORD curSize = sizeof(L_WORD);
  L_BYTE Buf[MAX_BUF_LEN];

  L_WORD recCnt = 2; // Число записей в одной порции пакета
  L_WORD ColumnCount; // Число столбцов в выборке данных
  GETA_OUT* ansDesc = NULL; // Описатель столбцов выборки данных
  L_BYTE PackBuf[MAX_BUF_LEN] = {0}; // Буфер для размещения
 добавляемой порции пакета
  L_BYTE *CurAdr = PackBuf + sizeof(recCnt);

  *((L_WORD *)PackBuf) = recCnt; // Первым пишем число записей в
 порции пакета

  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority,
 PrzExe);
  if (Err != NORMAL)
    PrintError(&CBLconnect); /* Если ошибка, то всегда exit(1)! */
  printf("Connect to RDBMS Linter\n");

  // Создадим таблицу с использованием всех типов данных СУБД
 ЛИНТЕР:
  Err = LinterNotSelect(
    &CBLconnect,
    "create or replace table TEST(INT_COLUMN int,"
                    "SMALLINT_COLUMN smallint,"
                    "BIGINT_COLUMN bigint,"
                    "CHAR_COLUMN char(5),"
```

```
                        "VARCHAR_COLUMN varchar(10),"
                        "BYTE_COLUMN byte(4),"
                        "VARBYTE_COLUMN varbyte(5),"
                        "NCHAR_COLUMN nchar(5),"
                        "NCHAR_VARYING_COLUMN nchar varying(10),"
                        "DATE_COLUMN date,"
                        "REAL_COLUMN real,"
                        "DOUBLE_COLUMN Double,"
                        "DEC_COLUMN decimal,"
                        "BOOL_COLUMN boolean,"
                        "EXTFILE_COLUMN extfile);");
 if (Err != NORMAL)
   PrintError(&CBLconnect);

 Err = LinterSLCT(&CBLconnect, PrzExe, Query, Buf, sizeof(Buf),
NULL);
 Err = LinterAnsColNum(&CBLconnect, &ColumnCount);
 if (Err != NORMAL)
   PrintError(&CBLconnect);

 ansDesc = (GETA_OUT*)calloc(ColumnCount, sizeof(GETA_OUT));
 if (ansDesc == NULL)
 {
   printf("Error: not memory\n"); exit(1);
 }

 // Получаем полные описания столбцов, но нам потребуется только
тип и длина:
 Err = LinterGETA(&CBLconnect, 0, ansDesc,
ColumnCount*sizeof(GETA_OUT));
 if (Err != NORMAL)
   PrintError(&CBLconnect);

 for (i = 0; i < ColumnCount; i++)
   CurAdr = fillCharBuffer(ansDesc[i].Length, ansDesc[i].Type,
CurAdr, &curSize);
 for (i = 0; i < ColumnCount; i++)
   CurAdr = fillCharBuffer(ansDesc[i].Length, ansDesc[i].Type,
CurAdr, &curSize);

 printf("PUTM AS CHAR");
 Err = LinterNotSelect(
   &CBLconnect,
   "start append into TEST char(INT_COLUMN, SMALLINT_COLUMN,
BIGINT_COLUMN, "
                                "CHAR_COLUMN, VARCHAR_COLUMN, "
```

```
                                  "BYTE_COLUMN, VARBYTE_COLUMN, "
                                  "NCHAR_COLUMN,
 NCHAR_VARYING_COLUMN, "
                                  "DATE_COLUMN, "
                                  "REAL_COLUMN, DOUBLE_COLUMN,
 DEC_COLUMN, "
                                  "BOOL_COLUMN, "
                                  "EXTFILE_COLUMN);");
  if (Err != NORMAL)
    PrintError(&CBLconnect);

  Err = LinterPUTM(&CBLconnect, PackBuf, curSize);
  if (Err != NORMAL)
    PrintError(&CBLconnect);

  if (LinterNotSelect(&CBLconnect, "end append into TEST;") !=
 NORMAL)
    PrintError(&CBLconnect);
  printf(" - OK\n");
  free(ansDesc);
  Err = LinterCLOS(&CBLconnect);
  if (Err != NORMAL)
    PrintError(&CBLconnect);
  return 0;
} // Main
```

Приложение 11
Идентификаторы выполняющихся в канале процессов

Идентификатор процесса	Описание процесса
0	Канал не активен, запрос еще не обрабатывается
1	SELECT
2	INSERT
3	UPDATE
4	DELETE
5	CREATE TABLE
6	CREATE VIEW
7	DROP TABLE
8	CREATE INDEX
9	DROP INDEX
10	GRANT: 1 form
11	GRANT: 2 form
12	REVOKE: 1 form
13	REVOKE: 2 form
14	ADD COLUMNS
15	ADD DATA FILE
16	ADD INDEX FILE
17	ADD BLOB FILE
18	MODIFY DATA FILE
19	MODIFY INDEX FILE
20	MODIFY BLOB FILE
21	DROP DATA FILE
22	DROP INDEX FILE
23	DROP BLOB FILE
24	PRESS TABLE
25	REBUILD TABLE
26	GETF (получить первую запись текущей выборки)
27	GETL (получить последнюю запись текущей выборки)
28	GETP (получить предыдущую запись текущей выборки)
29	GETN (получить следующую запись текущей выборки)
30	GETS (получить указанную запись текущей выборки)
31	UPDATE CURRENT CURSOR
32	DELETE CURRENT CURSOR
33	UPDATE CURRENT
34	DELETE CURRENT
35	CREATE NODE

Идентификатор процесса	Описание процесса
36	DROP NODE
37	CREATE EVENT
38	DROP EVENT
39	GET EVENT
40	WAIT EVENT
41	SET EVENT
42	CLEAR EVENT
43	CREATE ROLE
44	DROP ROLE
45	GRANT ROLE
46	REVOKE ROLE
47	GRANT USER
48	REVOKE USER
49	LOCK TABLE
50	UNLOCK TABLE
51	GET MANY ROWS
52	EXECUTE PROCEDURE
53	START AUDIT
54	STOP AUDIT
55	COMMIT
56	ROLLBACK
57	ALTER TABLE SET READ LEVEL | WRITE LEVEL
58	CREATE PROCEDURE
59	DROP PROCEDURE
60	ALTER PROCEDURE
61	CREATE GROUP
62	CREATE LEVEL
63	ALTER GROUP
64	ALTER LEVEL
65	GRANT ACCESS ON GROUP
66	REVOKE ACCESS ON GROUP
67	SOME AUDIT COMMAND
68	CREATE STATION
69	DROP STATION
70	ALTER STATION
71	CREATE DEVICE
72	DROP DEVICE
73	ALTER DEVICE
74	GRANT ACCESS ON STATION

Идентификатор процесса	Описание процесса
75	REVOKE ACCESS ON STATION
76	GRANT ACCESS ON DEVICE
77	REVOKE ACCESS ON DEVICE
78	DEBUG PROCEDURE
79	START APPEND
80	END APPEND
81	ALTER TABLE ADD PRIMARY KEY
82	ALTER TABLE ADD FOREIGN KEY
83	ALTER TABLE ADD UNIQUE
84	ALTER TABLE DROP PRIMARY KEY
85	ALTER TABLE DROP FOREIGN KEY
86	ALTER TABLE DROP UNIQUE
87	ALTER TABLE SET COLUMN READ LEVEL | WRITE LEVEL
88	CREATE REPLICATION RULE
89	DROP REPLICATION RULE
90	ALTER REPLICATION RULE
91	CREATE TRIGGER
92	DROP TRIGGER
93	ALTER TRIGGER
94	SET TRANSACTION READ ONLY
95	SET TRANSACTION ISOLATION LEVEL
96	SET SAVEPOINT
97	PUT SEVERAL ROWS
98	ALTER TABLE ALTER COLUMN SET DEFAULT
99	ALTER TABLE ALTER COLUMN DROP DEFAULT
100	ALTER TABLE RENAME COLUMN
101	ALTER TABLE ALTER COLUMN SIZE
102	ALTER TABLE RENAME
103	ALTER TABLE ADD CHECK
104	ALTER TABLE DROP CHECK
105	ALTER TABLE ALTER COLUMN ADD CHECK
106	ALTER TABLE ALTER COLUMN DROP CHECK
107	Создание составного индекса
108	Удаление составного индекса
109	ALTER TABLE ALTER COLUMN ENABLE NULL
110	ALTER TABLE ALTER COLUMN DISABLE NULL
111	SET PRIORITY
112	TEST TABLE
113	CREATE PHRASE INDEX

Идентификатор процесса	Описание процесса
114	DROP PHRASE INDEX
115	REBUILD PHRASE INDEX
116	BACKUP
117	CREATE FILTER
118	DROP FILTER
119	ALTER FILTER SET MODULE
120	SET DEFAULT FILTER FOR EXTENSION
121	CANCEL DEFAULT FILTER FOR EXTENSION
122	ALTER TABLE ALTER COLUMN SET DEFAULT FILTER
123	ALTER TABLE ALTER COLUMN CANCEL DEFAULT FILTER
124	SYNCHRONIZE RULE
125	ALTER TABLE ALTER COLUMN ADD RANGE
126	GET COMPLEX EVENT
127	WAIT COMPLEX EVENT
128	ALTER PHRASE INDEX
129	SET LOG
130	SET L_TTRUE COMMIT
131	SET ROW QUANT
132	SET INDEX QUANT
133	SET WORKSPACE LIMIT
134	CREATE TEMPORARY INDEX (INTERNAL)
135	CREATE SEQUENCE
136	DROP SEQUENCE
137	GRANT PROCEDURE
138	REVOKE PROCEDURE
139	CREATE CHARACTER SET
140	DROP CHARACTER SET
141	CREATE TRANSLATION
142	DROP TRANSLATION
143	SET NAMES
144	SET DATABASE NAMES
145	Один из процессов модификация параметров функционирования БД: SET RECORD SIZE LIMIT SET CHANNEL MEMORY LIMIT SET PHRASE INDEX FIRST CACHE SIZE SET PHRASE INDEX SECOND CACHE SIZE
146	BACKUP
147	SET CHANNEL AUTOCOMMIT
148	SET CHANNEL WAIT|NOWAIT
149	PURGE TABLE

Идентификатор процесса	Описание процесса
150	ADD BLOB
151	CLEAR BLOB
152	GET LINTER DUMP
153	CREATE ALIAS
154	DROP ALIAS
155	ALTER CHARACTER SET SET DESCRIPTION
156	ALTER CHARACTER SET DROP DESCRIPTION
157	SET DATABASE DEFAULT CHARACTER SET
158	SET OPTIMIZATION ENABLE/DISABLE
159	TRUNCATE TABLE
160	SET CONSTRAINTS ALL DEFERRED/IMMEDIATE
161	SET TRANSACTION READ WRITE
162	SET SORTPOOL LIMIT/UNLIMITED
163	TEST INTERNAL DATA
164	SET SESSION BLOB LOG {OFF|ON}
165	SAVE TABLE
166	RESTORE TABLE
167	ALTER TABLE [NOT] IN-MEMORY
168	SET SESSION PROCEDURE EXECUTE BY DEFAULT AS {DEFINER|CURRENT_USER}
169	SET DATABASE GEODATA VALIDITY CHECKING ON/OFF
170	SET CONNECTION GEODATA VALIDITY CHECKING ON/OFF
171	SET SESSION DEFAULT SECURITY
172	ALTER SEQUENCE
173	SET SESSION QUANT TIMEOUT {LIMIT|UNLIMITED}
174	SET QUANT TIMEOUT {LIMIT|UNLIMITED} FOR USER
175	ALTER TABLE RENAME INDEX
176	SET DATABASE QUANTUM
177	SET SESSION QUANTUM
178	CORRECT INDEX|CONVERTER|BITMAP
179	Канал ожидает отсылки данных утилите LHB
180	ALTER PROCEDURE DROP SOURCE TEXT
181	ALTER TRIGGER DROP SOURCE TEXT
182	ALTER TABLE ENABLE/DISABLE KEY
183	ALTER EVENT
222	ALTER TABLE {SET|CANCEL} RECORDS LIMIT
223	EXECUTE BLOCK
225	DROP COLUMNS
226	PREPARE
227	MERGE

Идентификатор процесса	Описание процесса
229	SET SCHEMA
230	CREATE COMMENT
231	CREATE VARIABLE
232	DROP VARIABLE

Приложение 12

Пример анализа результатов трансляции хранимой процедуры

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inter.h"
#include "exlib.h"

#define SIZE 8192


static void    PrintDescErr(TCBL * pCBL, L_LONG  CodErr);
static L_BYTE    Load(L_CHAR * Buf, const L_CHAR * fname, L_LONG  size);
static void    ProcPrintError(TCBL * pCBL);

#ifndef WINCE
int    main(int argc, const char * argv[])
#else
int    excreat()
#endif
  {
  TCBL    CBLconnect;
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;
  L_CHAR    Buf[SIZE];

  if (argc != 2)
    {
    printf("Usage:ExCreat input_file\n");
    exit(1);
    }

  memset(&CBLconnect,0,sizeof(TCBL));
  Err = LinterOPEN(&CBLconnect, Name_Pass, Node, Priority, PrzExe);
  if (Err != NORMAL)
    ProcPrintError(&CBLconnect);
  printf("Connect to RDBMS Linter\n");

  if (Load(Buf, argv[1], SIZE))
    {
```



```
    Err = LinterNotSelect(&CBLconnect, Buf);
    if (Err != NORMAL)
      ProcPrintError(&CBLconnect);
    printf("created/alterd procedure\n");
    }
  else
    printf("can't Load text from file %s", argv[1]);

  printf("End Example\n");

  return 0;
  }

static void    PrintDescErr(TCBL * pCBL, L_LONG  CodErr)
  {
  L_CHAR    Str[256];

  if (NORMAL != LinterGETE(pCBL, CodErr, Str, 255))
    {
    printf("LinterGETE -- Error=%ld\n", pCBL->CodErr);
    exit(1);
    }
  printf("%s", Str);
  }

static void  ProcPrintError(TCBL * pCBL)
  {
  ERRS_DESCR *ErrDesc;
  L_BYTE    Buf[4096];
  L_WORD    Len,
          CntErr,
          i;
  L_WORD    dp;
  L_CHAR    cp[101];
  L_LONG  Err;

  if (pCBL->CodErr != NORMAL)
    {
    printf("Linter Error: %ld\n", pCBL->CodErr);
    if (pCBL->CodErr != Proc_Translation_Error)
      exit(1);
    }
  else
    return;

  Err = LinterGETE(pCBL, Proc_Translation_Error, Buf, sizeof(Buf));
```

```
  if (Err != NORMAL)
    {
    printf("ProcPrintError -- Linter error %ld\n", Err);
    exit(1);
    }

  CntErr = *((L_WORD *) Buf);
  ErrDesc = (ERRS_DESCR *) (Buf + sizeof(L_WORD));

  for (i = 0; i < CntErr; i++)
    {
    if (ErrDesc->Row != -1)
      printf("Error in line %ld position %ld: ",
     ErrDesc->Row, ErrDesc->Position);
    else
      printf("Error: ");
    switch (ErrDesc->ArgType)
      {
      case PROC_ERR_ARGSHORT:
        dp = *((L_WORD *) (ErrDesc + 1));
        PrintDescErr(pCBL, ErrDesc->CodErr + PROC_ERR_BASE);
        printf(": %d", dp);
        ErrDesc = (ERRS_DESCR *) ((L_WORD *) (ErrDesc + 1) + 1);
        break;
      case PROC_ERR_ARGSTRING:
        Len = *((L_WORD *) (ErrDesc + 1));
        memcpy(cp, (L_WORD *) (ErrDesc + 1) + 1, Len);
        cp[Len] = 0;
        PrintDescErr(pCBL, ErrDesc->CodErr + PROC_ERR_BASE);
        printf(": %s", cp);
        ErrDesc = (ERRS_DESCR *) ((L_BYTE *) (ErrDesc + 1)
             + sizeof(L_WORD) + Len);
        break;
      default:
        PrintDescErr(pCBL, ErrDesc->CodErr + PROC_ERR_BASE);
        ErrDesc++;
        }
    printf("\n");
    }
  printf("End Example\n");
  exit(1);
  }
static L_BYTE  Load(L_CHAR * Buf, const L_CHAR * fname, L_LONG  size)
  {
  L_LONG  ret;
  FILE   *file;
```



```
file = fopen(fname, "r");
if (file == NULL)
  {
  printf("Error: can't open file %s\n", fname);
  exit(1);
  }

ret = fread(Buf, size, 1, file);
if (ret > 0)
  ret = feof(file);

fclose(file);

if (ret)
  return 0;

return 1;
}
```

Приложение 13

Коды исключений системы исполнения хранимых процедур и триггеров

Коды всех исключений (системных и пользовательских), возвращаемых системой исполнения хранимых процедур и триггеров, имеют отрицательное значение. С помощью этого признака их можно, при необходимости, отделить от кодов завершения, возвращаемых ядром СУБД ЛИНТЕР, которые всегда имеют положительное значение.

Имя	Код	Причина исключения	Комментарий
EXC_NONE	0	Нормальное завершение	
EXC_DIVZERO	-2	Деление на нуль	
EXC_UNDEFPROC	-3	Неизвестная процедура	В операторе вызова процедуры **CALL PROC** указано имя несуществующей в БД процедуры
EXC_BADPARAM	-4	Неправильный параметр	Несоответствие количества или типов данных объявленных и переданных параметров
EXC_BADINDEX	-5	Неправильный индекс	Для типа данных BYTE[i] использован недопустимый номер элемента
EXC_BADRETVAL	-6	Неправильное возвращаемое значение	Несоответствие типа данных объявленного и переданного выходного параметра
EXC_NULLDATA	-7	Недопустимое использование NULL-значений	В арифметической операции или операции сравнения (на неравенство) один из операндов имеет NULL-значение
EXC_NOMEM	-8	Нет свободной памяти	Исчерпана свободная дисковая или оперативная память системы исполнения хранимых процедур и триггеров. Эта ситуация возникает, как правило, при неограниченном рекурсивном вызове процедуры (триггера)
EXC_BADCURSOR	-9	Неправильный курсор	Несоответствие между объявленным курсором и фактически выполненным SELECT-запросом (по

Имя	Код	Причина исключения	Комментарий
			количеству выбираемых полей или по типам данных)
EXC_CURNOTOPEN	-10	Курсор не открыт	Попытка выполнить операторы **FETCH**, **CLOSE** для неоткрытого курсора
EXC_BADCODE	-11	Плохой код процедуры	Испорчен или не соответствует данной версии СУБД ЛИНТЕР оттранслированный код процедуры (триггера). Для исправления ошибки следует заново оттранслировать процедуру (триггер)
EXC_TRIGQUERY	-12	Зарезервирован	
EXC_ APPENDNOTSTARTED	-14	Пакетное добавление не активировано	Попытка выполнить команду **PUTM** или **END APPEND** без предварительного выполнения команды **START APPEND**
EXC_ QUERYWHENAPPEND	-15	Недопустимая операция при пакетном добавлении	Подан запрос, отличный от **PUTM** и **END APPEND**, в процессе пакетного добавления
EXC_ APPENDACTIVE	-16	Повторное инициирование пакетного добавления	Попытка повторно выполнить команду **START APPEND**, когда пакетное добавление уже идет
EXC_ APPLICATIONERROR	-17	Зафиксировано пользовательское исключение	Исключение сгенерировано вызовом **RAISE_ERROR**()
EXC_INVTRSTATE	-18	Ошибка в выполнении транзакционного оператора в хранимой процедуре	Проблемы при выполнении в хранимых процедурах операторов **BEGIN TRANSACTION**, **COMMIT**, **ROLLBACK**
EXC_CUSTOM	-100	Реально данное исключение не генерируется	Определяет границу системных и пользовательских исключений

Приложение 14
Пример выполнения хранимой процедуры

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "decimals.h"
#include "inter.h"
#include "tick.h"
#include "exlib.h"

static void AnalizeResult(L_BYTE * Buf);
static void show_value(L_CHAR * text, ARGPROC_OUT * ParInfo,
 L_BYTE * Buf);
static void show_except(L_LONG  except);

#ifndef WINCE
int    main(int argc, const char * argv[])
#else
int    exexec()
#endif
  {
  TCBL    CBL;
  L_BYTE    Buf[8192];
  L_CHAR    Name_Pass[] = "SYSTEM/MANAGER8";
  L_CHAR    Node[] = "        ";
  L_WORD    Priority = 0;
  L_LONG  PrzExe = M_EXCLUSIVE | Q_ENCODE | M_BINARY;
  L_LONG  Err;

  if (argc < 2)
    {
    fprintf(stderr, "Usage: exeproc <proccaller>\n");
    exit(-1);
    }

  memset(&CBL,0,sizeof(TCBL));
  Err = LinterOPEN(&CBL, Name_Pass, Node, Priority, PrzExe);
  if (Err != NORMAL)
    {
    fprintf(stderr, "Error open channel: %d\n", CBL.CodErr);
    exit(-1);
    }

  sprintf((L_CHAR *) Buf, "execute %s;", argv[1]);
```



```
  Err = LinterExec(&CBL, (L_CHAR *) Buf, Buf, sizeof(Buf));
  if (Err != NORMAL)
    {
    printf("Linter error code %d\n", Err);
    if (Err == Proc_Raised_Exception)
      show_except(CBL.SysErr);
    }
  if (Err == NORMAL || Err == Proc_Raised_Exception)
    AnalizeResult(Buf);

  LinterCLOS(&CBL);

  return 0;
  }

void    AnalizeResult(L_BYTE * Buf)
  {
  ARGPROC_OUT *ParInfo;
  L_WORD   *pCntArg;
  L_WORD    i;
  L_CHAR    str[100];
  L_WORD    Len;
  L_BYTE   *ptr;

  ParInfo = (ARGPROC_OUT *) Buf;
  show_value("Return value", ParInfo, Buf);
  pCntArg = (L_WORD *) (ParInfo + 1);
  ParInfo = (ARGPROC_OUT *) (pCntArg + 1);

  for (i = 0; i < *pCntArg; ++i)
    {
    if (ParInfo->Flags & PARM_FLAG_NAMESENT)
      {
      ptr = Buf + ParInfo->Value;
      Len = *((L_WORD *) ptr);
      memcpy(str, ptr + sizeof(L_WORD), Len);
      str[Len] = 0;
      ParInfo->Value += sizeof(L_WORD) + Len;
      }
    else
      sprintf(str, "Parameter number %d value",
         (int) ParInfo->Reserv);
    show_value(str, ParInfo, Buf);
    ParInfo++;
    }
  }
```

```
void    show_value(L_CHAR * text, ARGPROC_OUT * ParInfo, L_BYTE *
 Buf)
  {
  L_BYTE   *ptr = Buf + ParInfo->Value;
  CR_INFO *cr;
  L_WORD    i;

  L_CHAR    StrBuf[4000];
  L_WORD    SmallBuf;
  L_LONG  IntBuf;
  L_REAL    RealBuf;
  L_DOUBLE DblBuf;
  L_DECIMAL DecBuf;
  L_DOUBLE NumBuf;

  printf(text);
  printf(" = ");
  if (ParInfo->Flags & PARM_FLAG_ISNULL)
    printf("NULL");
  else if (ParInfo->Flags & PARM_FLAG_ISCURSOR)
    {
    cr = (CR_INFO *) ptr;
    printf("Cursor: NMRKAN = %d, RowId = %ld, KOLKOR = %ld,
        LNBUFKOR = %d, CODERR = %ld\n", cr->NumChan, cr->RowId,
           cr->RowCount, cr->LnBufRow, cr->CodErr);
    }
  else
    switch (ParInfo->Type.ntyp)
      {
      case DT_CHAR:
        memcpy(StrBuf, ptr, ParInfo->Type.length);
        StrBuf[ParInfo->Type.length] = 0;
        printf(StrBuf);
        break;
      case DT_INTEGER:
        if (ParInfo->Type.length == sizeof(L_WORD))
          {
          memcpy(&SmallBuf, ptr, sizeof(L_WORD));
          printf("%d", SmallBuf);
          }
        else
          {
          memcpy(&IntBuf, ptr, sizeof(L_LONG ));
          printf("%ld", IntBuf);
          }
```



```
        break;
      case DT_REAL:
        if (ParInfo->Type.length == sizeof(L_REAL))
          {
          memcpy(&RealBuf, ptr, sizeof(L_REAL));
          printf("%g", RealBuf);
          }
        else
          {
          memcpy(&DblBuf, ptr, sizeof(L_DOUBLE));
          printf("%g", DblBuf);
          }
        break;
      case DT_DATE:
        memcpy(DecBuf, ptr, sizeof(L_DECIMAL));
        TICKTOSTRF(DecBuf, "dd.mm.yyyy:hh:mi:ss", StrBuf);
        printf(StrBuf);
        break;
      case DT_DECIMAL:
        memcpy(DecBuf, ptr, sizeof(L_DECIMAL));
        DecToDbl(DecBuf, &NumBuf);
        printf("%g", NumBuf);
        break;
      case DT_BYTE:
        for (i = 0; i < ParInfo->Type.length; ++i)
          printf("%2x ", (L_WORD) ptr[i]);
        break;
      case DT_BLOB:
        printf("@:BLOB:@");
        break;
        }
  printf("\n");
  }

void    show_except(L_LONG  except)
  {
  switch (except)
    {
 case EXC_DIVZERO:
      printf("DIVZERO");
      break;
    case EXC_UNDEFPROC:
      printf("UNDEFPROC");
      break;
    case EXC_BADPARAM:
      printf("BADPARAM");
```

```
      break;
    case EXC_BADRETVAL:
      printf("BADRETVAL");
      break;
    case EXC_NULLDATA:
      printf("NULLDATA");
      break;
    case EXC_NOMEM:
      printf("NOMEM");
      break;
    case EXC_BADCURSOR:
      printf("BADCURSOR");
      break;
    case EXC_CURNOTOPEN:
      printf("CURNOTOPEN");
      break;
    case EXC_BADCODE:
      printf("BADCODE");
      break;
    case EXC_TRIGQUERY:
      printf("SQL is still not supported in triggers");
      break;
    case EXC_APPENDNOTSTARTED:
      printf("APPENDNOTSTARTED");
      break;
    case EXC_QUERYWHENAPPEND:
      printf("QUERYWHENAPPEND");
      break;
    case EXC_APPENDACTIVE:
      printf("APPENDACTIVE");
      break;
    case EXC_APPLICATIONERROR:
      printf("APPLICATIONERROR");
      break;
    case EXC_INVTRSTATE:
      printf("Invalid transaction state");
      break;
    default:
      if (except <= EXC_CUSTOM)
        printf("CUSTOM (%d)", -except+EXC_CUSTOM+1);
      else
        printf("%d", except);
      break;
      }
  printf(" exception has been raised\n");
  }
```


Указатель команд