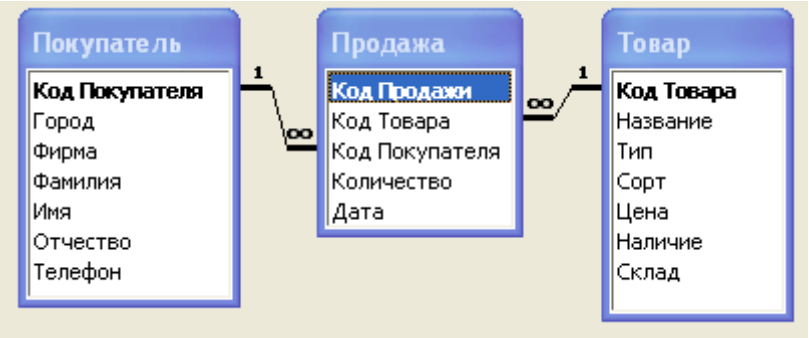
О языке SQL

Язык SQL предназначен для доступа к информации и управления реляционной базой данных. Управление различными реляционными базами данных осуществляют программы, называемые СУБД - системы управления базами данных (DBMS - DataBase Management System). Сама реляционная база данных представляет собой хранилище определенным образом организованной информации и СУБД. Однако на практике термин СУБД часто заменяют термином БД (База Данных). Для того чтобы с различными базами данных - такими как Oracle, Microsoft SQL Server, Informix, DB2, Access, MySQL - можно было общаться на одном языке, был разработан язык SQL.
Начиная с 1986 года, комитеты ISO (International Organization for Standardization) и ANSI (American National Standards Institute) приступили к созданию ряда стандартов языка SQL, которые впоследствии были приняты и получили следующие названия: SQL86, SQL89, *SQL92* и *SQL99*

Реализация SQL в конкретной СУБД может отличаться, хотя и очень незначительно, от стандарта.
Все приведенные ниже примеры можно посмотреть в базе данных формата Access2000: SQLпримеры.mdb



Предложение SELECT

Оператор `SELECT` – один из наиболее важных и самых распространенных операторов SQL. Он позволяет производить *выборки* данных из таблиц и преобразовывать к нужному виду полученные результаты. При его помощи можно реализовать сложные и громоздкие условия отбора данных из различных таблиц.

Оператор `SELECT` имеет следующий формат:

```
SELECT [ALL | DISTINCT ] {*|[имя_столбца
  [AS   новое_имя]]} [,...n]
  FROM  имя_таблицы [[AS] псевдоним] [,...n]
  [WHERE  <условие_поиска>]
  [GROUP BY  имя_столбца [,...n]]
  [HAVING <критерии выбора групп>]
  [ORDER BY имя_столбца [,...n]]
```

Оператор `SELECT` определяет поля (столбцы), которые будут входить в *результат выполнения запроса*. В списке они разделяются запятыми и приводятся в такой очередности, в какой должны быть представлены в *результате запроса*. **Если используется имя поля, содержащее пробелы или разделители, его следует заключить в квадратные скобки.** Символ `*` используется для выбора всех полей..

Если обрабатывается несколько таблиц, то **для различия одноименных полей разных таблиц в списке полей используется полная спецификация поля, т.е.** **`Имя_таблицы.Имя_поля`**.

Предложение FROM

Предложение `FROM` задает имена таблиц, которые содержат поля, перечисленные в операторе `SELECT`. Необязательный параметр псевдоним – это сокращение, устанавливаемое для имени таблицы.

Обработка элементов оператора `SELECT` выполняется в следующей последовательности:

- `FROM` – определяются имена используемых таблиц;
- `WHERE` – выполняется *фильтрация строк* объекта в соответствии с заданными условиями;
- `GROUP BY` – образуются *группы строк* , имеющих одно и то же значение в указанном столбце;
- `HAVING` – фильтруются группы строк объекта в соответствии с указанным условием;
- `SELECT` – устанавливается, какие столбцы должны присутствовать в выходных данных;
- `ORDER BY` – определяется упорядоченность результатов выполнения операторов.

Порядок предложений и фраз в операторе `SELECT` не может быть изменен. Только два предложения `SELECT` и `FROM` являются обязательными, все остальные могут быть опущены.

Пример №1. Составить список данных о всех Покупателях.

```
SELECT * FROM Покупатель
```

Параметр `WHERE` определяет критерий отбора записей из входного набора. Но в таблице могут присутствовать повторяющиеся записи (дубликаты). Предикат `ALL` задает включение в выходной набор всех дубликатов, отобранных по критерию `WHERE`. Нет необходимости указывать `ALL` явно, поскольку это значение действует по умолчанию.

Пример №2. Составить список всех фирм.

```
SELECT ALL Покупатель.Фирма FROM Покупатель

Или

SELECT Покупатель.Фирма FROM Покупатель
```

Результат выполнения запроса может содержать *дублирующиеся значения*, поскольку оператор `SELECT` не исключает повторяющихся значений при выполнении *выборки данных*.

Предикат `DISTINCT` следует применять в тех случаях, когда требуется отбросить блоки данных, содержащие *дублирующие записи* в выбранных полях. Значения для каждого из приведенных в инструкции `SELECT` полей должны быть уникальными, чтобы содержащая их запись смогла войти в выходной набор.

Причиной ограничения в применении `DISTINCT` является то обстоятельство, что его использование может резко замедлить выполнение запросов.

Откорректированный **Пример №2** выглядит следующим образом:

```
SELECT DISTINCT Покупатель.Фирма FROM Покупатель
```

Предложение WHERE

За ключевым словом `WHERE` следует перечень *условий поиска*, определяющих те строки, которые должны быть выбраны при выполнении запроса. Существует пять основных типов *условий поиска* (или предикатов):

- *Сравнение*: сравниваются результаты вычисления одного выражения с результатами вычисления другого.
- *Диапазон*: проверяется, попадает ли результат вычисления выражения в заданный *диапазон* значений.
- *Принадлежность множеству*: проверяется, принадлежит ли результат вычислений выражения заданному множеству значений.
- *Соответствие шаблону*: проверяется, отвечает ли некоторое строковое значение заданному шаблону.
- *Значение* `NULL`: проверяется, содержит ли данный столбец определитель `NULL` (неизвестное значение).

Сравнение

В языке SQL можно использовать следующие операторы *сравнения*: `=` – равенство; `<` – меньше; `>` – больше; `<=` – меньше или равно; `>=` – больше или равно; `<>` – не равно.

Пример №3. Показать все операции отпуска товаров объемом больше 100.

```
SELECT * FROM Продажа
WHERE Количество>100
```

Более сложные предикаты могут быть построены с помощью логических операторов `AND`, `OR` или `NOT`, а также скобок, используемых для определения порядка вычисления выражения. Вычисление выражения в *условиях* выполняется по следующим правилам:

- Выражение вычисляется слева направо.
- Первыми вычисляются подвыражения в скобках.
- Операторы `NOT` выполняются до выполнения операторов `AND` и `OR`.
- Операторы `AND` выполняются до выполнения операторов `OR`.

Для устранения любой возможной неоднозначности рекомендуется использовать скобки.

Пример №4. Вывести список товаров, цена которых больше или равна 100 и меньше или равна 150.

```
SELECT Название, Цена
FROM Товар
WHERE Цена>=100 And Цена<=150
```

Пример №5. Вывести список Покупателей из Москвы или из Самары.

```
SELECT Фамилия, Город
FROM Покупатель
WHERE Город="Москва" Or Город="Самара"
```

Диапазон

Оператор `BETWEEN` используется для поиска значения внутри некоторого интервала, определяемого своими минимальным и максимальным значениями. При этом указанные значения включаются в *условие поиска*.

Пример №6. Вывести список товаров, цена которых лежит в диапазоне от 100 до 150 (запрос эквивалентен **Примеру №4**).

```
SELECT Название, Цена
FROM Товар
WHERE Цена Between 100 And 150
```

При использовании отрицания `NOT BETWEEN` требуется, чтобы проверяемое значение лежало вне границ заданного *диапазона*.

Пример №7. Вывести список товаров, цена которых не лежит в диапазоне от 100 до 150.

```
SELECT Товар.Название, Товар.Цена
FROM Товар
WHERE Товар.Цена Not Between 100 And 150
```

Или

```
SELECT Товар.Название, Товар.Цена
FROM Товар
WHERE (Товар.Цена<100) OR (Товар.Цена>150)
```

Принадлежность множеству

Оператор `IN` используется для *сравнения* некоторого значения со списком заданных значений, при этом проверяется, соответствует ли результат вычисления выражения одному из значений в предоставленном списке. При помощи оператора `IN` может быть достигнут тот же результат, что и в случае применения оператора `OR`, однако оператор `IN` выполняется быстрее.

Пример №8. Вывести список Покупателей из Москвы или из Самары (запрос эквивалентен **Примеру №5**).

```
SELECT Фамилия, Город
FROM Покупатель
WHERE Город IN ("Москва", "Самара")
```

`NOT IN` используется для отбора любых значений, кроме тех, которые указаны в представленном списке.

Пример №9. Вывести список Покупателей, проживающих не в Москве и не в Самаре.

```
SELECT Фамилия, Город
FROM Покупатель
WHERE Город Not IN ("Москва","Самара")
```

Соответствие шаблону

С помощью оператора `LIKE` можно выполнять *сравнение* выражения с заданным шаблоном, в котором допускается использование символов-заменителей:

- Символ `%` – вместо этого символа может быть подставлено любое количество произвольных символов. **(В MS Access используется символ *)**
- Символ `_` заменяет один символ строки. **(В MS Access используется символ ?)**
- `[]` – вместо символа строки будет подставлен один из возможных символов, указанный в этих ограничителях.
- `[^]` – вместо соответствующего символа строки будут подставлены все символы, кроме указанных в ограничителях.

Пример №10. Найти Покупателей, у которых в номере телефона вторая цифра – 4.

```
SELECT Фамилия, Телефон
FROM Покупатель
WHERE Телефон Like "?4*"
```

Пример №11. Найти Покупателей, у которых в номере телефона вторая цифра – 2 или 4.

```
SELECT Фамилия, Телефон
FROM Покупатель
WHERE Телефон Like "?[24]*"
```

Пример №12. Найти Покупателей, у которых в номере телефона вторая цифра 2, 3 или 4.

```
SELECT Покупатель.Фамилия, Покупатель.Телефон
FROM Покупатель
WHERE Покупатель.Телефон Like "?[2-4]*"
```

Пример №13. Найти Покупателей, у которых в фамилии встречается слог "ро".

```
SELECT Покупатель.Фамилия
FROM Покупатель
WHERE Покупатель.Фамилия Like "*ро*"
```

Значение NULL

Оператор `IS NULL` используется для *сравнения* текущего значения со *значением* `NULL` – специальным значением, указывающим на отсутствие любого значения. `NULL` – это не то же самое, что знак пробела (пробел – допустимый символ) или ноль (0 – допустимое число). `NULL` отличается и от строки нулевой длины (пустой строки).

Пример №14. Найти сотрудников, у которых нет телефона (поле Телефон не содержит никакого значения).

```
SELECT Фамилия, Телефон
FROM Покупатель
WHERE Телефон Is Null
```

IS NOT NULL используется для проверки присутствия значения в поле.

Пример №15. Выборка сотрудников, у которых есть телефон (поле Телефон содержит какое-либо значение).

```
SELECT Покупатель.Фамилия, Покупатель.Телефон
FROM Покупатель
WHERE Покупатель.Телефон Is Not Null
```

Предложение ORDER BY

В общем случае строки в результирующей таблице SQL-запроса никак не упорядочены. Однако их можно требуемым образом отсортировать, для чего в оператор SELECT помещается фраза ORDER BY, которая сортирует данные выходного набора в заданной последовательности. Сортировка может выполняться по нескольким полям, в этом случае они перечисляются за ключевым словом ORDER BY через запятую. Способ сортировки задается ключевым словом, указываемым в рамках параметра ORDER BY следом за названием поля, по которому выполняется сортировка. По умолчанию реализуется сортировка по возрастанию. Явно она задается ключевым словом ASC. Для выполнения сортировки в обратной последовательности необходимо после имени поля, по которому она выполняется, указать ключевое слово DESC. Фраза ORDER BY позволяет упорядочить выбранные записи в порядке возрастания или убывания значений любого столбца или комбинации столбцов, независимо от того, присутствуют эти столбцы в таблице результата или нет. Фраза ORDER BY всегда должна быть последним элементом в операторе SELECT.

Пример №16.Вывести список Покупателей в алфавитном порядке.

```
SELECT Фамилия, Фирма
FROM Покупатель
ORDER BY Фамилия
```

Во фразе ORDER BY может быть указано и больше одного элемента. Главный (первый) ключ сортировки определяет общую упорядоченность строк результирующей таблицы. Если во всех строках результирующей таблицы значения главного ключа сортировки являются уникальными, нет необходимости использовать дополнительные ключи сортировки. Однако, если значения главного ключа не уникальны, в результирующей таблице будет присутствовать несколько строк с одним и тем же значением старшего ключа сортировки. В этом случае, возможно, придется упорядочить строки с одним и тем же значением главного ключа по какому-либо дополнительному ключу сортировки.

Пример №17. Вывести список фирм и Покупателей. Названия фирм упорядочить в алфавитном порядке, имена Покупателей в каждой фирме отсортировать в обратном порядке.

```
SELECT Покупатель.Фирма, Покупатель.Фамилия
FROM Покупатель
ORDER BY Покупатель.Фирма, Покупатель.Фамилия DESC
```

Построение вычисляемых полей

В общем случае для создания *вычисляемого (производного) поля* в списке SELECT следует указать некоторое выражение языка SQL. В этих выражениях применяются арифметические операции сложения, вычитания, умножения и деления, а также встроенные функции языка SQL. Можно указать имя любого столбца (поля) таблицы

или запроса, но использовать имя столбца только той таблицы или запроса, которые указаны в списке предложения FROM соответствующей инструкции. При построении сложных выражений могут понадобиться скобки.

Стандарты SQL позволяют явным образом задавать имена столбцов результирующей таблицы, для чего применяется фраза AS.

Пример №18. Рассчитать общую стоимость для каждой продажи. Этот запрос использует расчет результирующих столбцов на основе арифметических выражений.

```
SELECT Товар.Название, Товар.Цена, Продажа.Количество,
Товар.Цена * Продажа.Количество AS Стоимость
FROM Товар INNER JOIN Продажа
ON Товар.[Код Товара]=Продажа.[Код Товара]
```

Пример №19. Получить список фирм с указанием фамилии и инициалов Покупателей.

```
SELECT Фирма, Фамилия+" "+
Left(Имя,1)+"."+Left(Отчество,1)+"."AS ФИО
FROM Покупатель
```

В запросе использована встроенная функция Left, позволяющая вырезать в текстовой переменной один символ слева в данном случае.

Пример №20. Получить список товаров с указанием года и месяца продажи.

```
SELECT Товар.Название, Year(Продажа.Дата)
AS Год, Month(Продажа.Дата) AS Месяц
FROM Товар INNER JOIN Продажа
ON Товар.[Код Товара]=Продажа.[Код Товара]
```

В запросе использованы встроенные функции Year и Month для выделения года и месяца из даты.

Использование итоговых функций

С помощью *итоговых (агрегатных) функций* в рамках SQL-запроса можно получить ряд обобщающих статистических сведений о множестве отобранных значений выходного набора.

Пользователю доступны следующие основные *итоговые функции*:

- Count (Выражение) - определяет количество записей в выходном наборе SQL-запроса;
- Min/Max (Выражение) - определяют наименьшее и наибольшее из множества значений в некотором поле запроса;
- Avg (Выражение) - эта функция позволяет рассчитать среднее значение множества значений, хранящихся в определенном поле отобранных запросом записей. Оно является арифметическим средним значением, т.е. суммой значений, деленной на их количество.
- Sum (Выражение) - вычисляет сумму множества значений, содержащихся в определенном поле отобранных запросом записей.

Чаще всего в качестве выражения выступают имена столбцов. Выражение может вычисляться и по значениям нескольких таблиц.

Все эти функции оперируют со значениями в единственном столбце таблицы или с арифметическим выражением и возвращают единственное значение. Функции `COUNT`, `MIN` и `MAX` применимы как к числовым, так и к нечисловым полям, тогда как функции `SUM` и `AVG` могут использоваться только в случае числовых полей, за исключением `COUNT(*)`. При вычислении результатов любых функций сначала исключаются все пустые значения, после чего требуемая операция применяется только к оставшимся конкретным значениям столбца. Вариант `COUNT(*)` - особый случай использования функции `COUNT`, его назначение состоит в подсчете всех строк в результирующей таблице, независимо от того, содержатся там пустые, дублирующиеся или любые другие значения.

Если до применения обобщающей функции необходимо исключить дублирующиеся значения, следует перед именем столбца в определении функции поместить ключевое слово `DISTINCT`. Оно не имеет смысла для функций `MIN` и `MAX`, однако его использование может повлиять на результаты выполнения функций `SUM` и `AVG`, поэтому необходимо заранее обдумать, должно ли оно присутствовать в каждом конкретном случае. Кроме того, ключевое слово `DISTINCT` может быть указано в любом запросе не более одного раза.

Очень важно отметить, что *итоговые функции* могут использоваться только в списке предложения `SELECT` и в составе предложения `HAVING`. Во всех других случаях это недопустимо. Если список в предложении `SELECT` содержит *итоговые функции*, а в тексте запроса отсутствует фраза `GROUP BY`, обеспечивающая объединение данных в группы, то ни один из элементов списка предложения `SELECT` не может включать каких-либо ссылок на поля, за исключением ситуации, когда поля выступают в качестве аргументов *итоговых функций*.

Пример №21. Определить первое по алфавиту название товара.

```
SELECT Min(Товар.Название) AS Первое_Название
FROM Товар
```

Пример №22. Определить количество продаж.

```
SELECT Count(*) AS Количество_продаж
FROM Продажа
```

Пример №23. Определить суммарное количество проданного товара.

```
SELECT Sum(Продажа.Количество)
AS Количество_товара
FROM Продажа
```

Пример №24. Определить среднюю цену проданного товара.

```
SELECT Avg(Товар.Цена) AS Средняя_Цена
FROM Товар INNER JOIN Продажа
ON Товар.[Код Товара]=Продажа.[Код Товара]
```

Пример №25. Подсчитать общую стоимость проданных товаров.

```
SELECT Sum(Товар.Цена*Продажа.Количество)
AS Стоимость
FROM Товар INNER JOIN Продажа
ON Товар.[Код Товара]=Продажа.[Код Товара]
```

Предложение GROUP BY

Часто в запросах требуется формировать промежуточные итоги, что обычно отображается появлением в запросе фразы "для каждого...". Для этой цели в операторе `SELECT` используется предложение `GROUP BY`. Запрос, в котором присутствует `GROUP BY`, называется группирующим запросом, поскольку в нем группируются данные, полученные в результате выполнения операции `SELECT`, после чего для каждой отдельной группы создается единственная суммарная строка. Стандарт SQL требует, чтобы предложение `SELECT` и фраза `GROUP BY` были тесно связаны между собой. При наличии в операторе `SELECT` фразы `GROUP BY` каждый элемент списка в предложении `SELECT` должен иметь единственное значение для всей группы. Более того, предложение `SELECT` может включать только следующие типы элементов: имена полей, *итоговые функции*, константы и выражения, включающие комбинации перечисленных выше элементов.

Все имена полей, приведенные в списке предложения `SELECT`, должны присутствовать и во фразе `GROUP BY` - за исключением случаев, когда имя столбца используется в *итоговой функции*. Обратное правило не является справедливым - во фразе `GROUP BY` могут быть имена столбцов, отсутствующие в списке предложения `SELECT`.

Если совместно с `GROUP BY` используется предложение `WHERE`, то оно обрабатывается первым, а *группированию* подвергаются только те строки, которые удовлетворяют условию поиска.

Стандартом SQL определено, что при проведении *группирования* все отсутствующие значения рассматриваются как равные. Если две строки таблицы в одном и том же группируемом столбце содержат значение `NULL` и идентичные значения во всех остальных непустых группируемых столбцах, они помещаются в одну и ту же группу.

Пример №26. Вычислить средний объем покупок, совершенных каждым покупателем.

```
SELECT Покупатель.Фамилия, Avg(Продажа.Количество)
AS Среднее_количество
FROM Покупатель INNER JOIN Продажа
ON Покупатель.[Код Покупателя]=Продажа.[Код Покупателя]
GROUP BY Покупатель.Фамилия
```

Фраза "каждым покупателем" нашла свое отражение в SQL-запросе в виде предложения `GROUP BY Покупатель.Фамилия`.

Пример №27. Определить, на какую сумму был продан товар каждого наименования.

```
SELECT Товар.Название,
Sum(Товар.Цена*Продажа.Количество)
AS Стоимость
FROM Товар INNER JOIN Продажа
ON Товар.[Код Товара]=Продажа.[Код Товара]
GROUP BY Товар.Название
```

Пример №28. Подсчитать количество сделок, осуществленных каждой фирмой.

```
SELECT Покупатель.Фирма, Count(Продажа.[Код Продажи])
AS Количество_продаж
FROM Покупатель INNER JOIN Продажа
```

```
ON Покупатель.[Код Покупателя]=Продажа.[Код Покупателя]
GROUP BY Покупатель.Фирма
```

Пример №29. Подсчитать общее количество купленного для каждой фирмы товара и его стоимость.

```
SELECT Покупатель.Фирма, Sum(Продажа.Количество) AS Общее_Количество,
Sum(Товар.Цена*Продажа.Количество) AS Стоимость
FROM Товар INNER JOIN  (Покупатель INNER JOIN Продажа
ON Покупатель.[Код Покупателя]=Продажа.[Код Покупателя])
ON Товар.[Код Товара]=Продажа.[Код Товара]
GROUP BY Покупатель.Фирма
```

Пример №30. Определить суммарную стоимость каждого товара за каждый месяц.

```
SELECT Товар.Название, Month(Продажа.Дата) AS Месяц,
Sum(Товар.Цена*Продажа.Количество) AS Стоимость
FROM Товар INNER JOIN Продажа
ON Товар.[Код Товара]=Продажа.[Код Товара]
GROUP BY Товар.Название, Month(Продажа.Дата)
```

Пример №31. Определить суммарную стоимость каждого товара первого сорта за каждый месяц.

```
SELECT Товар.Название, Month(Продажа.Дата) AS Месяц,
Sum(Товар.Цена*Продажа.Количество) AS Стоимость
FROM Товар INNER JOIN Продажа
ON Товар.[Код Товара]=Продажа.[Код Товара]
WHERE Товар.Сорт="Первый"
GROUP BY Товар.Название, Month(Продажа.Дата)
```

Предложение HAVING

При помощи `HAVING` отражаются все предварительно сгруппированные посредством `GROUP BY` блоки данных, удовлетворяющие заданным в `HAVING` условиям. Это дополнительная возможность "профильтровать" выходной набор.

Условия в `HAVING` отличаются от условий в `WHERE`:

- `HAVING` исключает из результирующего набора данных группы с результатами агрегированных значений;
- `WHERE` исключает из расчета агрегатных значений по группировке записи, не удовлетворяющие условию;
- в условии поиска `WHERE` нельзя задавать агрегатные функции.

Пример №32. Определить фирмы, у которых общее количество сделок превысило три.

```
SELECT Покупатель.Фирма, Count(Продажа.Количество) AS Количество_сделок
FROM Покупатель INNER JOIN Продажа
ON Покупатель.[Код Покупателя]=Продажа.[Код Покупателя]
GROUP BY Покупатель.Фирма
HAVING Count(Продажа.Количество)>3
```

Пример №33.Вывести список товаров, проданных на сумму более 10000 руб.

```
SELECT Товар.Название, Sum(Товар.Цена*Продажа.Количество) AS Стоимость
FROM Товар INNER JOIN Продажа
ON Товар.[Код Товара]=Продажа.[Код Товара]
GROUP BY Товар.Название
HAVING Sum(Товар.Цена*Продажа.Количество)>10000
```

Пример №34. Вывести список товаров, проданных на сумму более 10000 без указания суммы.

```
SELECT Товар.Название
FROM Товар INNER JOIN Продажа
ON Товар.[Код Товара]=Продажа.[Код Товара]
GROUP BY Товар.Название
HAVING Sum(Товар.Цена*Продажа.Количество)>10000
```