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. Будучи очень мощным, он способен выполнять действия, эквивалентные операторам реляционной алгебры, причем в пределах единственной выполняемой команды. При его помощи можно реализовать сложные и громоздкие условия отбора данных из различных таблиц.

Оператор `SELECT` – средство, которое полностью абстрагировано от вопросов представления данных, что помогает сконцентрировать внимание на проблемах доступа к данным. Примеры его использования наглядно демонстрируют один из основополагающих принципов больших (промышленных) *СУБД*: средства хранения данных и доступа к ним отделены от средств представления данных. *Операции* над данными производятся в масштабе наборов данных, а не отдельных записей.

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. Символом `*` можно выбрать все поля, а вместо имени поля применить *выражение* из нескольких имен.

Если обрабатывается ряд таблиц, то (при наличии одноименных полей в разных таблицах) в списке полей используется полная спецификация поля, т.е. `Имя_таблицы.Имя_поля` .

Предложение FROM

Предложение `FROM` задает имена таблиц и представлений, которые содержат поля, перечисленные в операторе `SELECT` . Необязательный *параметр* псевдонима – 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. `SELECT` – закрытая операция: *результат запроса* к таблице представляет собой другую таблицу. Существует множество вариантов записи данного оператора, что иллюстрируется приведенными ниже примерами.

Пример 4.1. Составить *список* сведений о всех клиентах.

```
SELECT * FROM Клиент
```

```
Пример 4.1. Список сведений о всех клиентах.
```

Параметр WHERE определяет *критерий* записей для выходного 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 4.2. Составить *список* всех фирм.

```
SELECT ALL Клиент.Фирма FROM Клиент

Или (что эквивалентно)

SELECT Клиент.Фирма FROM Клиент

Пример 4.2. Список всех фирм.
```

Результат выполнения запроса может содержать *дублирующиеся значения*, поскольку в отличие от *операций реляционной алгебры* 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 4.2 выглядит следующим образом:

```
SELECT DISTINCT Клиент.Фирма
   FROM Клиент
```

Предложение WHERE

С помощью *WHERE*-параметра *пользователь* определяет, какие блоки данных из приведенных в списке *FROM* таблиц появятся в *результате запроса*. 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 4.3. Показать все операции отпуска товаров объемом больше 20.

```
SELECT * FROM Сделка
    WHERE Количество>20

Пример 4.3. Операции отпуска товаров объемом больше 20.
```

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 4.4. 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 4.4. Список товаров, цена которых больше или равна 100 и меньше или равна 150.

Пример 4.5. Вывести список клиентов из Москвы или из Самары.

```
SELECT Фамилия, ГородКлиента
    FROM Клиент
    WHERE ГородКлиента="Москва" Or
          ГородКлиента="Самара"
```

Пример 4.5. Список клиентов из Москвы или из Самары.

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 4.6. Вывести список товаров, цена которых лежит в диапазоне от 100 до 150 (запрос эквивалентен примеру 4.4).

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

Пример 4.6. Список товаров, цена которых лежит в диапазоне от 100 до 150.

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

Пример 4.7. 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 (что эквивалентно)

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

Пример 4.7. Список товаров, цена которых не лежит в диапазоне от 100 до 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 4.8. Вывести список клиентов из Москвы или из Самары (запрос эквивалентен примеру 4.5).

```
SELECT Фамилия, ГородКлиента
   FROM Клиент
   WHERE ГородКлиента IN ("Москва", "Самара")
```

Пример 4.8. Список клиентов из Москвы или из Самары

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

Пример 4.9. Вывести список клиентов, проживающих не в Москве и не в Самаре.

```
SELECT Фамилия, ГородКлиента
    FROM Клиент
    WHERE ГородКлиента
          NOT IN ("Москва","Самара")
```

```
Пример 4.9. Список клиентов, проживающих не в Москве и не в Самаре.
```

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.

Символ `_` заменяет один символ строки.

`[]` – 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 4.10. Найти клиентов, у которых в номере телефона вторая цифра – 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
    FROM Клиент
    WHERE Клиент.Телефон LIKE '_4%'
```

```
Пример 4.10. Выборка клиентов, у которых в номере телефона вторая цифра – 4.
```

Пример 4.11. Найти клиентов, у которых в номере телефона вторая цифра – 2 или 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
    FROM Клиент
    WHERE Клиент.Телефон LIKE '_[2,4]%'
```

```
Пример 4.11. Выборка клиентов, у которых в номере телефона вторая цифра – 2 или 4.
```

Пример 4.12. Найти клиентов, у которых в номере телефона вторая цифра 2, 3 или 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
    FROM Клиент
    WHERE Клиент.Телефон LIKE '_[2-4]%'
```

```
Пример 4.12. Выборка клиентов, у которых в номере телефона вторая цифра 2, 3 или 4.
```

Пример 4.13. Найти клиентов, у которых в фамилии встречается слог "ро".

```
SELECT Клиент.Фамилия
    FROM Клиент
    WHERE Клиент.Фамилия LIKE "%ро%"
```

```
Пример 4.13. Выборка клиентов, у которых в фамилии встречается слог "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 4.14. Найти сотрудников, у которых нет телефона (поле Телефон не содержит никакого значения).

```
SELECT Фамилия, Телефон
    FROM Клиент
    WHERE Телефон IS NULL
```

Пример 4.14. Выборка сотрудников, у которых нет телефона (поле Телефон не содержит никакого значения).

`IS NOT NULL` используется для проверки присутствия значения в поле.

Пример 4.15. Выборка клиентов, у которых есть телефон (поле Телефон содержит какое-либо значение).

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

Пример 4.15. Найти клиентов, у которых есть телефон (поле Телефон содержит какое-либо значение).

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 4.16.Вывести список клиентов в алфавитном порядке.

```
SELECT Клиент.Фамилия, Клиент.Фирма
    FROM Клиент
    ORDER BY Клиент.Фамилия
```

Пример 4.16. Список клиентов в алфавитном порядке.

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 4.17. Вывести список фирм и клиентов. Названия фирм упорядочить в алфавитном порядке, имена клиентов в каждой фирме отсортировать в обратном порядке.

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

Пример 4.17. Список фирм и клиентов. Названия фирм в алфавитном порядке, имена клиентов в каждой фирме в обратном порядке.