

МОБИЛЬНАЯ  
РЕЛЯЦИОННАЯ  
СУБД

**ЛИНТЕР®**

Linter Standard  
Linter Bastion  
Linter RealTime  
Linter Multiversion

**QT-интерфейс**

НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

---

 **РЕЛЭКС®**

## **Товарные знаки**

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

## **Интеллектуальная собственность**

Правообладателем продуктов ЛИНТЕР®, НЕВОД®, ЛАВ™, ЛАКУНА является компания РЕЛЭКС (1990–2011). Все права защищены. Данный документ является собственностью компании РЕЛЭКС. Ни одна часть данного документа не может быть воспроизведена, передана, преобразована, сохранена в системе поиска информации, переведена на другой язык или компьютерный язык в какой-либо форме, какими-либо средствами, электронными, механическими, магнитными, оптическими, химическими, ручными или иными без предварительного разрешения компании РЕЛЭКС.

## **О документе**

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

## **Адрес**

394006, г. Воронеж, ул. 20-летия Октября, 119.  
Тел./факс: (473) 2-711-711, 2-778-333.  
e-mail: market@relex.ru.

## **Адрес для корреспонденции**

394000, г. Воронеж, а/я 137.

## **Техническая поддержка**

Отдел поддержки и сопровождения программных продуктов:

телефон: (473) 2-711-711 с 9:00 до 18:00 мск.  
e-mail: support@relex.ru, market@relex.ru.

С целью повышения качества разрабатываемых программных средств и предоставляемых услуг в компании РЕЛЭКС действует автоматизированная система учёта и обработки рекламаций. Обо всех обнаруженных недостатках и ошибках в программном продукте и/или документации на него просим сообщать нам на Internet–странице [рекламация](#).

# Оглавление

<b>Предисловие</b> .....	<b>1</b>
Назначение документа.....	1
Для кого предназначен документ.....	1
Необходимые предварительные знания.....	1
Принятые обозначения и соглашения.....	1
Дополнительные документы.....	3
<b>Общие сведения о QT-библиотеке</b> .....	<b>4</b>
Необходимые условия применения.....	4
Использование в среде UNIX.....	4
Сборка QTLinter-драйвера.....	4
Установка QTLinter-драйвера.....	5
Использование в среде Win32.....	5
Сборка QTLinter-драйвера.....	5
Установка QTLinter-драйвера.....	5
<b>QTLinter–интерфейс</b> .....	<b>6</b>
Установка соединения с БД.....	6
Инициализация соединения.....	6
Создание объекта-соединения.....	6
Создание копии объекта-соединения.....	6
Уничтожение объекта-соединения.....	7
Конфигурирование соединения.....	7
Подключить драйвер базы данных.....	7
Установить параметры соединения.....	7
Объявить имя ЛИНТЕР–сервера.....	9
Объявить пользователя соединения.....	10
Установить пароль пользователя.....	10
Просмотр параметров соединения.....	11
Получить параметры соединения.....	11
Получить имя ЛИНТЕР–сервера.....	12
Получить пользователя соединения.....	12
Получить имя драйвера.....	13
Проверить доступность драйвера.....	13
Проверить состояние соединения.....	13
Проверить результат соединения с БД.....	14
Проверить существование соединения.....	14
Получить пароль соединения.....	15
<b>Управление соединением</b> .....	<b>16</b>
Установить сконфигурированное соединение.....	16
Установить параметризуемое соединение.....	16

## Оглавление

---

Закрыть соединение .....	17
Начать транзакцию.....	17
Подтвердить транзакцию .....	18
Удалить именованное соединение .....	19
Откатить транзакцию .....	19
<b>Поддерживаемые типы данных .....</b>	<b>20</b>
<b>Обработка SQL-запросов.....</b>	<b>21</b>
<b>Получение информации об объектах БД .....</b>	<b>24</b>
Получить список таблиц БД .....	24
Получить описание первичного ключа таблицы .....	25
Получить описание строки таблицы.....	26
<b>Обработка кодов завершения.....</b>	<b>27</b>
Получить последний код завершения.....	27

# Предисловие

## Назначение документа

Документ содержит описание QTELintер–драйвера, выполняющего доступ к СУБД ЛИНТЕР из приложений, разработанных с использованием среды разработки QT.

Документ может использоваться для работы с любой версией СУБД ЛИНТЕР. Особенности конкретных версий оговариваются по тексту.

## Для кого предназначен документ

Документ предназначен для программистов, разрабатывающих приложения в среде разработки QT с использованием СУБД ЛИНТЕР.

## Необходимые предварительные знания

Для работы необходимо знать;



- знать основы реляционных баз данных и языка баз данных SQL;
- классы и методы модуля SQL среды разработки QT;
- уметь работать в соответствующей операционной системе на уровне простого пользователя.

## Принятые обозначения и соглашения

<u>Обозначение</u>	<u>Пример</u>	<u>Значение</u>
Курсив	<i>Растровым</i> называется изображение...	Новый термин в тексте
Полужирный шрифт	В этом случае необходимо переносить <b>все</b> физические файлы.	Выделение в тексте
Подчеркнутый шрифт	Подробную информацию о работе программы можно получить на сайте <a href="http://www.dmk.ru">www.dmk.ru</a> .	Адреса страниц Internet
Текст, разделенный знаком ⇒	Выполните команду <b>View ⇒ Properties</b> (Вид ⇒ Свойства).	Последовательность выполнения команд
Текст, заключенный в <>, со знаком + между ними	<Ctrl>+<C>	В <> заключаются клавиши клавиатуры, знак + означает сочетание клавиш
Крупный моноширинный текст	SQL> _q	Текст командной строки
Мелкий моноширинный текст	Page Time Count	Текст программы

## Предисловие

---

<u>Обозначение</u>	<u>Пример</u>	<u>Значение</u>
Заглавные буквы	BROWSE	Названия команд, слова, зарезервированные в SQL, ключевые слова
Курсив в < >	<return statement>	Определяемый элемент синтаксической конструкции
Символ ::=		Равенство по определению. Слева от знака стоит определяемое понятие, справа – собственно определение понятия
Квадратные скобки [ ]	DBSTORE [-d -r -t -u]	Необязательные элементы конструкции. В данном примере ключи не являются обязательными элементами команды
Вертикальная черта	<return value> ::= <value expression>   NULL	Указывает на то, что все предшествующие ей элементы списка являются необязательными и могут быть заменены любым другим элементом списка после этой черты
Фигурные скобки { }	CODEPAGE {866   1251   KOI8}	Указывают на то, что все находящееся внутри них является единым целым
Многоточие «...»	Характеристики столбца MAKE CHAR(20) MODEL CHAR(20) ... SQL>	Означает, что предшествующая часть может быть повторена любое количество раз
Многоточие, внутри которого находится запятая «,...»		Указывает, что предшествующая часть оператора, состоящая из нескольких элементов, разделенных запятыми, может иметь произвольное число повторений
Текст со знаком  на сером фоне	 Если конфигурация страницы-шаблона не учитывала свойств, команда будет выполнена некорректно.	Примечание

## Дополнительные документы

- СУБД ЛИНТЕР. Архитектура СУБД.
- СУБД ЛИНТЕР. Справочник по SQL.
- СУБД ЛИНТЕР. Справочник кодов завершения.
- документация на систему разработки QT (например, на сайте <http://doc.trolltech.com/4.1/index.html>)

# Общие сведения о QT-библиотеке

QT<sup>1</sup>-библиотека – мультиплатформенная C++ библиотека для разработки высококачественных графических интерфейсов приложений. QT является полностью объектно-ориентированной, легко расширяемой и простой в применении библиотекой. QT включает в себя большой набор средств, облегчающих процесс разработки приложений, наиболее важными из которых являются QT Designer, средство для визуального создания графического интерфейса приложений, и QTSQL, средство для создания платформенно-независимых приложений для работы с базами данных.

QT включает «родные» драйвера для Oracle, Microsoft SQL Server, Sybase Adaptive Server, IBM DB2, PostgreSQL, MySQL и ODBC-совместимых баз данных.

Драйвер QT для СУБД ЛИНТЕР расширяет список баз данных, с которыми поддерживается QT-интерфейс. Для работы с СУБД ЛИНТЕР драйвер использует модуль QTSQL, который присутствует в QT 3.x.x Enterprise Edition. Выполнение подготовленных запросов возможно начиная с QT версии 3.1.x. Выходные параметры поддерживаются, начиная с QT версии 3.2.x

Драйвер предназначен для работы только с QT 3.x, Qt 4.x. Qt 2.x не поддерживается.

Функциональность QTSQL полностью интегрирована с QT Designer, который может отображать данные из БД «вживую». QT включает специфичные для БД виджеты, а также поддерживает расширение для работы с БД любых встроенных или отдельно написанных виджетов.

Приложения на QT могут расширять свою функциональность с помощью модулей («plugins») и динамических библиотек. Модули включают дополнительные кодеки, драйвера баз данных, форматы изображений, стили и виджеты. Библиотеки могут предложить неограниченное расширение функциональности.

## Необходимые условия применения

Для выполнения QT-приложения на компьютере должен быть установлен полный пакет среды разработки QT.

## Использование в среде UNIX

### Сборка QTlinter-драйвера

Сборка QTlinter-драйвера выполняется вручную.

Для этого необходимо:

1. установить (если это не было сделано ранее) среду разработки QT;
2. установить на компьютере СУБД ЛИНТЕР (см. документ «Установка СУБД ЛИНТЕР на платформе типа Unix»);
3. проверить наличие переменной окружения QTDIR. Эта переменная создается при установке на компьютере среды разработки QT и должна содержать путь к установочному каталогу среды разработки QT;

---

<sup>1</sup> Аббревиатура QT образована от названия норвежской компании Quasar Technologies, которая была родоначальником разработки данного программного средства.

4. перейти в подкаталог `/linqt` установочного каталога СУБД ЛИНТЕР и выполнить команды:

```
qmake linqt.pro  
make
```

В результате в подкаталоге `/linqt` будет создан QTLinter-драйвер: `libqsqlinter.so`.

## Установка QTLinter-драйвера

Установка QTLinter-драйвера выполняется вручную.

Для этого необходимо разместить собранную версию QTLinter-драйвера в установочный каталог QT в подкаталог `/plugins/sqldrivers`.

## Использование в среде Win32

### Сборка QTLinter-драйвера

Сборка QTLinter-драйвера выполняется вручную.

Для этого необходимо:

1. установить (если это не было сделано ранее) среду разработки QT;
2. установить компилятор (например пакет MinGW);
3. установить на компьютере СУБД ЛИНТЕР (см. документ «Установка СУБД ЛИНТЕР на платформе Win32»);
4. проверить наличие переменных окружения:
  - QTDIR. Эта переменная должна содержать путь к установочному каталогу среды разработки QT;
  - Path. Эта переменная должна содержать путь к подкаталогу `\bin` установочного каталога QT, путь к подкаталогу `\bin` установочного каталога Линтер, путь к подкаталогу `\bin` установочного каталога компилятора (например MinGW);
  - QMAKESPEC. Эта переменная должна содержать компилятор, который будем использовать (например `win32-g++`).
5. перейти в подкаталог `\linqt` установочного каталога СУБД ЛИНТЕР и выполнить команды:

```
qmake linqt.pro  
mingw32-make (в зависимости от компилятора)
```

В результате в подкаталоге `\linqt` будет создан QTLinter-драйвер: `qsqlinter.dll`.

### Установка QTLinter-драйвера

Установка QTLinter-драйвера выполняется вручную.

Для этого необходимо разместить собранную версию QTLinter-драйвера в установочный каталог QT в подкаталог `\plugins\sqldrivers`.

# QTLinter–интерфейс

## Установка соединения с БД

Логический доступ к БД ЛИНТЕР из QT-приложения реализуется с помощью методов (функций) абстрактного класса QSqlDatabase, входящего в состав среды разработки QT. Реальный доступ к данным БД и управление пользовательскими SQL–запросами выполняет драйвер Qtlinter, входящий в состав СУБД ЛИНТЕР.

## Инициализация соединения

### Создание объекта-соединения

#### Синтаксические правила

```
QSqlDatabase::QSqlDatabase ();
```

#### Описание

Создает пустой QSqlDatabase-объект. Для дальнейшего использования объекта к нему необходимо применить методы addDatabase(), removeDatabase() и database().

#### Возвращаемое значение

Указатель на созданный QSqlDatabase-объект или NULL.

#### См. также

addDatabase(), removeDatabase() и database().

#### Пример

```
QSqlDatabase * db = QSqlDatabase::QSqlDatabase ();
```

### Создание копии объекта-соединения

#### Синтаксические правила

```
QSqlDatabase::QSqlDatabase ( <объект> )
```

Объект

Указатель на существующий QSqlDatabase-объект.

#### Описание

Создает копию заданного QSqlDatabase-объекта.

#### Возвращаемое значение

Указатель на созданную копию QSqlDatabase-объекта.

#### Пример

```
QSqlDatabase * conn_user1 = QSqlDatabase::QSqlDatabase ();  
QSqlDatabase * conn_user2 = QSqlDatabase::QSqlDatabase (conn_user1);
```

## Уничтожение объекта-соединения

### Синтаксические правила

```
QSqlDatabase::~~QSqlDatabase ()
```

### Описание

Уничтожает созданный объект-соединение и освобождает все выделенные ему ресурсы. Если уничтоженный объект был последним объектом-соединением, который использовался для доступа к БД, то соединение с БД автоматически закрывается.

### Возвращаемое значение

Нет.

### См. также

`close()`.

## Конфигурирование соединения

### Подключить драйвер базы данных

#### Синтаксические правила

```
QSqlDatabase::addDatabase (<имя драйвера> [, <имя соединения>]);
```

<имя драйвера> ::= символьный литерал или символьная переменная

<имя соединения> ::= внутреннее имя данного соединения в QT, в СУБД ЛИНТЕР не используется.

### Описание

Устанавливает тип драйвера, через который должен выполняться доступ к БД по данному соединению. Для СУБД ЛИНТЕР имя драйвера должно быть «QLINTER».

### Возвращаемое значение

Нет.

### Пример

```
QSqlDatabase * db = QSqlDatabase::addDatabase( "QLINTER" );
```

### См. также

`database ()`, `removeDatabase ()`.

## Установить параметры соединения

### Синтаксические правила

```
void QSqlDatabase::setConnectOptions (<параметры> )
```

<параметры> ::= символьный литерал или символьная переменная

### Описание

Устанавливает (или изменяет) специфичные для СУБД параметры соединения с БД. Эта операция должна выполняться до открытия соединения с БД, иначе она не имеет смысла.

Изменить параметры соединения можно и таким образом:

- закрыть соединение (функция `close()`);
- вызвать данную функцию для установки нужных параметров соединения;
- заново открыть соединение (функция `open()`).

Символьная строка, задающая значение аргумента <параметры>, должна иметь формат:  
<опция> [;<опция>...].

Значение <опция> зависит от СУБД, к которой выполняется соединение.

Для СУБД ЛИНТЕР допустимы следующие значения <опций>:

Обозначение опции	Значение опции	Примечание
AUTOCOMMIT	Задает режим транзакций	Значение по умолчанию.
OPTIMISTIC	Задает режим транзакций	
EXCLUSIVE	Задает режим транзакций	
READ_UNCOMMITTED	Задает режим транзакций	начиная с версии 6.1
READ_COMMITTED	Задает режим транзакций	начиная с версии 6.1
SERIALIZABLE	Задает режим транзакций	начиная с версии 6.1
ANSI	Задает кодировку соединения	
KOI8	Задает кодировка соединения	Значение по умолчанию.

 Если аргумент <параметры> содержит несколько однотипных опций (например, "PESSIMISTIC;AUTOCOMMIT;READ\_COMMITTED"), то будет использоваться последняя опция.

### Возвращаемое значение

Нет.

### Примеры

```
...
// Соединение с СУБД ЛИНТЕР
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
db.setUsername( "SYSTEM" );
db.setPassword( "MANAGER" );
db.setConnectOptions("OPTIMISTIC ;KOI8"); // установить опции соединения
if (!db.open()) {
    db.setConnectOptions(); // очистить строку опций соединения
}

// Соединение с MySQL
db.setConnectOptions("CLIENT_SSL=1;CLIENT_IGNORE_SPACE=1");
if (!db.open()) {
    db.setConnectOptions();
}
```

```

}

// Соединение через ODBC
db.setConnectOptions("SQL_ATTR_ACCESS_MODE=SQL_MODE_READ_ONLY;SQL_ATTR_TRACE=SQL_OPT_TRACE_ON");
if (!db.open()) {
    db.setConnectOptions();
}

```

### См. также

connectOptions().

## Объявить имя ЛИНТЕР-сервера

### Синтаксические правила

```
void QSqlDatabase::setDatabaseName (<имя>)
```

<имя> ::= символьный литерал или символьная переменная длиной не более 8 символов

### Описание

Задаёт ЛИНТЕР-сервер, к которому должен быть установлен доступ через соединение. ЛИНТЕР-сервер должен быть определен в файле сетевой конфигурации nodetab (см. документ “СУБД ЛИНТЕР. Сетевые средства”).

Назначение ЛИНТЕР-сервера должно выполняться до открытия соединения, в противном случае делать это не имеет смысла. Если же соединение уже открыто, необходимо выполнить метод close(), затем данную функцию и после этого снова открыть соединение с помощью функции open().

Если ЛИНТЕР-сервер не был установлен, соединение будет установлено с локальным ЛИНТЕР-сервером по умолчанию.

### Возвращаемое значение

Нет.

### Пример

```

QSqlDatabase db;
db.setDatabaseName("conn_DB_SALE"); // Объявить имя сервера
db.setUserName("SYSTEM");
db.setPassword("MANAGER");
db.open();

QSqlQuery query(db);
query.exec("SELECT count(*), id FROM AUTO WHERE make = 'BMW'");

```

### См. также

databaseName(), setUsername(), setPassword(), setHostName(), setPort(), setConnectOptions(), open().

# Объявить пользователя соединения

## Синтаксические правила

```
void QSqlDatabase::setUserName ( <пользователь> )
```

<пользователь> ::= символьный литерал или символьная переменная длиной не более 66 символов

## Описание

Задаёт пользователя, от имени которого должен выполняться доступ к БД (может быть пустым).

Назначение пользователя соединения должна выполняться до открытия соединения, в противном случае делать это не имеет смысла. Если же соединение уже открыто, необходимо вызвать функцию `close()`, затем данную функцию и после этого снова открыть соединение с помощью функции `open()`.

Значения по умолчанию нет.

## Возвращаемое значение

Нет.

## Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
db.setUserName( "SYSTEM" );
cout << "User name " << db.userName().toLocal8Bit().data() << "\n";
db.setPassword( "MANAGER" );
cout << "Password " << db.password().toLocal8Bit().data() << "\n";

if( !db.open() )
{
cout << db.lastError().driverText().toLocal8Bit().data() << endl;
return 1;
}
```

## См. также

`userName()`, `setDatabaseName()`, `setPassword()`, `setHostName()`.

# Установить пароль пользователя

## Синтаксические правила

```
void QSqlDatabase::setPassword ( <пароль> )
```

<пароль> ::= символьный литерал или символьная переменная длиной не более 18 символов


## Описание

Задаёт пароль, который должен использоваться при авторизации доступа к БД по заданному соединению (может быть пустым).

Назначение пароля должно выполняться до открытия соединения, в противном случае делать это не имеет смысла. Если же соединение уже открыто, необходимо вызвать

функцию `close()`, затем данную функцию и после этого снова открыть соединение с помощью функции `open()`.

Значения по умолчанию нет.

 Использование данной функции приводит к запоминая пароля внутри QT-интерфейса, что чревато его вскрытием. Чтобы избежать этого, необходимо использовать функции `open()`, в которой пароль передается как аргумент. В этом случае значение пароля в QT-интерфейсе задействовано только на время выполнения операции открытия соединения.

### Возвращаемое значение

Нет.

### Пример

См. `setDatabaseName()`.

### См. также

`password()`, `setUserName()`, `setDatabaseName()`, `setHostName()`, `setPort()`, `setConnectOptions()`, `open()`

## Просмотр параметров соединения

### Получить параметры соединения

#### Синтаксические правила

```
QSqlDatabase::connectOptions () const
```

#### Описание

Предоставляет параметры указанного соединения.

#### Возвращаемое значение

Символьная строка переменной длины, содержащая список опций соединения, разделенных знаком «; » (точка с запятой).

Возможные значения опций приведены в описании функции `setConnectOptions()`.

Возвращаемая строка может быть пустой (это означает, что используются опции по умолчанию).

#### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
db.setUserName( "SYSTEM" );
db.setPassword( "MANAGER" );
db.setConnectOptions("OPTIMISTIC ;KOI8");
cout << "options " << db.connectOptions().toLocal8Bit().data() << "\n";
```

В буфере будет возвращено `AUTOCOMMIT;KOI8;`

### Получить имя ЛИНТЕР-сервера

#### Синтаксические правила

```
QSqlDatabase::databaseName ();
```

#### Описание

Предоставляет имя ЛИНТЕР-сервера, к которому должен выполняться доступ по данному соединению.

#### Возвращаемое значение

Символьная строка длиной 8 символов. Если соединение установлено с ЛИНТЕР-сервером по умолчанию, строка содержит пробелы.

#### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );  
db.setDatabaseName( "conn_DB_SALE " );  
db.setUserName( "SYSTEM" );  
db.setPassword( "MANAGER" );  
cout << "databaseName " << db.databaseName().toLocal8Bit().data() << "\n";
```

#### См. также

`setDatabaseName()`

### Получить пользователя соединения

#### Синтаксические правила

```
SqlDatabase::userName ();
```

#### Описание

Предоставляет значение, которое должно использоваться в качестве имени пользователя БД при открытии соединения (т. е. имя пользователя, заданное ранее при помощи функции `setUserName()`).

#### Возвращаемое значение

Символьное значение длиной до 66 символов (может быть пустым).

#### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );  
db.setUserName( "SYSTEM" );  
cout << "User name " << db.userName().toLocal8Bit().data() << "\n";  
db.setPassword( "MANAGER" );  
cout << "Password " << db.password().toLocal8Bit().data() << "\n";  
В буфере будет возвращено SYSTEM.
```

#### См. также

`setUserName()`.

## Получить имя драйвера

### Синтаксические правила

```
SqlDatabase::driverName ();
```

### Описание

Предоставляет имя драйвера, используемого для доступа к БД по указанному соединению.

### Возвращаемое значение

Символьная строка длиной 8 символов.

### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
cout << "driver name" << db.driverName().toLocal8Bit().data() << "\n"; //на консоль
будет выведена строка QLINTER
```

### См. также

addDatabase(), driver().

## Проверить доступность драйвера

### Синтаксические правила

```
QSqlDatabase::isDriverAvailable (<имя драйвера> );
<имя драйвера> ::= символьный литерал или символьная переменная
```

### Описание

Предоставляет информацию о доступности указанного драйвера.

### Возвращаемое значение

Логическое значение:

- true – драйвер доступен (установлен в QT-интерфейсе);
- false – драйвер не доступен (неверное имя драйвера или драйвер не установлен в QT-интерфейсе).

### Пример

```
QSqlDatabase db = QSqlDatabase::QSqlDatabase();
cout << "is available Linter driver " << db.isDriverAvailable("QLINTER") << "\n";
```

### См. также

drivers().

## Проверить состояние соединения

### Синтаксические правила

```
bool QSqlDatabase::isOpen () const
```

### Описание

Предоставляет информацию о текущем состоянии указанного соединения.

### Возвращаемое значение

Логическое значение:

- true – соединение открыто (активно);
- false – соединение закрыто (не активно). QSqlDatabase -объект, соответствующий этому соединению, не уничтожен, и может быть использован повторно.

### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
db.setUsername( "SYSTEM" );
db.setPassword( "MANAGER" );
db.open();
cout << "is open Linter connection " << db.isOpen() << "\n";
```

## Проверить результат соединения с БД

### Синтаксические правила

```
bool QSqlDatabase::isOpenError () const
```

### Описание

Предоставляет информацию о результате выполнения операции соединения с БД.

### Возвращаемое значение

Логическое значение:

- true – соединение установлено успешно;
- false – соединение не установлено. Причину ошибки можно узнать с помощью функции lastError().

### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
db.setUsername( "SYSTEM" );
db.setPassword( "MANAGER" );
db.open();
cout << "is OpenError: " << db.isOpenError() << "\n";
```

## Проверить существование соединения

### Синтаксические правила

```
bool QSqlDatabase::isValid () const
```

### Описание

Предоставляет информацию о существовании именованного соединения.

## Возвращаемое значение

Логическое значение:

- true – соединение существует;
- false – соединение не существует.

## Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
db.setUserName( "SYSTEM" );
db.setPassword( "MANAGER" );

db.open();
cout << "connection valid: " << db.isValid() << "\n";

if( !db.open() )
{
    cout << db.lastError().driverText().toLocal8Bit().data() << endl;
    return 1;
}

cout << "connectionName " << db.connectionName().toLocal8Bit().data() << "\n";
QString str = db.connectionName();

db.removeDatabase(str);
cout << "connection valid: " << db.isValid() << "\n";
```

## Получить пароль соединения

### Синтаксические правила

QString QSqlDatabase::password () const

### Описание

Предоставляет пароль пользователя, от имени которого выполнено (или должно выполняться) соединение с БД.

### Возвращаемое значение

Символьная строка длиной до 18 символов.

Пустая строка возвращается в следующих случаях:

- пароль не был предварительно задан с помощью функции setPassword();
- пароль использовался временно (как аргумент функции open() при открытии соединения);
- пользователь, установивший соединение с БД, не имеет пароля.

### См. также

setPassword(), userName().

## Управление соединением

### Установить сконфигурированное соединение

#### Синтаксические правила

```
bool QSqlDatabase::open ()
```

#### Описание

Открывает соединение с БД в соответствии с параметрами, установленными при инициализации (конфигурировании) объекта-соединения.

#### Возвращаемое значение

Логическое значение:

- true – соединение с БД установлено;
- false – ошибка открытия соединения. Подробную информацию о причине ошибки можно получить с помощью функции `lastError()`.

#### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
if( !db.open("SYSTEM", "MANAGER") )
{
    cout << db.lastError().driverText().toLocal8Bit().data() << endl;
    return 1;
}
```

#### См. также

`lastError()`, `setDatabaseName()`, `setUserName()`, `setPassword()`, `setHostName()`, `setPort()`, `setConnectOptions()`.

### Установить параметризуемое соединение

#### Синтаксические правила

```
bool QSqlDatabase::open (<пользователь>, <пароль> )
```

<пользователь> ::= символьный литерал или символьная переменная длиной до 66 символов

<пароль> ::= символьный литерал или символьная переменная длиной до 18 символов

#### Описание

Открывает соединение с БД в соответствии с текущими параметрами `QSqlDatabase`-объекта, но с новыми регистрационными данными <пользователь> и <пароль>. Параметры <пользователь> и <пароль> не запоминаются в `QSqlDatabase`-объекте, а передаются непосредственно драйверу для открытия соединения, после чего становятся недоступными.

#### Возвращаемое значение

Логическое значение:

- true – соединение с БД установлено.

- False – ошибка открытия соединения. Подробную информацию о причине ошибки можно получить с помощью функций `lastError()`.

### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );

if( !db.open("SYSTEM", "MANAGER") )
{
    cout << db.lastError().driverText().toLocal8Bit().data() << endl;
    return 1;
}
```

### См. также

`lastError()`.

## Закреть соединение

### Синтаксические правила

```
void QSqlDatabase::close (); (<пользователь>, <пароль> )
```

### Описание

Закрывает соединение с БД, освобождая все задействованные ресурсы (в том числе и связанный с соединением `QSqlDatabase`- объект).

### Возвращаемое значение

Нет.

### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );

if( !db.open("SYSTEM", "MANAGER") )
{
    cout << db.lastError().driverText().toLocal8Bit().data() << endl;
    return 1;
}

db.close();
```

### См. также

`removeDatabase()`.

## Начать транзакцию

### Синтаксические правила

```
bool QSqlDatabase::transaction ();
```

### Описание

Объявляет о начале транзакции. Текущая транзакция (если есть) по умолчанию завершается с фиксацией изменений в БД (функция `commit()`).

### Возвращаемое значение

Логическое значение:

- true – успешный старт транзакции;
- false – транзакция не стартовала.

### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
db.setUserName( "SYSTEM" );
db.setPassword( "MANAGER" );

db.transaction();
QSqlQuery query;
query.exec("DELETE FROM AUTO WHERE make = 'FIAT'");
query.exec("SELECT model FROM AUTO WHERE make = 'FIAT'");
if (query.next())
// что-то не так - откатываем транзакцию
    db.rollback();
else
    db.commit();
```

### См. также

QSqlDriver::hasFeature(), commit(), rollback().

## Подтвердить транзакцию

### Синтаксические правила

```
bool QSqlDatabase::commit ();
```

### Описание

Завершает текущую транзакцию в соединении (если транзакция была ранее инициирована при помощи функции transaction()).

### Возвращаемое значение

Логическое значение:

- true – транзакция завершена успешно;
- false – ошибка при завершении транзакции.

### Пример

См. функцию transaction().

### См. также

QSqlDriver::hasFeature(), rollback().

## Удалить именованное соединение

### Синтаксические правила

`void QSqlDatabase::removeDatabase (<имя оединения>)`

<имя соединения> ::= символный литерал или символная переменная

### Описание

Удаляет объект-соединение с заданным <именем соединения> из списка декларированных соединений.

В момент вызова данной функции соединение не должно содержать обрабатываемых SQL-запросов, в противном случае возможна потеря выделенных для обработки запроса ресурсов.

### Возвращаемое значение

Нет.

### См. также

`database()`.

### Пример

```
// Неправильный код
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER", "connection_name");
if( !db.open("SYSTEM", "MANAGER") )
{
    cout << db.lastError().driverText().toLocal8Bit().data() << endl;
    return 1;
}
QSqlQuery query("SELECT count(*) FROM AUTO", db);
QSqlDatabase::removeDatabase("connection_name"); // будет выдано предупреждение

// "db" теперь ссылается на несуществующее соединение,
// "query" содержит неправильные данные

// Правильный код:
{
    QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER", "connection_name");
    if( !db.open("SYSTEM", "MANAGER") )
    {
        cout << db.lastError().driverText().toLocal8Bit().data() << endl;
        return 1;
    }

    QSqlQuery query("SELECT count(*) FROM AUTO", db);
}
// Объекты "db" и "query" уничтожены потому что они больше не нужны
QSqlDatabase::removeDatabase("connection_name"); // правильная операция
```

## Откатить транзакцию

### Синтаксические правила

`bool QSqlDatabase::rollback ();`

### Описание

Выполняет откат текущей транзакции в заданном соединении. Транзакция должна быть предварительно инициирована с помощью функции `transaction()`.

### Возвращаемое значение

Логическое значение:

- `true`- откат транзакции выполнен успешно;
- `false` – при откате транзакции зафиксирована ошибка.

### Пример

См. функцию `transaction()`.

### См. также

`QSqlDriver::hasFeature()`, `commit()`.

## Поддерживаемые типы данных

Соответствие типов данных СУБД ЛИНТЕР и стандартного QT-интерфейса приведено в таблицах 1, 2.

**Таблица 1. Соответствие типов данных при выборке из БД**

Тип данных LINAPI-интерфейса	Тип данных QT-интерфейса
<code>tSmallInt</code>	<code>QVariant::Int</code>
<code>tInt</code>	<code>QVariant::Int</code>
<code>tBigInt</code>	<code>QVariant::LongLong</code> (для Qt 3.2.0 и выше ) <code>QVariant::Double</code>
<code>tByte</code>	<code>QVariant::ByteArray</code>
<code>tVarByte</code>	
<code>tChar</code>	<code>QVariant::CString</code>
<code>tVarChar</code>	
<code>tNChar</code>	<code>QVariant::String</code>
<code>tNVarChar</code>	
<code>tDate</code>	<code>QVariant::Date</code>
<code>tDouble</code>	<code>QVariant::Double</code>
<code>tDecimal</code>	<code>QVariant::String</code>
<code>tBlob</code>	<code>QVariant::ByteArray</code>
<code>tBoolean</code>	<code>QVariant::Boolean</code>

**Таблица 2. Соответствие типов данных при добавлении (модификации) данных БД**

Тип данных QT-интерфейса	Тип данных LINAPI-интерфейса
QVariant::Int	Tint
QVariant::LongLong	tBigInt
QVariant::ByteArray	tByte
QVariant::CString	tChar
QVariant::String	tNChar
QVariant::Date	tDate
QVariant::Double	tDouble
QVariant::Boolean	tBoolean

## Обработка SQL-запросов

### Синтаксические правила

QSqlQuery QSqlDatabase::exec (<SQL-запрос>)

<SQL\_запрос> ::= символный литерал или символная переменная.

### Описание

Выполняет подготовленный (без параметров) SQL-запрос.

Для получения кода завершения выполненного SQL-запроса используется функция `lastError()`. Если текст <SQL-запроса> пуст, то запрос не выполняется и код завершения не генерируется.

### Возвращаемое значение

Указатель на QSqlQuery-объект.

### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
db.setUsername( "SYSTEM" );
db.setPassword( "MANAGER" );

if( !db.open() )
{
    cout << db.lastError().driverText().toLocal8Bit().data() << endl;
    return 1;
}

QSqlQuery q;
q.exec( QString( "select name from person;" ) );

while ( q.next() )
{
    cout << q.value(0).toString().toLocal8Bit().data() << "\n";
}
```

Первый вызов `next()` позиционирует `QSqlQuery` на **первую** запись в наборе данных. Последующие вызовы `next()` передвигают указатель на следующую запись и так до тех пор, пока не будет достигнут конец выборки. В этой точке `next()` вернет **false**.

Функция `value()` возвращает значение поля в виде `QVariant`. Поля нумеруются, начиная с 0, в порядке их следования в предложении `SELECT`. Класс `QVariant` может хранить большое количество типов данных языка C++ и QT, в том числе `int` и `QString`. Различные типы данных, которые могут храниться в БД, переводятся в соответствующие типы C++ и QT, и сохраняются в виде `QVariant`. Например, `VARCHAR` представляется в виде `QString`, а `DATETIME` -- как `QDateTime`.

Класс `QSqlQuery` предоставляет целый набор функций для навигации по набору данных: `first()`, `last()`, `prev()`, `seek()` и `at()`. Они удобны в использовании, но при больших объемах выборки могут оказаться медлительными и ресурсоемкими. С целью оптимизации при работе с большими наборами данных можно вызвать `QSqlQuery::setForwardOnly(true)` перед `exec()`, а затем выполнять просмотр набора данных с помощью `next()` (в этом случае получаем однонаправленные наборы данных, т.е. такие наборы, навигация по которым может осуществляться только вперед, с помощью `next()`).

SQL-запрос можно передавать не только как аргумент функции `exec()`, но и напрямую, конструктору `QSqlQuery`:

```
QSqlQuery query("SELECT title, year FROM cd WHERE year >= 1998");
```

Проверку на наличие ошибок и выдачу диагностического сообщения можно выполнить следующим образом:

```
if (!query.isActive())
    cout << db.lastError().driverText().toLocal8Bit().data() << endl;
```

Выполнение SQL-запросов манипулирования данными аналогично запросам выборки данных:

```
QSqlQuery query("INSERT INTO cd (id, artistid, title, year) " "VALUES (203, 102, 'Living in America', 2002)");
```

После выполнения такого запроса `QSqlQuery::numRowsAffected()` возвращает количество добавленных записей.

При необходимости вставить в SQL-запрос значения переменных или когда нежелательно, или невозможно перевести аргументы SQL-предложения в строковый вид, можно построить параметризованный запрос с помощью функции `prepare()`. Текст параметризованного запроса вместо реальных значений содержит параметры, которые заполняются фактическими значениями после создания запроса, например:

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );
db.setUserName( "SYSTEM" );
db.setPassword( "MANAGER" );
if( !db->open() )
{
    cout << db->lastError().driverText().toLocal8Bit().data() << endl;
    return 1;
}
QSqlQuery query(db);
query.prepare("INSERT INTO cd (year, make, model, color) "
             "VALUES (?, ?, ?, ?)");
query.addBindValue(70);
query.addBindValue(QString "FORD");
query.addBindValue(QString("PANTERA"));
query.addBindValue(QString("BLACK"));
query.exec();
```

После создания запроса (вызовом `prepare()`), параметры запроса заполняются фактическими значениями с помощью функции `bindValue()` или `addBindValue()`, после чего запрос исполняется вызовом `exec()`. Параметризованные запросы можно выполнять в цикле. Перед началом цикла создается запрос, а в теле цикла производится заполнение параметров новыми значениями и исполнение запроса.

Параметризованные запросы очень часто используются в тех случаях, когда в БД нужно записать двоичные данные или строки, которые содержат символы из наборов, не принадлежащих диапазону ASCII или Unicode.

Каждое соединение с БД может поддерживать только одну активную транзакцию, поэтому множественные подключения могут оказаться полезными в том случае, когда необходимо одновременно запустить несколько транзакций. При использовании нескольких соединений в QT-приложении имеется одно неименованное соединение, которое используется по умолчанию объектами `QSqlQuery`, если им явно не указать с каким соединением они должны работать.

В дополнение к `QSqlQuery`, Qt предоставляет класс `QSqlCursor`, производный от `QSqlQuery`. Этот класс расширяет функциональность предка большим числом дополнительных методов, которые позволяют отказаться от написания SQL-запросов для SQL-операций, таких как: `SELECT`, `INSERT`, `UPDATE` и `DELETE`.

Следующий пример демонстрирует выполнение и обработку результата `SELECT`-запроса с помощью методов класса `QSqlCursor`:

```
QSqlCursor cursor("cd");
cursor.select("year >= 1998");
```

Эквивалентный вариант с использованием `QSqlQuery`:

```
QSqlQuery query("SELECT id, artistid, title, year FROM cd " "WHERE year >= 1998");
```

Навигация по выборке выполняется точно так же, как и в `QSqlQuery`, за одним исключением – вместо порядкового номера поля функции `value()` можно передать его имя:

```
while (cursor.next()) {
    cout << cursor.value("title").toString().toLocal8Bit().data() << "\n";
}
```

Для вставки записи в таблицу предварительно нужно создать новую запись `QSqlRecord` с помощью вызова `primeInsert()`, а затем для каждого из полей вызвать функцию `setValue()`.

После всего этого можно выполнить вставку записи функцией `insert()`:

```
QSqlCursor cursor("cd");
QSqlRecord buffer = cursor.primeInsert();
buffer.setValue("year", 71);
buffer.setValue("model", "BUICK");
buffer.setValue("make", "FIAT");
buffer.setValue("color", "BLACK");
cursor.insert();
```

Чтобы изменить запись, нужно:

- позиционировать `QSqlCursor` на запись, которая должна подвергнуться изменениям (например, с помощью `select()` и `next()`);
- получить указатель на `QSqlRecord` вызовом `primeUpdate()`;

- записать новые значения функцией setValue()
- вызвать update(), чтобы отправить сделанные изменения в базу данных:

```
QSqlCursor cursor("cd");
cursor.select("personid = 125");
if (cursor.next()) {
    QSqlRecord buffer = cursor.primeUpdate();
    buffer.setValue("color", "BLACK");
    buffer.setValue("year", buffer.value("year").toInt() + 1);
    cursor.update();
}
```

Процедура удаления записи похожа на процедуру изменения:

```
QSqlCursor cursor("cd");
cursor.select("personid = 128");
if (cursor.next()) {
    cursor.primeDelete();
    cursor.del();
}
```

**См. также**

QSqlQuery, lastError().

## Получение информации об объектах БД

### Получить список таблиц БД

**Синтаксические правила**

QStringList QSqlDatabase::tables (<тип>QSql::TableType type = QSql::Tables) const  
<тип>:: = тип запрашиваемых таблиц (типы таблиц определены в перечислимом типе данных QSql::TableType, см. табл. 3).

**Таблица 3. Типы таблиц БД, о которых можно запрашивать информацию**

Тип данных QSql::TableType	Значение	Описание
QSql::Tables	0x01	Все таблицы, владельцем которых является пользователь, от имени которого выполнено соединение с БД
QSql::SystemTables	0x02	Системные таблицы
Sql::Views	0x04	Все представления, владельцем которых является пользователь, от имени которого выполнено соединение с БД
QSql::AllTables	0xff	Все системные таблицы БД, а также таблицы и представления, владельцем которых является пользователь, от имени которого выполнено соединение с

		БД
--	--	----

### Описание

Предоставляет список пользовательских или системных таблиц (представлений) БД.

### Возвращаемое значение

Список таблиц в виде QStringList-объекта.

### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );

if( !db.open("SYSTEM", "MANAGER") )
{
    cout << db.lastError().driverText().toLocal8Bit().data() << endl;
    return 1;
}

QStringList list = db.tables(QSql::Views);
cout << "Views list\n " << list.join(",").toLocal8Bit().data() << "\n";
```

### См. также

primaryIndex(), record().

## Получить описание первичного ключа таблицы

### Синтаксические правила

QSqlIndex QSqlDatabase::primaryIndex ([<имя владельца>.]<таблица> ) const  
 <таблица> ::= символный литерал или символная переменная в виде  
 [<имя владельца>.]<имя таблица>

### Описание

Предоставляет информацию о первичном ключе указанной таблицы.

### Возвращаемое значение

Указатель на QSqlIndex-объект. Если первичный ключ не создан, возвращается пустой QSqlIndex-объект.

### Пример

```
// Получить список столбцов, входящих в первичный ключ
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );

db.setUserName( "SYSTEM" );
db.setPassword( "MANAGER" );

if( !db.open() )
{
    cout << db.lastError().driverText().toLocal8Bit().data() << endl;
    return 1;
}
```

```
QSqlIndex ind = db.primaryIndex("PERSON");
cout << "Primary index of table person " << ind.name().toLocal8Bit().data() << "\n";
```

### См. также

tables(), record()

## Получить описание строки таблицы

### Синтаксические правила

```
QSqlRecord QSqlDatabase::record (<таблица> ) const
<таблица> ::= символный литерал или символьная переменная в виде
[<имя владельца>.]<имя таблица>
```

### Описание

Предоставляет описание строки таблицы или представления. Порядок расположения полей таблицы – произвольный.

### Возвращаемое значение

Указатель на QSqlRecord-объект. Если аргумент *<таблица>* задает несуществующую в БД таблицу (представление), возвращается пустой QSqlRecord объект (isEmpty будет true).

### Пример

```
QSqlDatabase db = QSqlDatabase::addDatabase( "QLINTER" );

db.setUserName( "SYSTEM" );
db.setPassword( "MANAGER" );

if( !db.open() )
{
    cout << db.lastError().driverText().toLocal8Bit().data() << endl;
    return 1;
}

QSqlRecord rec = db.record("PERSON");
int i;
cout << "count " << rec.count() << "\n";
for (i = 0; i < rec.count(); i++)
{
    cout << "record " << rec.fieldName(i).toLocal8Bit().data() << "\n";
}
}
```

### См. также

tables(), record()

# Обработка кодов завершения

## Получить последний код завершения

### Синтаксические правила

```
QSqlError QSqlDatabase::lastError () const
```

### Описание

Предоставляет информацию о коде завершения последней выполненной по соединению операции.

### Возвращаемое значение

Указатель на QSqlError-объект. Возможные значения кода завершения определены в перечислимом типе данных QSqlError::ErrorType (таблица 4) .

**Таблица 4. Коды завершения интерфейса**

Код завершения QsqlError::ErrorType	Значение	Описание
QSqlError::NoError	0x00	Нормальное завершение операции
QSqlError::ConnectionError	0x01	Ошибка при соединении с БД
QSqlError::StatementError	0x02	Синтаксическая ошибка SQL-оператора
QSqlError::TransactionError	0x03	Ошибка при обработке транзакции
QSqlError::UnknownError	0x04	Неизвестная ошибка



