## http://sql.ru/docs/sql/u\_sql/index.shtml

## <http://www.compress.ru/Article.asp?id=325>

## Работа с операторами SQL

* **DDL**
* **DML**
* **DCL**

В этом разделе мы изучим различные операторы SQL, включая операторы для выбора данных, их добавления, удаления или изменения, изменения метаданных и пр.

[Работа с операторами SQL](http://www.compress.ru/Temp/325/index1.htm" \l "11)

[Выбор данных](http://www.compress.ru/Temp/325/index1.htm" \l "12)

[Предложение FROM](http://www.compress.ru/Temp/325/index1.htm" \l "13)

[Предложение WHERE](http://www.compress.ru/Temp/325/index1.htm" \l "14)

[Операторы AND, OR и NOT](http://www.compress.ru/Temp/325/index1.htm" \l "15)

[Предложение ORDER BY](http://www.compress.ru/Temp/325/index1.htm" \l "16)

[Связывание таблиц](http://www.compress.ru/Temp/325/index1.htm" \l "17)

[Предложение GROUP BY](http://www.compress.ru/Temp/325/index1.htm" \l "18)

[Предложение HAVING](http://www.compress.ru/Temp/325/index1.htm" \l "19)

[Ключевые слова ALL и DISTINCT](http://www.compress.ru/Temp/325/index1.htm" \l "20)

[Ключевое слово TOP](http://www.compress.ru/Temp/325/index1.htm" \l "21)

[Модификация данных](http://www.compress.ru/Temp/325/index1.htm" \l "22)

[Оператор UPDATE](http://www.compress.ru/Temp/325/index1.htm" \l "23)

[Оператор DELETE](http://www.compress.ru/Temp/325/index1.htm" \l "24)

[Оператор INSERT](http://www.compress.ru/Temp/325/index1.htm" \l "25)

[Модификация метаданных](http://www.compress.ru/Temp/325/index1.htm" \l "26)

[Оператор](http://www.compress.ru/Temp/325/index1.htm" \l "27) CREATE TABLE

[Оператор](http://www.compress.ru/Temp/325/index1.htm" \l "28) ALTER TABLE

[Оператор DROP](http://www.compress.ru/Temp/325/index1.htm" \l "29)

[Другие операторы SQL](http://www.compress.ru/Temp/325/index1.htm" \l "30)

|  |  |
| --- | --- |
| [в начало](file:///C:\Users\Admin\Downloads\1%20семестр\1сем_8зан\index.htm#begin) | [в начало](file:///C:\Users\Admin\Downloads\1%20семестр\1сем_8зан\index.htm#begin) |

#### Оператор CREATE TABLE

Для создания новой таблицы необходимо использовать оператор CREATE TABLE, синтаксис которого имеет вид:

CREATE TABLE table

( column1 type1 [(size1)][CONSTRAINT \_

column-constraint1]

[, column2 type2 [(size2)][CONSTRAINT \_

column-constraint2]

[, ...]]

[CONSTRAINT table-constraint1 \_

[,table-constraint2 [, ...]]]);

В этом операторе следует указать имя поля, тип данных для него (тип данных должен поддерживаться данной СУБД), длину (для некоторых типов полей) и, если нужно, серверные ограничения (с применением ключевого слова CONSTRAINT). Например, следующий запрос создает таблицу с именем Simple с четырьмя колонками — LastName, FirstName, EMail и HomePage:

CREATE TABLE Simple

(FirstName varchar(50) NOT NULL,

LastName varchar(50) NOT NULL,

EMail varchar(50),

HomePage varchar(255)

)

Мы можем расширить эту таблицу добавлением поля PersonID, которое будет использовано как первичный ключ:

CREATE TABLE Simple

( PersonID Integer NOT NULL PRIMARY KEY,

FirstName varchar(50) NOT NULL,

LastName varchar(50) NOT NULL,

EMail varchar(50),

HomePage varchar(255)

)

и указать, что комбинация полей LastName и FirstName должна быть уникальна:

CREATE TABLE Simple

( PersonID Serial NOT NULL PRIMARY KEY,

FirstName varchar(50) NOT NULL,

LastName varchar(50) NOT NULL,

EMail varchar(50),

HomePage varchar(255),

CONSTRAINT SimpleConstraint UNIQUE

(FirstName, LastName)

)

CONSTRAINT CHECK (COLUMN < 100)

Примеры:

CREATE TABLE projx

(

projno int4 NOT NULL

, pname bit varying (14) CHECK (SUBSTR(pname,1,1) BETWEEN 'A' AND 'Z')

, bdate DATE DEFAULT TRUNC ( SYSDATE )

, budget numeric(10,2) CHECK (budget > 0)

, str text

, CONSTRAINT C1 CHECK (str like 'A\*') - такое лучше делать на уровне кода

);

CREATE TABLE lessons (

subjectname VARCHAR(30),

teachertnp NUMBER(2) references teachers(tnp),

foreign key (subjectname) references subjects(subjectname),

);

Используя предложение SELECT и ключевое слово INTO, мы можем создавать новые таблицы, основанные на условии, указанном в предложении WHERE. Например:

SELECT \*

INTO NewOrders

FROM Orders

WHERE OrderDate > 1/1/97

Этот запрос создаст новую таблицу NewOrders и заполнит ее данными о заказах начиная с 1 января 1997 года.

### Типы данных

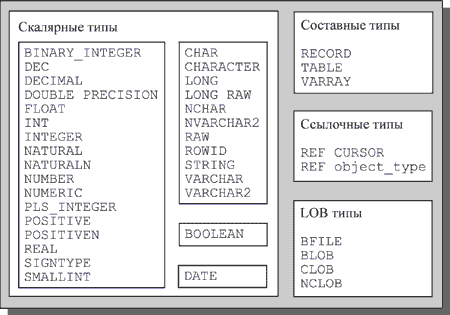
[**Детальное описание типов**](https://postgrespro.ru/docs/postgresql/9.4/datatype)

**Базовые типы данных (Postgres):**

| **Имя** | **Псевдонимы** | **Описание** |
| --- | --- | --- |
| bigint | int8 | знаковое целое из 8 байт |
| bigserial | serial8 | восьмибайтное целое с автоувеличением |
| bit [ (*n*) ] |  | битовая строка фиксированной длины |
| bit varying [ (*n*) ] | varbit | битовая строка переменной длины |
| boolean | bool | логическое значение (true/false) |
| box |  | прямоугольник в плоскости |
| bytea |  | двоичные данные ("массив байт") |
| character [ (*n*) ] | char [ (*n*) ] | символьная строка фиксированной длины |
| character varying [ (*n*) ] | varchar [ (*n*) ] | символьная строка переменной длины |
| cidr |  | сетевой адрес IPv4 или IPv6 |
| circle |  | круг в плоскости |
| date |  | календарная дата (год, месяц, день) |
| double precision | float8 | число двойной точности с плавающей точкой (8 байт) |
| inet |  | адрес узла IPv4 или IPv6 |
| integer | int, int4 | знаковое четырёхбайтное целое |
| interval [ *поля* ] [ (*p*) ] |  | интервал времени |
| json |  | текстовые данные JSON |
| jsonb |  | двоичные данные JSON, разобранные |
| line |  | прямая в плоскости |
| lseg |  | отрезок в плоскости |
| macaddr |  | MAC-адрес |
| money |  | денежная сумма |
| numeric [ (*p*, *s*) ] | decimal [ (*p*, *s*) ] | вещественное число заданной точности |
| path |  | геометрический путь в плоскости |
| pg\_lsn |  | Последовательный номер в журнале PostgreSQL |
| point |  | геометрическая точка в плоскости |
| polygon |  | замкнутый геометрический путь в плоскости |
| real | float4 | число одинарной точности с плавающей точкой (4 байта) |
| smallint | int2 | знаковое двухбайтное целое |
| smallserial | serial2 | двухбайтное целое с автоувеличением |
| serial | serial4 | четырёхбайтное целое с автоувеличением |
| text |  | символьная строка переменной длины |
| time [ (*p*) ] [ without time zone ] |  | время суток (без часового пояса) |
| time [ (*p*) ] with time zone | timetz | время суток с учётом часового пояса |
| timestamp [ (*p*) ] [ without time zone ] |  | дата и время (без часового пояса) |
| timestamp [ (*p*) ] with time zone | timestamptz | дата и время с учётом часового пояса |
| tsquery |  | запрос текстового поиска |
| tsvector |  | документ для текстового поиска |
| txid\_snapshot |  | снимок идентификатора транзакций |
| uuid |  | универсальный уникальный идентификатор |
| xml |  | XML-данные |

[Создание собственных типов](https://postgrespro.ru/docs/postgresql/9.4/sql-createtype)

**Базовые типы данных (Oracle):**



**Рис. 1.**  Список встроенных типов SQL

Скалярные типы описывают простые значения, не имеющие внутренних составляющих.

Составные типы описывают структуры, в которых имеются внутренние компоненты.

Ссылочные типы содержат значения. LOB типы содержат значения, называемые локаторами, которые определяют расположение больших объектов хранимых данных (например, графические файлы).

В следующей таблице приведено описание некоторых типов данных языка SQL.

|  |  |
| --- | --- |
| **Синтаксис** | **Диапазон значений** |
| **Числовые типы** |  |
| BINARY\_INTEGER PLS\_INTEGER (целое со знаком) | -2147483647 .. 2147483647. Тип PLS\_INTEGER требует меньше памяти и обрабатывается быстрее, чем другие числовые типы |
| NUMBER[(precision,scale)] (с плавающей точкой) | 1.0E-130 .. 9.99E125 |
| NUMERIC (с фиксированной точкой) | точность до 38 десятичных знаков |
| FLOAT (с плавающей точкой) | точность до 38 десятичных знаков |
| REAL (с плавающей точкой) | точность до 18 десятичных знаков |
| **Символьные типы** |  |
| CHAR[(maximum\_length)] (для строк постоянной длины) LONG (для строк переменной длины) | до 32767 байт Для столбца базы данных максимальный размер типа CHAR составляет 2000 байтов, а типа LONG - до 2 Гб |
| RAW(maximum\_length) LONG RAW (для двоичных данных или строк байтов) | до 32767 байт Для столбца базы данных типа RAW максимальный размер - 2000 байт. |
| VARCHAR2 (maximum\_length) (для строк символов переменной длины) | до 32767 байт |

Все встроенные типы данных являются базовыми типами.

#### Оператор ALTER TABLE

Для изменения структуры существующей таблицы можно использовать оператор ALTER TABLE. Применяя его, можно добавить или удалить поле или серверное ограничение. Существует четыре разновидности оператора ALTER TABLE.

Первая разновидность этого оператора используется для добавления колонки к таблице, и ее синтаксис имеет вид:

ALTER TABLE table ADD [COLUMN] column datatype

[(size)]

[CONSTRAINT sinlge-column-constraint]

В запросах такого вида определяется имя таблицы, имя нового поля, его тип данных и, если нужно, размер. Помимо этого можно указать серверное ограничение, связанное с данным полем. Например, для добавления поля Phone к таблице Simple, созданной ранее, можно выполнить следующий запрос:

ALTER TABLE Simple ADD Phone varchar(30)

Вторая разновидность оператора ALTER TABLE применяется для добавления серверных ограничений к таблице, а ее синтаксис имеет вид:

ALTER TABLE table ADD CONSTRAINT constraint

Такие запросы позволяют только добавлять индексы, позволяющие использовать соответствующие поля в качестве первичных или внешних ключей.

Третья разновидность предложения ALTER TABLE применяется для удаления поля из таблицы:

ALTER TABLE table DROP [COLUMN] column

Ключевое слово COLUMN использовать не обязательно. Например:

ALTER TABLE Simple DROP Phone

Обратите внимание на то, что для удаления проиндексированных полей следует сначала удалить индекс. Это можно сделать с помощью четвертой разновидности предложения ALTER TABLE:

ALTER TABLE table DROP CONSTRAINT index

Ниже приведен пример такого запроса:

ALTER TABLE Simple DROP CONSTRAINT PrimaryKey

#### Оператор DROP

Для удаления таблиц или индексов можно использовать оператор DROP, имеющий две разновидности. Первая из них применяется для удаления таблицы из базы данных:

DROP TABLE table

Вторая разновидность используется для удаления индекса:

DROP INDEX index ON table

#### Оператор INSERT

Для добавления записей в таблицы следует использовать оператор INSERT, синтаксис которого имеет вид:

INSERT [INTO] table

( [column\_list]

{ VALUES ( { DEFAULT | NULL | expression }

} [, …]

)

Например, для добавления нового клиента в таблицу Customers можно использовать следующий запрос:

INSERT INTO Customers

(CustomerID, CompanyName)

VALUES

(‘XYZFO’, ‘XYZ Deli’)

Вставка пустых указателей:

Если вам нужно ввести пустое значение(NULL), вы вводите его точно так- же как и обычное значение. Предположим, что еще не имелось пол city для мистера Peel. Вы можете вставить его строку со значением=NULL в это поле, следующим образом:

INSERT INTO Salespeople

VALUES (1001, 'Peel', NULL, .12);

Так как значение NULL - это специальный маркер, а не просто символьное значение, он не включается в одиночные кавычки.

Именование столбца:

Вы можете также указывать столбцы, куда вы хотите вставить значение имени. Это позволяет вам вставлять имена в любом порядке. Предположим что вы берете значения для таблицы Заказчиков из отчета выводимого на принтер, который помещает их в таком порядке: city, cname, и cnum, и для упрощения, вы хотите ввести значения в том же порядке:

INSERT INTO Customers (city, cnamе, cnum)

VALUES ('London', 'Honman', 2001);

Обратите внимание что столбцы rating и snum - отсутствуют. Это значит, что эти строки автоматически установлены в значение - по умолчанию. По умолчанию может быть введено или значение NULL или другое значе- ние определяемое как - по умолчанию. Если ограничение запрещает использование значения NULL в данном столбце, и этот столбец не установлен как по умолчанию, этот столбец должен быть обеспечен значением для любой команды INSERT которая относится к таблице( смотри [Главу 18](http://www.sql.ru/docs/sql/u_sql/ch18.shtml) для информации об ограничениях на NULL и на "по умолчанию" ).

Вставка результатов запроса:

Вы можете также использовать команду INSERT чтобы получать или вы- бирать значения из одной таблицы и помещать их в другую, чтобы использовать их вместе с запросом. Чтобы сделать это, вы просто заменяете предложение VALUES (из предыдущего примера) на соответствующий запрос:

INSERT INTO Londonstaff

SELECT \*

FROM Salespeople

WHERE city = 'London';

#### Оператор UPDATE

Для изменения значений в одной или нескольких колонках таблицы применяется оператор UPDATE. Синтакcис этого оператора имеет вид:

UPDATE tableSET column1 = expression1 [, column2 = expression2] [,…]

[WHERE criteria]

Выражение в предложении SET может быть константой или результатом вычислений. Например, для повышения цен всех продуктов, стоящих меньше 10 долл., можно выполнить следующий запрос:

Примеры:

UPDATE Products

SET UnitPrice = UnitPrice \* 1.1

WHERE UnitPrice < 10

UPDATE Salespeople

SET sname = 'Gibson',city = 'Boston',comm = .10

WHERE snum = 1004;

UPDATE Salespeople

SET comm = comm \* 2

WHERE city = 'London';

#### Оператор DELETE

Для удаления строк из таблиц следует использовать оператор DELETE, синтаксис которого имеет вид:

DELETE

FROM table

[WHERE criteria]

Внимание! Предложение WHERE не является обязательным, но если вы забудете его включить, из таблицы будут удалены все записи.

Например, для удаления из списка всех продуктов, которые больше не поставляются, можно выполнить следующий запрос:

DELETE

FROM Products

WHERE Discontinued = 1

Отметим, что полезно использовать оператор SELECT с тем же синтаксисом, что и оператор DELETE, чтобы проверить, какие именно записи будут удалены, прежде чем действительно их удалять. Ниже показан оператор SELECT для приведенного выше запроса на удаление данных:

SELECT ProductName

FROM Products

WHERE Discontinued = 1

Можно использовать в предложении WHERE более сложный критерий для определения того, какие записи должны быть удалены. Предположим, нам нужно удалить из списка клиентов тех из них, кто не имел заказов до определенной даты. Сначала для этого следует выполнить следующий SELECT, чтобы определить, что именно мы удаляем:

SELECT CompanyName

FROM Customers

WHERE Customers.CustomerID NOT IN

(SELECT CustomerID FROM Orders WHERE OrderDate > 01/01/96)

а затем заменить оператор SELECT на оператор DELETE:

DELETE FROM Customers

WHERE Customers.CustomerID NOT IN

(SELECT CustomerID FROM Orders WHERE OrderDate > 01/01/96)

Замечание. При использовании в операторах SQL даты или времени, а также полей, содержащих такие данные, следует уточнить синтаксис таких предложений в документации из комплекта поставки используемой СУБД.

### Выбор данных

Выбор данных представляет собой наиболее часто встречающуюся операцию, выполняемую с помощью SQL. Оператор SELECT — один из самых важных операторов этого языка, применяемый для выбора данных. Синтаксис этого оператора имеет следующий вид:

SELECT column-list

FROM table-list

[WHERE where-clause]

[ORDER BY order-by-clause]

Операторы SELECT должны содержать слова SELECT и FROM; другие ключевые слова, такие как WHERE или ORDER BY, являются необязательными.

За ключевым словом SELECT следуют сведения о том, какие именно поля необходимо включить в результирующий набор данных. Звездочка (\*) обозначает все поля таблицы, например:

SELECT \*

Для выбора одной колонки применяется следующий синтаксис:

SELECT CompanyName

Пример выбора нескольких колонок имеет вид:

SELECT CompanyName, ContactName, ContactTitle

Если выбор данных осуществляется из нескольких таблиц и при этом выбираются одноименные поля из разных таблиц, следует ссылаться на имена таблиц для полной идентификации полей, включаемых в результирующий набор данных, например:

SELECT Customers.CompanyName, Shippers.CompanyName

#### Предложение FROM

Для указания имен таблиц, из которых выбираются записи, применяется ключевое слово FROM, например:

[SELECT \* FROM Customers](http://www.compress.ru/Temp/325/pic3.htm)

Этот запрос возвратит все поля из таблицы Customers.

Если в результирующем наборе данных нужны только поля CompanyName и ContactName, мы можем ввести следующее предложение SELECT:

[SELECT CompanyName, ContactName FROM Customers](http://www.compress.ru/Temp/325/pic4.htm)

Пример запроса к более чем одной таблице приведен ниже:

[SELECT Customers.CompanyName, Shippers.CompanyName](http://www.compress.ru/Temp/325/pic5.htm)

[FROM Customers, Shippers](http://www.compress.ru/Temp/325/pic5.htm)

#### Предложение WHERE

Для фильтрации результатов, возвращаемых оператором SELECT, можно использовать предложение WHERE, синтаксис которого имеет вид:

WHERE expression1 [{AND | OR} expression2 […]]

Например, вместо получения полного списка продуктов можно ограничиться только теми из них, у которых значение поля CategoryID равно 4:

[SELECT \* FROM Products](http://www.compress.ru/Temp/325/pic6.htm)

[WHERE CategoryID = 4](http://www.compress.ru/Temp/325/pic6.htm)

В предложении WHERE можно использовать различные выражения, например:

SELECT \* FROM Products

WHERE CategoryID = 2 AND SupplierID > 10

или:

SELECT ProductName, UnitPrice FROM Products

WHERE CategoryID = 3 OR UnitPrice < 50

или:

SELECT ProductName, UnitPrice FROM Products

WHERE Discontinued IS NOT NULL

Выражение ‘IS NOT NULL’ означает, что соответствующая колонка результирующего набора данных не может содержать пустых значений.

В предложении WHERE можно использовать один из шести операторов отношений, определенных в SQL. Эти операторы приведены в табл. 8.

Таблица 8

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
| < | Меньше |
| <= | Меньше или равно |
| <> | Не равно |
| = | Равно |
| > | Больше |
| >= | Больше или равно |

Помимо перечисленных выше простых операторов сравнения, можно использовать и специальные операторы сравнения, приведенные в табл. 9.

Таблица 9

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
| ALL | Применяется совместно с операторами сравнения при сравнении со списком значений |
| ANY | Применяется совместно с операторами сравнения при сравнении со списком значений |
| BETWEEN | Применяется при проверке нахождения значения внутри заданного интервала (включая его границы) |
| IN | Применяется для проверки наличия значения в списке |
| LIKE | Применяется при проверке соответствия значения заданной маске |

Приведем несколько примеров применения этих операторов. Для сопоставления данных с маской применяется ключевое слово LIKE:

[SELECT CompanyName, ContactName](http://www.compress.ru/Temp/325/pic7.htm)

[FROM Customers](http://www.compress.ru/Temp/325/pic7.htm)

[WHERE CompanyName LIKE ‘M%’](http://www.compress.ru/Temp/325/pic7.htm)

В данной маске символ ‘%’ (процент) заменяет любую последовательность символов, а символ ‘\_’ (подчеркивание) — один любой символ. Тот же самый результат может быть получен следующим способом:

SELECT CompanyName, ContactName

FROM Customers

WHERE CompanyName BETWEEN ‘M’ AND ‘N’

В последнем примере мы можем расширить область поиска. В частности, при поиске компаний с именами, начинающимися с букв от A до C, можно выполнить следующий оператор SELECT:

SELECT CompanyName, ContactName

FROM Customers

WHERE CompanyName BETWEEN ‘A’ AND ‘D’

Используя оператор LIKE, мы можем сузить диапазон поиска, применив более сложную маску для сравнения. Например, чтобы найти компании, содержащие в своем названии подстроку bl, можно применить следующий запрос:

[SELECT CompanyName, ContactName](http://www.compress.ru/Temp/325/pic8.htm)

[FROM Customers](http://www.compress.ru/Temp/325/pic8.htm)

[WHERE CompanyName LIKE ‘%bl%’](http://www.compress.ru/Temp/325/pic8.htm)

Маска ‘%bl%’ показывает, что до и после искомой подстроки может быть любое количество произвольных символов.

Используя оператор IN, можно задать список значений, в котором должно содержаться значение поля:

SELECT CompanyName, ContactName

FROM Customers

WHERE CustomerID IN (‘ALFKI’, ‘BERGS’, ‘VINET’)

|  |  |
| --- | --- |
| [в начало](file:///C:\Users\Admin\Downloads\1%20семестр\1сем_8зан\index.htm#begin) | [в начало](file:///C:\Users\Admin\Downloads\1%20семестр\1сем_8зан\index.htm#begin) |

#### Операторы AND, OR и NOT

Мы уже рассматривали пример применения оператора AND для логических операций, связанных с требованием, чтобы запись удовлетворяла двум разным критериям. Рассмотрим следующий запрос:

SELECT CompanyName, ContactName

FROM Customers

WHERE CompanyName LIKE ‘S%’ AND Country = ‘USA’

Результатом выполнения этого запроса будет список заказчиков, находящихся в США, название которых начинается с буквы S.

Оператор OR позволяет выбрать записи, удовлетворяющие хотя бы одному из перечисленных условий, в то время как оператор NOT используется для исключения из набора данных записей, удовлетворяющих данному условию. Например, можно применить оператор OR для поиска всех заказчиков, либо находящихся в Калифорнии, либо имеющих название, начинающееся с буквы S (и при этом находящихся где угодно):

SELECT CompanyName, ContactName

FROM Customers

WHERE CompanyName LIKE ‘S%’ OR Region=’CA’

В этом случае результирующий набор данных будет содержать записи, в которых значение поля CompanyName удовлетворяет первому условию, плюс все записи, в которых значение поля Region удовлетворяет второму условию.

Теперь рассмотрим пример применения оператора NOT. Для исключения некоторых заказчиков из результирующего набора данных можно использовать запрос вида:

SELECT CompanyName, ContactName

FROM Customers

WHERE Country NOT IN (‘USA’, ‘UK’)

В результате выполнения этого запроса мы получим список заказчиков из всех стран, кроме США и Великобритании.

|  |  |
| --- | --- |
| [в начало](file:///C:\Users\Admin\Downloads\1%20семестр\1сем_8зан\index.htm#begin) | [в начало](file:///C:\Users\Admin\Downloads\1%20семестр\1сем_8зан\index.htm#begin) |

#### Предложение ORDER BY

Предложение ORDER BY (необязательное) применяется для сортировки результирующего набора данных по одной или нескольким колонкам. Для определения порядка сортировки используются ключевые слова ASC (по возрастанию) или DESC (по убыванию). По умолчанию данные сортируются по возрастанию. Синтаксис предложения ORDER BY имеет вид:

ORDER BY column1 [{ASC | DESC}]

[, column2 [{ASC | DESC}] [,…]

Например, для сортировки сотрудников по фамилии и затем по имени следует использовать следующий SQL-запрос:

SELECT LastName, FirstName, Title

FROM Employees

ORDER BY LastName, FirstName

Если сортировка данных требуется в убывающем порядке (например, требуется список продуктов в порядке убывания цен), используется ключевое слово DESC:

SELECT ProductName, UnitPrice

FROM Products

ORDER BY UnitPrice DESC

#### Связывание таблиц

Как мы уже убедились, можно создавать запросы, позволяющие извлечь данные из нескольких таблиц. Одна из возможностей сделать это заключается в связывании таблиц по одному или нескольким полям. Обратите внимание на то, что без связывания таблиц в результате запроса получится набор данных, содержащий все возможные комбинации строк каждой из исходных таблиц (известное также как декартово произведение):

SELECT ProductName, CategoryName

FROM Products, Categories

в то время как запрос, показанный ниже, приводит к отображению списка продуктов с указанием, к какой категории принадлежит данный продукт:

SELECT ProductName, CategoryName

FROM Products, Categories

WHERE Products.CategoryID = Categories.CategoryID

[Можно сравнить результаты этих двух запросов.](http://www.compress.ru/Temp/325/pic9.htm)

В общем случае синтаксис для связывания таблиц имеет вид:

SELECT column-list

FROM table1, table2

WHERE table1.column1=table2.column2

Следующие несколько примеров связывания таблиц характерны для Microsoft Access и Microsoft SQL Server и могут не работать с другими СУБД, однако мы полагаем, что иллюстрируемая ими функциональность достаточно важна.

Существует несколько типов связывания таблиц. Например, следующий оператор SQL осуществляет так называемое внутреннее соединение таблиц (inner join) — в этом случае в результирующем наборе данных содержатся записи, в которых значения в связанных полях совпадают:

SELECT ProductName, CategoryName

FROM Products INNER JOIN Categories

ON Products.CategoryID = Categories.CategoryID

Так называемые внешние соединения (outer joins) позволяют нам включить в результат запроса все строки из одной таблицы и соответствующие им строки из другой таблицы. Например:

SELECT ProductName, CategoryName

FROM Products LEFT OUTER JOIN Categories

ON Products.CategoryID = Categories.CategoryID

Это было так называемое левое внешнее соединение (left outer join). Существуют также правые внешние соединения (right outer join), возвращающие все строки из второй (то есть правой) таблицы и соответствующие им строки из другой таблицы:

SELECT ProductName, CategoryName

FROM Products RIGHT OUTER JOIN Categories

ON Products.CategoryID = Categories.CategoryID

Комбинируя левое и правое внешние соединения, можно получить полное внешнее соединение, возвращающее все данные из обеих таблиц:

SELECT ProductName, CategoryName

FROM Products FULL OUTER JOIN Categories

ON Products.CategoryID = Categories.CategoryID

Для получения всех комбинаций строк из обеих таблиц (декартова произведения) можно использовать ключевое слово CROSS JOIN без указания связываемых полей:

SELECT ProductName, CategoryName

FROM Products CROSS JOIN Categories

Если в запросе используется более трех таблиц, можно иcпользовать вложенные соединения.

**Объединение множеств**

Оператор UNION позволяет объединить два множества (условно две таблицы). Но в отличие от inner/outer join объединения соединяют не столбцы разных таблиц, а два однотипных набора в один. Формальный синтаксис объединения:

SELECT\_выражение1

UNION [ALL]

SELECT\_выражение 2

[UNION [ALL] SELECT\_выражение N]

Пример:

SELECT FirstName, LastName FROM Customers

UNION

SELECT FirstName, LastName FROM Employees;

**Пересечение множеств**

Оператор **INTERSECT** позволяет найти общие строки для двух выборок, то есть данный оператор выполняет операцию пересечения множеств. Для его использования применяется следующий формальный синтаксис:

SELECT\_выражение 1

INTERSECT

SELECT\_выражение 2

Пример:

Найдем всех сотрудников банка, которые одновременно являются его клиентами:

SELECT FirstName, LastName FROM Employees

INTERSECT

SELECT FirstName, LastName FROM Customers;

**Разность множеств**

Оператор **EXCEPT** в PostgreSQL позволяет найти разность двух выборок, то есть те строки которые есть в первой выборке, но которых нет во второй. Для его использования применяется следующий формальный синтаксис:

SELECT\_выражение1

EXCEPT

SELECT\_выражение2

Пример:

Надо найти всех клиентов банка, которые не являются его сотрудниками:

SELECT FirstName, LastName FROM Customers  
EXCEPT   
SELECT FirstName, LastName FROM Employees;

#### Предложение GROUP BY

Для вычисления суммарных значений на основе данных одной или нескольких таблиц можно использовать предложение GROUP BY, имеющее такой синтаксис:

GROUP BY {column1} [, …]

Например, следующий запрос связывает две таблицы, сортирует их по полю CustomerID, для каждого значения CustomerID создает одну строку в результирующем наборе данных и вычисляет количество значений поля OrderID для каждого значения CustomerID:

SELECT Customers.CustomerID,

COUNT (Orders.OrderID)

FROM Customers INNER JOIN Orders

ON Customers.CustomerID = Orders.CustomerID

GROUP BY Customers.CustomerID

В приведенном выше примере запроса мы использовали в предложении SELECT агрегатную функцию COUNT, вычисляющую количество значений. В [табл. 10](http://www.compress.ru/Temp/325/tab2.htm) указан список наиболее часто используемых агрегатных функций.

Помимо перечисленных выше агрегатных функций можно использовать также математические и строковые функции, приведенные в [табл 11](http://www.compress.ru/Temp/325/tab3.htm).

#### Предложение HAVING

Предложение HAVING имеет назначение, сходное с предложением WHERE, но используется с агрегатными данными. Например:

SELECT Customers.CustomerID,

COUNT (Orders.OrderID)

FROM Customers INNER JOIN Orders

ON Customers.CustomerID = Orders.CustomerID

GROUP BY Customers.CustomerID

HAVING COUNT(Orders.OrderID) >= 10

Этот запрос аналогичен предыдущему, но в результирующий набор данных включены только заказчики, разместившие десять или более заказов.

#### Ключевые слова ALL, ANY и DISTINCT

До этого момента мы рассматривали, как извлечь все или заданные колонки из одной или нескольких таблиц. Для управления выводом дублирующихся строк результирующего набора данных можно использовать ключевые слова ALL или DISTINCT в предложении SELECT. Ключевое слово DISTINCT указывает, что строки результирующего набора данных должны быть уникальны, тогда как ключевое слово ALL указывает, что возвращать следует все строки. Например, для извлечения названий стран, в которых имеются заказчики, можно использовать следующий запрос:

SELECT DISTINCT Country FROM Customers

Отметим, что ключевое слово ALL используется по определению. Если в запросе требуется вывести более одной колонки и при этом использовано слово DISTINCT, то результирующий набор данных будет содержать различные строки, но некоторые значения одного и того же поля в разных строках могут совпадать.

ALL — сравнение будет производиться со всеми записями, которые возвращает подзапрос (или просто со всеми значениями в набор). True вернется только в том случае, если все записи, которые возвращает подзапрос, будут удовлетворять указанному вами условию. Кроме того, в Oracle значение True вернется в ситуации, когда подзапрос не вернет ни одной записи.

В качестве примера приведем такой запрос:

select \* from hr.employees where salary <= ALL(SELECT salary FROM hr.employees WHERE job\_id = 'SH\_CLERK')

Он вернет записи для всех сотрудников, для которых зарплата меньше или равна самой маленькой зарплате у сотрудников с должностью SH\_CLERK.

ANY — сравнение вернет True, если условию будет удовлетворять любая запись из набора (или подзапроса). Например, такой запрос вернет всех пользователей, зарплата которых совпадает с зарплатой клерка:

select \* from hr.employees where salary = ANY(SELECT salary FROM hr.employees WHERE job\_id = 'SH\_CLERK')

#### TOP N записей

Ключевые слова limit и fetch могут быть использованы для возврата первых n строк или первых n процентов таблицы. Например, запрос:

select \* from employees  
order by salary desc  
limit 10;  
  
возвращает первые 10 самых оплачиваемых сотрудников, тогда как запрос:

select \* from employees  
order by salary desc  
fetch first 25 percent rows only;  
  
вернет первую четверть записей таблицы.

Другие примеры:

select distinct \* from employees  
order by salary desc  
fetch first 5 rows;

--

select \* from employees  
order by salary desc  
offset 3 rows  
fetch next 5 rows only;

## Заключение

Мы рассмотрели все основные компоненты языка SQL. Основные выводы:

1. SQL — непроцедурный язык, предназначенный для управления данными в реляционных СУБД. Последний официальный стандарт был опубликован ANSI в 1992 году, и современная реализация SQL называется SQL92. Язык SQL поддерживается большинством производителей СУБД;
2. оператор SELECT следует использовать для извлечения данных из таблиц. Предложение WHERE можно применять для того, чтобы ограничить результирующий набор данных записями, удовлетворяющими заданному условию;
3. предложение GROUP BY может быть использовано для создания результирующего набора данных, содержащего суммарные данные из одной или нескольких таблиц;
4. для получения данных из нескольких таблиц можно использовать ключевое слово JOIN;
5. для изменения данных применяется операторы INSERT, UPDATE и DELETE;
6. операторы CREATE, ALTER и DROP могут быть использованы для создания, модификации и удаления баз данных и содержащихся в них объектов — таблиц, представлений и др.