**Лекция 9:**

**Создание физической модели хранилища данных**

**Цель лекции**

Изучив материал настоящей лекции, вы будете знать:

* основные объекты, из которых состоит физическая реляционная база данных;
* какие стандартные типы данных предусмотрены для реляционных баз данных;
* как создать *таблицу* реляционной базы данных;
* как определить *спецификацию колонки*;
* какие *ограничения* поддерживаются в реляционных базах данных;

сможете:

* построить *физическую модель данных* для хранилища данных;
* применять команды SQL *CREATE TABLE*, *CREATE INDEX*, *ALTER TABLE* ;
* определять *спецификации колонок* ;

и научитесь:

* создавать физическую модель хранилища данных.

**Литература**: [[2](https://intuit.ru/studies/courses/599/455/literature#literature.2)], [[3](https://intuit.ru/studies/courses/599/455/literature#literature.3)], [[37](https://intuit.ru/studies/courses/599/455/literature#literature.37)], [[38](https://intuit.ru/studies/courses/599/455/literature#literature.38)].

**Объекты физической модели данных**

**Введение**

В предыдущих лекциях мы изучали различные аспекты методов логического проектирования ХД. Методы логического проектирования основываются на абстрактном рассмотрении данных. Логическая модель никак не связана с конкретной реализацией модели в БД СУБД.

На практике ХД создаются и эксплуатируются как БД под управлением конкретной СУБД. БД, реализующие ХД, создаются на основе *физической модели данных*, разработанной проектировщиком ХД и реализованной в виде объектов БД.

*Физическая модель данных*, напротив, зависит от конкретной СУБД, в ней содержится информация обо всех объектах базы данных. Поскольку стандартов на объекты базы данных не существует (например, нет стандарта на типы данных), физическая модель зависит от конкретной реализации СУБД и ее диалекта SQL. Следовательно, одной и той же логической модели данных могут соответствовать несколько разных физических моделей.

Основными объектами логической модели данных являются сущности, атрибуты и взаимосвязи. *Физическая модель данных*, как правило, создается на основе логической, поэтому каждому объекту логической модели соответствует объект физической модели (хотя соответствие может быть неоднозначным). В *физической модели данных* сущности логической модели данных соответствует *таблица*, экземпляру сущности – строка в *таблице*, а атрибуту – колонка *таблицы*. Кроме перечисленных выше объектов, физическая модель может содержать объекты, тип которых зависит от СУБД: *индексы*, *представления*, последовательности, *триггеры*, процедуры и т.п. Если в логической модели данных не имеет большого значения, какой конкретно тип данных у атрибута, то в физической важно описать всю информацию о конкретных объектах.

Далее, при изложении материала, мы будем предполагать, что имеем дело с реляционными или объектно-реляционными СУБД и соответствующими им диалектами SQL.

Можно выделить два этапа создания *физической модели данных* [[48](https://intuit.ru/studies/courses/599/455/literature#literature.48)]. Основными целями первого этапа являются:

* удовлетворение потребности в хранении данных предметной области в рамках реляционной модели данных, т.е. должны быть созданы *базовые таблицы* для хранения информации обо всех сущностях предметной области;
* удовлетворение требования целостности данных, т.е. должны быть определены типы колонок и наложены *ограничения* на значения колонок *базовых таблиц*, которые бы следовали из бизнес-правил предметной области;
* удовлетворение требования **ссылочной целостности** (*referential integrity*, RI), т.е. в случае принятия решения о *поддержки ссылочной целостности* встроенными средствами СУБД должны быть наложены *ограничения* ссылочной целостности на *таблицы*, исходя из бизнес-правил ссылочной целостности предметной области;
* удовлетворение (частично) требования независимости *представления* данных для конечного *пользователя* от характера физического хранения данных.

На первом этапе в рамках требований реляционной модели создаются объекты хранения данных, соответствующие сущностям и взаимосвязям логической модели данных — *таблицы*, *индексы*, *представления* и т.д.

Главной целью второго этапа является обеспечение требуемого уровня производительности. Для достижения этой цели необходимо учитывать как особенности реализации СУБД, для которой создается физическая модель, так и особенности функционирования будущей информационной системы в целом. Обычно производительность БД измеряется в терминах **производительности транзакций** (transaction performance).

Для повышения *производительности транзакций* могут быть модифицированы объекты, созданные на первом этапе, или созданы новые объекты БД. Другими словами, разработка физической модели представляет собой итерационный процесс, причем итераций (создание объектов – анализ транзакций – модификация объектов – анализ транзакций) может быть несколько. Вопросы повышения *производительности транзакций* будут рассмотрены в одной из следующих лекций.

**Иерархия объектов реляционной базы данных**

Одной из главных задач, которые обязан решить проектировщик на стадии проектирования физической модели ХД, является задача превращения объектов логической модели данных в объекты реляционной БД. Для решения этой задачи проектировщику необходимо знать: а) какими объектами располагает реляционная база данных в принципе; б) какие объекты поддерживает конкретная СУБД, которая выбрана для реализации базы данных.

Таким образом, мы предполагаем, что решение о выборе СУБД уже принято руководителем ИТ-проекта и согласовано с заказчиком базы данных, т.е. СУБД задана. Проектировщик ХД должен ознакомиться с документацией, в которой описан диалект SQL, поддерживаемый выбранной СУБД. В настоящей лекции предполагается, что была выбрана СУБД семейства MS SQL Server компании Microsoft, хотя подавляющая часть материала охватывает объекты в любой промышленной реляционной СУБД.

Иерархия объектов реляционной БД прописана в стандартах по SQL, в частности, в *стандарте SQL-92*, на который мы будем ориентироваться при изложении материала настоящей лекции. Этот стандарт поддерживается практически всеми современными СУБД. Иерархия объектов БД показана на [рис. 11.1](https://intuit.ru/studies/courses/599/455/lecture/10170?page=1#image.11.1).



**Рис. 11.1.**Иерархия объектов реляционной базы данных, соответствующая стандарту SQL-92

На самом нижнем уровне находятся объекты, с которыми работает реляционная БД, — столбцы (колонки) и строки. Они, в свою очередь, группируются в *таблицы* и *представления*. Заметим, что в контексте лекции атрибуты, колонки, столбцы и поля считаются *синонимами*. То же относится и к терминам "строка", "запись" и "кортеж".

*Таблицы* и *представления*, которые представляют физическое отражение логической структуры БД, собираются в *схему*. Несколько *схем* собираются в *каталоги*, которые затем могут быть сгруппированы в *кластеры*.

Следует отметить, что ни одна из групп объектов стандарта SQL-92 не связана со структурами физического хранения информации в памяти компьютеров.

Помимо указанных на рисунке объектов в реляционной базе данных могут быть созданы *индексы*, *триггеры*, события, хранимые команды, *хранимые процедуры* и ряд других. Теперь перейдем к определению объектов БД.

**Основные объекты реляционной базы данных**

*Кластеры*, *каталоги* > и *схемы* не являются обязательными элементами стандарта и, следовательно, программной среды БД.

Под ***кластером*** **понимается группа** *каталогов*, **к которым можно обращаться через одно соединение с сервером базы данных** (программная компонента СУБД).

Обычно процедура создания *каталога* определяется реализацией СУБД на конкретной операционной платформе. Под ***каталогом*** **понимается группа схем**. На практике *каталог* часто ассоциируется с физической базой данных как набором физических файлов операционной системы, которые идентифицируются ее именем.

Для проектировщика БД *схема* – это общее логическое *представление* отношений законченной базы данных. С точки зрения SQL, *схема* – это контейнер для *таблиц*, *представлений* и других структурных элементов реляционной базы данных. *Принцип размещения элементов БД в каждой* *схеме* **полностью определяется проектировщиком**. На практике *схема* часто ассоциируется с объектами определенного *пользователя* — владельца физической БД.

Далее объекты БД будут определяться в контексте СУБД MS SQL Server 2005/2008. Такой подход принят потому, что проектирование физической модели БД выполняется для конкретной среды ее реализации.

На MS SQL Sever 2005 *схема* (Schema) представляет собой коллекцию объектов базы данных: *таблиц*, *представлений*, *хранимых процедур*, *триггеров*, – формирующих единое пространство имен. *Схемы* и *пользователи* являются различными объектами физической БД. У каждой *схемы* есть владелец – *пользователь*, или роль (см. ниже).

К числу основных объектов реляционных БД относятся *таблица*, *представление* и *пользователь*.

***Таблица*** (Table) **является базовой структурой реляционной БД**. Она представляет собой единицу хранения данных — отношение. *Таблица* представляет собой двумерный массив данных, в котором колонка определяет значение, а строки содержат данные. *Таблица* идентифицируется в БД своим уникальным именем, которое включает в себя идентификацию *пользователя*. *Таблица* может быть пустой или состоять из набора строк.

***Представление*** (View) — **это поименованная динамически поддерживаемая СУБД выборка из одной или нескольких таблиц базы данных**. Оператор выборки ограничивает видимые *пользователем* данные. Обычно СУБД гарантирует актуальность *представления*: его формирование производится каждый раз, когда *представление* используется. Иногда *представления* называют *виртуальными таблицами*.

***Пользователь*** (User) — **это объект, обладающий возможностью создавать или использовать другие объекты базы данных и запрашивать выполнение функций СУБД, таких как организация сеанса работы, изменение состояние базы данных** и т.д.

Для упрощения идентификации и именования объектов в базе данных поддерживаются такие объекты, как *синоним*, последовательность и определенные *пользователем* типы данных.

***Синоним*** (*Synonym*) — **это альтернативное имя (псевдоним) объекта реляционной базы данных, которое позволяет иметь доступ к данному объекту**. *Синоним* может быть общим и частным. Общий *синоним* позволяет всем *пользователям* базы данных обращаться к соответствующему объекту по его псевдониму. *Синоним* позволяет скрыть от конечных *пользователей* полную квалификацию объекта в базе данных.

***Определенные пользователем типы данных*** (*User-defined data types*) **представляют собой определенные пользователем типы атрибутов (домены), которые отличаются от поддерживаемых (встроенных) СУБД типов**. Они определяются на основе встроенных типов.

**Правила** (Rules) – **это декларативные выражения, ограничивающие возможные значения данных**. Для формулировки правила используются допустимые предикатные выражения SQL.

Для обеспечения эффективного доступа к данным в реляционных СУБД поддерживается ряд других объектов: *индекс*, табличная область, *кластер*, *секция*.

***Индекс*** (Index) — **это объект базы данных, создаваемый для повышения производительности выборки данных и контроля уникальности первичного ключа (если он задан для таблицы)**.

***Секция*** (Partition) — **это объект базы данных, который позволяет представить объект с данными в виде совокупности подобъектов, отнесенных к различным табличным пространствам**. Таким образом, секционирование позволяет распределять очень большие *таблицы* на нескольких жестких дисках.

Для обработки данных специальным образом или для реализации *поддержки ссылочной целостности* базы данных используются объекты: *хранимая процедура*, функция, команда, *триггер*. С помощью этих объектов базы данных можно выполнять так называемую *построчную обработку* (record processing) данных. С точки зрения приложений баз данных построчная обработка — это последовательная выборка данных по одной строке, ее обработка и переход к обработке следующей строки.

Данные объекты реляционной базы данных представляют собой программы, т.е. исполняемый код. Этого код обычно называют *серверным кодом* (server-side code), поскольку он выполняется компьютером, на котором установлено ядро реляционной СУБД. Планирование и разработка такого кода является одной из задач проектировщика реляционной базы данных.

***Хранимая процедура*** (Stored procedure) — **это объект базы данных, представляющий поименованный набор команд SQL и/или операторов специализированных языков обработки программирования базы данных**.

**Функция** (Function) — **это объект базы данных, представляющий поименованный набор команд SQL и/или операторов специализированных языков обработки программирования базы данных, который при выполнении возвращает значение — результат вычислений**.

***Триггер*** (Trigger) — **это объект базы данных, который представляет собой специальную** *хранимую процедуру*. Эта процедура запускается автоматически, когда происходит связанное с *триггером* событие (например, вставка строки в *таблицу* ).

Для эффективного управления разграничением доступа к данным поддерживается объект **роль**.

**Роль** (Role) – **объект базы данных, представляющий собой поименованную совокупность привилегий, которые могут назначаться** *пользователям*, **категориям** *пользователей* **или другим ролям**.

**Домены в физической модели данных**

В логической модели данных среда реализации не учитывается. В ней определяются атрибуты и их возможные значения, такие как строка, число или дата, в идеале атрибуту может назначаться **домен**. Домен — это просто тип атрибута, например, "Деньги" или "Рабочий день". Проектировщик может включить ряд проверок допустимости или правил обработки, например, требование, что значение должно быть положительным, ненулевым и иметь максимум два десятичных разряда (это полезно для вычисления сумм рублевых платежей, выставляемых банком на другой банк).

Использование доменов упрощает задачу обеспечения непротиворечивости на стадии логической модели данных. При переходе к проектированию *физической модели данных* проектировщику необходимо знать возможности выбранной СУБД по назначению типов данных колонок. В логической модели данных значения, которые может принимать атрибут отношения, также задаются доменом, который наследуется из информационной модели. В физической модели базы данных требуется, чтобы каждый атрибут отношения в базе данных обладал рядом свойств, которые диктуют, что в нем может храниться и что не может. Этими свойствами являются тип, размер и *ограничения*, которые могут еще более ограничивать допустимый набор значений столбца. Задача состоит в преобразовании домена в подходящий тип данных, поддерживаемый СУБД. Таким образом, проектировщик должен знать, какими типами данных он располагает при решении вышеуказанной задачи.

В контексте проектирования физической модели реляционной БД **домен – это выражение, определяющее разрешенные значения для колонок (атрибутов) отношения**. При описании *таблицы* реляционной БД каждой колонке назначается определенный тип данных. Практически, основу определения домена составляет тип данных, содержащихся в колонке, поскольку большинство встроенных типов задают разрешенный интервал значений данных.

**Пример 11.1**. Колонку в базе данных можно описать следующим образом:

amount NUMBER (8,2) NOT NULL CONSTRAINT cc\_limit\_amnt CHECK

(amount > 0)

В колонке "Сумма платежа" ( Amount ) можно размещать только числовые данные; точность этого значения — два значащих десятичных разряда ( NUMBER (8,2) ); она должна быть заполнена для каждой строки *таблицы* ( NOT NULL ); ее значение должно быть положительным CONSTRAINT cc\_limit\_amnt CHECK (amount > 0). Максимальное значение, которое может храниться в этом столбце, — 999999.99. В этом простом определении колонки мы фактически определили ряд неявных правил, проверку которых MS SQL Server принудительно включает при вводе данных в БД.

Как видно, дальнейшее определение домена колонки (после присвоения ей типа) выполняется проектировщиком с помощью уточнений правил изменения значений. Такие уточнения поддерживаются в SQL с помощью механизма *ограничений* в *спецификации колонки* в *таблице*.

В *стандарт SQL-92* введено понятие доменов, определенных *пользователем*. Определение таких доменов базируется на встроенных типах данных СУБД.

**Допустимые типы данных**

**Тип данных — это спецификация, определяющая, какого рода данные могут храниться в объекте БД**: целые числа, символы, данные *денежного типа*, метки времени и даты, двоичные строки и так далее.

Все допустимые типы данных описаны в *стандарте SQL-92*, но в большинстве диалектов поддерживается расширенный список типов данных. Однако любой диалект SQL поддерживают три общих типа данных: строковые, числовые и тип для *представления* даты и времени. Задание типа данных определяет значения и длину данных, а также формат их *представления* при визуализации.

Для всех типов данных определено так называемое **нуль-значение**, которое указывает на отсутствие данных в колонке указанного типа, т.е. то обстоятельство, что значение данных в текущий момент времени неизвестно.

Описание типов, данное в *таблице* ниже, относится к диалекту SQL для СУБД семейства MS SQL Server, которое имеет существенные отличия от предписаний стандарта SQL. Жирным шрифтом выделена часть зарезервированного слова для определения типа, которую можно использовать как аббревиатуру при определении типа в *спецификации колонки*.

MS SQL Server предоставляет набор системных типов данных, определяющих все типы данных, которые могут использоваться в нем. Можно также определять собственные типы данных в Transact-SQL или Microsoft .NET Framework. Псевдонимы типов данных основываются на системных типах. Пользовательские типы данных обладают свойствами, зависящими от методов и операторов класса, который создается для них на одном из языков программирования, поддерживающих .NET Framework.

При объединении одним оператором двух выражений с разными типами данных, параметрами сортировки, точностями, масштабами или длинами результат определяется следующим образом.

* Тип данных результата определяется применением правил очередности типов данных к входным выражениям.
* Параметры сортировки результата определяются правилами очередности параметров сортировки, если тип данных результата относится к char, varchar, text, nchar, nvarchar или ntext.
* Точность, масштаб и длина результата зависят от точности, масштаба и длины входных выражений.

Типы данных в СУБД семейства MS SQL Server объединены в следующие категории:

* точные числа;
* приблизительные числа;
* дата и время;
* символьные строки;
* символьные строки в Юникоде;
* двоичные данные;
* прочие типы данных.

|  |
| --- |
| Таблица 11.1. Допустимые типы данных в языке SQL |
| **Тип данных** | **Синтаксис** |
| **Точные числа** |
| bigint | Целые значения в диапазоне от -2^{63} до 2^{63} - 1 |
| int | Целые значения в диапазоне от -2^{31} до 2^{31}-1 |
| smallint | Целые значения в диапазоне от -2^{15} до 2^{15)-1 |
| tinyint | Целые значения в диапазоне от 0 до 255 |
| bit | Целое число, равное 1, 0 или NULL |
| decimal[ (p[ , s] )] и numeric[ (p[ , s] )] | Числа с фиксированной точностью и масштабом. При использовании максимальной точности числа могут принимать значения в диапазоне от -10^{38}+1 до 10^{38}-1.p (точность) - максимальное количество десятичных разрядов числа (как слева, так и справа от десятичной запятой). Точность должна принимать значение от 1 до 38. По умолчанию для точности принимается значение 18.s (масштаб) - максимальное количество десятичных разрядов числа справа от десятичной запятой. Масштаб может принимать значение от 0 до p. Масштаб может быть указан только совместно с точностью. По умолчанию масштаб принимает значение 0, поэтому 0 <= s <= p. Максимальный размер хранилища зависит от точности. |
| money | Денежные (валютные) значения в диапазоне от -922 337 203 685 477,5808 до 922 337 203 685 477,5807 |
| smallmoney | Денежные (валютные) значения в диапазоне от -214 748,3648 до 214 748,3647 |
| **Приблизительные числа** |
| floatfloat [ ( n ) ] | Числовые значения с плавающей запятой в диапазонах: -1,79E^{308} - -2,23E^{-308} и 2,23E^{-308} - 1,79E^{308}.n - это количество битов, используемых для хранения мантиссы числа в формате float при экспоненциальном *представлении*, которое определяет точность данных и размер для хранения. Значение параметра n должно лежать в пределах от 1 до 53. Значением по умолчанию для параметра n является 53 |
| real | Числовые значения с плавающей запятой в диапазонах: -3,40E^{38} - -1,18E^{-38} и 1,18E^{-38} - 3,40E^{38} |
| **Дата и время** |
| date | Дата в формате ГГГГ-ММ-ДД. Диапазон значений от 0001-01-01 до 9999-12-31, от 1 января 1 года до 31 декабря 9999 года |
| datetime | Определяет дату, включающую время дня с долями секунды в 24-часовом формате, 1 января 1753 года - 31 декабря 9999 года, от 00:00:00 до 23:59:590,997 |
| smalldatetime | Определяет дату, сочетающуюся со временем дня. Время представлено в 24-часовом формате с секундами, всегда равными нулю (:00), без долей секунд, от 01.01.1900 до 06.06.2079, 1 января 1900 года - 6 июня 2079 года, от 00:00:00 до 23:59:59 |
| time | Определяет время дня. Время без учета часового пояса в 24-часовом формате, от 00:00:00.0000000 до 23:59:59.9999999 |
| datetime2 | Определяет дату, объединенную со временем дня в 24-часовом формате. От 0001-01-01 до 9999-12-31, с 1 января 1 года нашей эры до 31 декабря 9999 года нашей эры, от 00:00:00 до 23:59:59.9999999 |
| datetimeoffset | Определяет дату, объединенную со временем дня, с учетом часового пояса в 24-часовом формате, от 0001-01-01 до 9999-12-31, с 1 января 1 года нашей эры до 31 декабря 9999 года нашей эры, от 00:00:00 до 23:59:59.9999999 |
| **Символьные строки** |
| varchar [( n| max )] | Символьные данные переменной длины, не в Юникоде. n может иметь значение от 1 до 8 000, max означает, что максимальный размер хранения равен 2^{31}-1 байт. Размер хранения равен фактической длине данных плюс два байта. Введенные данные могут иметь длину 0 символов. *Синонимами* по стандарту ISO для типа varchar являются типы char *varying* или *character varying* |
| char [ ( n ) ] | Символьные данные фиксированной длины, не в Юникоде, с длиной n байт. Значение n должно находиться в интервале от 1 до 8000. Размер хранения данных этого типа равен n байт. *Синонимом* по стандарту ISO для типа char является character |
| text | Этот тип данных представляет данные, отличные от данных Юникод, с использованием кодовой страницы сервера. Максимальная длина данных - 2^{31} - 1 (2 147 483 647) символов. Если в кодовой странице сервера используются двухбайтовые символы, объем занимаемого типом пространства все равно не превышает 2 147 483 647 байт. Он может быть менее 2 147 483 647 байт - в зависимости от строки символов |
| **Символьные строки в Юникоде** |
| nchar [ ( n ) ] | Символьные данные в Юникоде длиной в n символов. Аргумент n должен иметь значение от 1 до 4000. Размер хранилища вдвое больше n байт. *Синонимами* по стандарту ISO для типа nchar являются типы national char и *national character* |
| nvarchar [(n|max )] | Символьные данные в Юникоде переменной длины. Аргумент n может принимать значение от 1 до 4 000. Аргумент max указывает, что максимальный размер хранилища равен 2^{31}-1 байт. Размер хранилища в байтах вдвое больше числа введенных символов + 2 байта. Введенные данные могут иметь длину в 0 символов. *Синонимами* по стандарту ISO для типа nvarchar являются типы national char *varying* и *national character* *varying* |
| ntext | Этот тип данных представляет символьные данные в Юникоде переменной длины, включающие до 2^{30} - 1 (1 073 741 823) символов. Объем занимаемого этим типом пространства (в байтах) в два раза превышает число символов. В спецификации SQL-2003 *синонимом* типа данных ntext является тип national text |
| **Двоичные данные** |
| binary [ ( n ) ] | Двоичные данные фиксированной длины размером в n байт, где n - значение от 1 до 8000. Размер хранения составляет n байт |
| varbinary [(n|max)] | Двоичные данные переменной длины. n могут иметь значение от 1 до 8000; max означает максимальную длину хранения, которая составляет 2^{31}-1 байт. Размер хранения - это фактическая длина введенных данных плюс 2 байта. Введенные данные могут иметь размер 0 символов. В SQL-2003 *синонимом* для varbinary является binary *varying* |
| image | Этот тип представляет двоичные данные переменной длины, включающие от 0 до 2^{31} - 1 (2 147 483 647) байт |
| **Прочие типы данных** |
| timestamp | Это тип данных, который представляет собой автоматически сформированные уникальные двоичные числа в базе данных. Тип данных timestamp используется в основном в качестве механизма для отметки версий строк *таблицы*. Размер при хранении - 8 байт. Тип данных timestamp - всего лишь увеличивающееся значение, которое не сохраняет дату или время. Тип данных datetime используется для записи даты или времени |
| xml ([CONTENT| DOCUMENT] xml\_schema\_collection ) | Тип данных, в котором хранятся XML-данные. Можно хранить экземпляры xml в столбце либо в переменной типа xml.CONTENT. Экземпляр xml должен быть корректным XML-фрагментом. XML-данные могут содержать несколько (0 или больше) элементов верхнего уровня. Текстовые узлы разрешены на верхнем уровне. Это поведение установлено по умолчанию.DOCUMENT. Экземпляр xml должен быть корректным XML-документом. XML-данные должны содержать только один корневой элемент. Текстовые узлы на верхнем уровне запрещены.xml\_schema\_collection. Имя коллекции XML- *схем*. Чтобы создать типизированный столбец или переменную xml, можно дополнительно указать имя коллекции XML- *схем*. Дополнительные сведения о типизированном и нетипизированном XML |
| sql\_variant | Столбец типа sql\_variant может содержать строки различных типов данных. Например, столбец, определенный как sql\_variant, может хранить значения int, binary и char |
| table | Особый тип данных, который можно использовать для хранения результирующего набора с целью последующей его обработки. Тип table применяется, главным образом, для временного хранения набора строк, возвращаемого в качестве результирующего набора возвращающей табличное значение функции |
| cursor | Тип данных для переменных или выходных параметров *хранимых процедур*, которые содержат ссылку на курсор. Любая переменная, созданная с типом данных cursor, может принимать значение NULL |

Итак, мы рассмотрели основные объекты БД и допустимые типы данных, которыми располагает проектировщик ХД при использовании СУБД семейства MS SQL Server.

Далее поговорим об алгоритме создания физической модели ХД.

### Моделирование объектов физической модели хранилища данных

#### Описание учебного примера

Рассмотрим в качестве примера логическую модель ХД типа "звезда" на [рис. 11.2](https://intuit.ru/studies/courses/599/455/lecture/10170?page=3#image.11.2) и построим на ее основе физическую модель ХД.



**Рис. 11.2.**Логическая модель хранилища данных

Логическая модель ХД, приведенная на рисунке, была разработана для анализа продаж компании в разрезах товаров, продавцов, покупателей, времени продаж. Она включает в себя четыре сущности для измерений: "Время" (Time), "Покупатель" (Customer), "Товар" (Product), "Продавец" (Employee) – и одну сущность для фактов "Продажи" (Sale).

Как видно из приведенной *схемы*, для атрибутов определены домены "Целое число", "Десятичное число", "Текст" размером в 20 и 40 символов.

Описание атрибутов модели приведено в [табл. 11.2](https://intuit.ru/studies/courses/599/455/lecture/10170?page=2#table.11.1).

|  |
| --- |
| Таблица 11.2. Описание атрибутов модели хранилища данных |
| **Атрибут** | **Значение** | **Сущность** |
| Time\_ID | Идентификатор времени, ключ сущности | "Время" (Time) |
| Year | Год | "Время" (Time) |
| Quarter | Квартал года | "Время" (Time) |
| Cust\_ID | Идентификатор покупателя, ключ сущности | "Покупатель" (Customer) |
| FName | Имя покупателя | "Покупатель" (Customer) |
| LName | Фамилия покупателя | "Покупатель" (Customer) |
| Address | Адрес покупателя | "Покупатель" (Customer) |
| Company | Место работы | "Покупатель" (Customer) |
| Prod\_ID | Идентификатор товара, ключ сущности | "Товар" (Product) |
| Name | Наименование товара | "Товар" (Product) |
| Size | Габариты товара | "Товар" (Product) |
| Unit\_Price | Цена за единицу товара | "Товар" (Product) |
| Empl\_ID | Идентификатор продавца, ключ сущности | "Продавец" (Employee) |
| Empl\_FName | Имя продавца | "Продавец" (Employee) |
| Empl\_LName | Фамилия продавца | "Продавец" (Employee) |
| City | Населенный пункт | "Продавец" (Employee) |
| Address | Адрес месторасположения | "Продавец" (Employee) |
| Sale\_ID | Идентификатор продаж, ключ сущности | "Продажи" (Sale) |
| Amount | Сумма платежа | "Продажи" (Sale) |
| Quantity | Количество | "Продажи" (Sale) |

После того как были рассмотрены документы, описывающие логическую модель ХД, задача проектировщика состоит в построении физической модели ХД, которая включает в себя следующие действия.

* Разработка физической модели ХД:
	+ определение *базовых таблиц* БД;
	+ определение колонок в *таблицах* ;
	+ определение типов данных для колонок;
	+ назначение первичных ключей *таблицам* ;
	+ задание *ограничений* NOT NULL на значения колонок;
	+ создание связей между *таблицами*.
* Разработка скрипта создания ХД:
	+ формирование *команд CREATE TABLE* для *таблиц* ХД;
	+ определение *ограничений* на колонки *таблиц* ХД;
	+ формирование дополнительных *индексов* *командой CREATE INDEX*.

#### Моделирование таблиц хранилища данных

##### Определение базовых таблиц

Как указывалось выше, первый шаг в построении физической модели ХД есть идентификация *таблиц* БД. При решении этой задачи проектировщик ХД имеет на входе отношения логической модели данных ХД, представляющие сущности предметной области, а на выходе этапа моделирования должен иметь определения *таблиц*, их колонок, ключей и *индексов* и т.д.

*Базовые таблицы* создаются для каждого отношения логической модели и являются главными объектами хранения данных в БД. Для каждой *базовой таблицы* определяется длинный идентификатор, который уникально идентифицирует *таблицу* в БД. Это имя должно соответствовать стандартам наименований сущностей предметной области, если такие стандарты были разработаны *администратором данных* на стадии анализа предметной области.

Для нашего учебного примера создадим пять *таблиц*: четыре *таблицы* измерений "Время" (Time), "Покупатель" (Customer), "Товар" (Product), "Продавец" (Employee) и одну *таблицу* фактов "Продажи" (Sale). Имена *таблиц* будут соответствовать именам сущностей логической модели ХД ([рис. 11.3](https://intuit.ru/studies/courses/599/455/lecture/10170?page=3#image.11.3)).



**Рис. 11.3.**Создание таблиц физической модели хранилища данных

Когда проектировщик заканчивает обработку всех отношений логической модели данных, он должен еще раз проверить, соответствует ли число *базовых таблиц* числу отношений логической модели данных (т.е. не меньше, чем число сущностей предметной области). Таким образом, при создании *базовых таблиц* проектировщик придерживается принципа: каждому отношению логической модели данных — по *базовой таблице*.

##### Определение колонок в таблицах

Следующим шагом проектировщика ХД является определение колонок для *базовых таблиц*. **Колонки** *таблицы* должны представлять атрибуты отношений логической модели реляционной ХД. Эти атрибуты необходимо преобразовать в *спецификации колонок* в *команде CREATE TABLE*. *Спецификация колонки* *таблицы* имеет следующий синтаксис: имя колонки, тип данных для значений, сохраняемых в колонке, список *ограничений*.

Сначала рассмотрим задачу добавления колонок. Колонка должна иметь имя. Имена атрибутов соответствующих отношений логической модели преобразуются в имена колонок в соответствии с правилами именования объектов, принятых в конкретной СУБД. Обычно, как указывалось выше, это *ограничение* на длину имени и применение в имени специальных символов. Например, в некоторых СУБД допускается использовать знак доллара в имени, однако этот знак обычно не распознается в командах выборки данных – SELECT.



**Рис. 11.4.**Определение имен колонок таблиц физической модели хранилища данных

Имеется еще одна проблема в именовании колонок – имена колонок должны интерпретироваться *пользователем* однозначно. Например, если проектировщик назначит для фамилии сотрудника короткое имя LN, то, наверное, потребуется комментарий, в котором необходимо указать, что это фамилия, а не линия (например, в смысле "линия производства"). Если невозможно использовать по каким-то причинам длинные имена полей, то следует использовать словарь данных для интерпретации введенных аббревиатур.

Для нашего учебного примера решение этой задачи сводится к перенесению атрибутов сущностей в поля соответствующих *таблиц* БД. Заметим, что атрибуты, претендующие в кандидаты на внешние ключи, не переносятся. На [рис. 11.4](https://intuit.ru/studies/courses/599/455/lecture/10170?page=3#image.11.4) показан результат проделанной работы.

Заметим, что для соблюдения принципа уникальности имен полей в рамках модели при определении полей на основе имен атрибутов сущностей мы изменили некоторые имена. Так, имя атрибута "Адрес" (Address) сущности "Продавцы" (Employee) в соответствующей *таблице* "Продавцы" (Employee) физической модели стало "Адрес продавца" (Empl\_Address). Аналогично имя атрибута "Адрес" (Address) сущности "Покупатели" (Customer) в соответствующей *таблицы* "Покупатели" (Customer) физической модели стало "Адрес покупателя" (Cust\_Address).

##### Определение типов данных для колонок

После идентификации колонок необходимо задать их тип в соответствии с допустимыми для данной СУБД типами данных. Эта задача упрощается, если в отношениях логической модели определены домены атрибутов. Некоторые из доменов могут быть определены уже в терминах СУБД. Для таких атрибутов практически ничего делать не нужно. Определение домена в терминах типа данных СУБД нужно просто перенести в *спецификацию колонки*. Возможно, проектировщику будет нужно уточнить второстепенные параметры типа. Например, если задан домен как DEC(9,2), а из контекста предметной области следует, что в этой колонке будет накапливаться итоговая сумма расходов за год, то, может быть, целесообразно определить тип как DEC(15,2), чтобы избежать возможного переполнения при работе приложений базы данных.

Если домен определен не в терминах СУБД, проектировщик базы данных должен преобразовать его в подходящий тип данных. При выполнении таких преобразований следует учитывать ряд факторов.

* Следует уточнить, как СУБД физически хранит данные того или иного предопределенного типа, и затем уточнить интервалы изменения значений колонок. Например, если тип переменной — varchar(3), и она содержит код, значение которого изменяется в интервале от '10A' до '99Z', то целесообразно с точки зрения хранения изменить тип этой переменной на char(3). Это объясняется тем, что тип varchar при физическом хранении занимает на байт-два больше, чем тип char при одной и той же объявленной длине.
* Для числовых значений фиксированной длины предпочтительнее использовать тип DEC. Он обрабатывается процессором быстрее, чем тип FLOAT. Исключение составляют данные для научных расчетов, где *представление* чисел в экспоненциальной форме бывает необходимо.
* Используйте INT и SMALLINT исключительно для счетчиков.
* Избегайте использовать тип CHAR для *представления* числовых данных. Во-первых, может потребоваться дополнительная проверка, а во-вторых, могут возникнуть проблемы при сортировке таких колонок, поскольку число, заданное строкой '11' будет находиться выше, чем число, заданной строкой '9', при упорядочивании по возрастанию.
* Используйте типы DATE и TIME только для хранения хронологических данных.
* Используйте тип DATETIME исключительно для целей управления данными.



**Рис. 11.5.**Определение типов данных для колонок таблиц физической модели хранилища данных

В нашем учебном примере для всех атрибутов задан домен. Для суррогатных первичных *идентификаторов сущностей* это числовое значение. Учитывая, что ХД будет хранить большие объемы данных, для *представления* таких атрибутов в БД целесообразно выбрать тип данных bigint, а для остальных атрибутов — тип данных integer. Домен "Текст" можно представить типом данных varchar(n), для *представления* десятичных чисел использовать тип данных numeric (p, s ).

Результат определения типов колонок *таблиц* физической модели ХД показан на [рис. 11.5](https://intuit.ru/studies/courses/599/455/lecture/10170?page=3#image.11.5).

**Назначение первичных ключей таблицам**

После определения всех колонок и их типов следует перейти к идентификации **первичных ключей таблицы**. Согласно требованиям реляционной теории каждая строка *таблицы* (кортеж) должна иметь уникальный **первичный ключ**. Обычно хорошим кандидатом на **первичный ключ** *таблицы* является *первичный ключ отношения* логической модели. Поскольку предполагается, что в отношении логической модели задан первичный ключ, обладающий свойствам минимальности, его просто нужно определить в *команде CREATE TABLE*. Такое определение первичного ключа *таблицы* для многих *таблиц* не является окончательным. Переопределение первичного ключа может происходить на следующих этапах физического проектирования базы данных.

Задание колонки как первичного ключа в контексте многих СУБД, в том числе и семейства MS SQL Server, считается *ограничением* на значение колонки (см. далее настоящую лекцию).

Для нашего примера мы уже определили первичные ключи *таблиц* физической модели ХД как первичные ключи соответствующих сущностей.

После выполнения вышеперечисленных действий задачу определения первичных ключей *базовых таблиц* в первом приближении можно считать законченной и перейти к решению следующей очень важной задачи определения *таблиц* – определению *ограничений* на значения колонок.

**Задание ограничений NOT NULL на значения колонок**

При определении *спецификаций колонок* *таблиц* проектировщик должен рассмотреть *ограничения*, которые могут быть наложены на значения колонок. В реляционных СУБД предусмотрено достаточно много подобных *ограничений*. Здесь мы остановимся на одном из главных – на обязательности присутствия значения в колонке. Такое *ограничение* на значения колонки называется **NOT NULL-ограничением**.

Предопределенное значение колонки, равное NULL, означает, что в данный конкретный момент для данной конкретной строки (экземпляра сущности предметной области) значение не определено, или неизвестно, или отсутствует. Проектировщику базы данных необходимо идентифицировать возможность колонки принимать NULL-значения, т.к. у *пользователей* могут возникать проблемы при использовании таких колонок.

Примером проблемы может служить ситуация, в которой *пользователю* требуется выполнить соединение двух *таблиц* по колонкам, имеющим NULL-значения. При выполнении соединений любые строки, которые содержат NULL-значения в колонках соединения в любой из *таблиц*, не будут показаны в результирующей выборке для запроса. Потеря данных может привести к тому, что *пользователь* получит неправильную выборку на запрос, особенно если ему необходимо видеть все строки хотя бы одной из *таблиц*.

При назначении NULL-значений колонкам проектировщику необходимо принимать во внимание следующие факторы.

* Колонки, являющиеся частью составного первичного ключа, должны иметь всегда *ограничение* NOT NULL, т.к. согласно реляционной теории значения колонок первичного ключа должны быть определены и уникальны для каждого кортежа.
* Внешние ключи должны также определяться как NOT NULL, и поскольку *дочерняя таблица* зависит от *родительской*, внешний ключ *родительской* не может иметь NULL-значения. Это следует из того, что существование строки *дочерней таблицы* без соответствующей строки *родительской таблицы* нарушает правило зависимости связи (о внешних ключах, *родительских* и *дочерних таблицах* см. далее).
* Только внешние ключи для *таблицы* с опциональной связью могут рассматриваться как кандидаты на наличие NULL-значений, чтобы показать, что для данной комбинации родительской и дочерних строк в этих *таблицах* связи нет.
* Внешние ключи с правилом удаления *SET NULL* должны определяться со спецификацией NULL.
* Используйте спецификацию NOT NULL WITH DEFAULT для колонок с типами данных DATE или TIME, чтобы сохранять текущие даты и текущее время автоматически.
* Разрешайте использовать NULL-значения только для тех колонок, которые действительно могут иметь неопределенные значения.
* Используйте NOT NULL WITH DEFAULT для всех колонок, которые не подпадают под перечисленные выше правила.

Для нашего учебного примера все колонки *таблиц* физической модели ХД, может быть, за исключением адресов, должны иметь *ограничение* NOT NULL ([рис. 11.6](https://intuit.ru/studies/courses/599/455/lecture/10170?page=4#image.11.6)).



**Рис. 11.6.**Задание ограничений NOT NULL на значения колонок

**Создание связей между таблицами**

Следующим шагом в моделировании физической модели ХД является установление взаимосвязи между *таблицами* модели ХД.

*Таблицы* измерения и *таблица* фактов в *многомерной модели* данных находятся в отношении "родитель-потомок". Первичный и соответствующий ему внешний ключ позволяют реализовать отношение "родитель-потомок" (parent/child relationship) между *таблицами* реляционной БД. Они отражают взаимосвязь между объектами предметной области (представленными кортежами *таблиц* ) через значения некоторых их атрибутов по принципу иерархического подчинения, когда объект-родитель определяет существование объектов-потомков. Сами объекты-потомки могут также выступать в качестве родителей для других объектов (descendents).

**Таблица реляционной базы данных, содержащая первичный ключ, называется** ***таблицей-родителем (parent table), или родительской таблицей***, **а таблица, содержащая соответствующий первичному ключу внешний ключ, —** ***таблицей-потомком (child table), или дочерней таблицей***. Таблица измерений "Товары" (Product) учебного примера является *таблицей-родителем* для таблицы фактов "Продажи" (Sale).

Отношение "родитель-потомок" между *таблицами* реализуется через атрибуты-ключи соответствующих строк. Строка, принадлежащая *таблице-родителю*, называется **родительской строкой**, а строка в *таблице-потомке*, на которую ссылается родительская строка, называется **строкой-потомком**, или дочерней строкой. Строка-потомок должна иметь по крайней мере один ненулевой атрибут внешнего ключа.

Отношения "родитель-потомок" между двумя *таблицами* отражают взаимосвязь по включению на доменах соответствующих атрибутов.

Установив связь "родитель-потомок" между *таблицами* измерений и *таблицей* фактов нашего учебного примера, получим физическую модель ХД ([рис. 11.7](https://intuit.ru/studies/courses/599/455/lecture/10170?page=4#image.11.7))



**Рис. 11.7.**Задание ограничений NOT NULL на значения колонок

Вопрос, который необходимо решить при моделировании ХД, состоит в том, будут ли внешние ключи измерений элементами составного первичного ключа *таблицы* фактов.

В нашем примере *таблица* фактов имеет уникальный суррогатный ключ "Идентификатор продажи" ( Sale\_ID ), поэтому внешние ключи, реализующие отношение "родитель-потомок" между *таблицами* измерений и *таблицей* фактов, не обязательно включать в составной первичный ключ *таблицы* фактов. Любая строка *таблицы* фактов однозначно идентифицируется уже назначенным первичным ключом.

После создания физической модели ХД проектировщик ХД может перейти к решению задачи разработки скрипта для создания ХД. Эта задача может быть решена и вручную, как мы сделаем это в следующем разделе настоящей лекции, и при помощи CASE-средств.

**Разработка скрипта для создания объектов физической модели хранилища данных**

**Команды SQL для создания таблиц базы данных**

Для определения и создания *таблиц* в SQL предусмотрена *команда CREATE TABLE*, которая определяет имя *таблицы*, имена и физический порядок колонок для нее, тип каждой колонки, а также некоторые указания для СУБД, такие как определение первичного или внешнего ключа, требования на запрет неопределенных значений в колонке *таблицы* и т.п.

*Команда CREATE TABLE* создает новую *таблицу* в БД. Синтаксис этой команды для диалекта SQL (Transact-SQL) СУБД семейства MS SQL Server приведен ниже.

CREATE TABLE

 [ database\_name . [ schema\_name ] . | schema\_name . ] table\_name

 ( { <column\_definition> | <computed\_column\_definition>

 | <column\_set\_definition> }

 [ <table\_constraint> ] [ ,...n ] )

 [ ON { partition\_scheme\_name ( partition\_column\_name ) | filegroup

 | "default" } ]

 [ { TEXTIMAGE\_ON { filegroup | "default" } ]

 [ FILESTREAM\_ON { partition\_scheme\_name | filegroup

 | "default" } ]

 [ WITH ( <table\_option> [ ,...n ] ) ]

[ ; ]

<column\_definition> ::=

column\_name <data\_type>

 [ FILESTREAM ]

 [ COLLATE collation\_name ]

 [ NULL | NOT NULL ]

 [

 [ CONSTRAINT constraint\_name ] DEFAULT constant\_expression ]

 | [ IDENTITY [ ( seed ,increment ) ] [ NOT FOR REPLICATION ]

 ]

 [ ROWGUIDCOL ] [ <column\_constraint> [ ...n ] ]

 [ SPARSE ]

<data type> ::=

[ type\_schema\_name . ] type\_name

 [ ( precision [ , scale ] | max |

 [ { CONTENT | DOCUMENT } ] xml\_schema\_collection ) ]

<column\_constraint> ::=

[ CONSTRAINT constraint\_name ]

{ { PRIMARY KEY | UNIQUE }

 [ CLUSTERED | NONCLUSTERED ]

 [

 WITH FILLFACTOR = fillfactor

 | WITH ( < index\_option > [ , ...n ] )

 ]

 [ ON { partition\_scheme\_name ( partition\_column\_name )

 | filegroup | "default" } ]

 | [ FOREIGN KEY ]

 REFERENCES [ schema\_name . ] referenced\_table\_name [ ( ref\_column ) ]

 [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

 [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

 [ NOT FOR REPLICATION ]

 | CHECK [ NOT FOR REPLICATION ] ( logical\_expression )

}

<computed\_column\_definition> ::=

column\_name AS computed\_column\_expression

[ PERSISTED [ NOT NULL ] ]

[

 [ CONSTRAINT constraint\_name ]

 { PRIMARY KEY | UNIQUE }

 [ CLUSTERED | NONCLUSTERED ]

 [

 WITH FILLFACTOR = fillfactor

 | WITH ( <index\_option> [ , ...n ] )

 ]

 | [ FOREIGN KEY ]

 REFERENCES referenced\_table\_name [ ( ref\_column ) ]

 [ ON DELETE { NO ACTION | CASCADE } ]

 [ ON UPDATE { NO ACTION } ]

 [ NOT FOR REPLICATION ]

 | CHECK [ NOT FOR REPLICATION ] ( logical\_expression )

 [ ON { partition\_scheme\_name ( partition\_column\_name )

 | filegroup | "default" } ]

]

<column\_set\_definition> ::=

column\_set\_name XML COLUMN\_SET FOR ALL\_SPARSE\_COLUMNS

< table\_constraint > ::=

[ CONSTRAINT constraint\_name ]

{

 { PRIMARY KEY | UNIQUE }

 [ CLUSTERED | NONCLUSTERED ]

 (column [ ASC | DESC ] [ ,...n ] )

 [

 WITH FILLFACTOR = fillfactor

 |WITH ( <index\_option> [ , ...n ] )

 ]

 [ ON { partition\_scheme\_name (partition\_column\_name)

 | filegroup | "default" } ]

 | FOREIGN KEY

 ( column [ ,...n ] )

 REFERENCES referenced\_table\_name [ ( ref\_column [ ,...n ] ) ]

 [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

 [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

 [ NOT FOR REPLICATION ]

 | CHECK [ NOT FOR REPLICATION ] ( logical\_expression )

}

<table\_option> ::=

{

 DATA\_COMPRESSION = { NONE | ROW | PAGE }

 [ ON PARTITIONS ( { <partition\_number\_expression> | <range> }

 [ , ...n ] ) ]

}

<index\_option> ::=

{

 PAD\_INDEX = { ON | OFF }

 | FILLFACTOR = fillfactor

 | IGNORE\_DUP\_KEY = { ON | OFF }

 | STATISTICS\_NORECOMPUTE = { ON | OFF }

 | ALLOW\_ROW\_LOCKS = { ON | OFF}

 | ALLOW\_PAGE\_LOCKS ={ ON | OFF}

 | DATA\_COMPRESSION = { NONE | ROW | PAGE }

 [ ON PARTITIONS ( { <partition\_number\_expression> | <range> }

 [ , ...n ] ) ]

}

<range> ::=

<partition\_number\_expression> TO <partition\_number\_expression>

Рассмотрим элементы и аргументы команды CREATE TАBLE.

* database\_name. Имя БД, в которой создается *таблица*. В качестве аргумента database\_name должно быть указано имя существующей БД. Если аргумент database\_name не указан, по умолчанию *таблица* создается в текущей БД. Имя входа для текущего соединения должно быть связано с идентификатором *пользователя*, который существует в БД, указанной аргументом database\_name, и этот *пользователь* должен обладать разрешениями CREATE TABLE.
* schema\_name. Имя *схемы*, которой принадлежит новая *таблица*.
* table\_name. Имя новой *таблицы*. Имена *таблиц* должны соответствовать правилам для идентификаторов. Аргумент table\_name может состоять не более чем из 128 символов, за исключением имен локальных временных *таблиц* (имена с *префиксом номера* #), длина которых не должна превышать 116 символов.
* column\_name. Имя столбца в *таблице*. Имена столбцов должны соответствовать правилам для идентификаторов и быть уникальными в данной *таблице*. Аргумент column\_name может содержать от 1 до 128 символов. При создании столбцов с типом данных timestamp аргумент column\_name может быть пропущен. Если аргумент column\_name не указан, столбцу типа timestamp по умолчанию присваивается имя timestamp.
* computed\_column\_expression. Выражение, определяющее значение вычисляемого столбца. Вычисляемый столбец представляет собой виртуальный столбец, физически не хранящийся в *таблице*, если для него не установлен признак PERSISTED. Значение столбца вычисляется на основе выражения, использующего другие столбцы той же *таблицы*. Например, определение вычисляемого столбца может быть следующим: cost AS price \* qty. Выражение может быть именем невычисляемого столбца, константой, функцией, переменной или любой их комбинацией, соединенной одним или несколькими операторами. Выражение не может быть вложенным запросом или содержать типы данных "псевдонимы".

Вычисляемые столбцы могут использоваться в списках выборки, предложениях WHERE, ORDER BY и в любых других местах, в которых могут применяться обычные выражения, за исключением следующих случаев.

1. Вычисляемый столбец нельзя использовать ни в качестве определения *ограничения* DEFAULT или FOREIGN KEY, ни вместе с определением *ограничения* NOT NULL. Однако вычисляемый столбец может использоваться в качестве ключевого столбца *индекса* либо части какого-либо *ограничения* PRIMARY KEY или UNIQUE, если значение этого вычисляемого столбца определяется детерминистическим выражением и тип данных результата разрешен в столбцах *индекса*.
2. Например, если *таблица* содержит целочисленные столбцы a и b, вычисляемый столбец a+b может быть включен в *индекс*, а вычисляемый столбец a+DATEPART(dd, GETDATE()) — не может, так как его значение может изменяться при последующих вызовах.
3. Вычисляемый столбец не может быть целевым столбцом инструкций INSERT или UPDATE.
	* PERSISTED. Указывает, что SQL Server *Database Engine* будет физически хранить вычисляемые значения в *таблице* и обновлять их при изменении любого столбца, от которого зависит вычисляемый столбец. Указание PERSISTED для вычисляемого столбца позволяет создать *индекс* по вычисляемому столбцу, который будет детерминистическим, но неточным. Любые вычисляемые столбцы, используемые в качестве столбцов секционирования в секционированной *таблице*, необходимо явно пометить признаком PERSISTED. Если указан признак PERSISTED, аргумент computed\_column\_expression должно быть детерминистическим.
	* ON { <partition\_scheme> | filegroup | "default" }. Указывает *схему* секционирования или файловую группу, в которой хранится *таблица*. Если аргумент <partition\_scheme> указан, *таблица* будет разбита на *секции*, сохраняемые в одной или нескольких файловых группах, указанных аргументом <partition\_scheme>. Если указан аргумент filegroup, *таблица* сохраняется в файловой группе с таким именем. Это должна быть существующая файловая группа в базе данных. Если указано значение "default", или параметр ON не определен вообще, *таблица* сохраняется в установленной по умолчанию файловой группе. Механизм хранения *таблицы*, указанный в инструкции CREATE TABLE, изменить в дальнейшем невозможно.

Параметр ON {<partition\_scheme> | filegroup | "default"} может также указываться в *ограничении* PRIMARY KEY или UNIQUE. С помощью этих *ограничений* создаются *индексы*. Если указан аргумент filegroup, *индекс* сохраняется в файловой группе с таким именем. Если указано значение "default" или параметр ON не определен вообще, *индекс* сохраняется в той же файловой группе, что и *таблица*. Если *ограничение* PRIMARY KEY или UNIQUE создает кластеризованный *индекс*, страницы данных *таблицы* сохраняются в той же файловой группе, что и *индекс*. Если *ограничение* создает кластеризованный *индекс* (с помощью параметра CLUSTERED или другим способом), то указанный аргумент <partition\_scheme> отличается от аргументов <partition\_scheme> и filegroup из определения *таблицы*, либо наоборот, принимается во внимание только определение *ограничения*, а все остальное не учитывается.

* TEXTIMAGE\_ON { filegroup | "default" }. Ключевые слова, указывающие, что столбцы типов text, ntext, image, xml, varchar(max), nvarchar(max), varbinary(max), а также пользовательских типов среды CLR хранятся в определенной файловой группе.

Параметр TEXTIMAGE\_ON недопустим, если в *таблице* нет столбцов с большими значениями. Нельзя указывать параметр TEXTIMAGE\_ON одновременно с параметром <partition\_scheme>. Если указано значение "default" или параметр TEXTIMAGE\_ON не определен вообще, столбцы с большими значениями сохраняются в установленной по умолчанию файловой группе. Способ хранения любых данных столбцов с большими значениями, определенный инструкцией CREATE TABLE, изменить в дальнейшем невозможно.

* FILESTREAM\_ON { partition\_scheme\_name | filegroup | "default" }. Задает файловую группу для данных FILESTREAM.

Если *таблица* содержит данные FILESTREAM и является секционированной, необходимо включить предложение FILESTREAM\_ON и указать *схему* секционирования файловых групп файлового потока. В этой *схеме* секционирования должны использоваться те же функции и столбцы секционирования, что и в *схеме* секционирования для *таблицы* ; в противном случае возникает ошибка.

Если *таблица* не секционирована, столбец FILESTREAM не может быть секционирован. Данные FILESTREAM для *таблицы* должны храниться в отдельной файловой группе. Эта файловая группа указывается в предложении FILESTREAM\_ON.

Если *таблица* не является секционированной и предложение FILESTREAM\_ON не указано, используется файловая группа FILESTREAM, для которой задано свойство DEFAULT. При отсутствии файловой группы FILESTREAM возникает ошибка.

Как и в случае с предложениями ON и TEXTIMAGE\_ON, значение, указанное с помощью инструкции CREATE TABLE для предложения FILESTREAM\_ON, не может быть изменено, за исключением следующих ситуаций.

1. Инструкция CREATE INDEX преобразует кучу в кластеризованный *индекс*. В этом случае можно указать другую файловую группу FILESTREAM, *схему* секционирования или значение NULL.
2. Инструкция DROP INDEX преобразует кластеризованный *индекс* в кучу. В этом случае можно указать другую файловую группу FILESTREAM, *схему* секционирования или значение "default".
	* [ type\_schema\_name. ] type\_name. Указывает *тип данных столбца* и *схему*, к которой он принадлежит.

Тип данных может быть одним из следующих.

1. Системный тип данных.
2. Тип данных — псевдонимы на основе системного типа данных SQL Server. Прежде чем псевдонимы типов данных можно будет использовать в определении *таблицы*, их нужно создать с помощью инструкции *CREATE TYPE*. Состояние признака NULL или NOT NULL для типа данных – псевдонима может быть переопределено с помощью инструкции CREATE TABLE. Однако его длину изменить нельзя; длина типа данных – псевдонима не определяется инструкцией CREATE TABLE.
3. Пользовательский тип среды CLR. Прежде чем пользовательские типы среды CLR можно будет использовать в определении *таблицы*, их нужно создать с помощью инструкции *CREATE TYPE*. Для создания столбца с пользовательским типом среды CLR требуется разрешение REFERENCES на этот тип.
	* precision. Точность указанного типа данных.
	* Scale. Масштаб указанного типа данных.
	* Max. Применяется только к типам данных varchar, nvarchar и varbinary для хранения  байт символьных и двоичных данных или 2^30 байт данных в Юникоде.
* CONTENT. Указывает, что каждый экземпляр типа данных xml в столбце column\_name может содержать несколько элементов верхнего уровня. Аргумент CONTENT применим только к данным типа xml и может быть указан только в том случае, если одновременно указан аргумент xml\_schema\_collection. Если этот параметр не указан, CONTENT принимается в качестве поведения по умолчанию.
* DOCUMENT. Указывает, что каждый экземпляр типа данных xml в столбце column\_name может содержать только один элемент верхнего уровня. Аргумент DOCUMENT применим только к данным типа xml и может быть указан только в том случае, если одновременно указан аргумент xml\_schema\_collection.
* xml\_schema\_collection.. Применим только к типу данных xml для коллекции XML- *схем*, связанной с этим типом. До помещения столбца xml *схема* должна быть создана в БД при помощи инструкции CREATE XML SCHEMA COLLECTION.
* DEFAULT. Указывает значение, присваиваемое столбцу в случае отсутствия явно заданного значения при вставке. Определения DEFAULT могут применяться к любым столбцам, кроме имеющих тип timestamp или обладающих свойством IDENTITY. Если значение по умолчанию указывается для столбца определяемого *пользователем* типа, этот тип должен поддерживать неявное преобразование выражения constant\_expression в определяемый *пользователем* тип. Определения DEFAULT удаляются, когда *таблица* удаляется из памяти. В качестве значения по умолчанию могут использоваться только константы (например, символьные строки), скалярные функции (системные, определяемые *пользователем* или функции среды CLR) или значение NULL. Для совместимости с более ранними версиями SQL Server параметру DEFAULT может быть присвоено имя *ограничения*.
* constant\_expression. Константа, значение NULL или системная функция, используемая в качестве значения столбца по умолчанию.
* IDENTITY. Указывает, что новый столбец является столбцом идентификаторов. При добавлении в *таблицу* новой строки компонент *Database Engine* формирует для этого столбца уникальное последовательное значение. Столбцы идентификаторов обычно используются с *ограничением* PRIMARY KEY для поддержания уникальности идентификаторов строк в *таблице*. Свойство IDENTITY может быть назначено столбцам типов tinyint, smallint, int, bigint, decimal(p,0) или numeric(p,0). Для каждой *таблицы* можно создать только один столбец идентификаторов. Ограниченные значения по умолчанию и *ограничения* DEFAULT не могут использоваться в столбце идентификаторов. Необходимо указать как начальное значение, так и приращение, или же не указывать ничего. Если ни одно из значений не указано, то действительно значение по умолчанию — (1,1).
* Seed. Значение, используемое для самой первой строки, загружаемой в *таблицу*.
* Increment. Значение приращения, добавляемое к значению идентификатора предыдущей загруженной строки.
* NOT FOR REPLICATION. В инструкции CREATE TABLE предложение NOT FOR REPLICATION может указываться для свойства IDENTITY, а также *ограничений* FOREIGN KEY и CHECK. Если это предложение указано для свойства IDENTITY, значения в столбцах идентификаторов не приращиваются, когда вставку выполняют агенты репликации. Если *ограничение* сопровождается этим предложением, оно не выполняется, когда агенты репликации выполняют операции вставки, обновления или удаления.
* ROWGUIDCOL. Указывает, что новый столбец является столбцом идентификаторов GUID. В качестве столбца ROWGUIDCOL можно назначить только один столбец uniqueidentifier в *таблице*. Применение свойства ROWGUIDCOL позволяет ссылаться на столбец с помощью ключевого слова $ROWGUID. Свойство ROWGUIDCOL можно присвоить только столбцу, имеющему тип uniqueidentifier. Ключевое слово ROWGUIDCOL недопустимо, если уровень совместимости базы данных равен 65 или ниже. Ключевым словом ROWGUIDCOL нельзя обозначать столбцы пользовательских типов данных.

Свойство ROWGUIDCOL не обеспечивает уникальности значений, хранимых в столбце. Кроме того, при указании данного свойства автоматическое формирование значений для новых строк, вставляемых в *таблицу*, не выполняется. Для создания уникальных значений в каждом столбце следует применять в инструкциях INSERT функции NEWID или NEWSEQUENTIALID либо использовать эти функции по умолчанию для столбца.

* *SPARSE*. Указывает, что столбец является разреженным столбцом. Хранилище разреженных столбцов оптимизируется для значений NULL. Для разреженных столбцов нельзя указать параметр NOT NULL.
* FILESTREAM. Допустимо только для столбцов типа varbinary(max). Указывает хранилище FILESTREAM для данных BLOB типа varbinary(max).

*Таблица* также должна содержать столбец типа uniqueidentifier с атрибутом ROWGUIDCOL. Этот столбец не должен допускать значений NULL и должен иметь *ограничение*, относящееся к одному столбцу, UNIQUE или PRIMARY KEY. Значение идентификатора GUID для столбца должно быть предоставлено приложением во время вставки данных или *ограничением* DEFAULT, в котором используется функция NEWID ().

Столбец ROWGUIDCOL нельзя удалить и связанные *ограничения* не могут быть изменены, если в *таблице* определен столбец FILESTREAM. Столбец ROWGUIDCOL можно удалить только после удаления последнего столбца FILESTREAM.

Если для столбца указан атрибут хранилища FILESTREAM, то все значения для этого столбца хранятся в контейнере данных FILESTREAM в файловой системе.

* COLLATE collation\_name. Задает параметры сортировки для столбца. Имя параметров сортировки может быть либо именем параметров сортировки Windows, либо именем параметров сортировки SQL. Аргумент collation\_name применим только к столбцам типов данных char, varchar, text, nchar, nvarchar и ntext. Если этот аргумент не указан, столбцу назначаются либо параметры сортировки пользовательского типа (если столбец принадлежит к пользовательскому типу данных), либо установленные по умолчанию параметры сортировки для базы данных.
* CONSTRAINT. Необязательное ключевое слово, указывающее на начало определения *ограничения* PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY или CHECK.
* constraint\_name. Имя *ограничения*. Имена *ограничений* должны быть уникальными в пределах *схемы*, к которой принадлежит *таблица*.
* NULL | NOT NULL. Определяет, допустимы ли для столбца значения NULL. Параметр NULL не является *ограничением* в строгом смысле слова, но может быть указан так же, как и NOT NULL. *Ограничение* NOT NULL может быть указано для вычисляемых столбцов только в случае если одновременно указан параметр PERSISTED.
* PRIMARY KEY. *Ограничение*, которое обеспечивает сущностную целостность для указанного столбца или столбцов с помощью уникального *индекса*. Можно создать только одно *ограничение* PRIMARY KEY для *таблицы*.
* UNIQUE. *Ограничение*, которое обеспечивает сущностную целостность для указанного столбца или столбцов с помощью уникального *индекса*. *Таблица* может содержать несколько *ограничений* UNIQUE.
* CLUSTERED | NONCLUSTERED. Указывает, что для *ограничения* PRIMARY KEY или UNIQUE создается кластеризованный или некластеризованный *индекс*. Для *ограничений* PRIMARY KEY по умолчанию создается кластеризованный *индекс* ( CLUSTERED ), а для *ограничений* UNIQUE — некластеризованный ( NONCLUSTERED ).

В инструкции CREATE TABLE параметр CLUSTERED можно задать только для одного *ограничения*. Если для *ограничения* UNIQUE указан параметр CLUSTERED и, кроме того, указано *ограничение* PRIMARY KEY, то для PRIMARY KEY применяется по умолчанию значение NONCLUSTERED.

* FOREIGN KEY REFERENCES. *Ограничение*, которое обеспечивает ссылочную целостность данных в этом столбце или столбцах. *Ограничения* FOREIGN KEY требуют, чтобы каждое значение в столбце существовало в соответствующем связанном столбце или столбцах в связанной *таблице*. *Ограничения* FOREIGN KEY могут ссылаться только на столбцы, являющиеся *ограничениями* PRIMARY KEY или UNIQUE в связанной *таблице*, или на столбцы, на которые имеются ссылки в *индексе* UNIQUE INDEX связанной *таблицы*. Внешние ключи в вычисляемых столбцах должны быть также помечены как PERSISTED.
* [ schema\_name . ] referenced\_table\_name ]. Имя *таблицы*, на которую ссылается *ограничение* FOREIGN KEY, и *схема*, к которой она принадлежит.
* ( ref\_column [ ,... n ] ). Столбец или список столбцов из *таблицы*, на которую ссылается *ограничение* FOREIGN KEY.
* ON DELETE { NO ACTION | CASCADE | *SET NULL* | *SET DEFAULT* }. Определяет операцию, которая производится над строками создаваемой *таблицы*, если эти строки имеют ссылочное отношение, а строка, на которую имеются ссылки, удаляется из *родительской таблицы*. Значение по умолчанию — NO ACTION.
	+ NO ACTION. Компонент *Database Engine* инициирует ошибку, и производится откат операции удаления строки *родительской таблицы*.
	+ CASCADE. Если из *родительской таблицы* удаляется строка, соответствующие ей строки удаляются из ссылающейся *таблицы*.
	+ *SET NULL*. Все значения, составляющие внешний ключ, при удалении соответствующей строки *родительской таблицы* устанавливаются в NULL. Для выполнения этого *ограничения* столбцы внешних ключей должны допускать существование значений NULL.
	+ *SET DEFAULT*. Все значения, составляющие внешний ключ, при удалении соответствующей строки *родительской таблицы* устанавливаются в значение по умолчанию. Для выполнения этого *ограничения* все столбцы внешних ключей должны иметь определения по умолчанию. Если столбец допускает значение NULL и множество значений по умолчанию не задано явно, NULL становится неявным значением по умолчанию для данного столбца.
* ON UPDATE { NO ACTION | CASCADE | *SET NULL* | *SET DEFAULT* }. Указывает, какое действие совершается над строками в изменяемой *таблице*, когда эти строки связаны ссылкой и строка *родительской таблицы*, на которую указывает ссылка, обновляется. Значение по умолчанию — NO ACTION.
	+ NO ACTION. Компонент *Database Engine* возвращает ошибку, и обновление строки *родительской таблицы* откатывается.
	+ CASCADE. Соответствующие строки обновляются в ссылающейся *таблице*, если эта строка обновляется в *родительской таблице*.
	+ *SET NULL*. Всем значениям, составляющим внешний ключ, присваивается значение NULL, когда обновляется соответствующая строка в *родительской таблице*. Для выполнения этого *ограничения* столбцы внешних ключей должны допускать существование значений NULL.
	+ *SET DEFAULT*. Всем значениям, составляющим внешний ключ, присваивается их значение по умолчанию, когда обновляется соответствующая строка в *родительской таблице*. Для выполнения этого *ограничения* все столбцы внешних ключей должны иметь определения по умолчанию. Если столбец допускает значение NULL и множество значений по умолчанию не задано явно, NULL становится неявным значением по умолчанию для данного столбца.
* CHECK. *Ограничение*, обеспечивающее доменную целостность путем *ограничения* возможных значений, которые могут быть введены в столбец или столбцы. *Ограничения* CHECK в вычисляемых столбцах должны быть также помечены как PERSISTED.
* logical\_expression. Логическое выражение, возвращающее значения TRUE или FALSE. Типы данных "псевдонимы" частью выражения быть не могут.
* Column. Столбец или список столбцов (в скобках), который применяется в *ограничениях* *таблицы* для указания столбцов, используемых в определении *ограничения*.
* [ ASC | DESC ]. Указывает порядок сортировки столбца или столбцов, участвующих в *ограничениях* *таблицы*: ASC — по возрастанию, DESC — по убыванию. Значение по умолчанию — ASC.
* partition\_scheme\_name. Имя *схемы* секционирования, определяющей файловые группы, которым сопоставляются *секции* секционированной *таблицы*. Эта *схема* секционирования должна существовать в БД.
* [ partition\_column\_name. ]. Указывает столбец, по которому будет секционирована *таблица*. Столбец должен соответствовать по типу данных, длине и точности столбцу, который указан в *функции секционирования*, используемой аргументом partition\_scheme\_name. Вычисляемый столбец, участвующий в *функции секционирования*, должен быть явно обозначен ключевым словом PERSISTED.
* WITH FILLFACTOR = fillfactor. Указывает, насколько плотно компонент *Database Engine* должен заполнять каждую страницу *индекса*, используемую для хранения данных *индекса*. Пользовательские значения аргумента fillfactor могут находиться в диапазоне от 1 до 100. Если значение не задано, по умолчанию оно принимается равным 0. Значения фактора заполнения 0 и 100 во всех отношениях считаются равнозначными.
* column\_set\_name XML COLUMN\_SET FOR ALL\_SPARSE\_COLUMNS. Имя набора столбцов. Набор столбцов представляет собой нетипизированное XML-представление, в котором все разреженные столбцы *таблицы* объединены в структурированные выходные данные.
* < table\_option> ::= Указывает один или более параметров *таблицы*.
* DATA\_COMPRESSION. Задает режим сжатия данных для указанной *таблицы*, номера *секции* или диапазона *секций*. Ниже приведены доступные параметры.
* NONE. *Таблица* или указанные *секции* не сжимаются.
* ROW. *Таблицы* или указанные *секции* сжимаются, используя сжатие строк.
* PAGE. *Таблицы* или указанные *секции* сжимаются, используя сжатие страниц.
* ON PARTITIONS ( { <выражение\_номера\_секции> | <диапазон> } [ ,...n ] ). Указывает *секции*, к которым применяется параметр DATA\_COMPRESSION. Если *таблица* не секционирована, аргумент ON PARTITIONS приведет к формированию ошибки. Если не указано предложение ON PARTITIONS, параметр DATA\_COMPRESSION применяется ко всем *секциям* секционированной *таблицы*.

<Выражение\_номера\_секции> можно указать одним из следующих способов:

1. указав номер *секции*, например: ON PARTITIONS (2) ;
2. указав номера нескольких *секций*, разделив их запятыми, например: ON PARTITIONS (1, 5) ;
3. указав диапазоны *секций* и отдельные *секции*, например: ON PARTITIONS (2, 4, 6 TO 8).

<Диапазон> можно указать номерами *секций*, разделенными ключевым словом TO, например: ON PARTITIONS (6 TO 8).

Чтобы для разных *секций* задать разные типы сжатия данных, укажите параметр DATA\_COMPRESSION несколько раз:

* <index\_option> ::= Указывает один или более параметров *индекса* (см. далее).
* PAD\_INDEX = { ON | OFF }. Если указано значение ON, процент свободного места, определяемый параметром FILLFACTOR, применяется к страницам *индекса* промежуточного уровня. Если указано значение OFF или значение FILLFACTOR не указано, страницы промежуточного уровня заполняются до приблизительного объема, оставляющего достаточно места, как минимум, для одной строки максимального размера, которого может достигать *индекс*, при этом учитывается набор ключей на промежуточных страницах. Значение по умолчанию — OFF.
* FILLFACTOR = fillfactor. Указывает процентное соотношение, определяющее, насколько заполненным компонент *Database Engine* должен делать конечный уровень каждой страницы *индекса* при его создании или изменении. Аргумент fillfactor должен быть целым числом в диапазоне от 1 до 100. Значение по умолчанию равно 0. Значения фактора заполнения 0 и 100 во всех отношениях считаются равнозначными.
* IGNORE\_DUP\_KEY = { ON | OFF }. Определяет ответ на ошибку, случающуюся, когда операция вставки пытается вставить в уникальный *индекс* повторяющиеся значения ключа. Параметр IGNORE\_DUP\_KEY применяется только к операциям вставки, производимым после создания или перестроения *индекса*. Параметр не работает во время выполнения инструкции CREATE INDEX, ALTER INDEX или UPDATE. Значение по умолчанию — OFF.
	+ ON. Если в уникальный *индекс* вставляются повторяющиеся значения ключа, выводится предупреждающее сообщение. С ошибкой завершаются только строки, нарушающие *ограничение* уникальности.
	+ OFF. Если в уникальный *индекс* вставляются повторяющиеся значения ключа, выводится сообщение об ошибке. Будет выполнен откат всей операции INSERT.
	+ IGNORE\_DUP\_KEY. Нельзя установить в значение ON для *индексов*, создаваемых для *представлений*, неуникальных *индексов*, XML-индексов, пространственных *индексов* и фильтруемых *индексов*.
* STATISTICS\_NORECOMPUTE = { ON | OFF }. Если указано значение ON, автоматический пересчет устаревших статистик *индекса* не производится. Если указано значение OFF, включается автоматическое обновление статистик. Значение по умолчанию — OFF.
* ALLOW\_ROW\_LOCKS = { ON | OFF }. Если указано значение ON, при доступе к *индексу* допустимы блокировки строк. Необходимость в блокировке строк определяет компонент *Database Engine*. При значении OFF блокировки строк не используются. Значение по умолчанию — ON.
* ALLOW\_PAGE\_LOCKS = { ON | OFF }. Если указано значение ON, при доступе к *индексу* допустимы блокировки страниц. Необходимость в блокировке строк определяет компонент *Database Engine*. При значении OFF блокировки страниц не используются. Значение по умолчанию — ON.

Выше было дано описание аргументов *команды CREATE TABLE*. В настоящей лекции мы будем обсуждать не все из перечисленных аргументов. Секционирование *таблиц* будет рассмотрено в ["Метаданные в хранилищах данных"](https://intuit.ru/studies/courses/599/455/lecture/10173). Сейчас мы остановимся на использовании *ограничений*.

#### Ограничения и их использование в реляционной базе данных

В предыдущих разделах мы уже сталкивались с несколькими типами *ограничений* в *спецификациях колонок* — NOT NULL, и *ограничениях* в *таблицах* — PRIMARY KEY, FOREING KEY. В данном разделе мы изучим практически еще несколько типов *ограничений*, которые поддерживаются в реляционных БД. *Ограничения* являются важным инструментом проектировщика, с помощью которого он поддерживает целостность (strong) БД. Их можно использовать для того, чтобы быть уверенным в том, что колонка первичного ключа *таблицы* является уникальной и всегда содержит значения. *Ограничения* применяются также для *поддержки ссылочной целостности*. Последнее означает, что значения в колонке внешнего ключа должны существовать как некоторое значение в колонке первичного ключа другой *таблицы*.

*Ограничения* представляют собой способ применения бизнес-правил предметной области на уровне БД и гарантируют совместимость вводимых данных с теми, которые уже находятся в *таблицах*. В реляционной БД **под ограничением понимается правило (условие), которому должен удовлетворять некоторый элемент в БД**. Например, условия, которым должны дополнительно удовлетворять значения колонки *таблицы* в рамках определенного для нее типа данных (т.е. тип данных плюс правило), полностью воплощают концепцию домена в *физической модели данных* БД.

Как мы видели выше, *ограничения* могут применяться на уровне колонки ( *ограничения колонки* ) или на уровне *таблицы* ( *ограничения таблицы* ). *Ограничения* первичного ключа — это *ограничения*, действующие *на уровне таблицы*, а NOT NULL -ограничения — это *ограничение колонки*. Существуют три основных типа *ограничений*, используемых в реляционной БД, — *целостности данных*, целостности ссылок и *ограничения* первичного ключа. *Ограничения целостности данных* (data *integrity constraints*) относятся к значениям данных в некоторых колонках и определяются в *спецификации колонки* с помощью элементов NOT NULL, UNIQUE, CHECK. *Ограничения* целостности ссылок (referential constraints) относятся к связям между *таблицами* на основе связи первичного и внешнего ключа. *Ограничения* первичного ключа относится к значениям данных в колонках первичного ключа *таблицы* и должны налагаться на каждую *базовую таблицу* реляционной БД. В [табл. 11.3](https://intuit.ru/studies/courses/599/455/lecture/10170?page=6#table.11.3) приведен список *ограничений*, применяемых в реляционных БД.

|  |
| --- |
| Таблица 11.3. Ограничения на объекты реляционной базы данных |
|  | **Ограничение** | **Описание** |
|  | CHECK | Гарантирует, что значения находятся в границах специфицированного интервала, задаваемого предикатом |
| 2 | DEFAULT | Помещает значение по умолчанию в колонку. Гарантирует, что колонка всегда имеет значение |
| 3 | FOREING KEY | Гарантирует, что значения существуют как значения в колонке первичного ключа другой *таблицы*. Обеспечивает процедуры удаления дочерних строк при удалении связанных с ней родительских |
| 4 | NOT NULL | Гарантирует, что колонка всегда содержит значение |
| 5 | PRIMARY KEY | Гарантирует, что колонка всегда содержит значение и оно уникально в *таблице* |
| 6 | UNIQUE | Гарантирует, что значение будет уникальным в *таблице* |

Использование *ограничений* NOT NULL и PRIMARY KEY было рассмотрено выше в настоящей лекции. Использование *ограничения* FOREING KEY будет рассмотрено при обсуждении создания *таблицы* фактов.

**Ограничение CHECK** позволяет выполнять проверку содержимого колонки относительно некоторых условий и списка значений. Она налагается с помощью предложения CHECK. Для добавления этого *ограничения* нужно после объявления столбца в *спецификации колонки* определить синтаксическую конструкцию CHECK (предикат). Согласно требованиям стандарта с помощью ключевого слова VALUE в предикате вы ссылаетесь на значение колонки. Но практически во всех диалектах для этой цели используется имя колонки.

**Опция DEFAULT** заставляет СУБД размещать значение по умолчанию в колонке, когда кортеж вставляется в *таблицу* и никакого значения колонки не представлено. Чтобы указать значение по умолчанию, нужно в *спецификацию колонки* добавить ключевое слово DEFAULT и после него указать любое значение, являющееся достоверным экземпляром типа данных колонки.

**Ограничение UNIQUE** гарантирует уникальность значения данных в колонке. Оно применяется, если нужно следить за тем, чтобы значения колонки, не являющейся первичным ключом, были уникальны в *таблице*. При этом проверяется уникальность всех значений, отличных от NULL.

Примеры использования *ограничений* будут приведены ниже при анализе команд создания объектов БД для нашего учебного примера.

#### Анализ команд создания объектов базы данных

При решении задачи задания объектов БД для ХД проектировщик имеет на входе *физическую модель данных*, *таблиц* измерений и фактов, а на выходе должен создать набор *команд CREATE TABLE*, которые будут использоваться для создания *таблиц* и других объектов БД.

Для всех суррогатных ключей *таблиц* измерений и *таблицы* фактов мы применяем определение колонки с типом данных uniqueidentifier. Задание такого типа колонки позволит увеличивать значение суррогатного ключа автоматически. Для этих столбцов используется функция NEWSEQUENTIALID() в *ограничении* DEFAULT для указания значений для новых строк. Также к столбцам типа uniqueidentifier применяется свойство ROWGUIDCOL, чтобы на столбец можно было ссылаться с помощью ключевого слова $ROWGUID, и *ограничение* первичного ключа.

##### Создание таблиц измерений

*Команда CREATE TABLE* для создания *таблицы* измерения "Покупатель" (Customer) на диалекте SQL семейства СУБД MS SQL Server имеет вид:

create table Customer (

 Cust\_ID uniqueidentifier

 CONSTRAINT Guid\_Default\_1 DEFAULT NEWSEQUENTIALID() ROWGUIDCOL,

 FName varchar(20) not null,

 LName varchar(20) not null,

 Cust\_Address varchar(40) null,

 Company varchar(40) not null,

 constraint PK\_CUSTOMER primary key (Cust\_ID)

)

go

Никаких дополнительных *ограничений* на значения не требуется, поэтому скрипт для создания *таблицы* измерения "Покупатель" считаем законченным.

*Команда CREATE TABLE* для создания *таблицы* измерения "Продавец" (Employee) на диалекте SQL семейства СУБД MS SQL Server имеет вид:

create table Employee (

 Empl\_ID uniqueidentifier

 CONSTRAINT Guid\_Default\_2 DEFAULT NEWSEQUENTIALID() ROWGUIDCOL,

 Empl\_FName varchar(20) not null,

 Empl\_LName varchar(20) not null,

 City varchar(20) not null,

 Empl\_Address varchar(40) null,

 constraint PK\_EMPLOYEE primary key (Empl\_ID)

)

go

Никаких дополнительных *ограничений* на значения не требуется, поэтому скрипт для создания *таблицы* измерения "Продавец" считаем законченным.

*Команда CREATE TABLE* для создания *таблицы* измерения "Товар" (Product) на диалекте SQL семейства СУБД MS SQL Server имеет вид:

create table Product (

 Prod\_ID uniqueidentifier

 CONSTRAINT Guid\_Default\_3 DEFAULT NEWSEQUENTIALID() ROWGUIDCOL,

 Name varchar(80) not null,

 Size varchar(20) not null,

 Unit\_Price numeric(8,2) not null,

 constraint PK\_PRODUCT primary key (Prod\_ID)

)

go

Наименование товара является уникальным значением, поэтому применим к этой колонке *ограничение*, изменив строку *спецификации колонки* "Наименование товара" на следующую:

Name varchar(80) not null UNIQUE NONCLUSTERED,

Цена товара является величиной положительной, поэтому целесообразно ввести проверку вводимого значения цены товара. Из анализа предметной области следует, что цена товаров, продаваемых компанией, находится в пределах от 15 руб. до 1500 руб. и можно ввести проверку этого значения на диапазон. Изменим строку, определяющую колонку "Цена товара" (Unit\_Price), как показано ниже:

Unit\_Price numeric(8,2) not null CHECK (Unit\_Price >= 15 and Unit\_Price <= 1500),

*Команда CREATE TABLE* для создания *таблицы* измерения "Время" (Time) на диалекте SQL семейства СУБД MS SQL Server имеет вид:

create table Time (

 Time\_ID uniqueidentifier

 CONSTRAINT Guid\_Default\_4 DEFAULT NEWSEQUENTIALID() ROWGUIDCOL,

 Year integer not null,

 Quarter integer not null,

 constraint PK\_TIME primary key (Time\_ID)

)

go

По решению руководства компании данные в ХД будут заноситься, начиная с 2007 года. Поэтому целесообразно ввести проверку на значения колонки "Год" (Year). Число кварталов в году – четыре. Введем проверку на значения колонки "Квартал" (Quarter). Строки стрипта, определяющие колонки "Год" (Year) и "Квартал" (Quarter), теперь выглядят, как показано ниже.

Year integer not null CHECK (Year >= 2007),

Quarter integer not null CONSTRAINT Q\_CHK CHECK (Quarter IN ('1', '2', '3', '4''),

##### Создание таблицы фактов

*Команда CREATE TABLE* для создания *таблицы* фактов "Продажи" (Sale) на диалекте SQL семейства СУБД MS SQL Server имеет вид:

create table Sale (

 Sale\_ID uniqueidentifier

 CONSTRAINT Guid\_Default\_5 DEFAULT NEWSEQUENTIALID() ROWGUIDCOL,

 Time\_ID uniqueidentifier null,

 Cust\_ID uniqueidentifier null,

 Prod\_ID uniqueidentifier null,

 Empl\_ID uniqueidentifier null,

 Amount numeric(9,2) not null,

 Quantity integer not null,

 constraint PK\_SALE primary key (Sale\_ID)

)

go

Значения колонок "Количество" (Quantity) и "Сумма платежа" (Amount) не могут быть нулевыми. Поэтому наложим соответствующие *ограничения* на значения этих колонок, как показано ниже.

Amount numeric(9,2) not null CHECK (Amount >= 15),

Quantity integer not null CHECK (Quantity >= 1),

Колонки "Идентификатор времени" (Time\_ID), "Идентификатор покупателя" (Cust\_ID\_ "Идентификатор товара" (Prod\_ID)) и "Идентификатор продавца" (Empl\_ID) являются значениями первичных ключей *таблиц* измерений и поэтому могут служить внешними ключами в *таблице* фактов. Наложим на значения этих колонок *ограничения* внешнего ключа.

##### Ограничения внешнего ключа в таблице фактов

Заметим, что *ограничения* задаются в *спецификациях колонки* или спецификациях ключей при создании *таблицы* в командах SQL *CREATE TABLE* или налагаются после создания *таблицы* в командах SQL *ALTER TABLE*. Как добавить *ограничения* в *таблицу* с помощью *команды CREATE TABLE*, мы уже знаем.

Воспользуемся *командой ALTER TABLE* для наложения *ограничений* внешнего ключа в *таблице* фактов. Синтаксис *команды ALTER TABLE* приведен ниже. Эта команда изменяет определение *таблицы* путем изменения, добавления или удаления столбцов и *ограничений*.

ALTER TABLE table\_name

{ [ ALTER COLUMN column\_name

 {DROP DEFAULT

 | SET DEFAULT constant\_expression

 | IDENTITY [ ( seed , increment ) ]

 }

| ADD

 { < column\_definition > | < table\_constraint > } [ ,...n ]

| DROP

 { [ CONSTRAINT ] constraint\_name

 | COLUMN column }

] }

< column\_definition > ::=

 { column\_name data\_type }

 [ [ DEFAULT constant\_expression ]

 | IDENTITY [ ( seed , increment ) ]

 ]

 [ROWGUIDCOL]

 [ < column\_constraint > ] [ ...n ] ]

< column\_constraint > ::=

 [ NULL | NOT NULL ]

 [ CONSTRAINT constraint\_name ]

 {

 | { PRIMARY KEY | UNIQUE }

 | REFERENCES ref\_table [ (ref\_column) ]

 [ ON DELETE { CASCADE | NO ACTION | SET DEFAULT |SET NULL } ]

 [ ON UPDATE { CASCADE | NO ACTION | SET DEFAULT |SET NULL } ]

 }

< table\_constraint > ::=

 [ CONSTRAINT constraint\_name ]

 { [ { PRIMARY KEY | UNIQUE }

 { ( column [ ,...n ] ) }

 | FOREIGN KEY

 ( column [ ,...n ] )

 REFERENCES ref\_table [ (ref\_column [ ,...n ] ) ]

 [ ON DELETE { CASCADE | NO ACTION | SET DEFAULT |SET NULL } ]

 [ ON UPDATE { CASCADE | NO ACTION | SET DEFAULT |SET NULL } ]

 }

Листинг .

Не будем приводить описания тех аргументов, которые присутствуют в команде *команде CREATE TABLE*. Дадим описание только специфичных для этой команды аргументов.

* ALTER COLUMN. Указывает, что определенный столбец будет изменен или модифицирован.
* ADD. Указывает, что добавлено одно или несколько определений столбца или *ограничений* *таблицы*.
* DROP { [CONSTRAINT] constraint\_name| COLUMN column}. Указывает, что из *таблицы* будет удален constraint\_name или column\_name.

Чтобы добавить *ограничения* внешнего ключа в *таблицу* фактов "Продажи" (Sale) с помощью *команды ALTER TABLE*, можно поступить следующим образом.

alter table Sale

 add constraint FK\_SALE\_REFERENCE\_TIME foreign key (Time\_ID)

 references Time (Time\_ID)

go

alter table Sale

 add constraint FK\_SALE\_REFERENCE\_CUSTOMER foreign key (Cust\_ID)

 references Customer (Cust\_ID)

go

alter table Sale

 add constraint FK\_SALE\_REFERENCE\_PRODUCT foreign key (Prod\_ID)

 references Product (Prod\_ID)

go

alter table Sale

 add constraint FK\_SALE\_REFERENCE\_EMPLOYEE foreign key (Empl\_ID)

 references Employee (Empl\_ID)

go

Теперь проектировщик хранилища данных может перейти к созданию *индексов*.

**Создание индексов хранилища данных**

Когда вы определяете PRIMARY KEY при создании *таблицы*, многие реляционные СУБД, в том числе СУБД семейства MS SQL Server, требуют обязательного создания уникального *индекса первичного ключа*. *Индексы*, так же как и *таблицы*, являются объектами реляционной <L (но не реляционной модели данных). Логически *индексы* представляют собой *таблицу*, в которой каждому значению индексируемой колонки ставится в соответствие некоторая информация, связанная с ее месторасположением на физическом носителе. Индексы предназначены для организации быстрого доступа к строкам *таблицы* и обеспечения контроля целостности данных (механизм *индексов* будет блокировать БД от повторного ввода строк в *таблицу* с одинаковыми значениями индексируемых атрибутов).

*Индекс* создается с помощью *команды CREATE INDEX*. Эта команда генерирует реляционный *индекс* или *представление* для указанной *таблицы*. *Индекс* может быть создан до появления данных в *таблице*. Реляционные *индексы* для *таблиц* или *представлений* могут быть созданы в другой базе данных, если указать ее полное имя.

Синтаксис *команды CREATE INDEX* приведен ниже.

Create Relational Index

CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index\_name

 ON <object> ( column [ ASC | DESC ] [ ,...n ] )

 [ INCLUDE ( column\_name [ ,...n ] ) ]

 [ WHERE <filter\_predicate> ]

 [ WITH ( <relational\_index\_option> [ ,...n ] ) ]

 [ ON { partition\_scheme\_name ( column\_name )

 | filegroup\_name

 | default

 }

 ]

 [ FILESTREAM\_ON { filestream\_filegroup\_name | partition\_scheme\_name | "NULL" } ]

[ ; ]

<object> ::=

{

 [ database\_name. [ schema\_name ] . | schema\_name. ]

 table\_or\_view\_name

}

<relational\_index\_option> ::=

{

 PAD\_INDEX = { ON | OFF }

 | FILLFACTOR = fillfactor

 | SORT\_IN\_TEMPDB = { ON | OFF }

 | IGNORE\_DUP\_KEY = { ON | OFF }

 | STATISTICS\_NORECOMPUTE = { ON | OFF }

 | DROP\_EXISTING = { ON | OFF }

 | ONLINE = { ON | OFF }

 | ALLOW\_ROW\_LOCKS = { ON | OFF }

 | ALLOW\_PAGE\_LOCKS = { ON | OFF }

 | MAXDOP = max\_degree\_of\_parallelism

 | DATA\_COMPRESSION = { NONE | ROW | PAGE}

 [ ON PARTITIONS ( { <partition\_number\_expression> | <range> }

 [ , ...n ] ) ]

}

<filter\_predicate> ::=

 <conjunct> [ AND <conjunct> ]

<conjunct> ::=

 <disjunct> | <comparison>

<disjunct> ::=

 column\_name IN (constant ,…)

<comparison> ::=

 column\_name <comparison\_op> constant

 <comparison\_op> ::=

 { IS | IS NOT | = | <> | != | > | >= | !> | < | <= | !< }

<range> ::=

<partition\_number\_expression> TO <partition\_number\_expression>

Листинг .

Значения аргументов команды следующие.

* UNIQUE. Создает уникальный *индекс* для *таблицы* или *представления*. Уникальным является *индекс*, в котором не может быть двух строк с одним и тем же значением ключа *индекса*. Кластеризованный *индекс* *представления* должен быть уникальным.

Компонент *Database Engine* не позволяет создать уникальный *индекс* по столбцам, уже содержащим повторяющиеся значения, даже если параметру IGNORE\_DUP\_KEY присвоено значение ON. При попытке написания такого *индекса* компонент *Database Engine* выдает сообщение об ошибке. Прежде чем создавать уникальный *индекс* по такому столбцу или столбцам, необходимо удалить все повторяющиеся значения. Столбцы, используемые в уникальном *индексе*, должны иметь свойство NOT NULL, т.к. при создании *индекса* значения NULL рассматриваются как повторяющиеся.

* CLUSTERED. Создает *индекс*, в котором логический порядок значений ключа определяет физический порядок соответствующих строк в *таблице*. На нижнем (конечном) уровне такого *индекса* хранятся действительные строки данных *таблицы*. Для *таблицы* или *представления* в каждый момент времени может существовать только один кластеризованный *индекс*.

Если аргумент CLUSTERED не указан, создается некластеризованный *индекс*.

* NONCLUSTERED. Создание *индекса*, задающего логическое упорядочение для *таблицы*. Логический порядок строк в некластеризованном *индексе* не влияет на их физический порядок.
* index\_name. Имя *индекса*. Имена *индексов* должны быть уникальными в пределах *таблицы* или *представления*, но не обязательно должны быть уникальными в пределах базы данных.
* Column. Колонка или колонки, на которых основан *индекс*. Имена одной или нескольких колонок для создания комбинированного *индекса*. Колонки, которые должны быть включены в составной *индекс*, указываются в скобках за аргументом table\_or\_view\_name в порядке сортировки.

В один *составной ключ* *индекса* могут входить до 16 колонок. Все колонки *составного ключа* *индекса* должны находиться в одной *таблице* или одном и том же *представлении*. Максимальный общий размер значений составного *индекса* равен 900 байт.

* [ ASC | DESC ]. Определяет сортировку значений заданного столбца *индекса*: по возрастанию или по убыванию. Значение по умолчанию — ASC.
* INCLUDE ( column [ ,... n ] ). Указывает неключевые столбцы, добавляемые на конечный уровень некластеризованного *индекса*. Некластеризованный *индекс* может быть уникальным или неуникальным.

Имена столбцов в списке INCLUDE не могут повторяться и не могут использоваться одновременно как ключевые и неключевые.

* WHERE <filter\_predicate>. Создает отфильтрованный *индекс* путем указания строк для включения в *индекс*. Отфильтрованный *индекс* должен быть некластеризованным *индексом* для *таблицы*. Также создается статистика фильтрации для строк данных отфильтрованного *индекса*.
* ON partition\_scheme\_name ( column\_name ). Задает *схему* секционирования, которая определяет файловые группы, соответствующие *секциям* секционированного *индекса*. *Схема* секционирования должна быть уже создана в базе данных с помощью инструкции CREATE PARTITION SCHEME или ALTER PARTITION SCHEME. Аргумент column\_name задает столбец, по которому будет секционирован *индекс*. Этот столбец должен соответствовать типу данных, длине и точности аргумента *функции секционирования*, которую использует *схема* partition\_scheme\_name. Аргумент column\_name может указывать на столбцы, не входящие в определение *индекса*. Можно указать любой столбец *базовой таблицы*, за исключением случая секционирования *индекса* UNIQUE, когда столбец column\_name должен быть выбран из используемых в уникальном ключе. Это *ограничение* дает возможность компоненту *Database Engine* проверять уникальность значений ключа только в одной *секции*.
* ON filegroup\_name. Создает заданный *индекс* в указанной файловой группе. Если местоположение не указано и *таблица* или *представление* не секционированы, *индекс* использует ту же файловую группу, что и *базовая таблица* или базовое *представление*. Файловая группа должна существовать.
* ON "default". Создает заданный *индекс* в файловой группе, используемой по умолчанию.
* [ FILESTREAM\_ON { filestream\_filegroup\_name | partition\_scheme\_name | "NULL" }]. Указывает размещение данных FILESTREAM для *таблицы* при создании кластеризованного *индекса*. Предложение FILESTREAM\_ON позволяет перемещать данные FILESTREAM в другую файловую группу FILESTREAM или *схему* секционирования.
* <object>::= Полное или неполное имя индексируемого объекта.
* database\_name. Имя базы данных.
* schema\_name. Имя *схемы*, к которой принадлежит *таблица* или *представление*.
* table\_or\_view\_name. Имя индексируемой *таблицы* или *представления*.
* <relational\_index\_option>::= Указывает параметры, которые должны использоваться при создании *индекса*.
* PAD\_INDEX = { ON | OFF }. Определяет заполнение *индекса*. Значение по умолчанию — OFF.
* FILLFACTOR = fillfactor. Указывает, на сколько процентов должен компонент *Database Engine* заполнить страницы конечного уровня при создании или перестройке *индекса*. fillfactor должен быть целым числом от 1 до 100. Значение по умолчанию — 0. Если fillfactor равен 100 или 0, компонент *Database Engine* создает *индексы* с полностью заполненными страницами конечного уровня.
* SORT\_IN\_TEMPDB = { ON | OFF }. Указывает, сохранять ли временные результаты сортировки в базе данных tempdb. Значение по умолчанию — OFF.
* IGNORE\_DUP\_KEY = { ON | OFF }. Определяет ответ на ошибку, случающуюся, когда операция вставки пытается вставить в уникальный *индекс* повторяющиеся значения ключа.
* STATISTICS\_NORECOMPUTE = { ON | OFF }. Указывает, выполнялся ли перерасчет статистики распределения. Значение по умолчанию — OFF.
* DROP\_EXISTING = { ON | OFF }. Указывает, что названный существующий кластеризованный или некластеризованный *индекс* удаляется и перестраивается. Значение по умолчанию — OFF.
* ONLINE = { ON | OFF }. Определяет, будут ли *базовые таблицы* и связанные *индексы* доступны для запросов и изменения данных во время операций с *индексами*. Значение по умолчанию — OFF.
* ALLOW\_ROW\_LOCKS = { ON | OFF }. Указывает, разрешена ли блокировка строк. Значение по умолчанию — ON.
* ALLOW\_PAGE\_LOCKS = { ON | OFF }. Указывает, разрешена ли блокировка страниц. Значение по умолчанию — ON.
* MAXDOP = max\_degree\_of\_parallelism. Переопределяет параметр конфигурации максимальной степени параллелизма на время операций с *индексами*. MAXDOP можно применять для *ограничения* числа процессоров, используемых в одновременном выполнении планов. Максимальное число процессоров — 64.
* DATA\_COMPRESSION. Задает режим сжатия данных для указанного *индекса*, номера *секции* или диапазона *секций*. Доступные параметры: NONE — *индекс* или заданные *секции* не сжимаются; ROW — для *индекса* или заданных *секций* производится сжатие строк; PAGE — для *индекса* или заданных *секций* производится сжатие страниц.
* ON PARTITIONS ( { <partition\_number\_expression> | <range> } [ , ...n ] ). Указывает *секции*, к которым применяется параметр DATA\_COMPRESSION. Если *индекс* не секционирован, аргумент ON PARTITIONS создаст ошибку. Если не указано предложение ON PARTITIONS, то параметр DATA\_COMPRESSION применяется ко всем *секциям* секционированного *индекса*.

Предложение CREATE INDEX определяет имя *индекса* ; предложение ON определяет имя *таблицы* и колонок, для которой и по которым строится *индекс* ; ключевое слово UNIQUE указывает, что индексируемые значения колонок должны быть уникальными для *таблицы*, т. е. исключается дублирование значений в индексируемой колонке. *Таблица* должна быть уже создана и содержать определения индексируемых столбцов. Спецификация UNIQUE опциональна, и вы можете также создавать и неуникальные *индексы*.

Для диалекта SQL СУБД семейства MS SQL Server *индексы* на колонки с *ограничением* первичного ключа и UNIQUE создаются автоматически. Поэтому проектировщику ХД нужно создать *индексы* для тех колонок, для которых он считает наличие *индекса* целесообразным.

Колонками – кандидатами для создания дополнительных *индексов* являются в нашем случае, хотя это можно и оспорить, "Наименование товара" (Name) *таблицы* измерения "Товар" (Product) и "Фамилия продавца" (Empl\_LName) *таблицы* измерения "Продавцы" (Employee). Создадим эти *индексы*, как показано ниже.

CREATE UNIQUE CLUSTERED INDEX Idx1 ON Product(Name);

go

CREATE UNIQUE CLUSTERED INDEX Idx2 ON Employee (Empl\_LName);

go

После выполнения вышеперечисленных действий задачу создания физической модели в первом приближении можно считать законченной. Теперь можно запустить разработанный скрипт для созданной БД и считать, что ХД создано.

**Резюме**

В этой лекции мы рассмотрели принципы разработки физической модели ХД. Создание физической модели ХД состоит в моделировании и создании объектов для хранения данных в *БД* конкретной *СУБД*. Эта задача сводится к моделированию и созданию *таблиц* и объектов в *БД*, в которых будет храниться *информация* о сущностях *предметной области* ХД. Решая эту задачу, проектировщик отображает отношения логической модели данных ХД в *таблицы* и *индексы* *БД*. Для выполнения этой задачи используется *подмножество* команд *SQL* – язык определения данных *DDL* (*Data Definition Language*).

ХД данных создается в реляционной *БД*. *Физическая модель* реляционной *БД* есть такое *представление* отношений *БД* и связей между ними, которое воплощено в последовательность команд *SQL*. Выполнение этой последовательности команд создает конкретную *БД* и ее объекты.

Сначала создаются *таблицы* *БД*. Напомним, что *таблицы* в реляционных *СУБД* состоят из одной или более колонок или полей. Колонки представляют собой поименованные ячейки в записи, которые содержат значения. Колонки определяются посредством спецификации, которая вводит формат колонки и ее характеристики, задаваемые с помощью *ограничений*.

Общий *алгоритм* построения физической модели ХД включает в себя следующие действия.

* Разработка физической модели ХД:
	+ определение *базовых таблиц* БД;
	+ определение колонок в *таблицах* ;
	+ определение типов данных для колонок;
	+ назначение первичных ключей *таблицам* ;
	+ задание *ограничений* NOT NULL на значения колонок;
	+ создание связей между *таблицами*.
* Разработка скрипта создания ХД:
	+ формирование *команд CREATE TABLE* для *таблиц* ХД;
	+ определение *ограничений* на колонки *таблиц* ХД;
	+ формирование дополнительных *индексов* *командой CREATE INDEX*.