# Создание отношений в среде SQL Server Management Studio

**Цель работы:** Изучить основные средства, которые предоставляет среда SQL Server Management Studio для создания отношений без использования языка запросов T-SQL

## Теоретические указания:

### 2.1 Введение

После создания базы данных, необходимо определить механизм сохранения информации в БД. И для этого первым делом необходимо определить отношения (таблицы) – то место, где будет храниться информация о сущностях, имеющих связь с предметной областью, которая рассматривается при проектировании БД.

Чтобы быть функциональной, БД достаточно одного отношения. Но обычно в БД содержится множество отношений, связанных между собой и формирующих в совокупности схему БД. Так как отношения – самый важный элемент БД, необходимо чётко представлять себе структуру отношений и их содержимое. Проектирование отношений и связей между ними – сложная задача и требует от разработчиков, архитекторов, администраторов БД тщательного анализа и знаний теории проектирования БД (используя формальные методы). Далее речь пойдёт о структуре отношений и о средствах SQL Server Management Studio по созданию отношений удобным способом, не требующим специфических знаний.

Существуют три основных метода создания отношений в БД:

1. Создание отношений, используя **конструктор таблиц**.
2. Создание отношений с помощью **дизайнера диаграмм** БД – специальной утилиты, облегчающей создание схем БД.
3. Создание отношений с помощью шаблонов – готовых заготовок Transact-SQL, имеющихся в среде SQL Server Management Studio, сгруппированных по назначению в отдельные узлы. Несмотря на то, что здесь используется Transact- SQL, для создания отношений не понадобится ни одной строчки Transact-SQL написанной вручную. Чтобы открыть окно содержащее доступные в SQL Server шаблоны, выберите пункт View->TemplateExplorer главного меню.

| |
|---|
| Часто вместо слова **отношение** используется слово **таблица**, чтобы облегчить понимание структуры отношений, т.к. организация данных в виде таблиц давно всем знакома. Также это имеет место и для **столбцов** таблицы или же **атрибутов**. |

Итак, отношение – это хранилище данных, где каждая единица информации (строка таблицы, кортеж), характеризующая какую-либо сущность из предметной области, представлена совокупностью атрибутов (столбцов). Отношение может содержать как минимум один атрибут. Количество кортежей в отношении неограниченно (может быть нулевым). Таким образом, отношение представляется как множество атрибутов, образующих схему отношения, и множество кортежей (строк), являющихся носителями информации. На практике отношения содержат немного атрибутов, а именно такое количество, которое необходимо для отражения одной сущности из предметной области (например, сущность «продукт» может содержать такие атрибуты как номер продукта, цена, наименование, вес одной единицы и т. д.). Количество кортежей в отношении практически ограничены лишь размером дискового пространства, где храниться БД.

Главным отличием отношений от других «обычных» таблиц – это наличие строго типизированных данных в столбцах. В отношении невозможно изменить тип данных атрибута, не повлияв при этом на содержимое всех кортежей отношения. Поэтому очень важным является правильный выбор типа данных (или **домена** – понятие из реляционной теории) атрибута, т. к. тип данных будет определять множество значений, которые может принимать атрибут.

Назначение отношений – хранение специфической информации. Именно поэтому отношениям нужно назначать понятные имена. Также нужно назначать понятные имена атрибутам и правильно устанавливать тип данных, имеющих смысл для данного атрибута.

### 2.2 Свойства атрибутов в SQL Server. Типы данных

Прежде чем ознакомиться с методами создания отношений в SQL Server Management Studio нужно изучить типы данных, которые поддерживает SQL Server. SQL Server поддерживает множество типов данных, каждый из которых имеет своё назначение, и предназначен для использования в определённых случаях для определенных атрибутов. SQL Server не ограничивается стандартными типами данных и предоставляет возможность создавать свои, назначать им имена и использовать в определении отношений. Следует с особой осторожностью подбирать тип данных для атрибута, так как многие типы данных имеют схожую структуру и как следствие могут использоваться не по назначению. Например, если атрибут используется для хранения имён, то подходящим для него типом данных будет строковый тип (последовательность символов), но существуют различные варианты строк (Unicode, Non-Unicode).

Прежде, чем перейти к обсуждению типов данных, рассмотрим простое правило, используя которое можно правильно подобрать тип данных для столбца: *при выборе типа данных для столбца следует отдавать предпочтение типу, который позволит хранить любые возможные для этого столбца значения и занимать при этом минимальное место на диске*.

Типы данных SQL Server подразделяют на семь основных категорий (таблица 1.1):

Таблица 1.1 – Категории типов данных

| Категория типа данных | Описание |
|---|---|
| Точный числовой | Хранит точные числовые значения с десятичными точками или без них |
| Приближенный числовой | Хранит числовые значения с десятичными точками или без них |
| Денежный | Хранит числовые значения с десятичными точками; используется специально для денежных значений; максимальная точность составляет четыре знака после запятой |
| Дата и время | Хранит информацию о дате и времени и выполняет хронологическую проверку |
| Символьный | Хранит символьные значения переменной длины |
| Двоичный | Хранит данные в строго двоичном представлении (0 и 1) |
| Специальный | Составные типы данных, требующие специальной обработки, например XML-документы или глобально уникальные идентификаторы (GUID) |

**Точные числовые типы данных**

К таким числам можно применять любые математические операции - никакая специальная обработка для этого не требуется. Объем занимаемой памяти также строго определен, поэтому данные, хранящиеся в этих типах, имеют одинаковые значения как в архитектуре процессора Intel, так и в архитектуре AMD. В таблице 1.2 приведён список точных числовых типов данных, поддерживаемых SQL Server.

Таблица 1.2 – Числовые типы данных

| Тип данных | Занимаемая память | Диапазон значений | Описание |
|---|---|---|---|
| *bigint* | 8 байт | от -2E63 до 2E63-1 | Хранение очень больших целых чисел – как положительных, так и отрицательных |
| *int* | 4 байта | от -2E3I до 2E3I-1 | Хранение положительных и отрицательных целых чисел |
| *smallint* | 2 байта | от -32768 до 32767 | Хранение положительных и отрицательных целых чисел |
| *tinyint* | 1 байт | от 0 до 255 | Хранение небольшого диапазона положительных целых чисел |
| *decimal(p,s)* | от 5 до 17 байт (в зависимости от точности) | от -10E38+1 до 10E38-1 | Хранение десятичных чисел с максимальным числом разрядов |
| *numeric(p,s)* | от 5 до 17 байт (в зависимости от точности) | от -10E38+1 до 10E38-1 | По своим функциям эквивалентен типу *decimal.* |

**Приближённые числовые типы данных**

В приближённых числовых типах данных можно хранить десятичные значения. Однако точность данных, хранимых в типах *float* или *real,* ограничивается значением, указанным в определении типа данных. При хранении данных SQL Server округляет все разряды справа, поэтому вычисления, в которых используют эти типы данных, содержат ошибки округления. Перемещение баз данных с таблицами, содержащими эти типы данных, между платформами Intel и AMD также приведёт к ошибкам. В таблице 1.3 перечислены приближённые числовые типы данных, поддерживаемые в SQL Server.

Таблица 1.3 – Приближенные числовые типы данных

| Тип данных | Занимаемая память | Диапазон значений | Описание |
|---|---|---|---|
| *float(p)* | 4 или 8 байт | от -2.23E308 до 2.23E308 | Хранение больших чисел с плавающей точкой, которые выходят за границы диапазона типа данных *decimal* |
| *real* | 4 байта | от -3.4E38 до 3.4E38 | Все еще поддерживается, но заменён на *float*, чтобы удовлетворить требованиям стандарта SQL-92 |

**Денежные типы данных**

Денежные тины данных созданы для хранения денежных значений с четырьмя десятичными разрядами после запятой. В таблице 1.4 представлены денежные типы данных, поддерживаемые в SQL Server.

Таблица 1.4 – Денежные типы данных

| Тип данных | Занимаемая память | Диапазон значений | Описание |
|---|---|---|---|
| *money* | 8 байт | от -922337203685477.5808 до 922337203685477.5807 | Хранение больших денежных значений |
| *smallmoney* | 4 байта | от -214748.3648 до 214748.3647 | Хранение небольших денежных значений |

**Типы данных «дата и время»**

При хранении данных ничто не вызывает таких разногласий, как способ хранения даты и времени. В некоторых приложениях нужно хранить только дату, в других — только время, а в третьих – и то и другое. Типы данных, относящиеся к данной категории:

– *datetime* – способен хранить любую дату и время с 1 января 1753 года до 31 декабря 9999 года. Если задается только дата, то время принимает по умолчанию значение 12:00:00.

- *datetime2* – как и тип данных datetime, datetime2 используется для хранения даты и времени, но datetime2 способен хранить более точные значения времени. Формат хранения данных: YYYY-MM-DD hh:mm:ss[.nnnnnnn].
- *smalldatetime* – схож с типом datetime, но хранит меньший диапазон дат: с 1 января 1900
- года до 6 июня 2079 года.
- *date* – хранит дату без указания времени.
- *time* – хранит только время неассоциированное ни с какой датой.
- *datetimeoffset* - определяет дату, объединенную со временем, с учетом часового пояса в
- 24-часовом формате.

**Символьные типы данных**

Для хранения символьных данных служат специально предназначенные для этого типы. В этих типах один символ занимает в памяти один или два байта в зависимости от используемой кодировки — ANSI или Unicode. В таблице 1.5 перечислены символьные типы данных, поддерживаемые в SQL Server.

Таблица 1.5. Символьные типы данных

| Тип данных | Занимаемая память | Количество символов | Назначение |
|---|---|---|---|
| *char(n)* | 1 – 8000 байт | До 8000 символов | Тип данных ANSI фиксированного размера |
| *nchar(n)* | 2 – 8000 байт | До 4000 символов | Тип данных Unicode фиксированного размера |
| *varchar(n)* | 1 – 8000 байт | До 8000 символов | Тип данных ANSI переменного размера |
| *varchar(max)* | До 2 Гб | До 1073741824 символов | Тип данных ANSI переменного размера |
| *nvarchar(n)* | 2 – 8000 байт | До 4000 символов | Тип данных Unicode переменного размера |
| *nvarchar(max)* | До 2 Гб | До 536870912 символов | Тип данных Unicode переменного размера |
| *text* | До 2 Гб | До 1073741824 символов | Тип данных ANSI переменного размера |
| *ntext* | До 2 Гб | До 536870912 символов | Тип данных Unicode переменного размера |

**Двоичные типы данных**

Довольно часто в базах данных нужно хранить двоичные данные. Поэтому SQL Server поддерживает три типа данных, позволяющих хранить в таблице различные объёмы двоичных данных. В таблице 1.6 перечислены двоичные типы данных, поддерживаемые в SQL Server.

Таблица 1.6 – Двоичные типы данных

| Тип данных | Занимаемая память | Назначение |
|---|---|---|
| *binary(n)* | 1 – 8000 байт | Хранение двоичных данных фиксированного размера |
| *varbinary(n)* | 1 – 8000 байт | Хранение двоичных данных переменного размера |
| *varbinary(max)* | До 2 Гб | Хранение двоичных данных переменного размера |
| *image* | До 2 Гб | Хранение двоичных данных переменного размера |

**Специальные типы данных**

Помимо уже перечисленных стандартных типов данных в SQL Server есть семь дополнительных типов данных узкого назначения. Их описание приведено в таблице 1.7.

Таблица 1.7 – Специальные типы данных

| Тип данных | Назначение |
|---|---|
| *bit* | Хранение 0, 1 или null. Применяется для основных значений флага. TRUE преобразуется в 1, а FALSE – в 0. |
| *timestamp* | Автоматически генерируемое значение. В каждой базе данных есть внутренний счетчик относительного времени, не связанного с реальным. В таблице может быть только один столбец с типом данных *timestamp,* которому присваивается значение временной метки базы данных при каждом добавлении или изменении строки. |
| *uniqueidentifie r* | 16-битный GUID, служащий для глобальной идентификации строки в базах данных, экземплярах и серверах. |
| *sql_variant* | Может изменять тип данных в зависимости от хранящихся данных. Максимальный объем хранимых данных – 8 000 байт. |
| *cursor* | Встречается в приложениях, работающих с курсорами, и содержит ссылку на курсор, которая применяется для выполнения операций. Этот тип данных нельзя использовать в таблицах. |
| *table* | Предназначен для хранения набора результатов с целью последующей обработки. Этот тип данных нельзя использовать в столбцах. Он применяется только при объявлении табличных переменных в триггерах, хранимых процедурах и функциях. |
| *xml* | Хранение XML-документов размером до 2 Гб. Задаваемые параметры позволяют хранить в столбцах только согласованные документы. |

**Возможность ввода пустых значений**

Второй аспект определения атрибута – возможность хранения в нём пустых значений. Для указания отсутствия значения в базах данных служит специальная конструкция *NULL,* означающая «неизвестный» или «отсутствующий», NULL не является значением и не требует памяти для своего хранения. При определении атрибута можно указать возможность ввода пустых значений. Если их вводить нельзя, пользователь должен ввести данные. Отсутствие чего-либо нельзя сравнить с отсутствием чего-то другого, точно так же одно пустое значение (NULL) несравнимо с другим. Поэтому следует учесть, что пустые значения нельзя использовать в сравнениях.

**Свойство *DEFAULT***

Могут возникнуть ситуации, когда для атрибута нет возможности определения пустых значений, но атрибут является необязательным для ввода. Применение данного свойства к атрибуту заставляет СУБД вставлять значение по умолчанию при занесении нового кортежа в отношение.

**Свойство *IDENTITY***

Для одного столбца таблицы может быть задано свойство *IDENTITY.* При активации этого свойства SQL Server создает в соответствующем столбце автоматически приращаемое значение.
Это свойство имеет два параметра: исходное значение и инкремент. Исходное значение — это начальное значение, используемое SQL Server. Инкремент определяет, насколько SQL Server увеличивает эту начальную величину при генерировании каждого последующего значения. Свойство *IDENTITY* применимо к точным числовым типам: *bigint, int. smallint, tinyint, decimal* и *numeric.* Для использования с типами данных *decimal* или *numeric* необходимо приравнять в них число разрядов после запятой к нулю.

**Свойство *UNIQUE***

Для того чтобы обеспечить уникальность значений некоторого атрибута, можно применить к атрибуту свойство уникальности UNIQUE. В этом случае СУБД будет гарантировать уникальность значений при добавлении записей в отношение.

**Свойство *CHECK***

В СУБД имеется возможность ограничить множество возможных значений, которые может принимать атрибут, задавая определённый диапазон. Тип данных атрибута уже определяет множество допустимых значений для атрибута, но, когда этого недостаточно, нужно применить свойство CHECK. Например, атрибут «Скидка» может иметь тип данных tinyint, и, следовательно, принимать значения от 0 до 255. Но, чтобы ограничить значения атрибута в диапазоне от 0 до 100 необходимо применить свойство CHECK.

**Вычисляемые столбцы**

В SQL Server можно создавать специальные типы столбцов, которые называются *вычисляемыми* (computed) и содержат вычисления, использующие один или несколько других столбцов таблицы. По умолчанию в вычисляемом столбце хранится определение вычислений, а не физические данные. При получении данных к ним применяется вычисление и возвращаются результаты.

**Свойства ключа. Первичный ключ**

Понятие ключей относится к основополагающим понятиям проектирования и управления базой данных, поэтому оно является очень важным. Основной задачей реляционных баз данных является обеспечение реляционной связи между данными (т.е. связи на уровне отношений между множествами). Поэтому важным требованием к реляционным базам данных является то, что большинство таблиц (за очень редкими исключениями) должно иметь уникальный идентификатор для каждой строки. Уникальный идентификатор позволяет надежно устанавливать связь между строками различных таблиц в базе данных и тем самым формировать отношение между двумя таблицами.

Первичные ключи представляют собой уникальные идентификаторы для каждой строки. Столбец первичного ключа должен содержать уникальные значения (и поэтому в этом столбце не допускается наличие null-значений). Первичные ключи очень важны для нормальной эксплуатации реляционной базы данных, поэтому являются наиболее фундаментальными объектами базы данных по сравнению со всеми прочими ключами и ограничениями. Первичный ключ гарантирует уникальность сочетания значений столбцов, объявленных как принадлежащие к этому первичному ключу. Сами эти уникальные значения служат в качестве идентификаторов для каждой строки в таблице.

**Альтернативный (потенциальный) ключ**

Альтернативный ключ – это множество атрибутов, значения которых уникально в пределах отношения. В отношении может быть несколько альтернативных ключей. Проектировщик БД выбирает в качестве первичного наиболее подходящий альтернативных ключ, т.е. каждый из альтернативных ключей потенциально может стать первичными. Выбирается обычно в качестве первичного ключа, ключ с наименьшей избыточностью, т.е. требующий наименьшее количество памяти для хранения своих значений. Избыточность не является единственным фактором при выборе первичного ключа, большое значение имеют также связи между отношениями, в которых принимает участие первичный ключ. В SQL Server ключом может стать любой атрибут, для которого задано свойство UNIQUE, гарантирующее уникальность значений атрибута, а также отменена возможность ввода пустых значений (NOT NULL), обеспечивающих ограничение сущности.

**Внешний ключ**

Внешние ключи не только обеспечивают целостность данных, но и указывают на наличие связей между таблицами. После задания внешнего ключа на таблице устанавливается зависимость между таблицей, для которой определяется внешний ключ (так называемой *ссылающейся таблицей),* и таблицей, на которую ссылается внешний ключ (так называемой *таблицей, упомянутой в ссылке).* После задания на ссылающейся таблице внешнего ключа любая строка, вставляемая в эту таблицу, должна соответствовать одному из следующих двух условий: иметь согласующуюся с ней строку в столбце (столбцах), которому соответствует внешний ключ таблицы, указанной в ссылке, или иметь значение столбца (столбцов) внешнего ключа NULL.

Свойства CHECK, UNIQUE, DEFAULT, первичный ключ (PRIMARY KEY), внешний ключ (FOREIGN KEY) являются специальными видами ограничений в SQL Server. В данной лабораторной работе эти ограничения рассматриваются как свойства атрибутов.

**2.3 Создание отношений**

После того, как БД создана, необходимо определить отношения, которые будут содержаться в ней. Ниже перечислены отношения БД Automobiles и их атрибуты (рисунок 2.1):

| **Автомобили** | Номер_Двигателя |
|---|---|
| | Марка_Автомобиля |
| | Номер_Автомобиля |
| | Пробег |
| | Дата_Регистрации |
| | Номер_Паспорта |

| **Владельцы** | Номер_Паспорта |
|---|---|
| | ФИО |
| | Адрес |
| | Номер_Кредитной_Карточки |

| **Банковские_Реквизиты** | Номер_Кредитной_Карточки |
|---|---|
| | Тип_Кредитной_Карточки |

| **ТО** | Номер_Талона |
|---|---|
| | Номер_Двигателя |
| | Номер_Паспорта |
| | Дата_ТО |

| **Продажи** | Номер_Заказа |
|---|---|
| | Дата_Покупки |
| | Номер_двигателя |
| | Цена_автомобиля |

Рисунок 2.1 – Отношения БД Automobiles и их атрибуты

Шаги создания отношения «Automobiles», используя окно *Object Explorer* среды SQL Server Management Studio:

Запустите SQL Server Management Studio.

*1)* Перейдите к окну Object Explorer и раскройте раздел Databases.

*2)* Найдите в списке БД «Automobiles» и раскройте содержимое этого узла, выделив раздел **Tables** (таблицы).

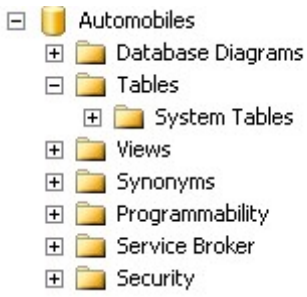



Рисунок 2.2 – Раздел Tables БД Automobiles

*3)* Выберите пункт меню New Table из контекстного меню раздела Tables для запуска конструктора таблиц (**Table Designer**).

*4)* С помощью конструктора таблиц можно создать всю структуру отношения (создать все необходимые атрибуты), а также задать все необходимые свойства для атрибутов. В столбце **Column Name** конструктора вводится название атрибута отношения. При определении названий атрибутов в SQL Server рекомендуется не оставлять пробелы в названиях, т.к. при использовании таких атрибутов приходится использовать квадратные скобки. Например, [Номер двигателя].

*5)* Далее, необходимо определить тип данных (Date Type) для атрибута.

*6)* По умолчанию, SQL Server оставляет возможность ввода пустых значений для атрибута. Если данные атрибута должны присутствовать в каждой строке (в каждом кортеже), то необходимо снять галочку в столбце Allow Nulls или выбрать эту опцию из вкладки Column Properties, как показано на рисунке 2.3.

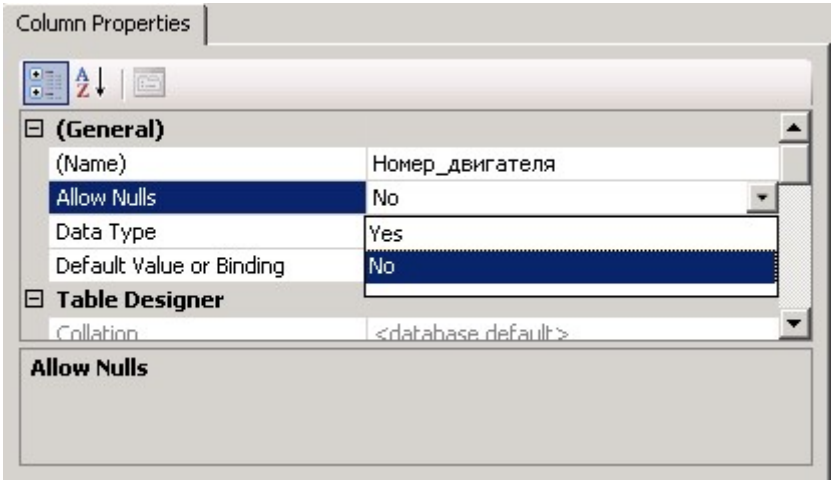



Рисунок 2.3 – Отмена возможности ввода пустых значений

*7)* Другим важным свойством атрибута является возможность задания для него свойства ключа. «Номер_Двигателя» - подходящий кандидат на свойство ключа, так как от него зависят все другие атрибуты отношения и он гарантирует, что в отношении будет храниться только один кортеж с таким ключом. Для задания свойства ключа нажмите правым кликом на строке, где задается имя атрибута и выберите пункт из контекстного меню Set Primary Key как показано на рисунке:

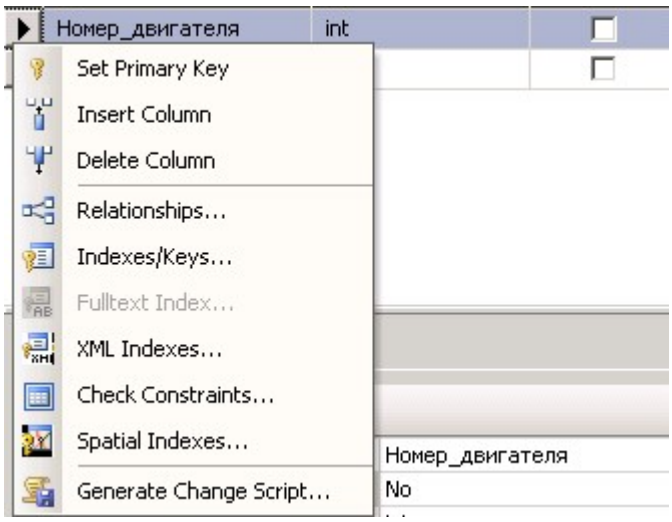



Рисунок 2.4 – Задание свойства ключа для атрибута

*8)* Часто ключевые атрибуты наделяют свойством IDENTITY, что обеспечивает возможность авто-инкремента значения атрибута при добавлении нового кортежа в отношение. Свойство IDENTITY может использоваться не только для ключевых атрибутов. Для задания свойства IDENTITY для атрибута «Номер_Двигателя» нужно перейти в раздел Identity Specification вкладки Column Properties (рисунок 2.5).

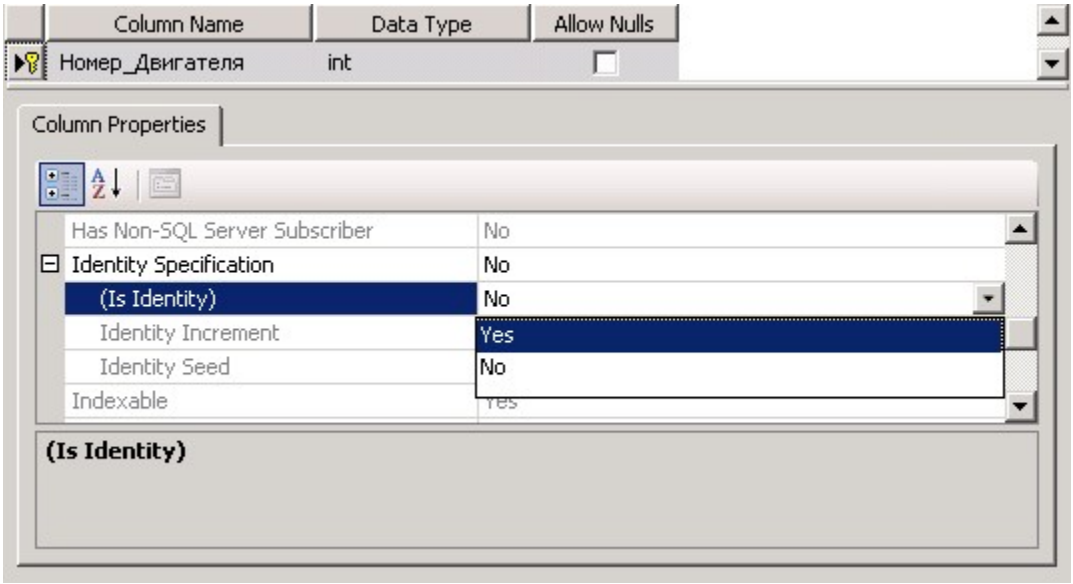



Рисунок 2.5 – Задание свойства *IDENTITY* для атрибута Номер_Двигателя

*9)* Т.к. номера автомобилей уникальны, то есть смысл применить для атрибута «Номер автомобиля» свойство UNIQUE. Из контекстного меню атрибута выбирается пункт **Indexes/Keys…** В результате появится диалоговое окно Indexes/Keys (рисунок 2.6).

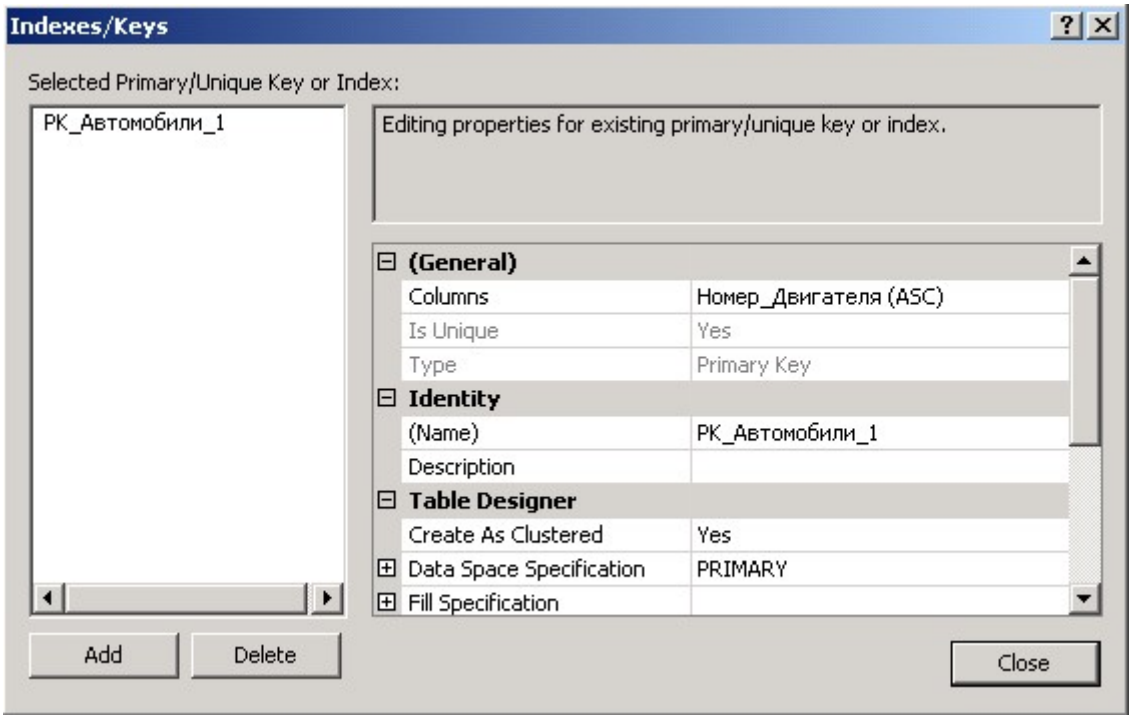



Рисунок 2.6 – Диалоговое окно Indexes/Keys

В данном диалоговом окне имеется возможность задания свойств первичных ключей и уникальных значений для атрибутов. Как видно из рисунка у нас имеется список ограничений. Пока в списке присутствует только одно ограничение, имеющее название PK_Автомобили_1. Это ограничение было автоматически создано SQL Server когда был определён первичный ключ для атрибута «Номер_двигателя».

Чтобы добавить ограничение уникальности (свойство уникальности) для атрибута «Номер_Автомобиля» нажмите кнопку **Add** диалогового окна Indexes/Keys. Далее необходимо задать свойство уникальности атрибуту «Номер_Автомобиля». На рисунке 2.7 показаны необходимые установки свойств для обеспечения уникальности атрибута «Номер_ Автомобиля».

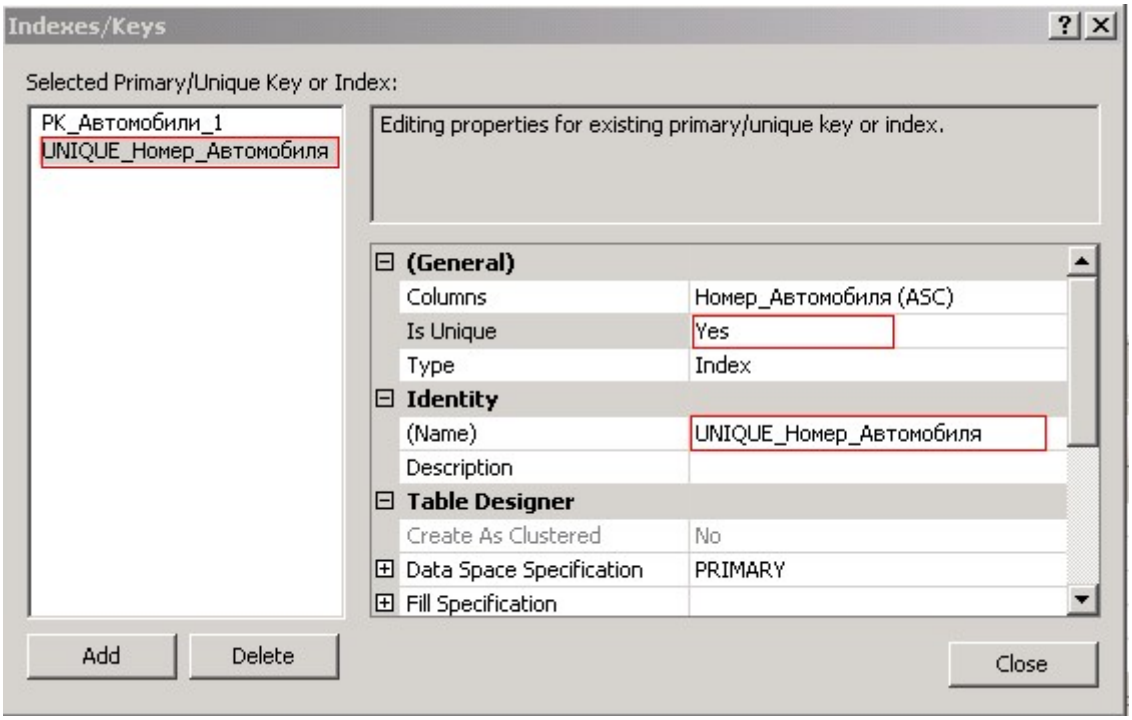



Рисунок 2.7 – Добавление свойства уникальности через окно Indexes/Keys

*11)* В отношении «Автомобили» атрибут «Дата_Регистрации» фиксирует дату регистрации автомобиля. Эта дата может совпадать с датой внесения записи об автомобиле в БД. Поэтому значение этого атрибута может быть задано по умолчанию, с помощью свойства DEFAULT. Для задания даты по умолчанию в SQL Server имеется специальная функция, которая возвращает текущее время. Ее можно использовать для генерации значений по умолчанию для атрибута. Для задания свойства DEFAULT выделим атрибут

«Дата_Регистрации» мышкой в конструкторе таблиц. Далее во вкладке Column Properties зададим в поле Default value or Binding значение GETDATE(), что показано на рисунке 2.8.

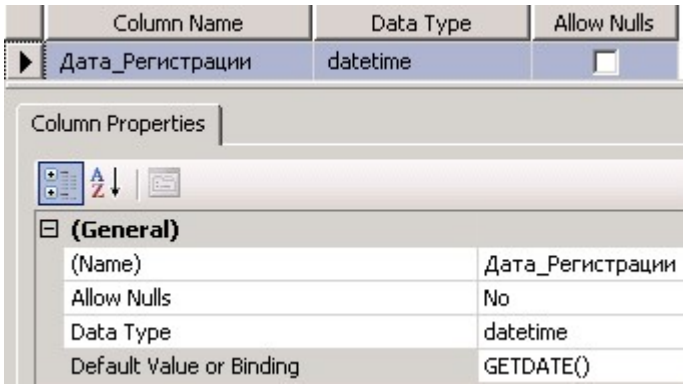



Рисунок 2.8 – Задание свойства *DEFAULT*

*12)* В СУБД должен существовать механизм защиты данных от некорректного ввода. Одно из правил проектирования гласит, что нельзя доверять пользовательскому вводу, поэтому нужно всегда предусматривать случаи появления неправильных данных и соответствующей реакции СУБД на некорректный ввод. Для этих целей и существует свойство (ограничение) CHECK. С его помощью

можно организовать т.н. бизнес-правила. Атрибут «Пробег» имеет тип *smallint*. Диапазон возможных значений для данного типа от -32768 до 32767. Но пробег не может быть отрицательным. Поэтому целесообразно ограничить значения от 0 до 32767. Для задания свойства CHECK для атрибута выберем из контекстного меню пункт Check Contraints. В результате появится диалоговое окно Check Constraints. Для того чтобы добавить новое ограничение CHECK нажмем кнопку Add (Добавить). После этого в списке свойств зададим имя ограничения, а также выражение CHECK. В нашем случае атрибут «Пробег» должен принимать значения, большие 0 и выражение примет вид **Пробег >= 0** (рисунок 2.9).

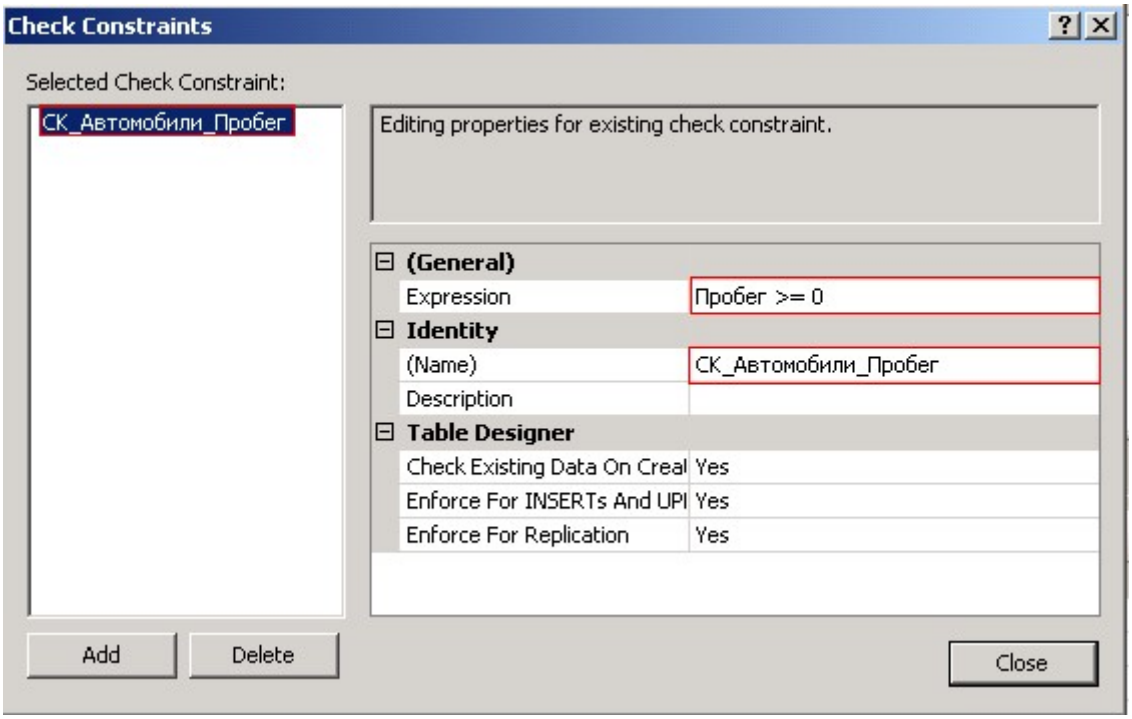



Рисунок 2.9 – Диалоговое окно Check Constraints

*13)* Определяем все остальные атрибуты отношения «Автомобили». В результате конструктор таблиц принимает следующий вид:

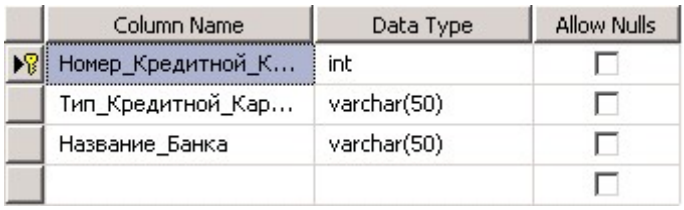
| Column Name | Data Type | Allow Nulls |
|---|---|---|
| Номер_Двигателя | int | ☐ |
| Марка_Автомобиля | varchar(50) | ☐ |
| Номер_Автомобиля | varchar(50) | ☐ |
| Пробег | smallint | ☐ |
| Дата_Регистрации | datetime | ☐ |
| Номер_Паспорта | varchar(50) | ☐ |
| | | ☐ |

Рисунок 2.10 – Конструктор таблиц после добавления всех атрибутов

*14)* После определения всех атрибутов отношения и их свойств необходимо сохранить отношение в БД. Для этого нужно нажать кнопку сохранения в панели инструментов среды SQL Server Management Studio.

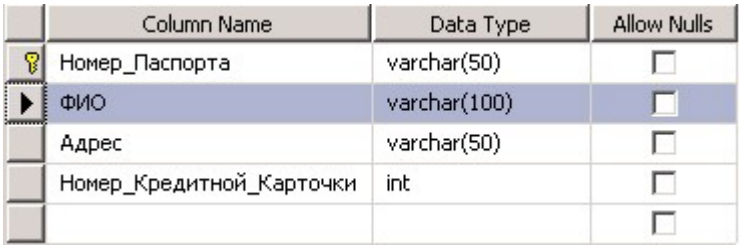
*15)* После создания отношения можно просмотреть его свойства, выбрав пункт контекстного меню Properties(свойства) отношения Автомобили. В результате появится диалоговое окно Table Properties.

Изучив основы создания отношений с помощью конструктора таблиц, создадим остальные отношения БД «Automobiles». Ниже представлены изображения конструкторов таблиц после создания всех отношений (рисунки 2.11 – 2.14):



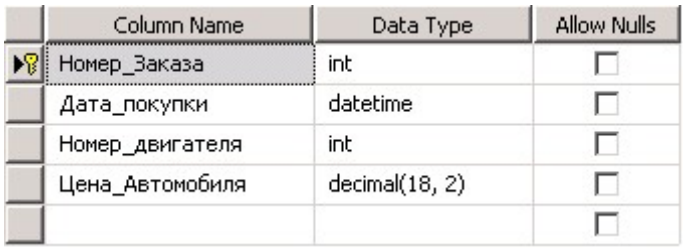
| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| 🔑 | Номер_Кредитной_К... | int | ☐ |
| | Тип_Кредитной_Кар... | varchar(50) | ☐ |
| | Название_Банка | varchar(50) | ☐ |
| | | | ☐ |

Рисунок 2.11 – Отношение «Банковские_Реквизиты»



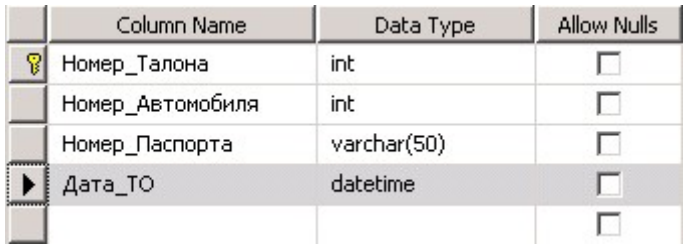
| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| 🔑 | Номер_Паспорта | varchar(50) | ☐ |
| ▶ | ФИО | varchar(100) | ☐ |
| | Адрес | varchar(50) | ☐ |
| | Номер_Кредитной_Карточки | int | ☐ |
| | | | ☐ |

Рисунок 2.12 – Отношение «Владельцы»



| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| 🔑 | Номер_Заказа | int | ☐ |
| | Дата_покупки | datetime | ☐ |
| | Номер_двигателя | int | ☐ |
| | Цена_Автомобиля | decimal(18, 2) | ☐ |
| | | | ☐ |

Рисунок 2.13 – Отношение «Продажи»



| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| 🔑 | Номер_Талона | int | ☐ |
| | Номер_Автомобиля | int | ☐ |
| | Номер_Паспорта | varchar(50) | ☐ |
| ▶ | Дата_ТО | datetime | ☐ |
| | | | ☐ |

Рисунок 2.14 – Отношение «ТО»

### 2.4 Создание связей между отношениями

**Связь один ко многим**

К этому моменту мы имеем готовые структуры пяти таблиц базы данных Automobile: «Автомобили», «Банковские_Реквизиты», «Владельцы», «Продажи» и «ТО». Для каждой таблицы описаны атрибуты и определены первичные ключи. Тем не менее мы пока не определили никаких зависимостей между данными. Связь между владельцами и автомобилями может описываться отношением один ко многим: каждый автомобиль имеет единственного владельца, в то время как владелец может иметь несколько автомобилей. Отношение один ко многим часто записывается в виде 1:N.

Структура БД Automobiles уже спроектирована с учетом зависимостей между отношениями. Поэтому остается пояснить почему структура отношений сформирована именно так, а не иначе. Возьмем, к примеру, два отношения: «Автомобили» и «Владельцы». На первый взгляд между ними нет никакой связи, хотя нам уже известно, что между этими сущностями существует связь один-ко- многим. Но как найти владельца автомобиля из таблицы «Автомобили»? Ведь в отношении «Автомобили» нет атрибутов, содержащих, например, имя владельца. Но мы можем увидеть, что в отношении «Автомобили» есть атрибут «Номер_Паспорта», а по номеру паспорта мы можем найти имя владельца в отношении «Владельцы». Мы знаем, что по номеру паспорта мы однозначно можем определить владельца автомобиля (ведь «Номер_Паспорта» является первичным ключом и гарантирует уникальность владельца в БД). Таким образом атрибут «Номер_Паспорта» в отношении «Автомобили» **ссылается** на данные, находящиеся в отношении «Владельцы». В результате мы не храним имя владельца, его адрес и номер кредитной карточки в отношении «Автомобили» и, значит, не дублируем эту информацию. Мы добились **неизбыточности** данных путем введения **внешнего ключа** в отношение «Автомобили», который и обеспечивает связь двух отношений по принципу один ко многим.

Итак, внешние ключи – важный механизм, с помощью которого реализуются связи между отношениями, а также решаются многие проблемы, связанные с избыточностью данных.

**Связь многие ко многим**

Классическим примером данной связи является связь между отношениями «Книги» и «Авторы». Некоторые книги написаны несколькими авторами, к тому же некоторые авторы выпустили более одной книги. Другими словами, авторы и книги связаны отношением многие ко многим, которое часто записывается в виде *N:M*. Связь многие ко многим на практике реализуется с помощью двух связей один ко многим и дополнительного отношения для хранения этих связей.

**Связь один к одному**

Если вы обнаружите между таблицами отношение типа 1:1, то, скорее всего, их лучше объединить в одну таблицу. Основной причиной использования связи 1:1 является увеличение скорости исполнения запросов. Например, если вы редко используете какую-либо информацию о книге (заметки об авторских правах или список измененных страниц), ее можно поместить в отдельную таблицу, чтобы не обрабатывать её а общих запросах.

Покажем пример создания связей в среде SQL Server Management Studio между отношениями «Автомобили» и «Владельцы».

1) Первым делом необходимо определить отношение, которое будет содержать внешний ключ, т.е. будет ссылаться на информацию, содержащуюся в другом отношении. Это отношение, которое выступает со стороны многих – «Автомобили». Далее, необходимо открыть конструктор таблиц и из контекстного меню выбрать пункт **Relationships** (Связи), чтобы вызвать конструктор связей (**Relationship Designer**).

2) После выбора пункта меню Relationships появится диалоговое окно **Foreign Key Relationship.** С его помощью можно добавлять новые связи или удалять ненужные. Чтобы добавить новую связь нажмем кнопку **Add** (Добавить). В результате в диалоговом окне появится новая связь, для которой можно задавать свойства. Первое свойство, которое мы установим – имя связи. Важно назначать понятное имя связи, чтобы по нему стало ясно какие отношения вовлечены в связь. Принято устанавливать имя по следующему формату: FK_[Имя отношения со стороны одного]_[Имя отношения со стороны многих]. Таким образом, наша связь получит имя FK_Владельцы_Автомобили:

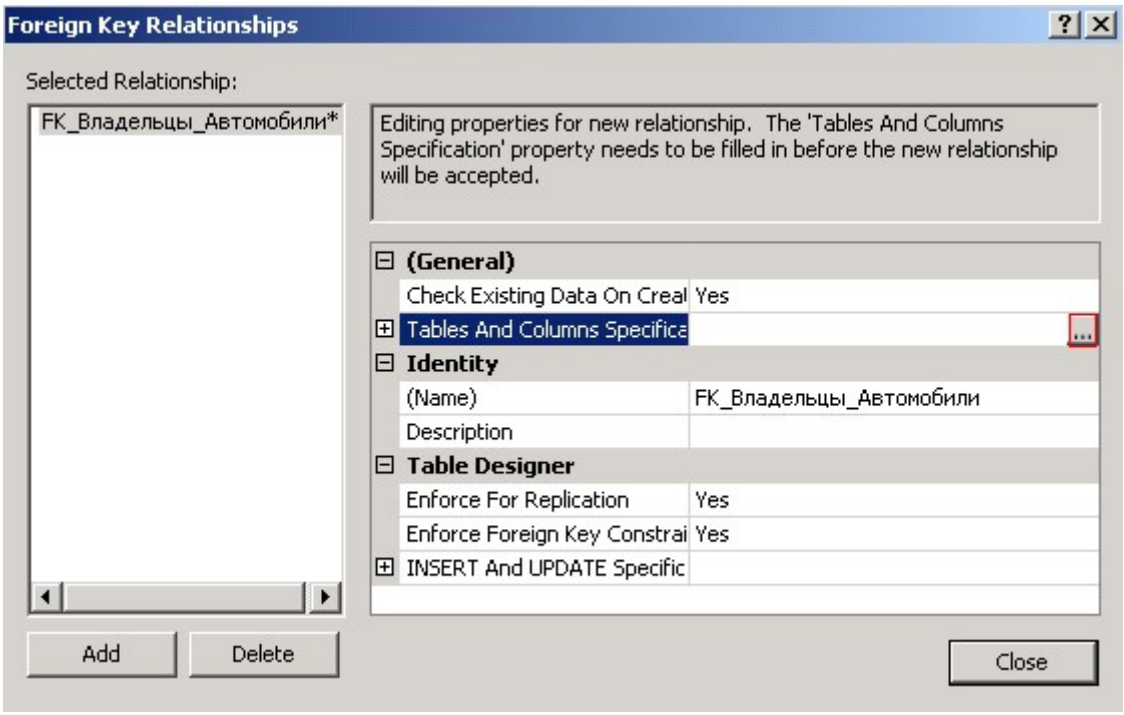



Рисунок 2.15 – Задание имени для связи

3) Далее необходимо определить отношение, содержащее первичный атрибут, с которым нужно построить связь один ко многим. Открывается окно Tables and Columns (позволяет связать

атрибуты отношений, участвующих в связи) нажатием на кнопку возле параметра Tables And Columns Specification. Связываемые атрибуты: со стороны одного – ключевой атрибут «Номер_Паспорта» отношения «Владельцы» (Primary Key Table), со стороны многих – атрибут (внешний ключ) «Номер_Паспорта» отношения «Автомобили» (Foreign Key Table).

Всё это показано на рисунке 2.16.

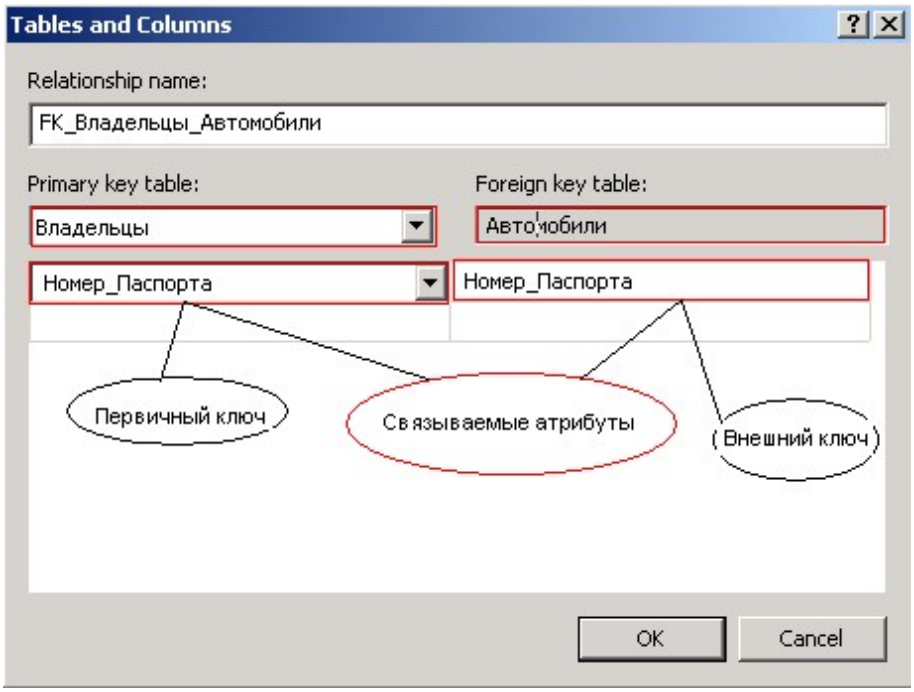



Рисунок 2.16 – Диалоговое окно Tables and Columns

4) После того, как связываемые атрибуты определены, нужно определить один важный момент. Для этого ответим на вопрос: “Что случится с записями дочернего отношения, если мы удалим запись из родительского отношения?” Эта ситуация может возникнуть когда мы удаляем запись о владельце из отношения «Владельцы». В итоге в отношении «Автомобили» могут появиться записи, внешний ключ которых ссылается на недействительные данные, т.е. мы не сможем определить владельца автомобиля по его номеру паспорта. Такая же ситуация может возникнуть когда мы изменяем номер паспорта в одной из записей отношения «Владельцы». Такие случаи СУБД должна корректно обрабатывать.

По умолчанию СУБД SQL Server защищает от удаления такие строки родительской таблицы, которым соответствуют строки, существующие в дочерней таблице. Но иногда предпочтительный способ организации работы приложения состоит в том, чтобы удаление всех зависимых строк, препятствующих удалению строки, на которую они ссылаются, происходило автоматически. Аналогичный принцип применяется к обновлению строк, поскольку требуется, чтобы в зависимых строках автоматически восстанавливалась ссылка на вновь обновленные строки (*опция Cascade диалогового окна Foreign Key Relationship*).

Немного реже встречается ситуация, в которой требуется перевести ссылающуюся строку в какое-то другое заведомо известное состояние. В последнем случае имеется возможность либо задать

в зависимой строке NULL-значение (*опция Set Null диалогового окна Foreign Key Relationship*), либо применить для соответствующего столбца то значение, которое предусмотрено по умолчанию (*опция Set Default диалогового окна Foreign Key Relationship*). Процесс автоматического осуществления подобных операций удаления и обновления известен под названием *каскадного выполнения действий.* Фактически этот процесс, особенно применительно к операциям удаления, может проходить через несколько уровней зависимостей (когда одна строка зависит от другой, вторая зависит от третьей и т.д.).

Каскадное выполнение действий предусмотрено и в СУБД SQL Server.

Все описанные выше опции по каскадному выполнению действий можно выбрать из диалогового окна Foreign Key Relationship в разделе **INSERT and UPDATE Specification**, как показано на рисунке 2.17:

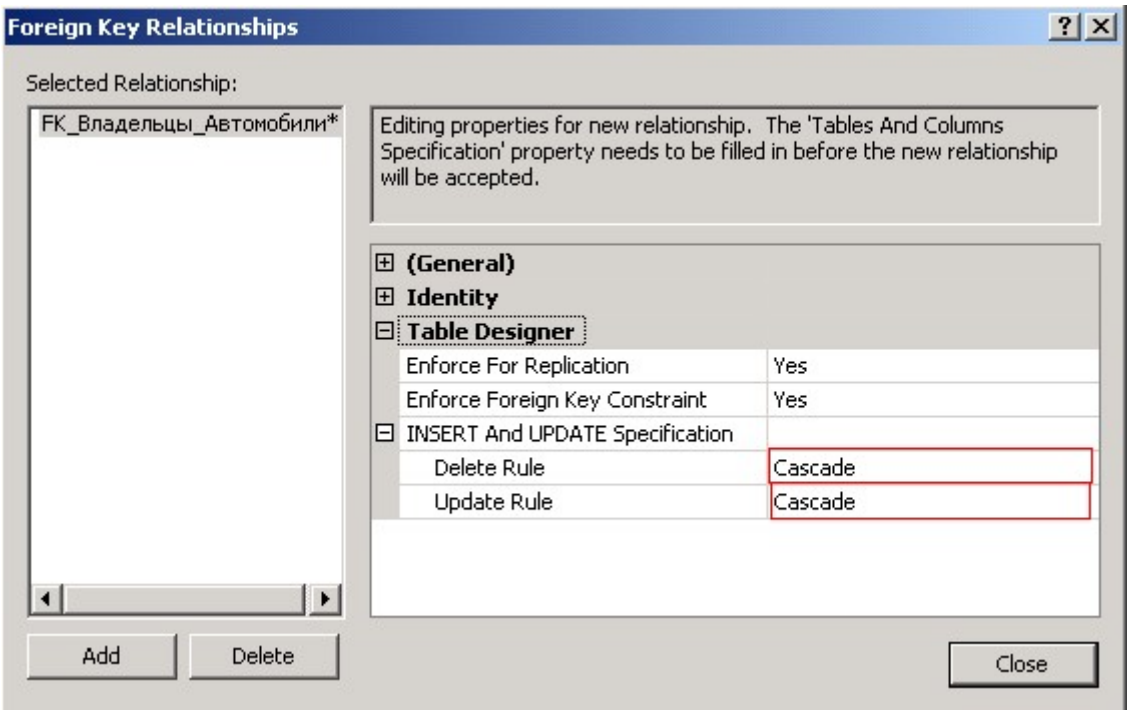



Рисунок 2.17 – Каскадное выполнение действий

5) Далее необходимо определить все остальные связи: выявить внешние ключи отношений и связать их с первичными ключами родительских отношений.

### 2.5 Заполнение БД данными

После того как БД создана и в ней определены все необходимые отношения и установлены связи, можно приступить к заполнению БД данными. Для того чтобы внести новые записи (кортежи) в отношение «Владельцы» из контекстного меню окна Object Explorer выберем пункт Edit Top 200 Rows (Редактировать первые 200 строк).

В результате появится таблица, в которую можно вводить новые записи о владельцах.

Введем одну пробную запись. Но при вводе новой записи возникает ошибка (исключение).

Появляется диалоговое окно описания исключения, где написано, что наш ввод нарушает ограничение целостности по ссылке при попытке ввести данные в столбец Номер_Кредитной_Карточки (рисунок 2.18). Но ведь так и должно быть – мы ещё не ввели никаких данных о кредитных карточках и поэтому не можем ссылаться на номера кредитных карточек.

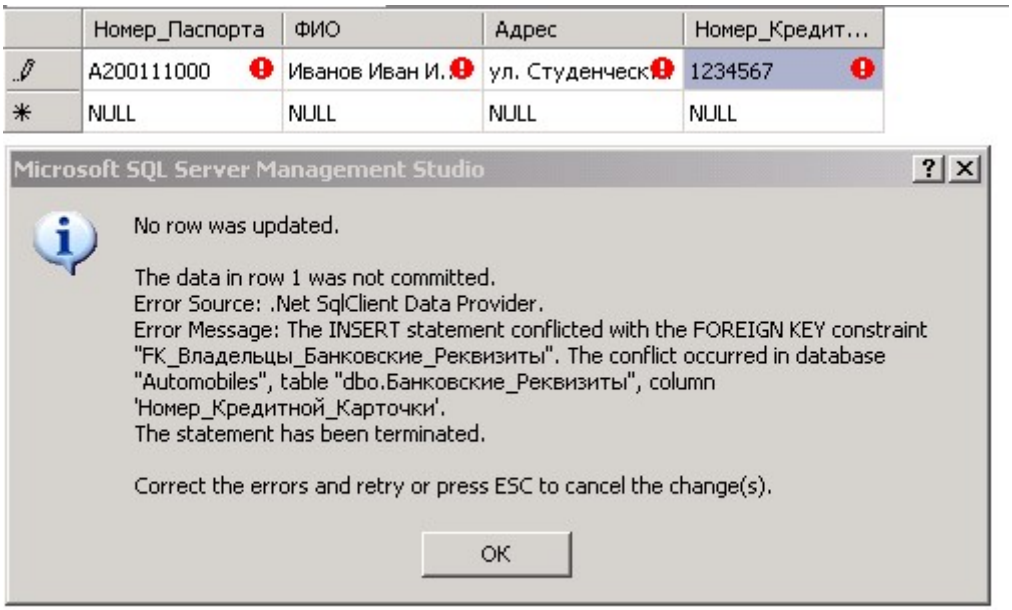



Рисунок 2.19 – Ошибка при вводе пробной записи в таблицу «Владельцы»

После появления данной ошибки становится ясно, что вводит данные вначале необходимо в те отношения, у которых нет внешних ключей, иначе необходимо наличие актуальных данных в БД на которые можно сослаться с помощью внешнего ключа. Таким образом, прежде чем заполнять информацией таблицу «Владельцы» нужно заполнить данными таблицу «Банковские_Реквизиты».

Откроем таблицу «Банковские_Реквизиты» и введем несколько строк данных:

| Номер_Кредит... | Тип_Кредитно... | Название_Банка |
|---|---|---|
| 123 | VISA | AGROINDBANK |
| 132 | VISA | AGROINDBANK |
| 231 | MAESTRO | EUROCREDITBANK |
| 321 | VISA ELECTRON | MOBIASBANK |
| NULL | NULL | NULL |

Рисунок 2.20 – Заполнение данными отношения «Банковские_Реквизиты»

После заполнения таблицы «Банковские_Реквизиты» данными мы можем приступить к внесению информации в таблицу «Владельцы», соблюдая при этом ограничения внешнего ключа «Номер_Кредитной_Карточки».

### 2.6 Создание отношений с помощью дизайнера диаграмм

В SQL Server Management Studio имеется удобное средство для визуального представления схемы БД – дизайнер диаграмм (database diagramming). С помощью этой удобной утилиты можно наглядно рассмотреть схему БД, структуру каждого отношения, а также связи между ними. Более

того, с помощью этой утилиты можно также создавать отношения, определять их структуру, задавать свойства атрибутам и. т.д. Поэтому дизайнер диаграмм является одним из альтернативных способов создания схем БД в среде SQL Server Management Studio.

По сравнению с созданием отношений с помощью конструктора таблиц в дизайнере диаграмм возможно построение всей схемы БД в одном месте. Также дизайнер диаграмм помогает в документировании проекта – можно наглядно представить результаты проектирования БД.

Как уже упоминалось выше, дизайнер диаграмм позволяет определять структуру БД. Но ограничением является то, что создавать с его помощью можно только таблицы БД. Такие объекты как хранимые процедуры, триггеры, представления нельзя спроектировать при помощи дизайнера диаграмм.

Для создания схемы перейдите к БД «Automobiles» в окне Object Explorer и раскройте содержимое БД. Раскройте узел Database Diagrams окна Object Explorer. Для того, чтобы добавить диаграммы БД SQL Server должен установить специальные объекты, которые будут поддерживать их работу. По умолчанию SQL Server Management Studio не устанавливает поддержу диаграмм. Поэтому, когда мы раскроем узел Database Diagrams, проявиться диалоговое окно с предложением установить поддержку для диаграмм.

После этого, у нас появляется возможность добавления новых диаграмм БД. Из контекстного меню узла Database Diagrams выбираем пункт New Database Diagram. После того, как выбран пункт меню New Database Diagram открывается дизайнер диаграмм. Прежде чем начать работу, дизайнер предлагает нам добавить в диаграмму существующие в БД отношения. На рисунке 2.21. представлено диалоговое окно Add Table, с помощью которого можно осуществить этот выбор.

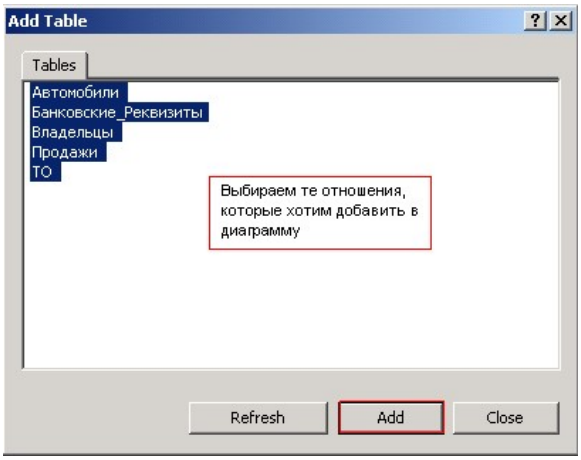



Рисунок 2.21 – Добавление отношений в диаграмму

После нажатия кнопки Add(Добавить), все выбранные отношения добавляются в диаграмму (рисунок 2.22), на которой представлены отношения и их атрибуты, а также все существующие между отношениями связи. Со стороны «одного» связь обозначается знаком ключа, а со стороны «многих» – знаком бесконечности.

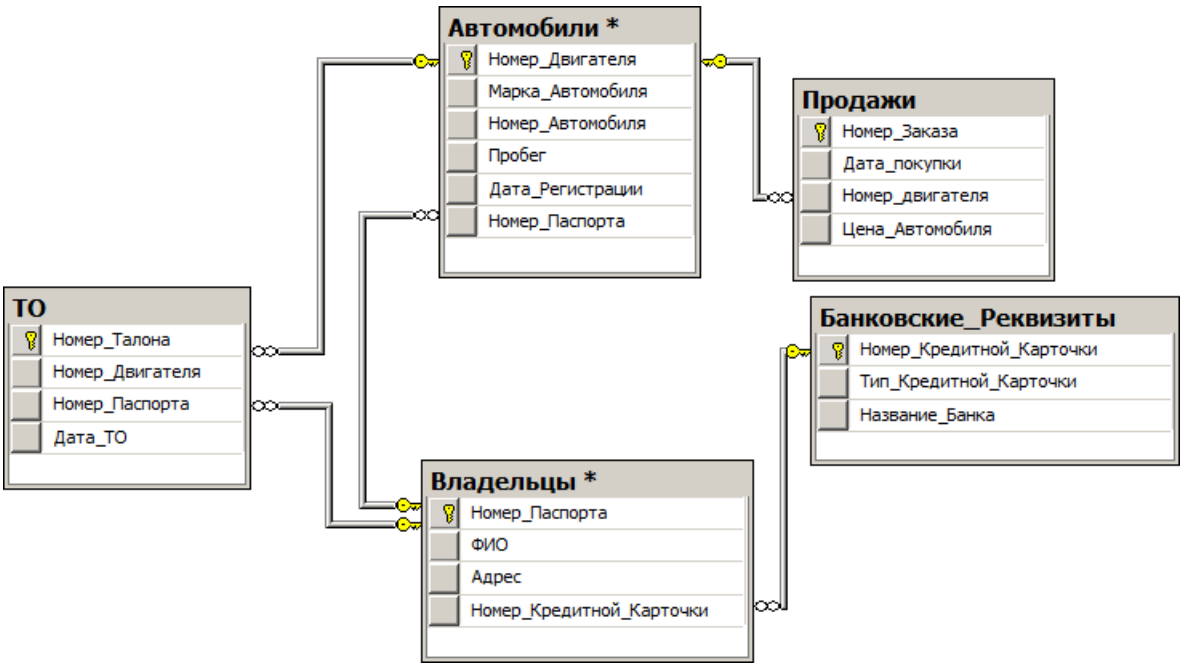



Рисунок 2.22 – Диаграмма БД «Automobiles»

Способ создания отношений схож с тем, что мы рассматривали в предыдущих разделах, только в данном случае этот процесс более наглядный. Создадим отношение «Автоклубы».

Для создания отношения «Автоклубы» выберем пункт New Table из контекстного меню дизайнера диаграмм. Дизайнер диаграмм предложит ввести название создаваемого отношения. В нашем случае мы вводим в диалоговое окно Choose Name (Выбор названия) название «Автоклубы». После определения названия отношения, в дизайнере диаграмм появиться форма для определения структуры отношения (схожая с представлением конструктора таблиц). Определим все необходимые атрибуты и их свойства (количество свойств атрибутов, отображаемых на диаграмме, можно изменить, выбрав пункт контекстного меню Table View).

После того как были определены все атрибуты отношения «Автоклубы», необходимо определить связь между отношениями «Автоклубы» и «Владельцы». Как упоминалось выше, для организации связи многие ко многим необходимо построить еще одно отношение, которое будет вступать в связь один ко многим с отношениями со стороны «многих». Определим отношение

«Владельцы_Автоклубы» с атрибутами «Регистрационный_Номер» и «Номер_Паспорта». В данном случае первичным ключом будет пара атрибутов «Регистрационный_Номер» и «Номер_Паспорта», т.к. именно пара значений этих атрибутов является уникальной в пределах отношения. Также атрибут «Регистрационный_Номер» является внешним ключом, ссылающимся на запись из отношения «Автоклуб», а атрибут «Номер_Паспорта» ссылается на владельца автомобиля из отношения «Владельцы». Таким образом, связь N:M между отношениями «Автоклуб» и «Владельцы» организуется на практике путем создания нового отношения и двух связей 1:N. Реализуем сказанное выше с помощью дизайнера диаграмм.

Для начала создадим отношение «Владельцы_Автоклубы» с помощью дизайнера диаграмм: определим атрибуты и их свойства, создадим первичный ключ, состоящий из двух атрибутов. Для задания первичного ключа нужно выделить все первичные атрибуты из и контекстного меню выбрать пункт меню Set Primary Key.

Далее нужно создать две связи один ко многим. В дизайнере диаграмм это реализуется путем выбора атрибута (внешнего ключа) нажатием левой кнопки мыши и его перетаскивания указателем мыши на первичный атрибут отношения, выступающего со стороны многих. Этот процесс наглядно демонстрируется на рисунке 2.23.

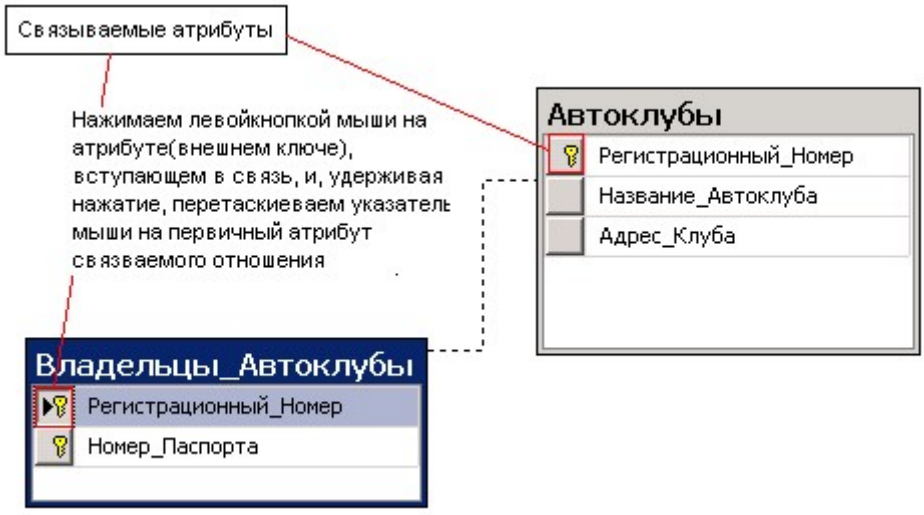



Рисунок 2.23 – Создание связи один ко многим

После того как кнопка мыши будет отпущена (в области связываемого отношения на диаграмме), появится диалоговое окно Tables and Columns, в котором можно задать имя связи, а также атрибуты, участвующие в связи. На рисунке 2.24 показано диалоговое окно Tables and Columns, где красной рамкой выделена область, в которой SQL Server отображает имя создаваемой связи, предложенное СУБД. В большинстве случаев имя связи, задаваемое по умолчанию SQL Server, соответствует соглашению по именованию, которое мы приняли в предыдущем разделе.

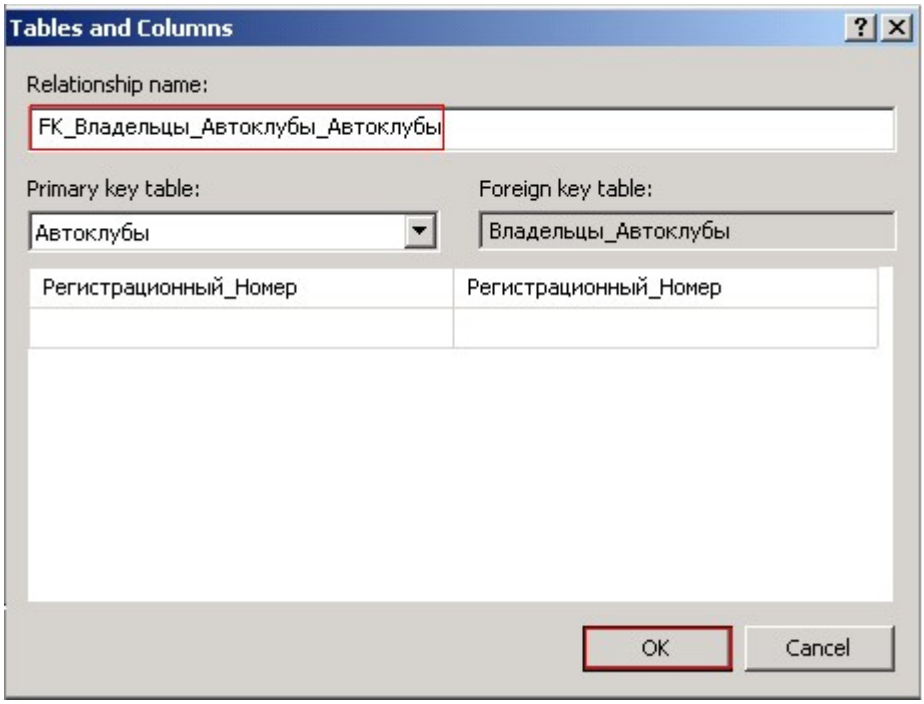



Рисунок 2.24 – Определение названия связи

Далее после нажатия OK открывается Foreign Key Relationships, которое также нам известно. Здесь можно установить свойства связи, в частности, можно выбрать одну из опций каскадного выполнения действий. В нашем случае мы выберем опцию Cascade в разделе INSERT And UPDATE Specification для обоих правил каскадирования: Update Rule (обновление) и Delete Rule (удаление).

После сохранения изменений, мы можем увидеть результат реализации связи многие ко многим между отношениями «Владельцы» и «Автоклубы»:

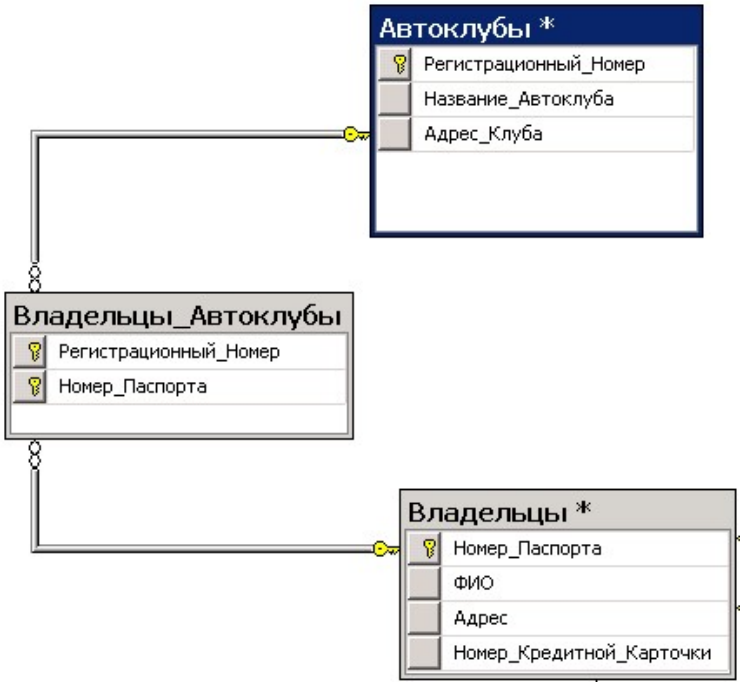



Рисунок 2.25 – Связь многие ко многим между отношениями «Владельцы» и «Автоклубы»

## Контрольные вопросы:

1. Назовите категории типов данных, которые поддерживаются в SQL Server.
2. Объясните значение свойств IDENTITY, CHECK и DEFAULT.
3. Какие методы создания отношений предоставляются средой SQL Server Management Studio?
4. Какие преимущества и недостатки у метода создания отношений при помощи дизайнера диаграмм?
5. Какие ключи участвуют в построении связей между отношениями?
6. Какие типы связей бывают между отношениями?
7. Какие опции предоставляет SQL Server для обеспечения каскадного выполнения ограничений ссылочной целостности?
8. Каким образом следует заполнять БД данными?

## Задания к лабораторной работе:

**Задание 1. Создание базы данных и отношений**

Создайте БД в среде SQL Server Management Studio, содержащую минимум пять отношений.

**Задание 2. Связывание отношений**

Создайте связи между отношениями. В БД должны присутствовать все типы связей: 1:1, 1:N и N:M. С помощью дизайнера диаграмм создайте два отношения, участвующие в связи N:M, остальные отношения создайте с помощью конструктора таблиц.

**Задание 3. Установка свойств атрибутов в отношениях**

Задайте, где это необходимо, свойства атрибутов: *IDENTITY* (авто-инкремент), *UNIQUE* (уникальные значения, альтернативные ключи), *DEFAULT* (значение по умолчанию), *CHECK* (условие, которому должны удовлетворять значения атрибутов), а также возможность ввода пустых значений.

**Задание 4.** Заполните все отношения кортежами (минимум 10 кортежей в каждое) с соблюдением ограничений ссылочной целостности.

## Дополнительная информация:

**1.** Vitalie Cotelea, Marian Cotelea. Microsoft SQL Server 2019: Pas cu pas. – Universitatea Tehnică a Moldovei, Departamentul “Ingineria Software si Automatică”. - Chişinău: S.n., 2020 (Tipografia “Foxtrot”). -474 p.