**Модуль 4. Многотабличные базы данных (3 пары)**

**Цель занятия:** изучить принципы создания многотабличной базы данных, рассмотреть правила нормализации таблиц, а также принципы формирования многотабличного запроса.

**План занятия:**

1. Актуализация знаний.

2. Аномалии взаимодействия с однотабличной базой данных:

* аномалии обновления;
* аномалии вставки;
* аномалии удаления.

2. Принципы создания многотабличной базы данных:

* причины создания многотабличной базы данных;
* внешний ключ;
* связи. Типы связей;
* целостность данных;
* нормализация:
* необходимость нормализации;
* понятие нормальной формы;
* первая нормальная форма;
* вторая нормальная форма;
* третья нормальная форма;
* нормальная Форма Бойса-Кодда.

3. Многотабличные запросы:

* принципы создания многотабличного запроса;
* декартовое произведение.

4. Домашнее задание.

1. **Актуализация знаний.**
2. **Аномалии взаимодействия с однотабличной базой данных**

Вы уже начали рассматривать строение SQL Server, рассмотрели строение и создание баз данных и таблиц. Это просто Здорово. Вы даже начали писать запросы!!! Но. В жизни вряд ли вся информация (как в Books) храниться в одной таблице и наверняка найдены модели, которые позволяют наиболее оптимально хранить и использовать хранящиеся данные. Вот об этом и пойдет речь в этом уроке.

# Аномалии взаимодействия с однотабличной базой данных.

Что из себя представляет аномалия в обычной жизни. Каждый понимает это по своему, общая же идея такова: Аномалия это то, что не является нормой и в связи с этим считается определенной странностью и исключением. Впрочем, с понятием исключений вы встречались на программировании, например, когда программа выполняет недопустимую операцию (например деление на 0) и закрывается. Это аномалия работы приложения.

При взаимодействии с однотабличной базой данных может также произойти ряд аномалий. Т.е. логически все построено, вроде бы, правильно, но возникает ошибка, которая может повлечь нарушение всей структуры базы данных.

Список аномалий, которые могут вам встретиться:

1. Аномалия добавления.
2. Аномалия обновления.
3. Аномалия удаления.

### Аномалия добавления.

Давайте внимательнее рассмотрим базу данных books. На первый взгляд все здорово и понятно. Но, изменив некоторые параметры, вы поймете, на сколько удобно с ней работать сейчас и какие варианты аномалий с ней могли бы происходить.

Допустим, что все столбцы таблицы обязательны для заполнения. Но вам поступает заказанная кем-то книга, которую необходимо занести в базу. При этом вы не знаете ни названия издательства, ни цену. Для этого требовалось бы сделать следующий запрос.

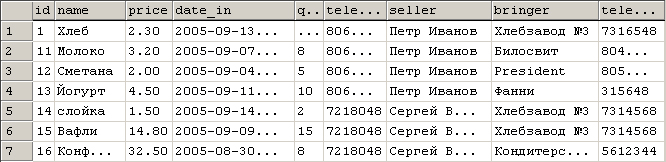
Insert into books

values(772,8901,1,'Аномалии при работе с базой данных',null,null,

550,'80\*20/15',2005-01-09,25000,'базы данных','аномалии')

Ошибок в синтаксисе нет, но ошибка будет в том, что занести данные нужно во все поля и подставить null запрещено, т.к. в противном случае информация о потенциально новом издательстве может просто отсутствовать в базе данных.

И еще один пимер, понагляднее. Вот информация из базы данных Магазин, в которой храниться информация о продуктах, продавцах и поставщиках.



Как видите, вся эта информация хранится в одной таблице. Проблема: Товар нужно добавить, а телефон поставщика не знаем. Это поле является обязательным для заполнения и поэтому товар не может быть добавлен.

### Аномалия обновления.

Еще одна аномалия. Представьте, что кроме названия издательства в базе данных еще необходимо хранить к каждому из издательств коды тех книг, которые они издали. При этом у какой-то одной книги меняется ее код. Одного запроса для этого не хватит, да и может произойти ситуация, когда в каких-то полях номер изменен не будет, что приведет к нарушению целостности всей базы данных.

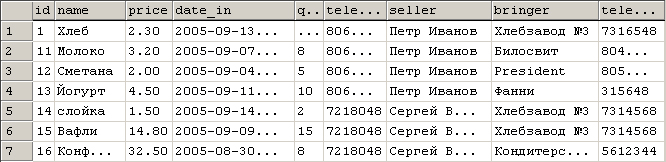
### Аномалия удаления.

При удалении какой-либо информации может произойти ситуация, когда будет случайно удалена нужная. Например, вы хотите из базы данных удалить издательства, а при этом удаляется несколько категорий. Т.о. мы теряем из списка категорий те, которые понадобятся для работы в дальнейшем. Или, удаляя категории, теряем тематику и т.д.

Давайте снова рассмотрим базу данных Магазин. Представте, что производится удаление из базы данных всех товаров, которые хранятся на складе больше 2 недель и еще Магазин поссорился с Хлебзаводом №3 и все записи об этом хлебзаводе также нужно удалить.

delete from shop where date\_in<getdate()-14 or bringer ='Хлебзавод №3'

Посмотрите на таблицу.



Такой запрос удалит все записи о Сергее Васильеве, т.е. в результате пропал сотрудник.

Получается, что при удалении одних данных мы непременно теряем те, которые удалять не стоило.

Все эти 3 аномалии говорят о том, то структура нашей существующей базы данных не является оптимальной как по мере использования, так и по мере хранения информации, т.к. одна и та же информация повторяется очень много раз, что отражается на размере базы данных.

Выход один. Нужно разбить существующую таблицу на несколько отдельных.

Т.о. мы переходим к созданию многотабличных баз данных.

# Внешние ключи. Связи. Целостность данных.

Вы увидели возможные варианты ошибок при проектировании баз данных и учли, что однотабличная база данных - это плохо. Поэтому, давайте разбираться с созданием многотабличных баз данных. Но нужно не забывать, что, хоть информация и разделяется на несколько таблиц, необходимо определить, как они должны взаимодействовать между собой. Для этого необходимо определить несколько ключевых понятий.

## Первичный ключ.

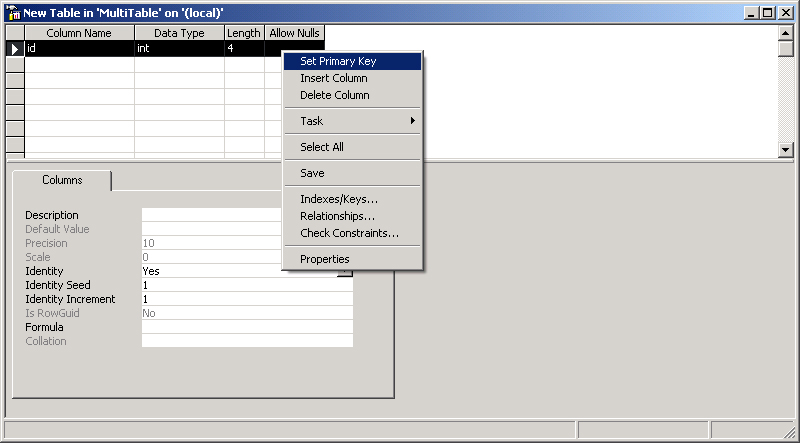
**Первичный ключ** - это поле таблицы, которое служит для идентификации строк. Самый лучший пример первичного ключа - это нумерация строк в игре морской бой. Представьте, какая была бы путаница, если бы на поле для игры в морской бой было бы 2 строки с номером 3, 4 строки с номером 5 и т.д. Тогда точно идентифицировать строку просто невозможно.

В реальной жизни может возникнуть ситуация, когда в базе данных, которая хранит информацию о людях, появится несколько человек с одинаковыми именами и фамилиями. В этой ситуации однозначно идентифицировать человека нельзя, поэтому единственный выход - создать поле, в котором будут храниться неповторяющиеся номера строк, и по этому полю определять требуемую строку.



Т.о. столбцы идентифицируются по имени столбца (если вы помните, то в одной таблице имена столбцов также не могут повторяться), а стоки идентифицируются по первичному ключу. Обычно этому полю дают имя Id. Таким полем обязана обладать любая таблица реляционной базы данных.

Для установки первичного ключа необходимо при создании таблицы, в контекстном меню к полю, выбрать пункт меню Set Primary Key.

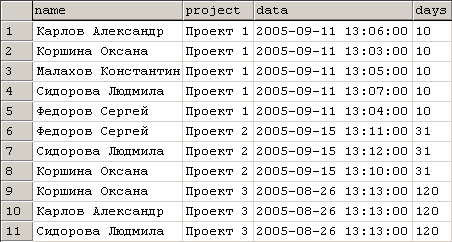


Заметьте, что тип данных у поля установлен int, а дополнительный параметр - Identity: Yes, Identity Seed: 1, Identity Increment: 1. (Если вы забыли - Identity это счетчик). Для первичного ключа это незаменимая опция, т.к. при добавлении информации вы вряд ли будете помнить, какой уникальный номер был в последнем добавленном элементе, а так каждое последующее значение будет на 1 больше чем предыдущее.

### Составной первичный ключ.

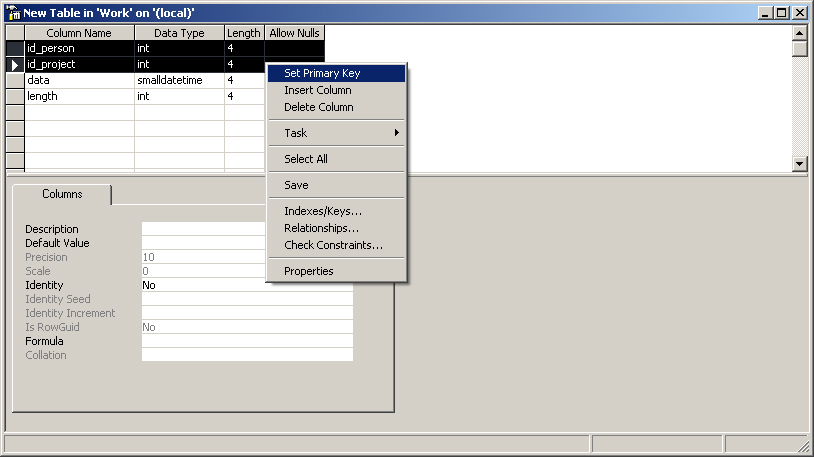
Существует возможность создать таблицу, у которой первичный ключ будет основываться одновременно на нескольких полях. Тогда уникальным будет не значение в каждом из них, а ***уникальны будут сочетания значений этих полей***.

Рассмотрим таблицу, хранящую информацию о проектах и людях, которые их реализуют.



Первых два поля определяют составной первичный ключ. Т.е. любой человек может участвовать в любом проекте, либо во всех сразу. Составной первичный ключ, при этом, будет обеспечивать проверку, чтобы ни один человек не участвовал 2 раза в одном проекте. Это возможно за счет того, что одинаковых комбинаций первого и второго поля невозможны.

Чтобы создать составной первичный ключ есть 2 способа. Один из них - при помощи запроса, мы рассмотрим в одном из следующих уроков. Пока остановимся на его определении при помощи Enterprise Manager.



Все что нужно сделать - при создании таблицы выделить те поля, которые вы хотите включить в составной первичный ключ, после чего выбрать в контекстном меню к выделенным полям Set Primary Key. **Важно** помнить, что все поля, которые включаются в составной первичный ключ, должны быть обязательными для заполнения (не должна стоять галочка Allow Nulls).

## Внешний ключ.

**Внешним ключом** называется поле, в котором хранятся значения первичного ключа другой таблицы.

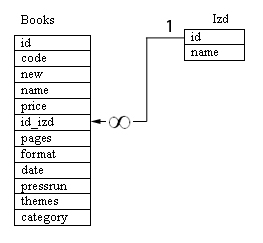
Давайте рассмотрим хранение информации в базе данных books. Если вы заметили, то поле Izd хранит издательства и при этом названия издательств повторяются. Происходит следующее. Учитывая, что тип данных поля Izd nvarchar(255), а самое короткое имя у издательства "Век", то минимальное кол-во занимаемых одной строкой байт - 6 (а максимальное теоретически 500 байт).

Чтобы избежать такого перенасыщения таблицы повторяющейся информацией, обычно создают отдельную таблицу, хранящую список издательств без повторений. В основной таблице, при этом, добавляется поле, в которое помещают значения первичного ключа из таблицы с издательствами. Список издательств с основной таблицы, после этого удаляется.

Получается, что таблица books

C:\Users\admin\Google Диск\Мои документы\ШАГ\MS SQL\MS SQL\Уроки ПСТ\Урок 3\program\img\pict2.jpg

преобразуется в несколько таблиц.



Единицей обычно обозначают первичный ключ, а знаком бесконечность - внешний ключ.

Не сложно догадаться, что желательно еще отдельно вынести тематики и категории.

Еще один пример.

Допустим, что у нас есть две таблицы в базе данных: первая таблица содержит данные о продавцах, а вторая - о покупателях. В таблице с данными о покупателях есть столбец, в котором записано имя продавца, продавшего товар. Было бы очень скучно и непродуктивно писать по несколько раз одно и то же имя продавца (а что делать, если есть только 2-3 продавца, а сделок осуществлено более 1000 ?). Вот тут и возникает вопрос: может существует другая, более удобная возможность заполнения данного поля?

|  |  |
| --- | --- |
| Продавцы ID | Name |
| 1 | Brown |
| 2 | Smith |

|  |  |
| --- | --- |
| Покупатели ID | Name |
| 1 | Gates |
| 2 | Jackson |

|  |  |  |  |
| --- | --- | --- | --- |
| Сделки ID | ID\_cust | ID\_sal | Amt |
| 1 | 2 | 2 | 300 |
| 2 | 1 | 2 | 500 |

В данном случае ID\_cust и ID\_sal в таблице "сделки" являются внешними ключами которые соответствуют первичным ключам в таблице "покупатели"(ключевое поле ID) и в таблице "продавцы"(ключевое поле также ID).

## Связи.

Чтобы суметь нормально спроектировать базу данных нужно понять, что информация, хранящаяся в нескольких таблицах одной базы данных, все равно остается неразрывным целым. В связи с этим нужно рассмотреть типы взаимозависимости информации в таблицах - сокращенно связи.

### Типы связи.

Между двумя сущностям, например, А и В возможны четыре вида связей.

**Первый тип** – связь ОДИН-К-ОДНОМУ (1:1): в каждый момент времени каждому представителю (экземпляру) сущности А соответствует 1 или 0 представителей сущности В:

C:\Users\admin\Google Диск\Мои документы\ШАГ\MS SQL\MS SQL\Уроки ПСТ\Урок 3\program\img\pic2-2-1.gif

Студент может не "заработать" стипендию, получить обычную или одну из повышенных стипендий.

**Второй тип** – связь ОДИН-КО-МНОГИМ (1:М): одному представителю сущности А соответствуют 0, 1 или несколько представителей сущности В.

C:\Users\admin\Google Диск\Мои документы\ШАГ\MS SQL\MS SQL\Уроки ПСТ\Урок 3\program\img\pic2-2-2.gif

Квартира может пустовать, в ней может жить один или несколько жильцов.

Так как между двумя сущностями возможны связи в обоих направлениях, то существует еще два типа связи МНОГИЕ-К-ОДНОМУ (М:1) и МНОГИЕ-КО-МНОГИМ (М:N).

**Пример.** Если связь между сущностями МУЖЧИНЫ и ЖЕНЩИНЫ называется БРАК, то существует четыре возможных представления такой связи:



Существует еще один вид связей, так называемая "связь предок/потомок". Если первичный ключ данной таблицы присутствует в поле (полях) одной или нескольких таблиц, то эти таблицы являются "потомками" по отношению к данной таблице, а она в свою очередь является "предком" по отношению к ним.

Переходим к практике создания многотабличных баз дынных.

# Принципы создания многотабличной базы данных.

Чтобы не создавать новую базу данных мы будем видоизменять структуру базы данных books, которую вы импортировали. Учитывая, что информация повторяется много раз в полях Izd, Themes и Categories, вынесем эту информацию в 3 отдельных таблицы.

В таблицу books, при помощи Enterprise manager, добавляем 3 поля : id\_press, id\_theme, id\_category, также поле N нужно переименовать в Id и сделать первичным ключом, если оно таковым не является. Создаем таблицы для хранения издательств - Press, таблицу для тематик - Themes, таблицу для категорий - Categories. Поля с таблицы books нужно будет удалить только после экспортирования данных из нее.

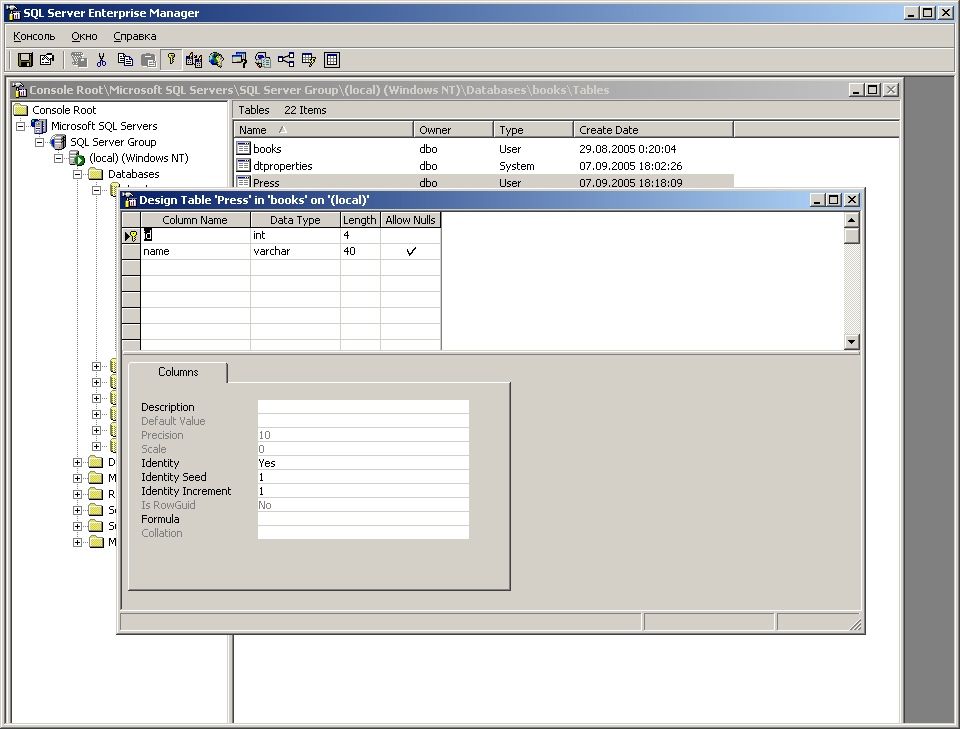
У вас должны получиться следующие таблицы в базе данных Books. При изменении, например, типов данных, возможно, появится предупреждение о потере данных. Также возможна ошибка установки первичного ключа, если в этом поле есть повторяющиеся значения.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BOOKS | | | | |
| Id | int | 4 | Not Null | Identity - yes, Seed - 1, Increment - 1 |
| Code | float | 8 | Nullable |  |
| New | bit | 1 | Not Null |  |
| Name | varchar | 40 | Nullable |  |
| Price | money | 8 | Nullable |  |
| Izd | varchar | 40 | Nullable |  |
| Pages | int | 4 | Nullable |  |
| Format | varchar | 20 | Nullable |  |
| Date | smalldatetime | 4 | Nullable |  |
| Pressrun | int | 4 | Nullable |  |
| Categories | varchar | 40 | Nullable |  |
| Themes | varchar | 40 | Nullable |  |
| id\_press | int | 4 | Nullable |  |
| id\_theme | int | 4 | Nullable |  |
| id\_category | int | 4 | Nullable |  |

Вот как должна выглядеть Books в Enterprise manager.

C:\Users\admin\Google Диск\Мои документы\ШАГ\MS SQL\MS SQL\Уроки ПСТ\Урок 3\program\img\pict4.jpg

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Press | | | | |
| Id | int | 4 | Not Null | Identity - yes, Seed - 1, Increment - 1 |
| Name | varchar | 40 | Nullable |  |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Themes | | | | |
| Id | int | 4 | Not Null | Identity - yes, Seed - 1, Increment - 1 |
| Name | varchar | 40 | Nullable |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Categories | | | | |
| Id | int | 4 | Not Null | Identity - yes, Seed - 1, Increment - 1 |
| Name | varchar | 40 | Nullable |  |

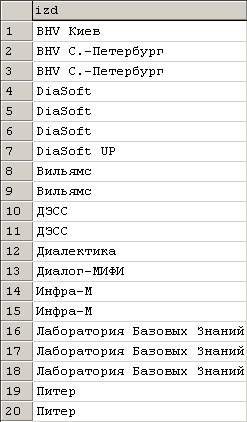
Надеюсь, что проблем с созданием у вас не возникло. Теперь нужно эти таблицы заполнить. При этом, нужно использовать информацию с таблицы books.

Чтобы начать работу с копированием данных необходимо изучить еще одно ключевое слово.

## Distinct

Иногда возникает необходимость показать значения столбца, исключив при показе повторяющиеся значения. Для этого перед именами (либо именем) поля указывается ключевое слово **Distinct**. Только вы должны помнить, что повторения убираются из столбца с максимальным кол-вом **неповторяющихся** значений.

Запрос **Select izd from books** выдаст в результате такую таблицу.

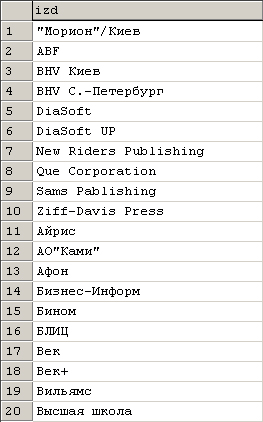


Если же вы хотите исключить повторения, тогда запрос будет выглядеть так:

Select Distinct izd

From books

И вы получите такой результат:



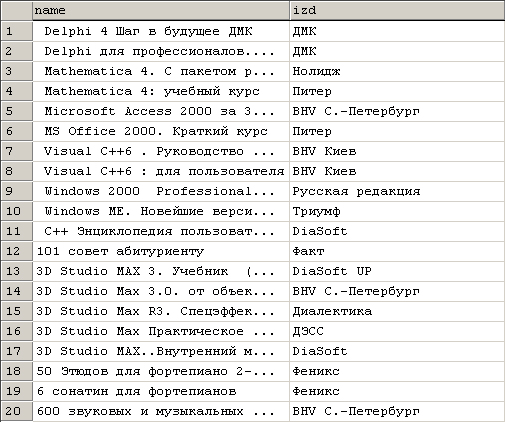
Результатом будут являться названия издательств, при этом повторяться они не будут. Но.

Пример 2.

Select Distinct name, izd

From books

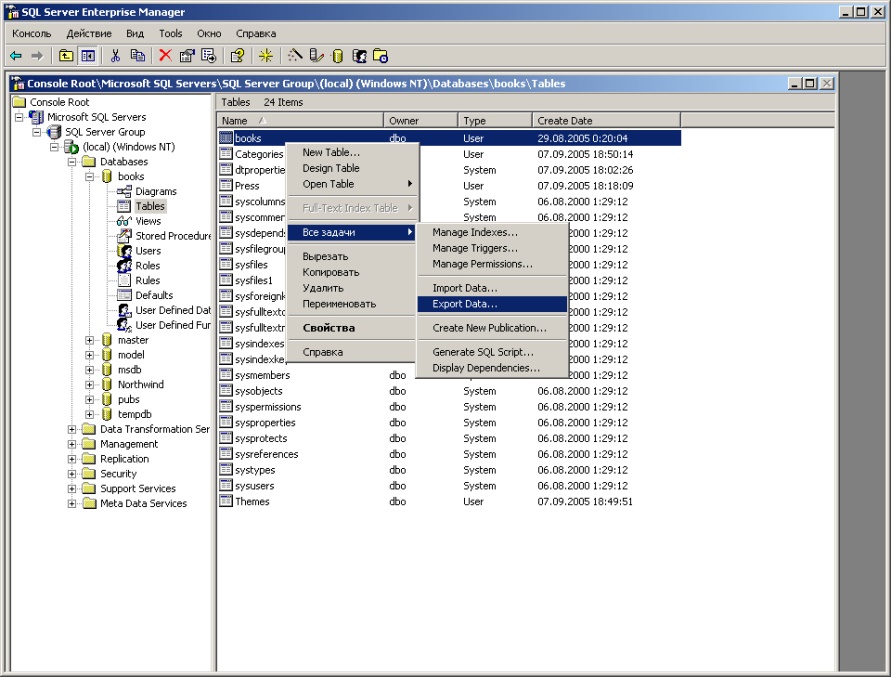
Теоретически должно быть два столбца: названия книг, без повторений и названия издательств, без повторений. Но книг без повторений около 800, а издательств без повторений около 60. Вывести несколько столбцов с разным кол-вом строк - невозможно. Поэтому записей в поле с издательствами будет столько же, сколько и в столбце с названиями книг. И, конечно же, названия издательств, при этом, будут повторяться.



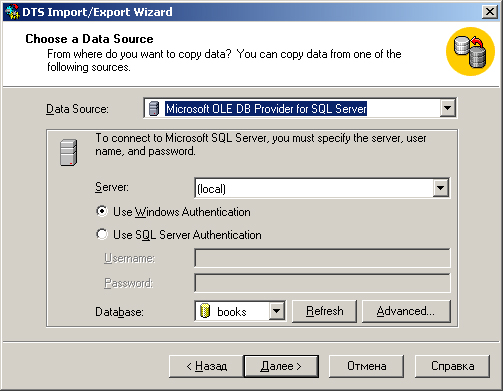
### Импортирование данных из одной таблицы в другую.

Начнем с копирования данных из таблицы books в таблицы press. Сразу определим, что нужно скопировать: столбец ***Izd*** таблицы **Books** должен быть скопирован в столбец ***Name*** таблицы **Press** без повторений.

Открываем Enterprise manager, выбираем пункт меню *Все задачи -> Export Data...* таблицы Books.

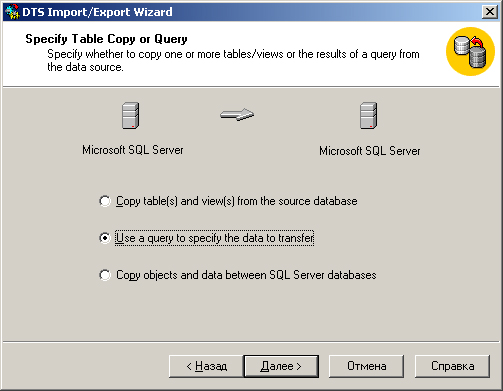
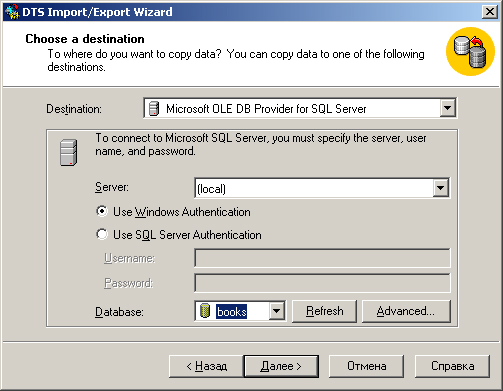


Появляется менеджер импортирования и экспортирования данных.



В этом окне определяется ресурс, откуда будет производиться импортирование информации. Конечно же, в этот раз это база данных Books.

В следующем окне необходимо указать базу данных, в которую будет производиться импортирование данных. Это наша же база данных Books.

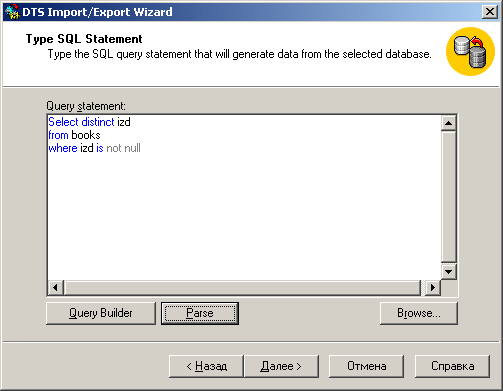


Далее необходимо определить тип копируемой информации. Выбираем Use a query to specify the data to transfer. Т.е. использовать запрос для определения требуемой информации. Учитывая, что мы копируем издательства, запрос следует указать такой:

Select distinct izd

from books

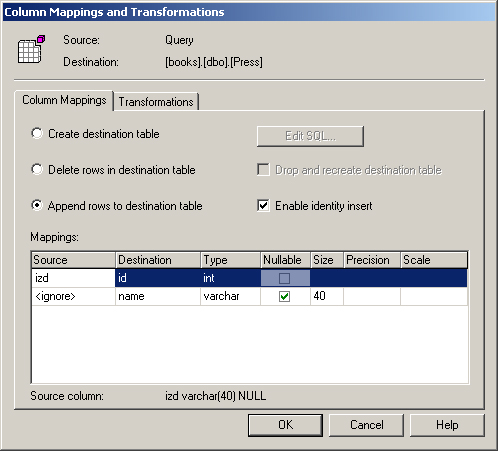
where izd is not null



Нажимаем Далее.

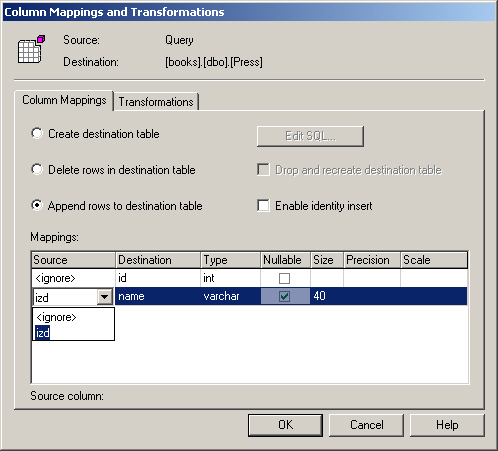


В появившемся окне нужно конкретно указать куда производить вставку информации. Для этого нужно нажать на выпадающий список Destination и выбрать требуемую таблицу той базы данных, которую указали во втором окне утилиты Импорта/Экспорта данных Chose a destination. В нашем случае это таблица Press. После этого нужно нажать на **...** столбца Transform.



Здесь необходимо:

1. Снять галочку Enable identity insert, определив тем самым, что данные в столбец первичного ключа будут добавляться автоматически.
2. В поле Source определить то поле, в которое нужно вставить данные (оно должно быть не <ignore>). Все ненужные, при этом, устанавливаем в <ignore>



После этого нажимаем Ok -> Далее -> Готово. После этого данные должны быть скопированы в новую таблицу.

Те же шаги проделываем для таблиц Categories и Themes. Заодно закрепляем процедуру экспортирования.

Осталось дело за малым. Нужно заполнить нужные значения в поля id\_press, id\_category, id\_theme. Для этого необходимо написать запрос.

Update books

set books.id\_press=press.id

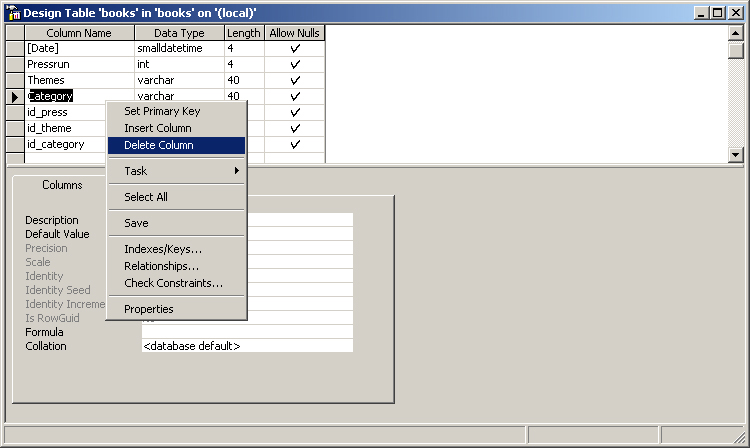
from books, press

where books.izd=press.name

Здесь указывается, что производится обновление таблицы books *(Update books)*, при этом значения в поле books.id\_press берутся из поля press.id, после чего указывается конкретно какое значение брать. Для каждой строки - это совпадение имен издательств (books.izd=press.name). Вы должны понимать на логическом уровне, что в определенной строке должен быть тот код, под которым указанное издательство находится в другой таблице.

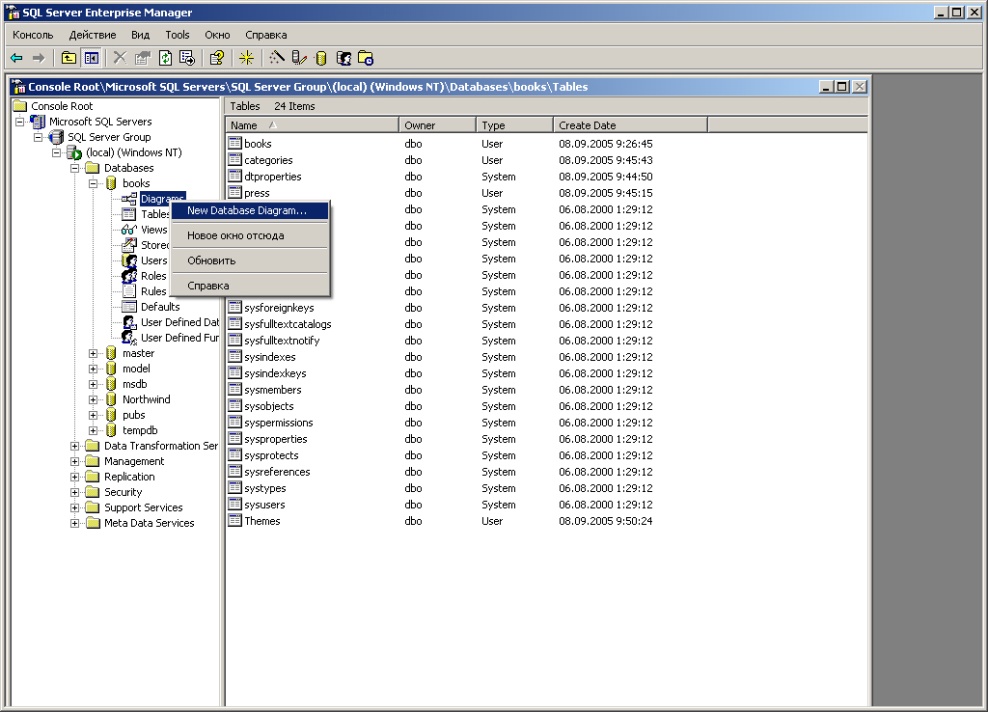
Такой вариант прописывания связей устарел. Более полное обоснование использования связей и их применения описывается в следующей главе. Пока вы должны понимать, что обращение к столбцу можно производить указав ***Имя\_таблицы.Имя\_поля***.

Такого же рода пишем запросы для Тематик и Категорий. После этого столбцы с названиями категорий, тематик и издательств можно удалять из таблицы books. Для этого выбираем в контекстном меню требуемого поля пункт меню Delete.

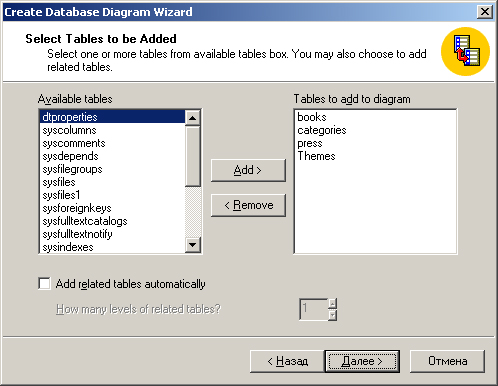


После проделанной работы получилась многотабличная база данных, что от нас и требовалось. Но, представьте, вы пришли в компанию и вам нужно разобраться с базой данных, в которой 40 таблиц, причем все друг с другом связаны и при этом как - никто не знает. Для упрощения служит построение диаграммы базы данных.

Для этого в Enterprise manager нужно выбрать в контекстном меню пункта Diagrams требуемой базы данных, New Database Diagram... .

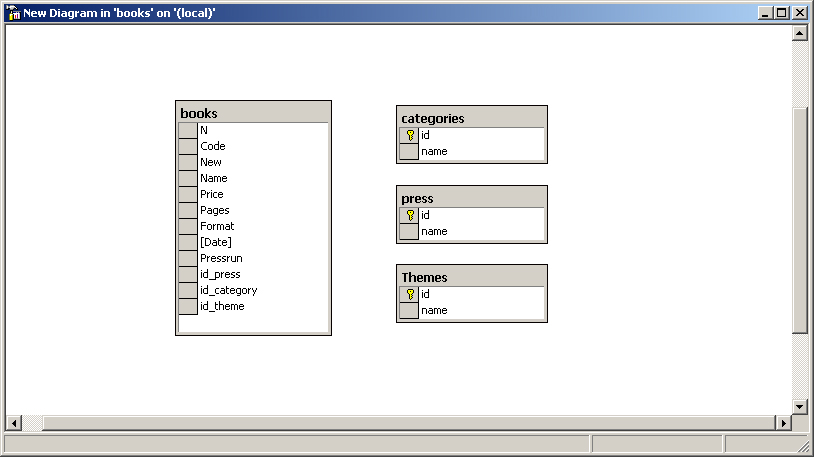


Появляется мастер создания диаграмм.

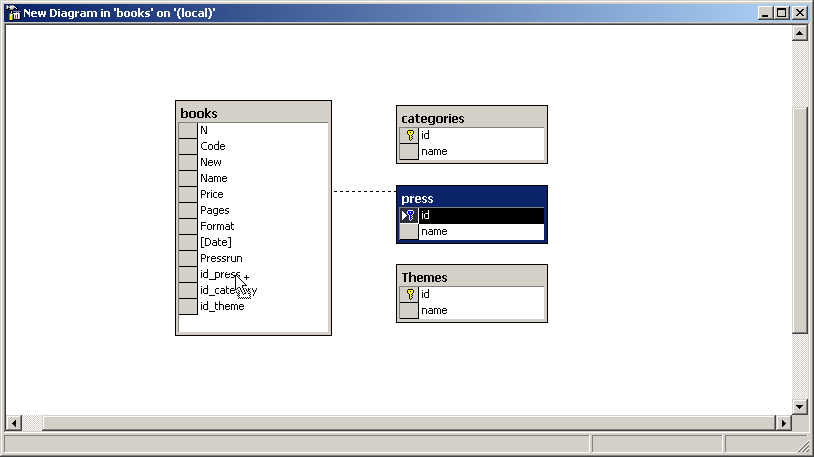


С левой колонки перемещаем требуемые таблицы в правую, при помощи кнопки Add. Remove перемещает таблицы из левой колонки в правую. Она используется, если вы случайно перетянули не нужную таблицу.

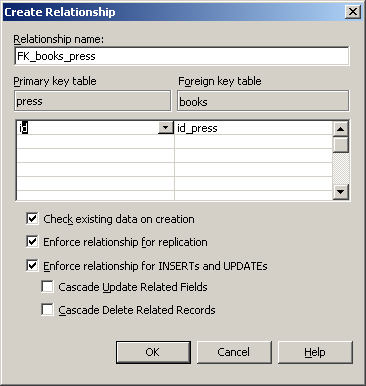
После этого нажимаем Далее -> Готово. На экране появятся выбранные таблицы.



Теперь нужно перетянуть первичный ключ одной таблицы (например Press) на внешний ключ другой (в нашем случае Books).



Появится окно - Определение связей.



В первом поле Relationship name указывается имя для создаваемой связи. Поле Primary key table - определяет таблицу, в которой первичный ключ, а поле Foreign key table - определяет таблицу, в которой находится внешний ключ. Если поля перетягивались правильно, то имена полей указываются автоматически. Если же нажать на другое поле при перетягивании, то в этом окошке вы можете это поменять. Для этого служит выпадающий список, раскрывается который при нажатии на стрелочку рядом с именем поля.

Ниже указаны настройки связи.

1. **Check existing data on creation** - определяет, что при создании текущей связи будет произведена проверка правильности, и наличия введенных данных в каждое из указанных полей.
2. **Enforce relationship for replication** - определяет, что данные связанных полей будут проверены, при проведении репликаций.
3. **Enforce relationship for INSERTs and UPDATEs** - определяет, что при добавлении, изменении либо удалении будет произведена проверка данных. Например, если значение первичного ключа присутствует в поле внешнего ключа, то обновить либо удалить эту запись в таблице, чей первичный ключ вынесен, нельзя.
   * ***Cascade Update Related Fields*** - определяет, что при обновлении значения первичного ключа связанной таблицы, значения внешнего ключа, которые совпадают, будут автоматически обновлены также.
   * ***Cascade Delete Related Records*** - определяет, что при удалении строки с таблицы, у которой вынесен первичный ключ, удаляться будут также строки основной таблицы, в которых удаляемое значение первичного и внешнего ключей совпадают.

Такие же связи необходимо указать и для двух оставшихся таблиц. После этого закрываете мастер создания диаграмм, указывая для него имя.

# Понятие Нормализации. Нормальные формы.

Рассмотрим подробней процесс проектирования базы данных. Итак, перед Вами стоит какая-то задача. Например, создать базу данных, в которой будет храниться какая-либо информация. Вы подумали... Создали базу данных. Однако, после проделанной работы необходимо еще задуматься о том, является ли созданная база данных оптимально спроектированной, а может есть какая-то избыточность (например в двух таблицах повторяется одна и та же информация) и т.д.. Как проверить базу на оптимальность? Не стоит изобретать велосипед и придумывать какие-то свои способы анализа. Нужно просто воспользоваться так называемыми нормальными формами.

### Первая нормальная форма.

Таблица находится в первой нормальной форме, если в ее ячейках хранятся атомарные значения (минимальные значения, описывающие данные)и в таблице нет повторяющихся (полностью идентичных) строк.

В принципе, любая таблица реляционной СУБД сразу находится в Первой Нормальной Форме.

Рассмотрим пример на базе данных CD, в которой хранится информация о сотрудниках и проданных ими дисках. Также там есть информация о группе, чей альбом был продан, о годе создания группы, а также о формате данных на проданном диске.

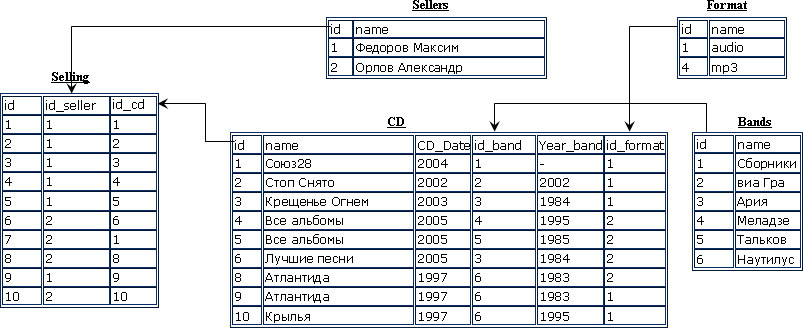
Как видите повторяющихся строк нет и в каждой ячейке хранятся атомарные значения. Соответственно эта таблица находится в первой нормальной форме.

Не сложно увидеть, что данные указаны в избытке, и поэтому можно сказать, что первой нормальной формы не достаточно для обеспечения оптимальности.

База данных находится во второй нормальной форме, если во всех существующих таблицах в полях, которые не являются частью составного первичного ключа, нет потенциальных первичных ключей (т.е. значений, которые можно вынести в отдельную таблицу, а на их место в начальной таблице подставить значения из созданной таблицы).



Т.о. необходимо разбить таблицу на несколько (произвести декомпозицию).



Теперь база данных находится во второй нормальной форме. Но давайте рассмотрим таблицу CD. В ней действительно нет полей, которые могут претендовать на то, чтобы быть вынесенными в другую таблицу, но ведь Year\_band функционально зависит не от компакт диска, а от Групп (Bands). Поэтому необходимо еще раз произвести декомпозицию таблиц.

База данных находится в Третьей Нормальной Форме, если она находится во второй нормальной форме и в полях всех таблиц находятся значения, которые зависят от первичного ключа только своей таблицы.



Как видите, информация хранится без повторений и структура базы данных исключает все аномалии, которые могут возникнуть.

Обычно этого достаточно для нормального функционирования базы данных. Однако иногда еще выделяют Четвертую и Пятую нормальные формы.

# Многотабличные запросы.

Надеюсь, вы поняли, как создавать многотабличные базы данных и понимаете их преимущества. Но, вполне вероятно, с информацией, которая в них храниться, нужно научиться работать как с одним целым.

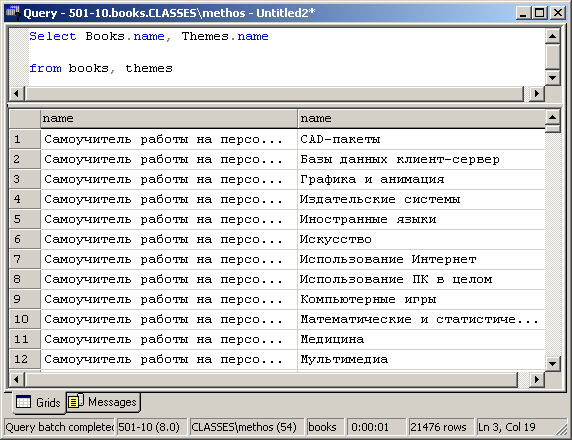
Предположим, что у вас есть 3 таблицы и в каждой из ним есть поле Name. При этом нужно вывести одновременно информацию по всем трем полям. Поймет ли SQL Server, что Name нужно брать из 3-х таблиц, да и еще в определенном порядке. Конечно же, нет. Поэтому для обращения к полям в многотабличных запросах используется обращение по полному имени: "**Имя\_таблицы.имя\_поля**"

Для начала давайте рассмотрим типичную ошибку при написании запросов. Нужно вычитать названия книг (из таблицы Books) и их тематик (таблица Themes). Теоретически этот запрос будет выглядеть следующим образом:

ОШИБКА!!!

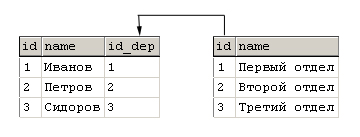
Select Books.name, Themes.name

from books, themes



На экране появятся записи, но получится, что каждая книга есть во всех тематиках. Это наглядный пример т.н. Декартового произведения двух таблиц. Это всевозможные комбинации строк первой и второй таблицы.

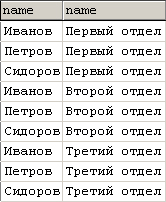
Рассмотрим Декартово произведение на меньшей базе данных.



В этой базе данных есть две таблицы. Одна содержит информацию о сотрудниках, вторая об отделах, в которых они работают. Если сделать запрос типа:

select workers.name,departments.name from workers,departments

Получится такой результат:

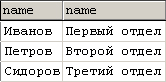


Т.е. каждый сотрудник есть в каждом отделе. Чтобы этого не было нужно указать связь между первичным ключом таблицы Departments и внешним ключом таблицы Workers. Запрос примет вид:

select workers.name,departments.name from workers,departments

where workers.id\_dep=departments.id

Вот что будет в результате:



Чтобы не допускать больше таких ошибок, давайте сразу определим набор правил.

При написании многотабличных запросов нужно:

1. К именам полей обращаться по полному имени
2. Чтобы не было Декартового произведения необходимо указывать связь между используемыми таблицами. Для этого добавляется в запрос ключевое слово Where, после чего указывается **Имя\_таблицы.Первичный\_ключ=Имя\_таблицы2.Внешний\_ключ**. И такие связи должны быть указаны для все таблиц, которые указываются в запросе.

Общий синтаксис:

Select имя\_таблицы1.имя\_поля, имя\_таблицы2.имя\_поля ... имя\_таблицыN.имя\_поля

From имя\_таблицы1, имя\_таблицы2, имя\_таблицыN

Where имя\_таблицы1.внешний\_ключ=имя\_таблицы2.первичный\_ключ And

имя\_таблицы1.внешний\_ключ=имя\_таблицыN.первичный\_ключ

And дополнительные\_условия

Связи первичного и внешнего ключа нужно определять по диаграмме. В примере они указаны для наглядности.

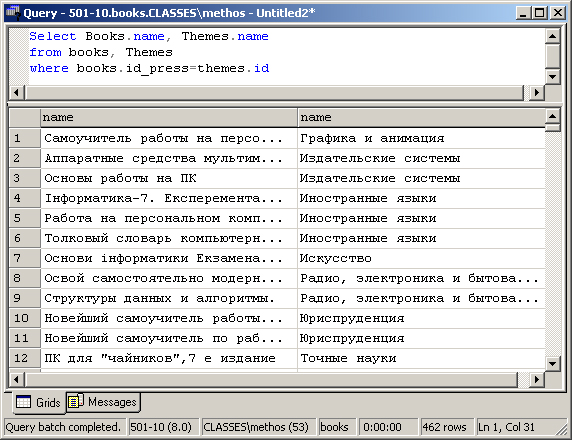
Возвращаемся к примеру. Нужно показать названия книг и название тематики.

Правильно!!!

Select Books.name, Themes.name

from books, Themes

where books.id\_press=themes.id



Пример 2.Нужно показать всю информацию о книгах издательства BHV.

Select Books.new, Books.name, Press.name, Books.price, Books.pages, Themes.name, Categories.name

from books, themes, categories, press

where books.id\_theme=themes.id and books.id\_category=categories.id and books.id\_press=press.id

and press.name like '%BHV%'

