Урок №6

Теория баз данных

Содержание

Перекрестные запросы

Перекрестные запросы являются более сложной категорией запросов на выборку. В них также используется группирование, но на этот раз двухмерное, то есть по записям (строкам) и по полям (столбцам). Другими словами — это выборка данных, которые сгруппированы по двум критериям. Как правило, такой тип запросов используется для формирования итоговых месячных, квартальных или годовых отчетов по продажам или поставкам товаров.

Данный тип запросов присущ только MS Access и отличается от обычных групповых запросов тем, что в результирующей таблице перекрестного запроса не только заголовки строк, но и заголовки столбцов ***определяются на основе значений полей, а не их названий***.

Для создания такого запроса нужно как минимум ***три элемента***:

1) поле для определения заголовков строк;
2) поле, которое определяет заголовки столбцов;
3) поле для выбора значений, с которыми будут непосредственно выполняться расчеты.

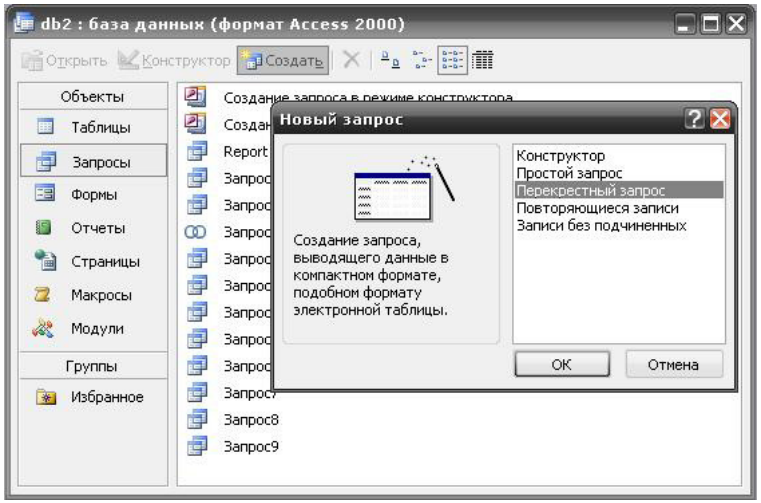
Поскольку, для понимания данного вида запроса его лучше один раз увидеть, чем бесконечно говорить, перейдем непосредственно к практике. Создадим классический перекрестный запрос квартальных продаж товаров за 2009 г., взяв за основу предварительно созданный многотабличный запрос. ***Базовый запрос*** (назовем его

Report) должен отражать общую стоимость для каждого заказанного товара и будет иметь следующий вид:

```
SELECT Product.IdProduct as [Код],
       Product.Name  as  [Товар],
       Sale.DateSale as [Дата продажи],
       Sale.Price*Sale.Quantity as [Стоимость]
FROM Product INNER JOIN Sale
ON Product.IdProduct = Sale.IdProduct
WHERE Sale.DateSale BETWEEN #01/01/2009# AND Date();
```

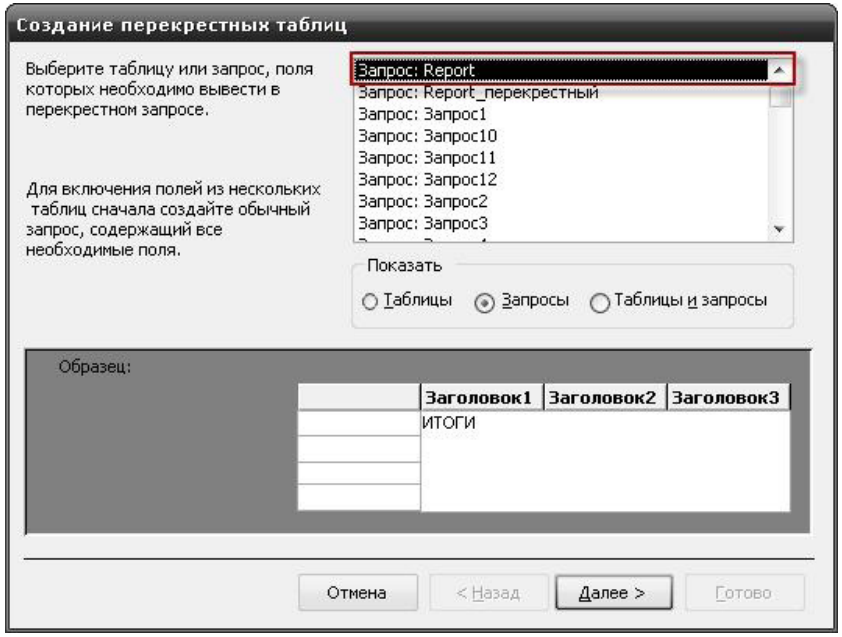
Сам перекрестный запрос имеет свои особенности синтаксиса, поэтому для лучшего понимания, создадим его сначала с помощью Визарда, а потом разберем код SQL, который его создает.

Итак, для того, чтобы создать перекрестный запрос, необходимо выбрать на панели инструментов окна базы данных кнопку «Создать» и в диалоговом окне выбора типа нового запроса выбрать пункт «Перекрестный запрос».

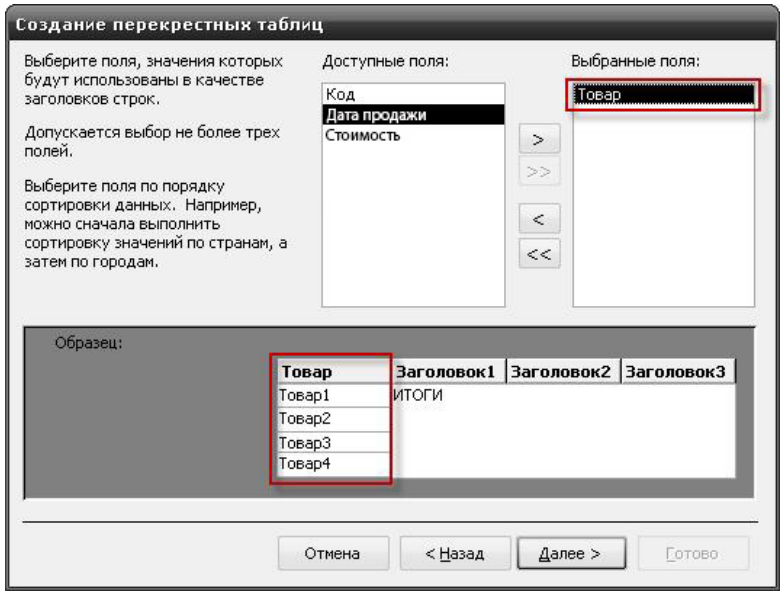


Далее визард предложит нам осуществить следующие шаги:

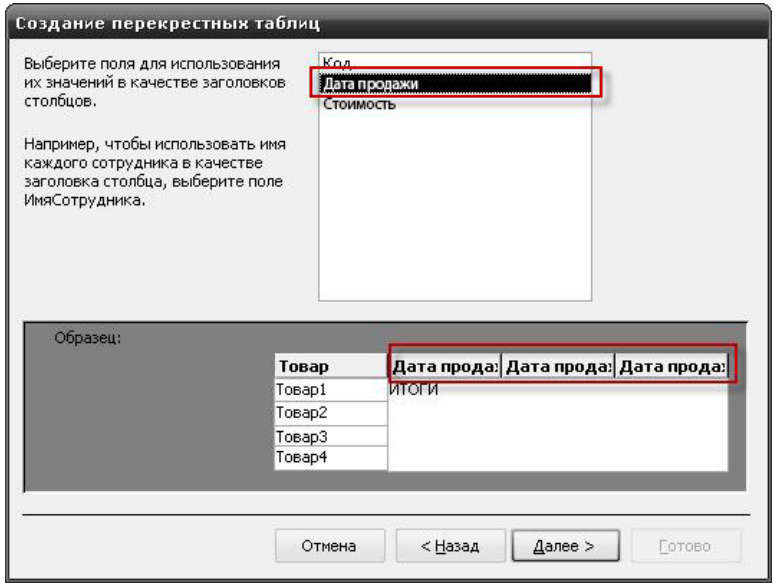
1. С самого начала нужно выбрать источник данных для перекрестного запроса. Для этого выберите в группе опций «Показать» элемент «Запросы» и из списка доступных запросов вашей базы данных выберите необходимый.



2. На следующем шаге выберите из списка «Доступные поля» те поля, значения которых будут использоваться как заголовки строк перекрестного запроса. Необходимые поля перенесите с помощью кнопок в список «Выбранные поля». В нашем случае, необходимо сформировать отчет по товарам, поэтому следует выбрать поле «Товар».

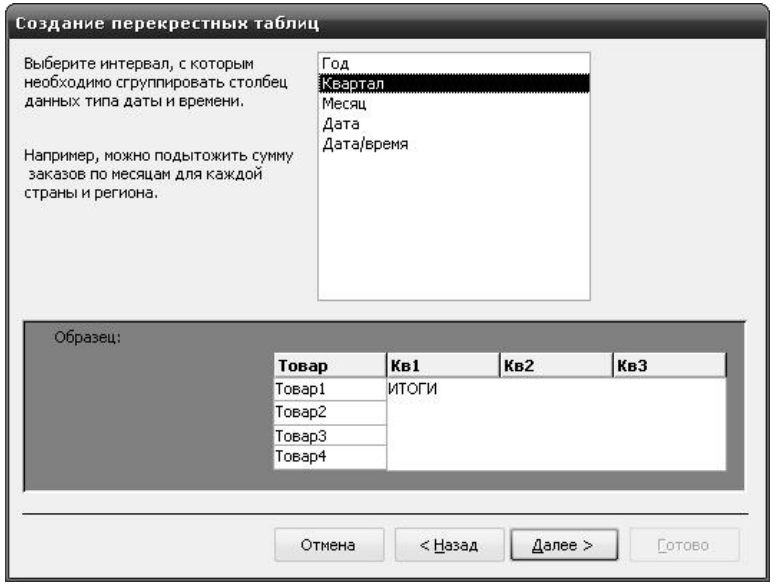


Далее необходимо выбрать поле, значение которого будет использовано в качестве заголовка колонок.

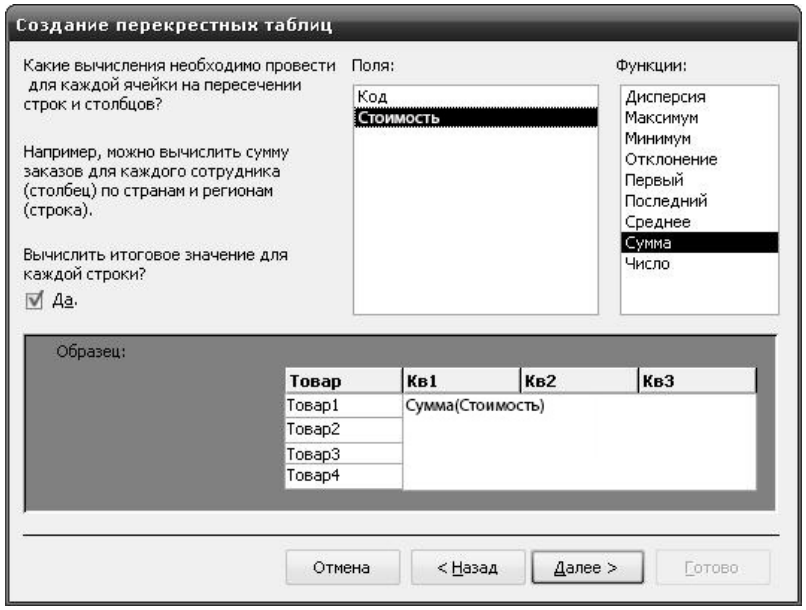


Поскольку квартальный отчет предусматривает, что подсчет продажи товаров будет осуществляться на каждую дату в пределах квартала, то следует выбрать поле «Дата продажи».

После этого нужно выбрать интервал группировки для столбцов, то есть для дат продажи. Поскольку отчет в нас квартальный, — выбираем «Квартал».



5. Последним существенным шагом является выбор итоговой операции, которую необходимо применить для обработки данных поля «Стоимость» (в нашем случае). Данная операция фактически будет отражать ту информацию, которая нас интересует: общую сумму продаж в разрезе квартала, среднюю стоимость и тому подобное. Нас интересует сумма, поэтому в списке «Функции» следует выбрать соответствующий пункт.



Вот и все. Результат будет иметь следующий вид:

Из результата видно, в поле «Товар» отображается список товаров проданых в течение 2009 года. Поле «Итоговое значение Стоимость» (его можно потом переименовать для удобства) содержит данные о том, на какую сумму за этот период каждого товара было продано, а в соседних полях («кв1», «кв2», «кв3») эта информация показывается в разрезе кварталов.

Создание перекрестного запроса с помощью Визард, кроме основного своей преимущества, — наглядности и удобства, имеет более существенные ***недостатки***. К ним следует ***отнести***:

- работать можно только с одним отчетом или таблицей;
- нельзя сортировать результирующую таблицу по значениям, которые содержаться в полях, поскольку в большинстве случаев одновременное упорядочение данных в полях по всем записям невозможно. Но вы можете задать сортировку заголовков записей (строк);
- в процессе создания запроса нельзя указывать дополнительные условия отбора.

В режиме SQL данный запрос иметь следующий вид:

```
TRANSFORM Sum(Report.Стоимость)
        AS [Sum-Стоимость]
SELECT Report.Товар, Sum(Report. Стоимость)
        AS [Итоговое значение Стоимость]
FROM Report
GROUP BY Report.Товар
PIVOT "Кв" & Format([Дата продажи], "q");
```

Обратите внимание на формирование самого запроса и использования ключевых слов **TRANSFORM** и **PIVOT**, которые, кстати, не являются зарезервированными в стандарте ANSI SQL. Операция **TRANSFORM** в данном запросе позволяет определить данные, которые содержатся в результирующей таблице, а в операции **PIVOT** задаются заголовки столбцов.

Полный синтаксис оператора **TRANSFORM**:

```
TRANSFORM функция_агрегирования
SELECT выборка
PIVOT поля_для_заголовков [IN (значение1 [, значение2 [,...]])];
```

Функция агрегирования у оператора **TRANSFORM** позволяет обработать отобранные данные. Оператор **SELECT**, который указывается следующим, указывает поля, которые будут использоваться в качестве названий записей (строк). Он ***обязательно*** должен содержать выражение **GROUP BY**, который указывает на способ группировки записей. Кроме выражения **GROUP BY**, оператор **SELECT** полноценно может содержать и другие конструкции (**WHERE**, **ORDER BY** и т.д.) и подзапросы.

У оператора **PIVOT**, как уже было сказано, указываются имена полей или выражения, которые будут использоваться в качестве заголовков столбцов в результирующей таблице. Например, если указать названия месяцев, то результирующий запрос будет содержать двенадцать столбцов. Этот список полей можно ограничить, создав заголовки из набора констант, которые перечисляются в выражении **IN**. Другими словами, значения, которые не входят в набор, будут отфильтрованы.

Подытоживая все вышесказанное, можно написать перекрестный запрос на формирование квартального отчета по продажам товаров, без использования промежуточного запроса. При этом, можно отсортировать нашу выборку.

```
TRANSFORM Sum(Sale.Price*Sale.Quantity) as [Стоимость]
SELECT Product.Name as [Товар],
```



```
    Sum(Sale.Price*Sale.Quantity)
    as [Итоговое значение Стоимость]
FROM Product INNER JOIN Sale
ON Product.IdProduct = Sale.IdProduct
WHERE Sale.DateSale BETWEEN #01/01/2009# AND
Date()
GROUP BY Product.Name
ORDER BY Product.Name
PIVOT "Кв" & Format(Sale.DateSale,"q");
```

Запросы с параметрами

Бывают случаи, когда значения критериев запроса неизвестны заранее, то есть не являются константными. Например, необходимо выводить список поставленных или проданных товаров на определенную дату, но эта дата каждый раз при формировании отчета другая. В таком случае, мы можем постоянно писать новый запрос, что недостаточно удобно, или же создать один ***параметризованный запрос***, который будет спрашивать пользователя значение необходимого критерия (-ев), в зависимости от которого (ых) будет выведен результат.

Итак, ***параметризованный запрос*** или ***запрос с параметрами*** — это специальный интерактивный тип запроса, который перед выполнением выводит диалоговое окно с просьбой ввести один или несколько параметров, необходимых для его работы. Эти параметры фактически позволяют пользователю накладывать условия отбора записей.

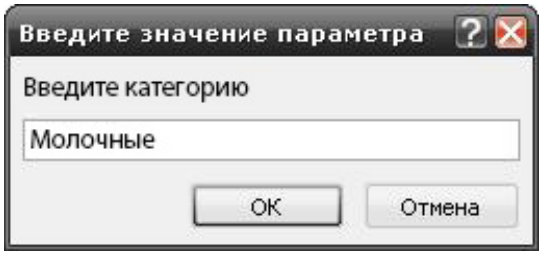
Например, напишем запрос, который выводит на экран список товаров, необходимой пользователю категории:

```
SELECT Product.Name as [Товар], Category.Name as [Категория]
FROM Category INNER JOIN Product
ON Category.IdCategory = Product.IdCategory
WHERE Category.Name=[Введите категорию];
```

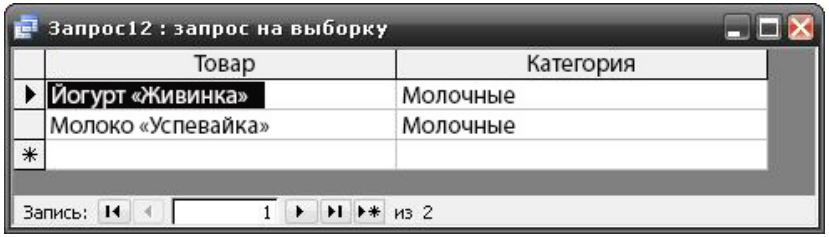
Обратите внимание на условие: Category.Name = [Введите категорию]. Для того, чтобы указать, что значение

критерия отбора будет получено от пользователя, в конструкции **WHERE** вместо конкретного значения указываются квадратные скобки. Если внутри них указать текст, то он будет подсказкой пользователю на то, что необходимо ввести.

Итак, после запуска данного запроса, перед вами появится диалоговое окно с просьбой ввести категорию, список товаров которой необходимо вывести на экран.



После нажатия на кнопку OK, будет сформирован список товаров введенной категории, то есть молочные продукты.



Еще один пример. Необходимо отобрать товары, которые были поставлены в промежутке определенных дат.

```
SELECT Product.Name as [Товар],
       Supplier.Name as [Поставщик],
       Delivery.DateDelivery as [Дата поставки]
```

```
FROM Supplier INNER JOIN
        (Product INNER JOIN Delivery
        ON Product.IdProduct = Delivery.IdProduct)
        ON Supplier.IdSupplier = Delivery.IdSupplier
WHERE Delivery.DateDelivery
        BETWEEN [Начальная дата]
        AND [Конечная дата];
```

Следует отметить, что параметризованные запросы такого вида присущи только СУБД MS Access. В большинстве других СУБД этот вид запросов представлен отдельным объектам базы данных, такими как хранимые процедуры и функции.

Запросы на создание таблицы.

(Конструкция TOP)

В результате работы данных запросов, осуществляется выборка данных из одной или нескольких таблиц, и ее результат помещается в новую таблицу. Как правило, запросы на создание новой таблицы используются для создания архивных копий таблиц, копий экспорта в другую базу данных или как источник данных для отчета. Например, отчет о ежемесячных продажах по регионам можно сделать на основе одного и того же запроса.

Для создания такого вида запросов, в синтаксисе оператора **SELECT** существует опция **INTO**. Данная опция указывается после списка **SELECT**, но перед указанием списка таблиц, из которых будет получена информация (конструкции **FROM**):

```
SELECT [ALL | DISTINCT] { * | поле_для_вибірки [, поле_
для_выборки] }
[INTO имя_новой_таблицы [IN внешняя_база_данных] ]
FROM { таблица | представление} [as] [псевдоним]
[, {таблица | представление} [as] [псевдоним] ]

[WHERE условие]
[GROUP BY [ALL] выражение_группирования]
[HAVING условие_на_группу]
[ORDER BY имя_поля | номер_поля [ ASC | DESC ] ];
```

Необязательно опция **IN** позволяет указать базу данных, в которую будет помещена новосозданная таблица. По умолчанию, такой базой является текущая, то есть новая таблица будет сохранена в нашей базе данных.

Структура новой таблицы будет соответствовать структуре списка **SELECT**, и поэтому во избежание проблем по идентификации полей вновь таблицы желательно указывать псевдонимы. Кроме того, стоит отметить, что при создании новой таблицы ее поля наследуют только тип и размер источника, другие же свойства (индексы, первичные ключи и т.д.) нужно создавать самостоятельно.

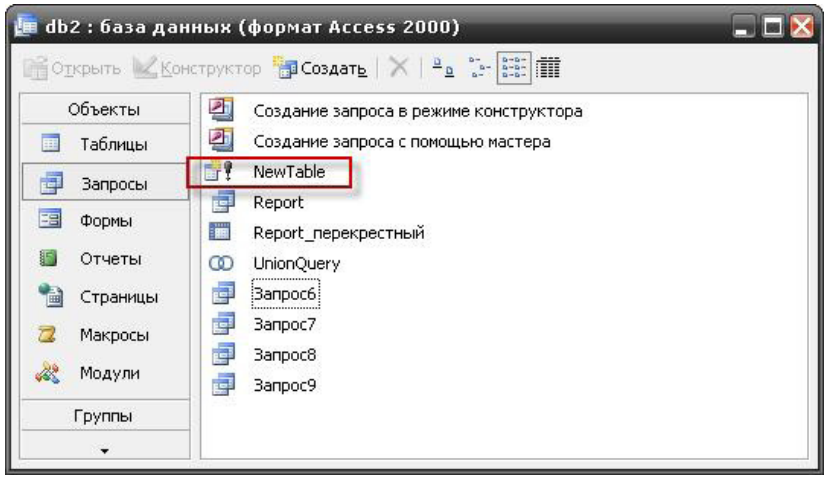
Итак, создадим запрос с целью создания ***новой таблицы***, содержащей информацию о названиях товаров, их цены и категории

```
SELECT Product.Name as [NameProduct],
       Format(Product.Price, '## ###.00 грн.') as [Price],
       Category.Name as [Category]
INTO ProductEx
FROM Product INNER JOIN Category
ON Product.IdCategory = Category.IdCategory;
```

В результате работы данного запроса образуется новая таблица «ProductEx», содержащая вышеуказанные данные (название товара, его цену и категорию). Имена полей новосозданной таблицы носят названия псевдонимов полей при написании запроса, то есть ***NameProduct***, ***Price*** и ***Category***.

В новые таблицы ProductEx вы можете написать произвольный запрос и изменения значений в ее записях не влияют на таблицы Product и Category.

Кстати, обратите внимание на пиктограмму нашего запроса в списке запросов базы данных. Она приобрела вид таблички с восклицательным знаком, указывая на то, что это требование не на выборку, а на создание новой таблицы.



Запросы на создание таблицы поддерживаются не всеми СУБД, причем каждая из тех, которые такие запросы поддерживает, осуществляет его различными способами. Поэтому, перед тем как написать запрос на

создание таблицы, перечитайте документацию по SQL для необходимой СУБД.

Следующая и последняя конструкция, которую стоит рассмотреть в синтаксисе оператора **SELECT** — это **TOP**. Она позволяет выбрать первые n-записей результирующей выборки. С данной опцией синтаксис оператора **SELECT** приобретает следующий вид:

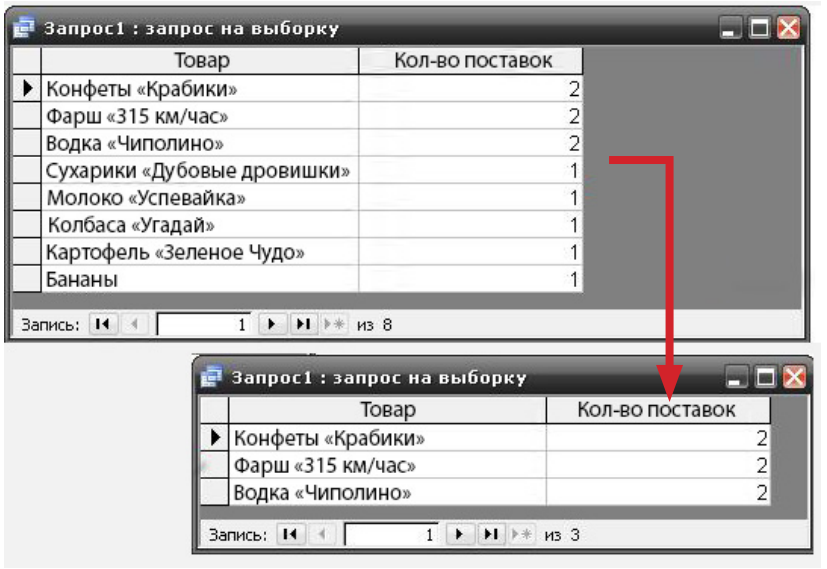
```
SELECT [ALL | DISTINCT]
        [TOP n] {* | поле_для_выборки [, поле_для_виборки]}
[INTO имя_новой_таблицы
[IN внешняя_база_данных]]
FROM {таблица | представление} [as] [псевдоним]
[, {Таблица | представление} [as] [псевдоним]]

[WHERE условие]
[GROUP BY [ALL] выражение_группировки]
[HAVING условие_на_группу]
[ORDER BY имя_поля | номер_поля [ASC | DESC]];
```

Например, напишем запрос, который будет содержать информацию о трех наиболее часто поставляемых товарах, то есть товарах, заказов которых было совершено наибольшее количество раз.

```
SELECT TOP 3 Product.Name AS [Товар],
        Count(Delivery.IdProduct)
        AS [Количество поставок]
FROM Product INNER JOIN Delivery
ON Product.IdProduct = Delivery.IdProduct
GROUP BY Product.Name
ORDER BY 2 DESC;
```


Для получения такой информации, сначала создается запрос на выборку, в котором подсчитывается, сколько раз был поставлен каждый товар. После этого результирующий запрос сортируется и на экран выводится первых (верхних) три записи (TOP 3).



Кстати, результат наших действий можно сохранить в отдельную таблицу, которая будет содержать отчетную информацию о троице чаще всего поставляемых товаров.

```
SELECT TOP 3 Product.Name AS [Товар],
        Count(Delivery.IdProduct) AS [Количество поставок]
INTO BestDelivery
FROM Product INNER JOIN Delivery
ON Product.IdProduct = Delivery.IdProduct
GROUP BY Product.Name
ORDER BY 2 DESC;
```

Модификация данных средствами DML.

(Запросы на добавление, удаление и обновление)

Переходим к следующей категории команд, а именно **DML** (*Data Manipulation Language — Язык Манипулирования Данными*). Команды данной категории позволяют манипулировать данными объектов базы данных. В нее входят операторы:

1. Вставки записей — **INSERT**;
2. Удаление записей — **DELETE**;
3. Изменения значений в существующих записях таблицы — **UPDATE**.

Рассмотрим, как работать с данными операторами.

Оператор INSERT

Оператор SQL **INSERT** используется для добавления записей в таблице. Различные варианты оператора **INSERT** позволяют добавлять в таблицу как один, так и множество записей. Это можно осуществлять как путем введения константных значений, так и методом считывания данных из другой таблицы. В каждом из перечисленных случаев ***необходимо учитывать структуру таблицы***:

- Количество полей;

- Тип данных каждого поля;
- Имена полей, в которые вносятся данные;
- Ограничение и свойства полей.

Синтаксис оператора **INSERT** следующий:

```
INSERT [INTO] Название таблицы [(поле1 [, поле2 [...]])]
VALUES (DEFAULT | 'значение 1 "[," значение "[...]])
```

Согласно данному синтаксису в указанную таблицу вставляется запись со значениями полей, указанными в перечне фразы VALUES (значения), причем 1-е значение соответствует 1-му полю в списке полей (поля, не указанные в списке, заполняются NULL-значениями). Если в списке VALUES указаны все поля модификуемои таблицы и порядок их перечня соответствует порядку полей в описании таблицы, то список полей при **INTO** можно проигнорировать.

Итак, добавим в список категорий новую категорию товаров:

```
INSERT INTO Category
VALUES (11, 'Рыбные продукты');
```

Таким образом, значения вставляются во все поля таблицы Category в соответствии с их физическим порядком. Если нужно вставить значение только в некоторые поля, тогда их нужно явно указывать.

Например, идентификатор товара указывать неуместно, он должен генерироваться автоматически, следует указывать только название категории. В таком случае запрос на добавление записей следует переписать:

```
INSERT INTO Category (Name)
VALUES ('Соки и воды');
```

А что делать, когда таблица имеет внешний ключ? Например, к производителям продукции нужно добавить данные о новом производителе. Таблица «Producer» имеет два основных поля: Name (название) и IdAddress (внешний ключ, который содержит ссылки на полный адрес производителя). Запрос на вставку данных будет иметь следующий вид:

```
INSERT INTO Producer (Name, IdAddress)
VALUES ('ЧП Постовалов И.В.', 3);
```

Для добавления нескольких записей путем выборки данных из другой таблицы используется модифицированный синтаксис оператора **INSERT**:

```
INSERT [INTO] название_таблицы
        [IN внешняя_база_даных] [( поле1 [, поле2 [,...]] )]
SELECT ( поле1 [, поле2 [,...]] )
FROM список_таблиц;
```

Выражение **IN** позволяет указать название внешней базы данных, в указанную таблицу которой будут вставлены новые данные. То, какие именно данные будут вставлены, определяется оператором **SELECT**.

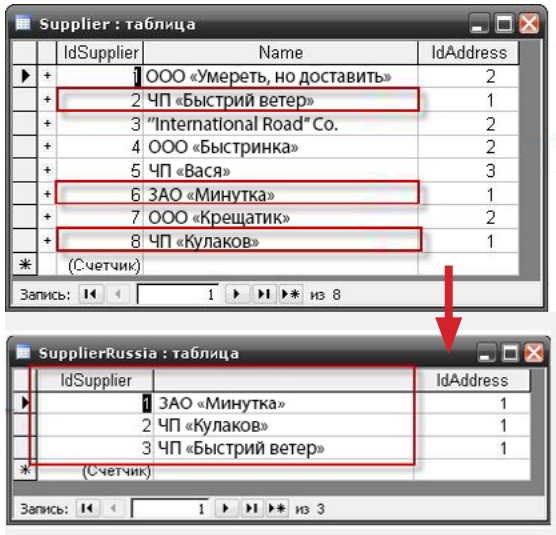
Допустим, у нас существует таблица SupplierArchive, которая содержит копию данных о поставщиках.

```
INSERT INTO SupplierArchive
SELECT *
FROM Supplier;
```

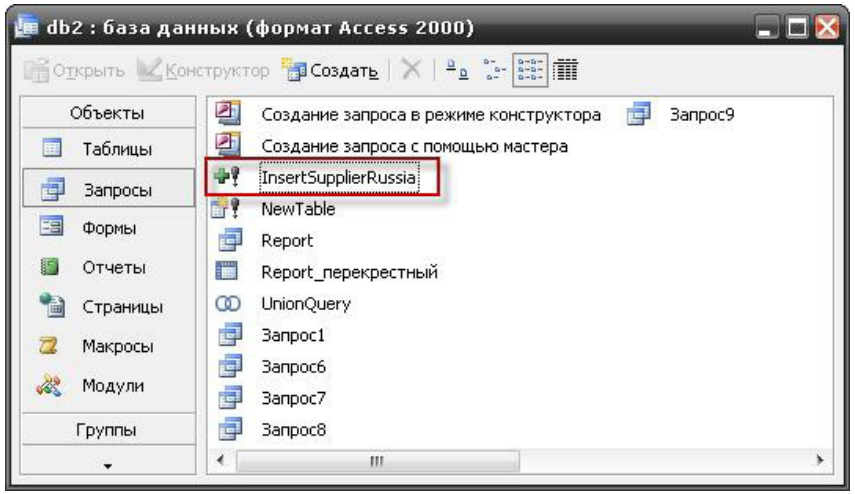

Если необходимо наложить условие на вставку данных, например, в новую таблицу SupplierRussia нужно вставить данные обо всех поставщиках из России, тогда запрос на встаки перепишется:

```
INSERT INTO SupplierRussia (Name, IdAddress)
SELECT Supplier.Name, Supplier.IdAddress
FROM Country INNER JOIN (Address INNER JOIN Supplier
                ON Address.IdAddress = Supplier.IdAddress)
ON Country.IdCountry = Address.IdCountry
WHERE Country.Name='Россия';
```

Список полей при **INTO** и **SELECT** в данном случае является обязательным, поскольку отобранные данные о поставщиках могут иметь неупорядоченные идентификаторы (например, 5 и 10), а новая таблица может иметь свои значения порядковых номеров для новых данных, а также требует, чтобы они шли по порядку.



Пиктограмма запросов на вставку данных будет отличаться от запросов на выборку и создание новой таблицы и будет иметь значок «+» с восклицательным знаком:



Оператор DELETE

С помощью оператора **DELETE** из таблицы можно удалить любое количество полей. Разрешается накладывать условие, критерий на удаляемые данные. Поскольку оператор **DELETE** удаляет запись полностью и данную операцию отменить невозможно (только через восстановление из резервной копии), следует проверять перед удалением условие, чтобы быть уверенным в верности удаления данных.

При удалении записей, существует один нюанс. Если удаляется запись из таблицы, которая входит в отношение 1: N (один-ко-многим) и со стороны 1 разрешено каскадное удаление, то записи, относящиеся к ней со

стороны N, также будут удалены. Например, если такое отношение вставленно для таблиц Customers и Orders, то удаление записи о клиенте автоматически вызовет удаление записи о заказах.

Синтаксис оператора DELETE следующий:

```
DELETE
FROM название_таблицы
[WHERE условие];
```

Если условие не указано, тогда будут удалены все записи указанной таблицы, соответствующей его полному очищению. Иначе удалению будут подлежать только те записи таблицы, которые удовлетворяют критерий.

Например, нужно удалить содержимое таблицы Category, то есть удалить все категории товаров:

```
DELETE
FROM Category;
```

После этого таблица Category будет пустой.

Следующий пример. Удалить из базы данных производителя «MicroChips» Ltd.

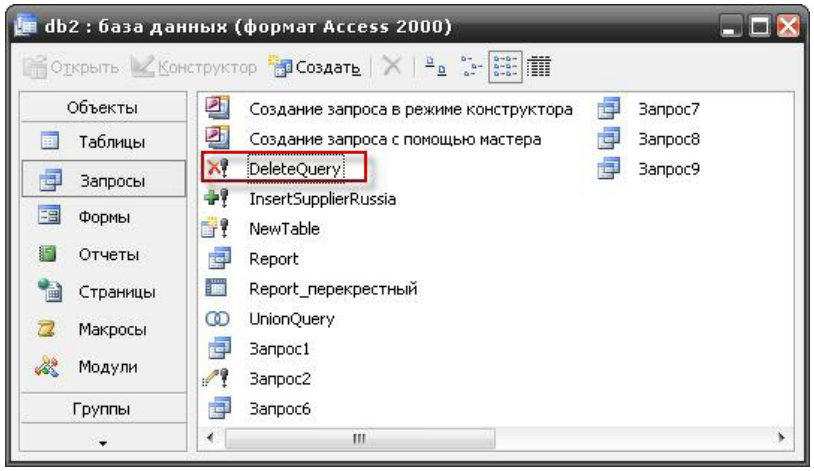
```
DELETE
FROM Producer
WHERE Name='"MicroChips" Ltd.';
```

Если при этом необходимо удалить все товары, производителем которых является «MicroChips» Ltd., тогда без подзапроса не обойтись:

```
DELETE
FROM Product
```

```
WHERE IdProducer =
                    (SELECT IdProducer
                     FROM Producer
                     WHERE Name='"MicroChips" Ltd.');
```

Вот и все. Пиктограмма запроса на удаление также будет отличаться от других запросов и будет иметь вид крестика с восклицательным знаком.



Оператор UPDATE

Третий оператор — это оператор обновления данных **UPDATE**. Он позволяет изменить значение поля (-ей) в указанной таблице, как правило, согласно указанному критерию. Данный оператор особенно полезен при изменении большого количества записей или записей в нескольких таблицах. Подобно оператору **DELETE**, операцию изменения нельзя отменить, поэтому перед ее запуском необходимо убедиться в верности построения критерия отбора.

Синтаксис оператора **UPDATE**:


```
UPDATE название_таблицы
SET название_поля = значения [,...]
[WHERE условие];
```

Инструкция **SET** определяет, какие поля должны быть изменены и на какие именно значения. Аналогично оператору **DELETE**, конструкция **WHERE** является необязательной и позволяет указать условие отбора записей, значение которых будет модифицированно. По умолчанию меняются все данные указанной таблицы.

Например, заменить название поставщика с кодом 3 на «Inter Ltd.»:

```
UPDATE Supplier
SET Name = 'Inter Ltd.'
WHERE IdSupplier = 3;
```

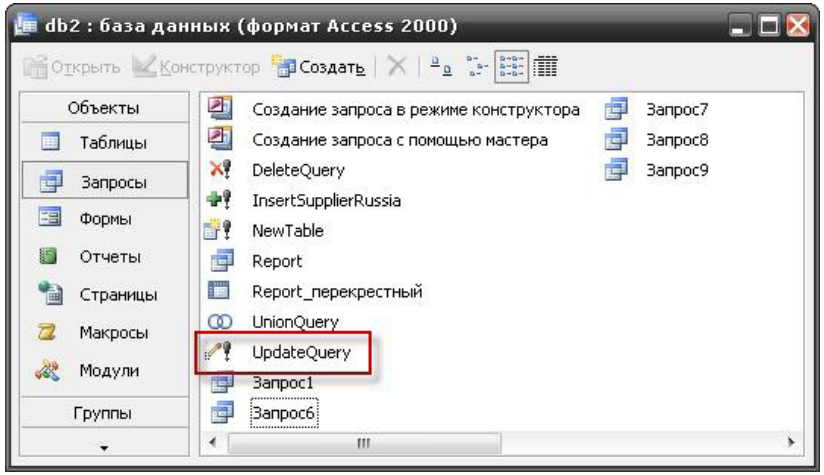
Или же нужно установить цену поставки равной 20 грн. для всех товаров, поставлялись поставщиками «ЧП Вася» и «ОАО «Быстрый ветер»:

```
UPDATE Delivery
SET Price = 20
WHERE IdSupplier IN
          (SELECT IdSupplier
           FROM Supplier
           WHERE Name IN ('ЧП Вася', ' ОАО "Бытрый ветер"'));
```

Изменить информацию о поставке йогурта «Живинка», которая была осуществлена 12/07/2009 следующим образом: цену поставки увеличить на 10%, а количество увеличить на 100 ед.

```
UPDATE Delivery
SET Price = Price * 0.2,
      Quantity = Quantity + 100
WHERE DateDelivery = #12/07/2009# AND IdProduct IN
                        (SELECT IdProduct FROM Product
                         WHERE Name='Йогурт "Живинка"');
```

Пиктограмма запросов на обновление примет вид карандаша с восклицательным знаком, отличая его от других запросов:



Домашнее задание

1. Сделать запрос на создание таблицы, содержащей информацию о 5 самых продаваемых товарах.
2. Написать параметризованный запрос, который возвращает 2 самых дорогих товары данной категории.
3. Написать запрос, который позволит увеличить дату поставки каждого товара, которая соответствует шаблону на 2 дня. Шаблон на товар задается пользователем, то есть передается в качестве параметра в запрос.
4. Написать запрос, который позволяет просмотреть все товары определенного производителя и необходимой категории, при этом данные о производителе и категорию указываются при вызове
5. Средствами SQL сделать вставку новых данных в таблицу Product и Country.
6. Создать таблицу Delivery2008, которая имеет структуру, аналогичную таблице Delivery. Данная таблица должна содержать информацию обо всех поставках, которые были осуществлены в течение 2008 года. С целью автоматизации процесса заполнения таблицы Delivery2008, написать необходимый запрос на вставку.
7. Уменьшить цены на товары, которые поставлялись двумя поставщиками (например, "Петров" и "Сидоров") на 10% и тип наценки при этом изменить на оптовую.
8. Увеличить на 20% цены продажи всех товаров,

производитель которых, например, "Бондарчук" и которые поставлялись определенным поставщиком (например, "Воробей").

9. Проставить дату поставки равной сегодняшней для всех товаров, в которых такая информация отсутствует.
10. Удалить из текущей базы данных все товары, имеющееся количество которых меньше 100 ед. (кг, шт. и т.п.), а цена не превышает 10 грн. за ед.
11. Удалить все товары кондитерской промышленности, продажа которых с начала года составляет более 1000 кг.
12. Удалить все товары, в которых цена больше, чем средняя.
13. Средствами SQL создать месячный отчет, который содержит информацию о том, от каких поставщиков поставку товаров текущего года в разрезе категорий. Примечание! Привязки к конкретным датам быть не должно, то есть текущий год определяется программно.
14. Средствами SQL создать годовой отчет о средних ценах продажи товаров определенной категории. Категория, по которой необходимо формировать отчет, задается пользователем каждый раз перед его созданием.