**4.6.1.****Типы данных SQL.**

* Символьные типы данных - содержат буквы, цифры и специальные символы.
  + **CHAR** или **CHAR(n)** -символьные строки фиксированной длины. Длина строки определяется параметром **n**. **CHAR** без параметра соответсвует **CHAR(1)**. Для хранения таких данных всегда отводится **n** байт вне зависимости от реальной длины строки.
  + **VARCHAR(n)** - символьная строка переменной длины. Для хранения данных этого типа отводится число байт, соответствующее реальной длине строки.
* Целые типы данных - поддерживают только целые числа (дробные части и десятичные точки не допускаются). Над этими типами разрешается выполнять арифметические операции и применять к ним агрегирующие функции (определение максимального, минимального, среднего и суммарного значения столбца реляционной таблицы).
  + **INTEGER** или **INT**- целое, для хранения которого отводится, как правило, 4 байта. *(Замечание: число байт, отводимое для хранения того или иного числового типа данных зависит от используемой СУБД и аппаратной платформы, здесь приводятся наиболее "типичные" значения)* Интервал значений от - 2147483647 до + 2147483648
  + **SMALLINT** - короткое целое (2 байта), интервал значений от - 32767 до +32768
* Вещественные типы данных - описывают числа с дробной частью.
  + **FLOAT** и **SMALLFLOAT** - числа с плавающей точкой (для хранения отводится обычно 8 и 4 байта соответсвенно).
  + **DECIMAL(p)** - тип данных аналогичный **FLOAT** с числом значащих цифр **p**.
  + **DECIMAL(p,n)** - аналогично предыдущему, **p** - общее количество десятичных цифр, **n** - количество цифр после десятичной запятой.
* Денежные типы данных - описывают, естественно, денежные величины. Если в ваша система такого типа данных не поддерживает, то используйте DECIMAL(p,n).
  + **MONEY(p,n)** - все аналогично типу **DECIMAL(p,n)**. Вводится только потому, что некоторые СУБД предусматривают для него специальные методы форматирования.
* Дата и время - используются для хранения даты, времени и их комбинаций. Большинство СУБД умеет определять интервал между двумя датами, а также уменьшать или увеличивать дату на определенное количество времени.
  + **DATE** - тип данных для хранения даты.
  + **TIME** - тип данных для хранения времени.
  + **INTERVAL** - тип данных для хранения верменного интервала.
  + **DATETIME** - тип данных для хранения моментов времени (год + месяц + день + часы + минуты + секунды + доли секунд).
* Двоичные типы данных - позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т.д.). Определения этих типов наиболее сильно различаются от системы к системе, часто используются ключевые слова:
  + **BINARY**
  + **BYTE**
  + **BLOB**
* Последовательные типы данных - используются для представления возрастающих числовых последовательностей.
  + **SERIAL** - тип данных на основе **INTEGER**, позволяющий сформировать уникальное значение (например, для первичного ключа). При добавлении записи СУБД автоматически присваивает полю данного типа значение, получаемое из возрастающей последовательности целых чисел.

**4.6.2.****DDL: Операторы создания схемы базы данных.**

При описании команд предполагается, что:

* текст, набранный строчными буквами (например, **CREATE TABLE**) является обязательным
* текст, набранный прописными буквами и заключенный в угловые скобки (например, **<имя\_базы\_данных>**) обозначает переменную, вводимую пользователем
* в квадратные скобки (например, **[NOT NULL]**) заключается необязательная часть команды
* взаимоисключающие элементы команды разделяются вертикальной чертой (например, **[UNIQUE | PRIMARY KEY]**).

**Операторы базы данных**

|  |  |
| --- | --- |
| **Команда** | **Описание** |
| **CREATE DATABASE <имя\_базы\_данных>** | Создание базы данных. |
| **DROP DATABASE <имя\_базы\_данных>** | Удаление базы данных. |

**Создание и удаление таблиц**

**Создание таблицы:**

**CREATE TABLE <имя\_таблицы>**

**(<имя\_столбца> <тип\_столбца>**

**[NOT NULL]**

**[UNIQUE | PRIMARY KEY]**

**[REFERENCES <имя\_мастер\_таблицы> [<имя\_столбца>]]**

**, ...)**

Пользователь обязан указать имя таблицы и список столбцов. Для каждого столбца обязательно указываются его имя и тип (см. таблицу в предыдущем разделе), а также опционально могут быть указаны параметры

* **NOT NULL** - в этом случае элементы столбца всегда должны иметь определенное значение (не NULL)
* один из взаимоисключающих параметров **UNIQUE** - значение каждого элемента столбца должно быть уникальным или **PRIMARY KEY** - столбец является первичным ключом.
* **REFERNECES <имя\_мастер\_таблицы> [<имя\_столбца>]** - эта конструкция определяет, что данный столбец является внешним ключом и указывает на ключ какой мастер\_таблицы он ссылается.

Контроль за выполнением указанных условий осуществляет СУБД.

*Пример: создание базы данных* ***publications:***

CREATE DATABASE publications;

CREATE TABLE authors (au\_id INT PRIMARY KEY,

author VARCHAR(25) NOT NULL);

CREATE TABLE publishers (pub\_id INT PRIMARY KEY,

publisher VARCHAR(255) NOT NULL,url VARCHAR(255));

CREATE TABLE titles (title\_id INT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

yearpub INT,

pub\_id INT REFERENCES publishers(pub\_id));

CREATE TABLE titleautors (au\_id INT REFERENCES authors(au\_id),

title\_id INT REFERENCES titles(title\_id));

CREATE TABLE wwwsites (site\_id INT PRIMARY KEY,

site VARCHAR(255) NOT NULL,

url VARCHAR(255));

CREATE TABLE wwwsiteauthors (au\_id INT REFERENCES authors(au\_id),

site\_id INT REFERENCES wwwsites(site\_id));

**Удаление таблицы:**   
**DROP TABLE <имя\_таблицы>**

**Модификация таблицы:**

|  |  |
| --- | --- |
| Добавить столбцы | **ALTER TABLE <имя\_таблицы> ADD**    **(<имя\_столбца> <тип\_столбца>**    **[NOT NULL]**    **[UNIQUE | PRIMARY KEY]**    **[REFERENCES <имя\_мастер\_таблицы> [<имя\_столбца>]]**    **,...)** |
| Удалить столбцы | **ALTER TABLE <имя\_таблицы> DROP (<имя\_столбца>,...)** |
| Модификация типа столбцов | **ALTER TABLE <имя\_таблицы> MODIFY**    **(<имя\_столбца> <тип\_столбца>**    **[NOT NULL]**    **[UNIQUE | PRIMARY KEY]**    **[REFERENCES <имя\_мастер\_таблицы> <имя\_столбца>]]**    **,...)** |

**4.6.3.****DDL: Операторы создания индексов.**

**Создание индекса:**

**CREATE [UNIQUE] INDEX <имя\_индекса> ON <имя\_таблицы> (<имя\_столбца>,...)**

Эта команда создает индекс с заданным именем для таблицы <имя\_таблицы> по столбцам, входящим в список, указанный в скобках. Индекс часто представляет из себя структуру типа B-дерева (см. [параграф 1.2](http://www.mstu.edu.ru/study/materials/zelenkov/ch_1_2.html)), но могут использоваться и другие структуры. Создание индексов значительно ускоряет работу с таблицами. В случае указания необязательного параметра **UNIQUE** СУБД будет проверять каждое значение индекса на уникальность.

Очень часто встает вопрос, какие поля необходимо индексировать. Обязательно надо строить индексы для первичных ключей, поскольку по их значениям осуществляется доступ к данным при операциях соединения двух и более таблиц. Также в ответе на этот вопрос поможет анализ наиболее частых запросов к базе данных. Например, для БД **publications** можно ожидать, что одним из наиболее частых запросов будет выборка всех публикаций данного автора. Для минимизации времени этого запроса необходимо посроить индекс для таблицы **authors** по именам авторов:

     CREATE INDEX au\_names ON authors (author);

Создание индексов для первичных ключей:

     CREATE INDEX au\_index ON authors (au\_id);   
     CREATE INDEX title\_index ON titles (title\_id);   
     CREATE INDEX pub\_index ON publishers (pub\_id);   
     CREATE INDEX site\_index ON wwwsites (site\_id);

Первоначальное определение структуры индексов производится разработчиком на стадии создания прикладной системы. В дальнейшем она уточняется администратором системы по результатам анализа ее работы, учета наиболее часто выполняющихся запросов и т.д.

**Удаление индекса:**

**DROP INDEX <имя\_индекса>**

**4.6.4.****DDL: Операторы управления правами доступа.**

По соображениям безопасности не каждому пользователю прикладной системы может быть разрешено получать информацию из какой-либо таблицы, а тем более изменять в ней данные. Для определения прав пользователей относительно объектов базы данных (таблицы, представления, индексы) в **SQL** определена пара команд GRANT и REVOKE. Синтаксис операции передачи прав на таблицу:

**GRANT <тип\_права\_на\_таблицу>**

**ON <имя\_таблицы> [<список\_столбцов>]**

**TO <имя\_пользователя>**

Права пользователя на уровне таблицы определяются следующими ключевыми словами (как мы увидим чуть позже эти ключевые слова совпадают с командами выборки и изменения данных):

* SELECT - получение информации из таблицы
* UPDATE - изменение информации в таблице
* INSERT - добавление записей в таблицу
* DELETE - удаление записей из таблицы
* INDEX - индексирование таблицы
* ALTER - изменение схемы определения таблицы
* ALL - все права

В поле <тип\_права\_на\_таблицу> может быть указано либо ключевое слово ALL или любая комбинация других ключевых слов. Например, предоставим все права на таблицу **publishers** пользователю **andy**:

GRANT ALL ON publishers TO andy;

Пользователю **peter** предоставим права на извлечение и дбавление записей на эту же таблицу:

GRANT SELECT INSERT ON publishers TO peter;

В том случае, когда одинаковые права надо предоставить сразу всем пользователям, вместо выполнения команды GRANT для каждого из них, можно вместо имени пользователя указать ключевое слово PUBLIC:

GRANT SELECT ON publishers TO PUBLIC;

Отмена прав осуществляется командой REVOKE:

**REVOKE <тип\_права\_на\_таблицу>**

**ON <имя\_таблицы> [<список\_столбцов>]**

**FROM <имя\_пользователя>**

Все ключевые слова данной команды эквивалентны оператору GRANT.

Большинство систем поддерживают также команду GRANT для назначения привилегий на базу данных в целом. В этом случае формат команды:

**GRANT <тип\_права\_на\_базу\_данных>**

**ON <имя\_базы данных>**

**TO <имя\_пользователя>**

К сожалению, способы задания прав на базу данных различны для разных СУБД, и точную их формулировку нужно уточнять в документации. В качестве примера приведем список прав на базу данных, поддерживаемых СУБД *Informix*:

* CONNECT - права на доступ к данным и их модификацию, если это разрешено на уровне таблицы;
* RESOURCE - права на управление ресурсами. Все перечисленное выше плюс права на создание новых объектов (таблиц, индексов и т.д.) и удаление и изменение тех объектов, которыми данный пользователь владеет;
* DBA - права на администрирование. Все права на управление ресурсами плюс права на удаление базы данных, удаление любых объектов, назначение и отмена прав других пользователей.

Отмена прав на базу данных осуществляется командой:

**REVOKE <тип\_права\_на\_базу\_данных> FROM <имя\_пользователя>**

**4.6.5.****DML: Команды модификации данных.**

К этой группе относятся операторы добавления, изменения и удаления записей.

**Добавить новую запись в таблицу:**

**INSERT INTO <имя\_таблицы> [ (<имя\_столбца>,<имя\_столбца>,...) ]**

**VALUES (<значение>,<значение>,..)**

Список столбцов в данной команде не является обязательным параметром. В этом случае должны быть указаны значения для всех полей таблицы в том порядке, как эти столбцы были перечислены в команде CREATE TABLE, например:

     INSERT INTO publishers VALUES (16,"Microsoft Press","http://www.microsoft.com");

Пример с указанием списка столбцов:

     INSERT INTO publishers (publisher,pub\_id)

            VALUES ("Super Computer Publishing",17);

**Модификация записей:**

**UPDATE <имя\_таблицы> SET <имя\_столбца>=<значение>,...**

**[WHERE <условие>]**

Если задано ключевое слово WHERE и условие, то команда UPDATE применяется только к тем записям, для которых оно выполняется. Если условие не задано, UPDATE применяется ко всем записям. Пример:

     UPDATE publishers SET url="http://www.superpub.com" WHERE pub\_id=17;

В качестве условия используются логические выражения над константами и полями. В условиях допускаются:

* операции сравнения: > , < , >= , <= , = , <> , != . В SQL эти операции могут применяться не только к числовым значениям, но и к строкам ( "<" означает раньше, а ">" позже в алфавитном порядке) и датам ( "<" раньше и ">" позже в хронологическом порядке).
* оперции проверки поля на значение NULL: IS NULL, IS NOT NULL
* операции проверки на вхождение в диапазон: BETWEEN и NOT BETWEEN.
* операции проверки на вхождение в список: IN и NOT IN
* операции проверки на вхождение подстроки: LIKE и NOT LIKE
* отдельные операции соединяются связями AND, OR, NOT и группируются с помощью скобок.

Подробно все эти ключевые слова будут описаны и проиллюстрированы в параграфе, посвященном оператору SELECT. Здесь мы ограничимся приведением несложного примера:

     UPDATE publishers SET url="url not defined" WHERE url IS NULL;

Эта команда находит в таблице **publishers** все неопределенные значения столбца **url** и заменяет их строкой "url not defined".

**Удаление записей**

**DELETE FROM <имя\_таблицы> [ WHERE <условие> ]**

Удаляются все записи, удовлетворяющие указанному условию. Если ключевое слово WHERE и условие отстутствуют, из таблицы удаляются все записи. Пример:

    DELETE FROM publishers WHERE publisher = "Super Computer Publishing";

Эта команда удаляет запись об издательстве Super Computer Publishing.

**4.6.6.****DML: Выборка данных.**

Для извлечения записей из таблиц в SQL определен оператор **SELECT**. С помощью этой команды осуществляется не только операция реляционной алгебры "выборка" (горизонтальное подмножество), но и предварительное соединение (join) двух и более таблиц. Это наиболее сложное и мощное средство SQL, полный синтаксис оператора **SELECT** имеет вид:

**SELECT [ALL | DISTINCT] <список\_выбора>**

**FROM <имя\_таблицы>, ...**

**[ WHERE <условие> ]**

**[ GROUP BY <имя\_столбца>,... ]**

**[ HAVING <условие> ]**

**[ORDER BY <имя\_столбца> [ASC | DESC],... ]**

Порядок предложений в операторе **SELECT** должен строго соблюдаться (например, GROUP BY должно всегда предшествовать ORDER BY), иначе это приведет к появлению ошибок.

Мы начнем рассмотрение **SELECT** с наиболее простых его форм. Все примеры, приведенные ниже, касающиеся базы данных [publications](http://www.mstu.edu.ru/study/materials/zelenkov/struct.html#_blank), можно выполнить самостоятельно, зайдя на [эту страничку](http://www.mstu.edu.ru/study/materials/zelenkov/sql/index.html#_blank), поэтому результаты запросов здесь не приводятся.

Этот оператор всегда начинается с ключевого слова SELECT. В кострукции <список\_выбора> определяется столбец или столбцы, включаемые в результат. Он может состоять из имен одного или нескольких столбцов, или из одного символа \* (звездочка), определяющего все столбцы. Элементы списка разделяются запятыми.

Пример: получить список всех авторов

        SELECT author FROM authors;

получить список всех полей таблицы **authors**:

        SELECT \* FROM authors;

В том случае, когда нас интересуют не все записи, а только те, котрые удовлетворяют некому условию, это условие можно указать после ключевого слова WHERE. Например, найдем все книги, опубликованные после 1996 года:

        SELECT title FROM titles WHERE yearpub > 1996;

Допустим теперь, что нам надо найти все публикации за интервал 1995 - 1997 гг. Это условие можно записать в виде:

        SELECT title FROM titles WHERE yearpub>=1995 AND yearpub<=1997;

Другой вариант этой команды можно получить с использованием логической операции проверки на вхождение в интервал:

        SELECT title FROM titles WHERE yearpub BETWEEN 1995 AND 1997;

При использовании конструкции NOT BETWEEN находятся все строки, не входящие в указанный диапазон.

Еще один вариант этой команды можно построить с помощью логической операции проверки на вхождение в список:

SELECT title FROM titles WHERE yearpub IN (1995,1996,1997);

Здесь мы задали в явном виде список интересующих нас значений. Конструкция NOT IN позволяет найти строки, не удовлетворяющие условиям, перечисленным в списке.

Наиболее полно преимущества ключевого слова IN проявляются во вложенных запросах, также называемых подзапросами. Предположим, нам нужно найти все издания, выпущенные компанией "Oracle Press". Наименования издательских компаний содержатся в таблице **publishers**, названия книг в таблице **titles**. Ключевое слово NOT IN позволяет объединить обе таблицы (без получения общего отношения) и извлечь при этом нужную информацию:

SELECT title FROM titles WHERE pub\_id IN

(SELECT pub\_id FROM publishers WHERE publisher='Oracle Press');

При выполнении этой команды СУБД вначале обрабатывает вложенный запрос по таблице **publishers**, а затем его результат передает на вход основного запроса по таблице **titles**.

Некоторые задачи нельзя решить с использованием только операторов сравнения. Например, мы хоти найти web-site издательтва "Wiley", но не знаем его точного наименования. Для решения этой задачи предназначено ключевое слово LIKE, его синтаксис имеет вид:

   WHERE <имя\_столбца> LIKE <образец> [ ESCAPE <ключевой\_символ> ]

Образец заключается в кавычки и должен содержать шаблон подстроки для поиска. Обычно в шаблонах используются два символа:

* % (знак процента) - заменяет любое количество символов
* \_ (подчеркивание) - заменяет одиночный символ.

Попробуем найти искомый web-site:

SELECT publiser, url FROM publishers WHERE publisher LIKE '%Wiley%';

В соотвествии с шаблоном СУБД найдет все строки включающие в себя подстроку "Wiley". Другой пример: найти все книги, название которых начинается со слова "SQL":

       SELECT title FROM titles WHERE title LIKE 'SQL%';

В том случае, когда надо найти значение, которое само содержит один из символов шаблона, используют ключевое слово ESCAPE и <ключевой\_символ>. Литерал, следующий в шаблоне после ключевого символа, рассматривается как обычный символ, все последующие символы имеют обычное значение. Например, нам надо найти ссылку на web-страницу, о которой известно, что в ее url содержится подстрока "my\_works":

      SELECT site, url FROM wwwsites WHERE url LIKE '%my@\_works%' ESCAPE '@';

В заключение заметим, что при выполнении оператора SELECT результирующее отношение может иметь несколько записей с одинаковыми значениями всех полей. Чтобы исключить повторяющиеся записи из выборки используется ключевое слово DISTINCT. Ключевое слово ALL указывает, что в результат необходимо включать все строки.

**4.6.7.****DML: Выборка из нескольких таблиц.**

Очень часто возникает ситуация, когда выборку данных надо производить из отношения, которое является результатом слияния (join) двух других отношений. Например, нам нужно получить из базы данных **publications** информацию о всех печатных изданиях в виде следующей таблицы:

------------------------------------------------

|название\_книги | год\_выпуска | издательство |

------------------------------------------------

| | | |

| | | |

Для этого СУБД предварительно должна выполнить слияние таблиц **titles** и **publishers**, а только затем произвести выборку из полученного отношения.

Для выполнения операции такого рода в операторе SELECT после ключевого слова FROM указывается список таблиц, по которым произвоится поиск данных. После ключевого слова WHERE указывается условие, по которому производится слияние. Для того, чтобы выполнить данный запрос, нужно дать команду:

    SELECT titles.title,titles.yearpub,publishers.publisher

        FROM titles,publishers

        WHERE titles.pub\_id=publishers.pub\_id;

А вот пример, где одновременно задаются условия и слияния, и выборки (результат предыдущего запроса ограничивается изданиями после 1996 года):

    SELECT titles.title,titles.yearpub,publishers.publisher

        FROM titles,publishers

        WHERE titles.pub\_id=publishers.pub\_id AND

              titles.yearpub>1996;

Следует обратить внимание на то, что когда в разных таблицах присутствуют одноименные поля, то для устранения неоднозначности перед именем поля указывается имя таблицы и знак "." (точка). (Хорошее правило: имя таблицы указывать всегда!)

Естественно, имеется возможность производить слияние и более чем двух таблиц. Например, чтобы дополнить описанную выше выборку именами авторов книг необходимо составить оператор следующего вида:

SELECT authors.author,titles.title,titles.yearpub,publishers.publisher

FROM titles,publishers,titleauthors

WHERE titleauthors.au\_id=authors.au\_id AND

titleauthors.title\_id=titles.title\_id AND

titles.pub\_id=publishers.pub\_id AND

titles.yearpub > 1996;

**4.6.8.****DML: Вычисления внутри SELECT.**

SQL позволяет выполнять различные арифметические операции над столбцами результирующего отношения. В конструкции <список\_выбора> можно использовать константы, функции и их комбинации с арифметическими операциями и скобками. Например, чтобы узнать сколько лет прошло с 1992 года (год принятия стандарта SQL-92) до публикации той или иной книги можно выполнить команду:

     SELECT title, yearpub-1992 FROM titles WHERE yearpub > 1992;

В арифметических вражения допускаются операции сложения (+), вычитания (-), деления (/), умножения (\*), а также различные функции (COS, SIN, ABS - абсолютное значение и т.д.). Также в запрос можно добавить строковую константу:

     SELECT 'the title of the book is', title, yearpub-1992

            FROM titles WHERE yearpub > 1992;

В SQL также определены так называемые агрегатные функции, которые совершают действия над совокупностью одинаковых полей в группе записей. Среди них:

* **AVG(<имя поля>)** - среднее по всем значениям данного поля
* **COUNT(<имя поля>)** или **COUNT (\*)** - число записей
* **MAX(<имя поля>)** - максимальное из всех значений данного поля
* **MIN(<имя поля>)** - минимальное из всех значений данного поля
* **SUM(<имя поля>)** - сумма всех значений данного поля

Следует учитывать, что каждая агрегирующая функция возвращает единственное значение. Примеры: определить дату публикации самой "древней" книги в нашей базе данных

SELECT MIN(yearpub) FROM titles;

подсчитать количество книг в нашей базе данных:

SELECT COUNT(\*) FROM titles;

Область действия данных функции можно ограничить с помощью логического условия. Например, количество книг, в названии которых есть слово "SQL":

SELECT COUNT(\*) FROM titles WHERE title LIKE '%SQL%';

**4.6.9.****DML: Групировка данных.**

Группировка данных в операторе SELECT осуществляется с помощью ключевого слова GROUP BY и ключевого слова HAVING, с помощью которого задаются условия разбиения записей на группы.

GROUP BY неразрывно связано с агрегирующими функциями, без них оно практически не используется. GROUP BY разделяет таблицу на группы, а агрегирующая функция вычисляет для каждой из них итоговое значение. Определим для примера количество книг каждего издательства в нашей базе данных:

SELECT publishers.publisher, count(titles.title)

FROM titles,publishers

WHERE titles.pub\_id=publishers.pub\_id

GROUP BY publisher;

Kлючевое слово HAVING работает следующим образом: сначала GROUP BY разбивает строки на группы, затем на полученные наборы накладываются условия HAVING. Например, устраним из предыдущего запроса те издательства, которые имеют только одну книгу:

SELECT publishers.publisher, count(titles.title)

FROM titles,publishers

WHERE titles.pub\_id=publishers.pub\_id

GROUP BY publisher

HAVING COUNT(\*)>1;

Другой вариант использования HAVING - включить в результат только те издательтва, название которых оканчивается на подстроку "Press":

SELECT publishers.publisher, count(titles.title)

FROM titles,publishers

WHERE titles.pub\_id=publishers.pub\_id

GROUP BY publisher

HAVING publisher LIKE '%Press';

В чем различие между двумя этими вариантами использования HAVING? Во втором варианте условие отбора записей мы могли поместить в раздел ключевого слова WHERE, в первом же варианте этого сделать не удасться, поскольку WHERE не допускает использования агрегирующих функций.

**4.6.10****.DML: Cортировка данных.**

Для сортировки данных, получаемых при помощи оператора SELECT служит ключевое слово ORDER BY. С его помощью можно сортировать результаты по любому столбцу или выражению, указанному в <списке\_выбора>. Данные могут быть упорядочены как по возрастанию, так и по убыванию. Пример: сортировать список авторов по алфавиту:

SELECT author FROM authors ORDER BY author;

Более сложный пример: получить список авторов, отсортированный по алфавиту, и список их публикаций, причем для каждого автора список книг сортируется по времени издания в обратном порядке (т.е. сначала более "свежие" книги, затем все более "древние"):

SELECT authors.author,titles.title,titles.yearpub,publishers.publisher

FROM authors,titles,publishers,titleauthors

WHERE titleauthors.au\_id=authors.au\_id AND

titleauthors.title\_id=titles.title\_id AND

titles.pub\_id=publishers.pub\_id

ORDER BY authors.author ASC, titles.yearpub DESC;

Ключевое слово DESC задает здесь обратный порядок сортировки по полю **yearpub**, ключевое слов ASC (его можно опускать) - прямой порядок сортировки по полю **author**.

**4.6.11.****DML: Операция объединения.**

В SQL предусмотрена возможность выполнения операции реляционной алгебры "ОБЪЕДИНЕНИЕ" (UNION) над отношениями, являющимися результатами оператора SELECT. Естественно, эти отношения должны быть определены по одной схеме.Пример: получить все Интеренет-ссылки, хранимые в базе данных **publications**. Эти ссылки хранятся в таблицах **publishers** и **wwwsites**. Для того, чтобы получить их в одной таблице, мы должны построить следующие запрос:

SELECT publisher,url FROM publishers

UNION

SELECT site,url FROM wwwsites;

**4.6.12.****Использование представлений.**

До сих пор мы говорили о таблицах, которые *реально* хранятся в базе данных. Это, так называемые, базовые таблицы (base tables). Существует другой вид таблиц, получивший название "представления" (иногда их называют"представляемые таблицы").

**Определение**:  
Представление (view) - это таблица, содержимое которой берется из других таблиц посредством запроса. При этом новые копии данных не создаются

Когда содержимое базовых таблиц меняется, СУБД автоматически перевыполняет запросы, создающие view, что приводит к соответствующи изменениям в представлениях.

Представление определяется с помощью команды

**CREATE VIEW <имя\_представления> [<имя\_столбца>,...]**

**AS <запрос>**

При этом должны соблюдаться следующие ограничения:

* представление должно базироваться на единcтвенном запросе (UNION не допустимо)
* выходные данные запроса, формирующего представление, должны быть не упорядочены (ORDER BY не допустимо)

Создадим представление, хранящее информацию об авторах, их книгах и издателях этих книг:

CREATE VIEW books AS

SELECT authors.author,titles.title,titles.yearpub,publishers.publisher

FROM authors,titles,publishers,titleauthors

WHERE titleauthors.au\_id=authors.au\_id AND

titleauthors.title\_id=titles.title\_id AND

titles.pub\_id=publishers.pub\_id

Теперь любой пользователь, чьих прав на доступ к данному представлению достаточно, может осуществлять выборку данных из **books**. Например:

SELECT titles FROM books WHERE author LIKE '%Date'

SELECT author,count(title) FROM books GROUP BY author

(Права пользователей на доступ в представлениям назначаются также с помощью команд GRANT / REVOKE.)

Из приведенного выше примера достаточно ясен смысл использования представлений. Если запросы типа "выбрать все книги данного автора с указанием издательств" выполняются достаточно часто, то создание представляемой таблицы **books** значительно сократит накладные расходы на выполнение соединеия четырех базовых таблиц **authors**, **titles**, **publishers** и **titleauthors**. Кроме того, в представлении может быть представлена информация, явно не хранимая ни в одной из базовых таблиц. Например, один из столбцов представления может быть вычисляемым:

CREATE VIEW amount (publisher, books\_count) AS

SELECT publishers.publisher, count(titles.title)

FROM titles,publishers

WHERE titles.pub\_id=publishers.pub\_id

GROUP BY publisher;

Здесь использована еще одна, ранее не описанная, возможность SQL - присвоение новых имен столбцам представления. В приведенном примере число изданий, осуществленных каждым издатетлем, будет хранится в столбце с именем **books\_count**. Заметим, что если мы хотим присвоить новые имена столбцам представления, нужно указывать имена для *всех* столбцов. Тип данных столбца представления и его нулевой статус всегда зависят от того, как он был определен в базовой таблице (таблицах).

Запрос на выборку данных к представлению выглядит абсолютно аналогично запросу к любой другой таблице. Однако на изменение данных в представлении накладываются ограничения. Кратко о них можно сказать следующее:

* Если представление основано на одной таблице, изменения данных в нем допускаются. При этом изменяются данные в связанной с ним таблице.
* Если представление основано более чем на одной таблице, то изменения данных в нем не допускаются, т.к. в большинстве случаев СУБД не может правильно восстановить схему базовых таблиц из схемы представления.

Удаление представления производится с помощью оператора:

**DROP VIEW <имя\_представления>**

**4.6.13.****Другие возможности SQL.**

Описываемые ниже возможности пока не стандартизованы, но представлены в той или иной мере практически во всех современных СУБД.

* **Хранимые процедуры**. Практический опыт создания приложений обработки данных показывает, что ряд операций над данными, реализующих общую для всех пользователей логику и не связанных с пользовательским интерфейсом, целесообразно вынести на сервер. Однако, для написания процедур, реализующих эти операции стандартных возможностей SQL не достаточно, поскольку здесь необходимы операторы обработки ветвлений, циклов и т.д. Поэтому многие поставщики СУБД предлагают собственные **процедурные** расширения SQL (PL/SQL компании Oracle и т.д.). Эти расширения содержат логические операторы (IF ... THEN ... ELSE), операторы перехода по условию (SWITCH ... CASE ...), операторы циклов (FOR, WHILE, UNTIL) и операторы предачи управления в процедуры (CALL, RETURN). С помощью этих средств создаются функциональные модули, которые хранятся на сервере вместе с базой данных. Обычно такие модули называют *хранимые процедуры*. Они могут быть вызваны с передачей параметров любым пользователем, имеющим на то соотвествующие права. В некоторых системах хранимые процедуры могут быть реализованы и в виде внешних по отношению к СУБД модулей на языках общего назначения, таких как *C* или *Pascal*. Пример для СУБД PostgreSQL:
* **CREATE FUNCTION <имя\_функции> ([<тип\_параметра1>,...<тип\_параметра2>])**
* **RETURNS <возвращаемые\_типы>**
* **AS [ <SQL\_оператор> | <имя\_объектного\_модуля> ]**

**LANGUAGE 'SQL' | 'C' | 'internal'**

Вызов созданной функции осуществялется из оператора SELECT (также, как вызываются функции агрегирования). Более подробно о хранимых процедурах см. статью Э.Айзенберга [Новый стандарт хранимых процедур в языке SQL](http://www.osp.ru/dbms/), СУБД N 5-6, 1996 г.

* **Триггеры**. Для каждой таблицы может быть назначена хранимая процедура без параметров, которая вызывается при выполнении оператора модификации этой таблицы (INSERT, UPDATE, DELETE). Такие хранимые процедуры получили название триггеров. Триггеры выполняются автоматически, независимо от того, что именно является причиной модификации данных - действия человека оператора или прикладной программы. "Усредненный" синтаксис оператора создания триггера:
* **CREATE TRIGGER <имя\_триггера>**
* **ON <имя\_таблицы>**
* **FOR { INSERT | UPDATE | DELETE }**
* **[, INSERT | UPDATE | DELETE ] ...**

**AS <SQL\_оператор>**

Ключевое слово ON задает имя таблицы, для которой определяется триггер, ключевое слово FOR указывает какая команда (команды) модификации данных активирует триггер. Операторы SQL после ключевого слова AS описывают действия, которые выполняет триггер и условия выполнения этих действий. Здесь может быть перечислено любое число операторов SQL, вызовов хранимых процедур и т.д. Использование триггеров очень удобно для выполнения операций контроля ограничений целостности (см. [главу 4.3](http://www.mstu.edu.ru/study/materials/zelenkov/ch_4_3.html)).

* **Мониторы событий**. Ряд СУБД допускает создание таких хранимых процедур, которые непрерывно сканируют одну или несколько таблиц на предмет обнаружения тех или иных событий (например, среднее значение какого-либо столбца достигает заданного предела). В случае наступления события может быть инициирован запуск триггера, хранимой процедуры, внешнего модуля и т.п. Пример: пусть наша база данных является частью автоматизированной системы управления технологическим процессом. В поле одной из таблиц заносятся показания датчика температуры, установленного на резце токарного станка. Когда это значение превышает заданный предел, запускается внешняя программа, изменяющая параметры работы станка.