В SQL Server безопасность построена на трёх основных понятиях:

1. Объекты, к которым осуществляется доступ - **Securables**
2. Субъекты, которые осуществляют доступ - **Principals**
3. Разрешения на доступ - **Permissions**

SQL Server поддерживает два основных метода аутентификации:

1. **Аутентификация Windows (SQL Server использует учетные данные текущего пользователя Windows)**
2. **Смешанная аутентификация (Windows + SQL Server Authentication)**

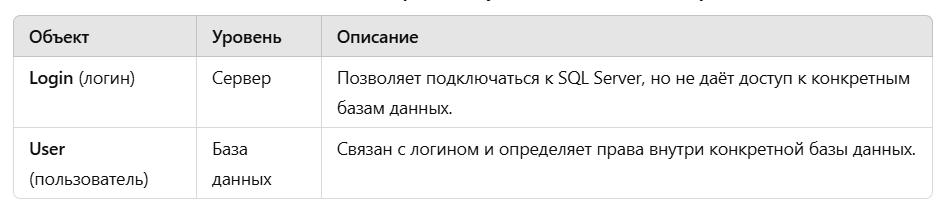
Логины в SQL Server можно создавать двумя способами:

1. **Через SQL Server Management Studio (SSMS)** – графический интерфейс.
2. **С помощью Transact-SQL (T-SQL)** – с использованием команд SQL.

**CREATE LOGIN MyUser WITH PASSWORD = 'StrongP@ssword123';**

Некоторые компоненты SQL Server создают логины автоматически при установке или настройке. Это нужно для их работы и взаимодействия с сервером.

В SQL Server логины и пользователи – это **разные сущности**, и их нельзя путать.



**USE MyDatabase; -- Переходим в нужную базу данных**

**CREATE USER MyUser FOR LOGIN MyLogin;**

В SQL Server есть два **специальных пользователя**, которые играют важную роль в системе безопасности:

1. **dbo (Database Owner)** – владелец базы данных, имеющий полный контроль. Автоматически создаётся в каждой базе данных.
2. **guest** – Позволяет логинам, у которых нет собственного пользователя в базе данных, получать ограниченный доступ.

**Автономные базы данных** – это базы, которые **не зависят от серверных логинов**. Все пользователи и их учетные данные хранятся **внутри самой базы**, что делает такие базы **переносимыми** и удобными для миграции.

**1. Разрешить использование автономных баз данных на сервере**

**sp\_configure 'contained database authentication', 1;**

**RECONFIGURE;**

**2. Создать автономную базу данных**

При создании базы нужно включить режим частичной автономности (CONTAINMENT = PARTIAL):

**CREATE DATABASE ContainedDB**

**CONTAINMENT = PARTIAL;**

Это значит, что база **частично автономна** – поддерживает как автономных пользователей, так и обычных.

**3. Создать автономного пользователя**

После создания базы можно добавить пользователя **без логина**:

**USE ContainedDB;**

**CREATE USER ContainedUser WITH PASSWORD = 'SecureP@ssword';**

Роль — это **группа** пользователей, которым назначены определённые права (разрешения).

Разрешение — это **конкретное право** на выполнение определённой операции на объекте в базе данных или сервере.

Одним из самых мощных разрешений является **CONTROL SERVER (Это разрешение предоставлено роли sysadmin)**, которое даёт **почти** полный доступ к серверу, но не делает пользователя sysadmin.

|  |
| --- |
| Член роли sysadmin игнорирует все ограничения безопасности, он всегда имеет полный доступ.  Пользователь с CONTROL SERVER имеет почти такие же права, но SQL Server всё равно проверяет разрешения.  Если у пользователя есть CONTROL SERVER, но **ему запретили доступ к определённой базе** (DENY CONNECT), он **не сможет войти в неё**. А вот **член sysadmin** сможет, потому что роль sysadmin игнорирует все запреты. |





В SQL Server можно создавать **пользовательские серверные роли** (не фиксированные), управлять их разрешениями и членством с помощью команд **CREATE SERVER ROLE**, **ALTER SERVER ROLE**, **DROP SERVER ROLE**, а также с помощью команд для управления правами: **GRANT TO**, **DENY TO**, **REVOKE TO**.

Команда **GRANT** используется для **предоставления разрешений** пользователю или роли.

Команда **DENY** используется для **отказа в разрешении**, и она имеет более высокий приоритет, чем команда **GRANT**. Если роль или пользователь имеет разрешение с помощью **DENY**, они **не смогут выполнить действие**, даже если разрешение было предоставлено другим способом.

Команда **REVOKE** отменяет **ранее предоставленные или отказанные** разрешения. Это позволяет удалить права без необходимости заново предоставлять разрешения через DENY или GRANT.

Управление ролями на **уровне базы данных**:

* Создание и удаление

CREATE ROLE

DROP ROLE

* Управление разрешениями

GRANT, DENY и REVOKE

* Управление членством

ALTER ROLE

Роль **db\_owner** — это одна из **фиксированных ролей** на уровне базы данных, которая предоставляет полные права на управление объектами базы данных и выполнение всех операций в базе данных.

* Пользователь **dbo** всегда является членом роли **db\_owner**, что позволяет ему иметь полный контроль над базой данных, включая управление объектами и правами других пользователей базы данных.

**Роль приложения** позволяет приложению работать с базой данных, используя свои собственные права, а не права пользователя, который подключается.

**Для создания роли приложения** используется команда

**CREATE APPLICATION ROLE role\_name WITH PASSWORD = 'your\_password';**

**Предоставление прав** роли:

**GRANT SELECT, INSERT ON table\_name TO role\_name;**

**Активация роли** через хранимую процедуру **sp\_setapprole**: Эта процедура меняет текущий контекст безопасности на контекст роли.

**DECLARE @context VARBINARY(512);**

**EXEC sp\_setapprole 'role\_name', 'your\_password', @context OUTPUT;**

**Возвращение к исходным правам** через **sp\_unsetapprole**: После работы с ролью мы восстанавливаем исходный контекст безопасности.

**EXEC sp\_unsetapprole @context;**

**Иерархия защищаемых объектов в SQL Server:**

1. **Сервер (Server)**
2. **База данных (Database)**
3. **Схема (Schema)**. Схема — это логическая структура, которая группирует объекты базы данных, такие как таблицы, представления и процедуры.
4. **Объект (Object)**. Объекты базы данных — это конкретные элементы, такие как таблицы, представления, хранимые процедуры, функции и индексы.

**Примеры разрешений для таблиц:**

* SELECT
* INSERT
* UPDATE
* DELETE
* REFERENCES – позволяет пользователю ссылаться на таблицу в ограничениях внешнего ключа

**Примеры разрешений для хранимых процедур:**

* EXECUTE
* ALTER
* VIEW DEFINITION – позволяет пользователю просматривать определение хранимой процедуры (т.е. её текст), но не изменять её

**Примеры разрешений для функций:**

* Для скалярных функций - EXECUTE
* Для табличных функций – SELECT – чтобы пользователь мог обращаться к возвращаемым данным функции

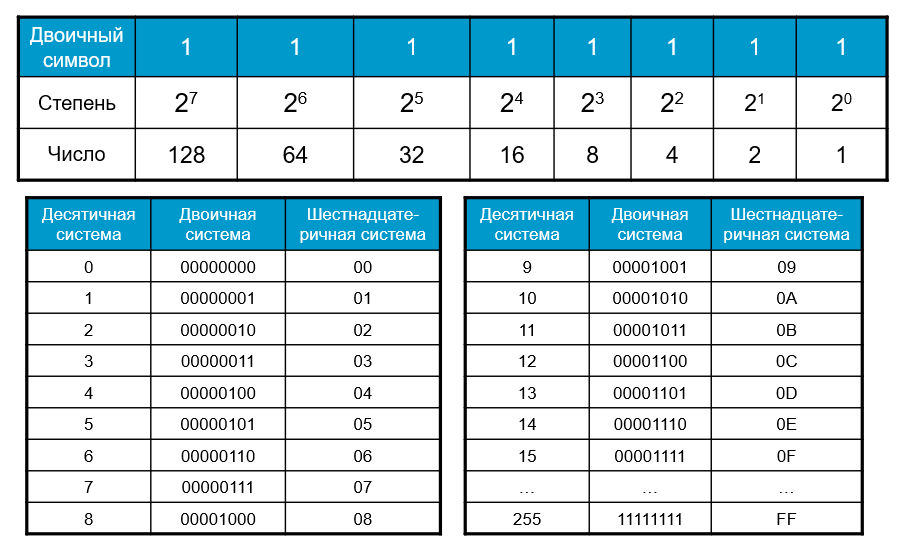
**Цепочка владельцев:**

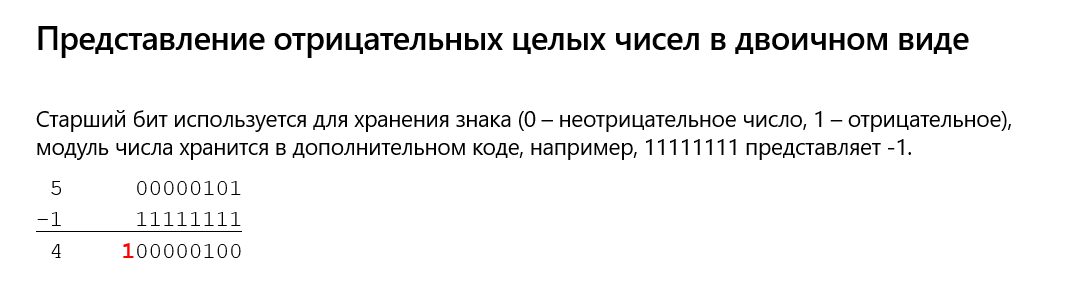
* Пользователь имеет разрешение на выполнение процедуры (объект B), которая обращается к таблице (объект A). Если оба объекта принадлежат одному владельцу, пользователю не нужно явно предоставлять разрешения на таблицу. Он сможет выполнить процедуру, и SQL Server автоматически проверит разрешения на таблицу в контексте процедуры.
* Пользователь имеет разрешение на выполнение процедуры (объект B), которая ссылается на таблицу (объект A). Если объекты принадлежат разным владельцам, то помимо разрешений на выполнение процедуры, пользователь должен также иметь разрешения на таблицу.

В SQL Server, если при выполнении запроса не указано полное имя объекта, система будет искать объект в различных местах по определенному порядку:

1. **Для хранимых процедур, начинающихся на sp\_\*:**
   * Хранимые процедуры с именами, начинающимися на sp\_, ищутся **в схеме sys** (независимо от текущего контекста пользователя или базы данных). Это системные хранимые процедуры, и их имена обычно начинаются с префикса sp\_ (например, sp\_help, sp\_configure и т.д.).
2. **Затем в схеме пользователя по умолчанию:**
   * Если объект не найден в схеме sys (например, для пользовательских хранимых процедур или других объектов), SQL Server продолжит поиск в схеме, которая настроена как **схема по умолчанию** для текущего пользователя. Эта схема указывается при создании пользователя.
3. **Затем в схеме dbo:**
   * Если объект не был найден в схеме по умолчанию, SQL Server будет искать его в схеме **dbo** (по умолчанию для объектов, если не указана другая схема).







Каждому из символов (букв, цифр, знаков препинания и т. д. каким-то образом присваивается порядковый номер. К примеру, букве "A" может быть присвоен номер 1; "Z" - 26 и так далее. Номер, представляющий символ как целое число, считается **кодом символа**, а **кодировка** - это, соответственно, набор символов в такой таблице, которую называют **кодовой таблицей**.

Минимальный набор символов около 80 символов:

* 26+26 буквы английского алфавита **(или латинского?)**
* 10 цифр
* Несколько десятков знаков препинания.

ASCII - это первая кодировка, применявшаяся еще в 1963 году. Кодировка **7-битная**, то есть определено 128 символов, 8-й бит полного байта использовался для проверки четности (поскольку в то время каналы были ненадежные, то предполагалось что будет проверяться каждый полученный байт). Первые 32 кода символов — служебные и непечатные.

Использование восьмого бита для кодирования символов предоставило возможность закодировать еще 128 символов. Традиционно часть из них использовалось для псевдографики, часть – для символов национальных языков.

ANSI-кодировки — по сути это те же расширенные версии ASCII, однако из них удалены различные псевдографические элементы (рамки, линии, блоки) и добавлены символы типографики, для которых ранее не хватало "свободных мест".

Первая версия Юникода представляла собой кодировку с фиксированным размером символа в 16 бит, то есть общее число кодов было 216 (65 536). С тех пор символы стали обозначать четырьмя шестнадцатеричными цифрами (например, U+04F0).

Для совместимости со старыми 16-битными системами была изобретена система UTF-16. UTF-16 может использовать **1 или 2** 16-битных числа для хранения символов.

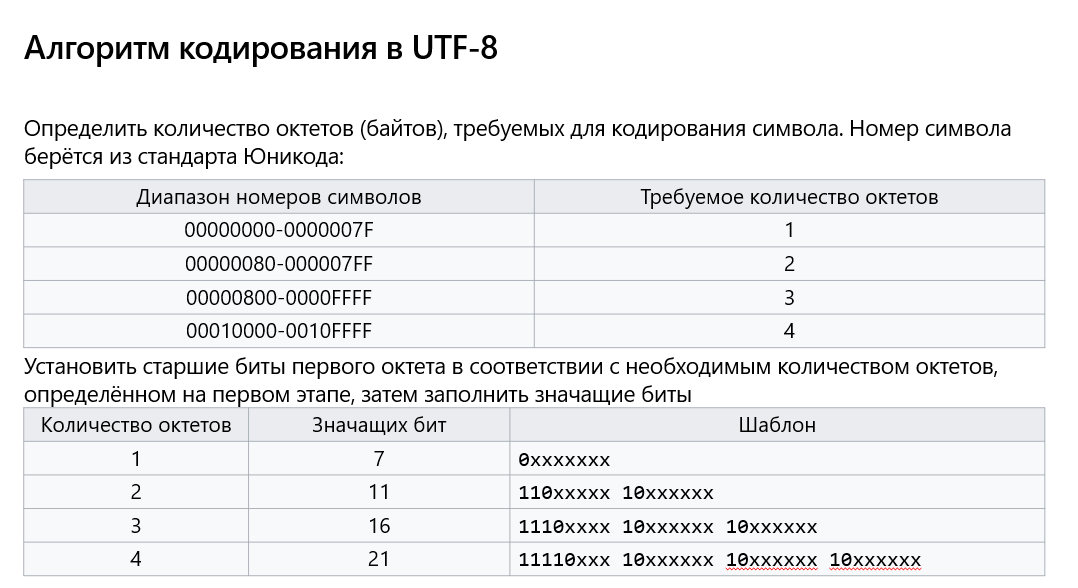
UTF-32 использует для кодирования любого символа ровно 32 бита.

✅ **Плюс:** Каждый символ занимает фиксированные **4 байта**, поэтому легко обращаться к любому символу по индексу.

❌ **Минус:** Неэффективен по памяти — даже частые символы занимают **4 байта**, что вдвое больше, чем в **UTF-16**, и в 3–4 раза больше, чем в **UTF-8**.

💡 **Вывод:** UTF-32 удобен для быстрого доступа, но редко используется из-за большого размера.

UTF-8 — распространённый стандарт кодирования символов, позволяющий более компактно хранить и передавать символы Юникода, используя переменное количество байт (от 1 до 4), и обеспечивающий полную обратную совместимость с 7-битной кодировкой ASCII.



Точные числовые типы (Exact numerics)

1. **bit**  
   Логический тип данных, принимает значения 0, 1 или NULL. **Размер:** 1 бит.
2. **tinyint**  
   Целое число от 0 до 255. **Размер:** 1 байт.
3. **smallint**  
   Целое число от -32,768 до 32,767. **Размер:** 2 байта.
4. **int**  
   Целое число от -2,147,483,648 до 2,147,483,647. **Размер:** 4 байта.
5. **bigint**  
   Целое число от -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807. **Размер:** 8 байт.
6. **smallmoney**  
   Денежное значение от -214,748.3648 до 214,748.3647 с точностью до 4 знаков после запятой. **Размер:** 4 байта.
7. **money**  
   Денежное значение от -922,337,203,685,477.5808 до 922,337,203,685,477.5807 с точностью до 4 знаков после запятой. **Размер:** 8 байт.
8. **decimal / numeric**. Число с фиксированной точностью и масштабом (до 38 знаков). **Размер:** 5–17 байт (в зависимости от точности).

**decimal / numeric. Описание:** Хранят числа с фиксированной точностью (p) и масштабом (s).

* **p** (точность) — общее количество цифр (от 1 до 38).
* **s** (масштаб) — количество цифр после запятой (от 0 до p).

**Пример:**

* decimal(10, 2) или numeric(10, 2) — число с 10 знаками в целом, 2 из которых после запятой (например, 12345678.99).\

1. float:

* 4 байта для **float** (с точностью до 7 знаков).
* 8 байт для **float** (с точностью до 15 знаков, по умолчанию).

**Иерархия хранения данных в SQL Server:**

1. **Физический или виртуальный сервер** – компьютер, где работает SQL Server.
2. **Экземпляр (Instance)** – отдельный запущенный SQL Server, может быть несколько на одном сервере. Для каждого экземпляра создается собственная структура папок и набор служб. На один и тот же физический сервер можно установить экземпляры различных версий SQL Server. Пример объекта уровня экземпляра – логин. Каждый экземпляр получает индивидуальный TCP-порт для получения запросов.
3. **База данных (Database)** – контейнер для данных внутри экземпляра.
4. **Файловая группа (Filegroup)** – логическая группировка файлов базы данных.
5. **Файл (File)** – физический файл на диске (.mdf, .ndf, .ldf).
6. **Интервал (Extent)** – группа из 8 страниц (64 КБ), минимальный блок аллокации.
7. **Страница (Page)** – базовая единица хранения данных, размер **8 КБ**.
8. **Экстент (Extent)** – объединение 8 страниц (64 КБ), используется для эффективного распределения.
9. **Запись (Record)** – строка данных внутри страницы.