**Лабораторная работа №4**

## Основы SQL. Типы данных. (Oracle Database 11g Express Edition)

|  |
| --- |
| Настройка SQL\*Plus. Необходимо включить режим ECHO и вывести протокол лаб. работы в файл *<Фамилия студента>.txt*. Этот файл является отчетом о проделанной лаб. работе.  SQL> set echo on  SQL> spool c:\spool. txt  ……………………………………..  Завершить протокол лаб. работы (команда spool off) и направить результаты преподавателю.  SQL> spool off |

## Основные команды SQL

Язык SQL состоит из операторов языка, называемых иногда командами языка SQL, типов данных, набора встроенных функций.

По своему логическому назначению операторы языка SQL часто разбиваются на следующие группы:

* + язык определения данных **DDL** (**Data Definition Language**);
  + язык манипулирования данными **DML** (**Data Manipulation Language**);
  + язык управления транзакциями **TCL** (**Transaction Control Language**);
  + язык доступа к данным **DCL** (**Data Control Language**).

Основные типы данных ORACLE используемые в SQL:

|  |  |
| --- | --- |
| **Тип данных** | **Описание типа данных** |
| **Blob** | Тип данных BLOB, содержащий двоичные данные максимального размера 4 гигабайта. |
| **Byte** | Целый тип для представления 8-разрядных целых чисел без знака, находящихся в диапазоне от 0 до 255. |
| **Char** | Тип данных Oracle CHAR, содержащий строку знаков фиксированной длины с максимальным размером 2000 байтов. |
| **Clob** | Тип данных CLOB, содержащий символьные данные, основан на стандартном наборе знаков на сервере, максимальный размер которого — 4 гигабайта. |
| **Date** | Тип данных DATE хранит значения в виде точек времени (т.е. дату и время). Тип данных DATE запоминает год (включая век), месяц, день, часы, минуты и секунды (после полуночи). ORACLE может хранить даты в диапазоне от 1 января 4712 года до н.э. до 31 декабря 4712 года нашей эры. Стандартный формат даты ORACLE для ввода и вывода имеет вид DD-MON-YY |
| **LongRaw** | Тип данных LONGRAW, содержащий двоичные данные переменной длины с максимальным размером 2 гигабайта. |
| **NChar** | Тип данных NCHAR, содержащий строку знаков фиксированной длины для хранения в национальном наборе знаков в базе данных максимального размера 2000 байтов (а не знаков). |
| **Number** | Тип данных NUMBER, содержащий числовые данные переменной длины с максимальной точностью и масштабом 38. |
| **Timestamp** | Тип данных TIMESTAMP, содержащий дату и время (включая секунды), с размером в диапазоне от 7 до 11 байтов. |
| **TimestampLocal** | Тип данных TIMESTAMP WITH LOCAL TIMEZONE содержащий дату, время и ссылку на исходный часовой пояс, с размером в диапазоне от 7 до 11 байтов |
| **VarChar2** | Тип данных VARCHAR2, содержащий строку знаков переменной длины с максимальным размером 4000 байтов |

### ЯЗЫК ОПРЕДЕЛЕНИЯ ДАННЫХ (DDL - Data Definition Language)

Язык определения данных (**DDL**) включает операторы, управляющие объектами базы данных. К объектам БД относятся таблицы, индексы, представления. Для каждой конкретной базы данных существует свой набор объектов базы данных, который может значительно расширять набор объектов, предусмотренный стандартом. В СУБД Oracle все объекты базы данных, принадлежащие одному пользователю, образуют схему базы данных.

Язык определения данных включает в себя три основные команды:

* + **CREATE** – создание объекта;
  + **ALTER** – изменение объекта;
  + **DROP** – удаление объекта.

В качестве объекта здесь может быть: таблица, представление, триггер, индекс, ограничение целостности, процедура, функция, пакет, объектные пользовательские типы данных, последовательности и т.д., т.е. фактически весь набор типов объектов СУБД Oracle. Важно отметить, что управление все параметрами работы СУБД осуществляется командой ALTER. Для каждого типа объекта БД и команды DDL существует определенный синтаксис и набор опций. Общий синтаксис для команд DDL выглядит так:

***Команда Объект Имя\_объекта Набор\_опций\_и\_операций***

***КОМАНДА СОЗДАНИЯ ТАБЛИЦЫ – CREATE TABLE.***

Оператор CREATE TABLE создает таблицу в базе данных. Созданная таблица будет пуста (если не был указан запрос в предложении AS). Строки данных, как правило, добавляются в таблицу с помощью оператора INSERT. В команде CREATE TABLE определяется имя таблицы и набор столбцов. Она также определяет типы данных и размеры столбцов. Каждая таблица должна иметь, по крайней мере, один столбец. Знак подчеркивания «\_» обычно используется для разделения слов в именах таблиц.

**Синтаксис создания таблицы:**

CREATE TABLE пользователь.имя\_таблицы

( описание\_столбца | эл-т\_таблицы,

описание\_столбца | эл-т\_таблицы,...

) [AS запрос]

**Пользователь *-*** владелец создаваемой таблицы.

**описание\_столбца *-*** описание столбцов.

***эл-т\_таблицы -*** описание ограничений для таблицы.

Рассмотрим пример создания таблицы «Физических лиц»:

|  |
| --- |
| CREATE TABLE s\_fiz\_lic  (kod NUMBER(7,0) NOT NULL,  fam VARCHAR2(30),  im VARCHAR2(25),  otch VARCHAR2(25),  p\_vidan VARCHAR2(80),  p\_ser\_nom VARCHAR2(20)  ); |

Порядок столбцов в таблице определяется порядком, в котором они указаны. Имена столбца не должны разделяться при переносе строки (что делается для удобочитаемости) и отделяются запятыми. Ограничение **NOT NULL** означает, что данное поле не может содержать пустое значение.

В нижеприведенном примере производится создание таблицы «Должности», в качестве дополнительных опций таблицы для столбца «KOD» - указывается опция NOT NULL, а так же служебные слова **PRIMARY KEY**, которые определяют, что необходимо создать первичный ключ по данному столбцу.

|  |
| --- |
| CREATE TABLE dolj  (kod NUMBER(10) NOT NULL PRIMARY KEY,  naimen VARCHAR2(50)); |

Кроме стандартного способа создания таблицы, в котором мы определяем структуру таблицы, существует способ, при котором можно создать таблицу на базе ранее созданной таблицы с данными. Структура создаваемой таблицы и данные определяется запросом. В данном примере мы создаем дубликат таблицы «s\_fiz\_lic» с именем «s\_fiz\_lic2».

|  |
| --- |
| CREATE TABLE s\_fiz\_lic2 as  (SELECT \* FROM s\_fiz\_lic); |

Рассмотрим более сложный пример создания таблицы физических лиц с имнем «s\_fiz\_lic3». При создании таблицы в этом примере мы применяем следующие опции для столцов:

* + поле «KOD» - не может быть пустым – NOT NULL, является первичным ключом – PRIMARY KEY;
  + поле «FIO» - не может быть пустым – NOT NULL, проверка вводимых данных в таблицу, значения поле должно вводится в верхнем регистре – CHECK(fio=UPPER(fio));
  + поле «POL» - по умолчанию значение поле равно 1 – DEFAULT 1;
  + поле «KOD\_DOLJ» - создается вторичный ключ с именем «fk\_dolj» ссылвющийся на поле «KOD» таблицы «DOLJ»

|  |
| --- |
| CREATE TABLE s\_fiz\_lic3  (kod NUMBER(10) NOT NULL PRIMARY KEY,  fio VARCHAR2(250) NOT NULL CHECK(fio=UPPER(fio)),  pol NUMBER(1) DEFAULT 1,  kod\_dolj NUMBER(10 ) CONSTRAINT fk\_dolj REFERENCES dolj (kod),  data\_r DATE); |

Просмотреть структуру таблицы можно командой **DESC** <**table\_name**>. Пример, DESC s\_fiz\_lic. В результате на экран будет выведена структруа таблицы:

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Пусто?** | **Тип данных** |
| KOD | NOT NULL | NUMBER(7) |
| FAM |  | VARCHAR2(30) |
| IM |  | VARCHAR2(25) |
| OTCH |  | VARCHAR2(25) |
| P\_VIDAN |  | VARCHAR2(80) |
| P\_SER\_NOM |  | VARCHAR2(20) |

***КОМАНДА СОЗДАНИЯ ПОСЛЕЛОВАТЕЛЬНОСТИ – CREATE SEQUENCE.***

Последовательностью называется объект ORACLE, генерирующий неповторяющиеся целые числа. Полученные из последовательности числа очень часто используются в качестве значений для первичных ключей.

**Синтаксис создания последовательности:**

CREATE SEQUENCE пользователь.имя\_последовательности

INCREMENT BY n

START WITH n

MAXVALUE n | NOMAXVALUE

MINVALUE n | NOMINVALUE

CYCLE | NOCYCLE

ORDER | NOORDER

При использовании последовательности:

* + для выбора следующего номера - используется вызов типа my\_seq.NEXTVAL, где my\_seq – имя последовательности;
  + для выбора текущего номера - используется вызов тип my\_seq.CURRVAL, где my\_seq – имя последовательности.

Рассмотрим пример создания последовательности с именем «my\_seq». В этом примере генерируемое значение последовательности начнется со 100, и будет увеличиваться на 1. Возвращаемые значения являются отсортированными – ORDER, последовательность является циклической – это означает, что при достижении свое максимального значения - 1000, она начнет возвращать значения со стартового значения.

|  |
| --- |
| CREATE SEQUENCE my\_seq  INCREMENT BY 1 START WITH 100  CYCLE MAXVALUE 1000 ORDER; |

В следующем примере приведены способы получения текущего и следующего значения последовательности, для этого используется системная таблица с именем «DUAL».

|  |
| --- |
| SELECT my\_seq.nextval FROM dual;  SELECT my\_seq.currval FROM dual; |

***КОМАНДА СОЗДАНИЯ ПРЕДСТАВЛЕНИЯ – CREATE VIEW.***

Представление есть виртуальная таблица, доступ к которой осуществляется так же как к реально существующей таблице.

**Синтаксис создания представления:**

CREATE VIEW пользователь.имя\_представления

(имя\_столбца1, имя\_столбца2 …)

AS запрос

Оператор CREATE VIEW создает “логическое окно” (виртуальную таблицу, представление данных) для одной или нескольких таблиц или таких представлений.

В нижеприведенном примере мы создаем представление с именем «v\_fiz\_lic», которое содержит все столбцы и данные таблицы «s\_fiz\_lic».

|  |
| --- |
| CREATE VIEW V\_FIZ\_LIC AS  (SELECT \* FROM S\_FIZ\_LIC); |

Рассмотрим более сложный пример создания представления с именем «v\_fiz\_lic2». Создаваемое представление основывается на двух таблицах «Физических лиц» и «Должностей», после служебного слова FROM через запятую перечисляют таблицы, в результате мы пересекаем строки 2-х таблиц и оставляем те записи, у которых значение поля «KOD» таблицы «DOLJ» равно значению поля «KOD\_DOLJ» таблицы «S\_FIZ\_LIC3».

|  |
| --- |
| CREATE VIEW V\_FIZ\_LIC2 (KOD, FIO, POL, DOLJ, DATA\_R) AS  (SELECT d.KOD, d.FIO, d.POL, f.naimen, d.DATA\_R  FROM S\_FIZ\_LIC3 d, dolj f  where d.kod\_dolj=f.kod); |

Данное представление позволяет отобразить список физических лиц и их должности из таблицы «Должности».

***КОМАНДА ИЗМЕНЕНИЯ ТАБЛИЦЫ – ALTER TABLE***

Оператор ALTER TABLE позволяет добавлять в таблицу столбцы и ограничения, изменять размер и тип данных существующих столбцов, изменять установки NULL/NOT NULL, удалять ограничения, изменять схему хранения таблицы.

**Синтаксис изменения таблицы:**

ALTER TABLE имя\_пользователя.таблица

ADD [CONSTRAINT] описание\_столбца | ограничение\_таблицы,

MODIFY [RENAME| CONSTRAINT] описание\_столбца, описание\_столбца,...

DROP COLUMN|CONSTRAINT ограничение, ограничение,...

Рассмотрим примеры использования команды ALTER применительно к таблице. В нижеприведенном примере мы производим добаление столбца «sem\_pol» с типом данных NUMBER. При добавлении столбца можно использовать дополнитльные опции, как при создании таблицы (DEFAULT, NOT NULL и т.д.), кроме командой ALTER можно добавить несколько столбцов одной командой, для этого нужно их перечислить через запятую.

|  |
| --- |
| ALTER TABLE s\_fiz\_lic3 ADD sem\_pol NUMBER(1); |

В следующий пример проиллюстрировано, как произвести перемименование столбца. В этом прмере меняем имя созданного столбца с «sem\_pol» на «sem\_poloj».

|  |
| --- |
| ALTER TABLE s\_fiz\_lic3 RENAME COLUMN sem\_pol TO sem\_poloj; |

В третьем примере мы производим учеличение размерности «sem\_poloj» с 1 до 10. Небходимо отметить, что если столбец таблицы содержит данные, то мы можем только увеличить значение размерности, уменьшить размерность мы можем в том случаи, если все строки по этому столбу содержат значение NULL.

|  |
| --- |
| ALTER TABLE s\_fiz\_lic3 MODIFY sem\_poloj NUMBER(10); |

В данный пример показывает как проивзодить удаление столбца таблицы.

|  |
| --- |
| ALTER TABLE s\_fiz\_lic3 DROP COLUMN sem\_poloj; |

Для индентификации строк и оптимизации таблицы используют первичный ключ. Первичный ключ может состоять из одного столбца или быть составным (несколько столбцов). Такой уникальный столбец (или уникальная группа столбцов), используемый для идентификации каждой строки и хранения всех строк по отдельности, называется первичным ключом таблицы.

Первичные ключи таблицы - важный элемент в структуре базы данных. Первичный ключ гарантируют, что ваши данные имеют определенную целостность. Если первичный ключ правильно используется и поддерживается, вы будете знать, что нет пустых строк таблицы, и что каждая строка отличается от любой другой строки.

Следующий пример показывает, как нужно добавлять перчиный ключ таблицы. Служебные слова USING INDEX используются опцинально, указывая эти слова, производится автоматическое создание системного индекса по данному столбцу таблицы.

|  |
| --- |
| ALTER TABLE s\_fiz\_lic ADD PRIMARY KEY (KOD) USING INDEX; |

В следующем примере мы производим удаление вторичного (внешнего) ключа.

|  |
| --- |
| ALTER TABLE s\_fiz\_lic3 DROP CONSTRAINT fk\_dolj; |

В следующем примере мы производим создание вторичного ключа, которые представляет собой ограничение целостности для изменяемой таблицы. Вторичный ключ определяет перечень тех значений, которые будет принимать столбец изменяемой таблицы. Общий синтаксис:

**ALTER** **TABLE** имя\_таблицы **ADD** **CONSTRAINT** имя\_вторичного\_ключа **FOREIGN** **KEY** (список\_столбцов) **REFERENCES** имя\_родит\_таблицы (список\_столбцов);

При создании вторичного (внешенго) ключа необходимо учесть, что тип данных и размерность столбцов обеих таблиц должны совпадать. У родительской таблицы (таблица, которая определяет перечень значений) по этому столбцу должен быть создан первичый ключ, данный столбец является родительским по отношению к столбцу другой таблицы.

|  |
| --- |
| ALTER TABLE s\_fiz\_lic3 ADD CONSTRAINT fk\_dolj FOREIGN KEY (kod\_dolj) REFERENCES dolj (KOD); |

Два следующих примера иллюстрируют, как производить включение и отключение вторичного ключа. Часто требуется производить отключением проверки вводимых значений, для этого необходимо произвести отключение вторичного ключа.

|  |
| --- |
| ALTER TABLE s\_fiz\_lic3 MODIFY CONSTRAINT fk\_dolj DISABLE;  ALTER TABLE s\_fiz\_lic3 MODIFY CONSTRAINT fk\_dolj ENABLE; |

***КОМАНДА УДАЛЕНИЯ – DROP***

Оператор DROP производит удаление объекта из БД.

**Синтаксис удаления объекта:**

DROP тип\_объекта имя\_объекта

Рассмотрим примеры удаления объектом БД, а так же опции, которые могут применяться при удалении таблицы. В первом примере произовдится удаление таблицы с именем «s\_fiz\_lic2» при этом таблица отправляется в спецмальную область для хранения удаленных объектов, называемой «Корзина».

|  |
| --- |
| DROP TABLE s\_fiz\_lic2; |

Для удаления таблицы полностью без отправки ее в корзину применяется опция PURGE. Если производится удаление таблицы, на которую ссылается другая таблица, то есть существует ограничение целостности, то такую таблицу нельзя удалить без опции CASCADE CONSTRAINTS, данная опция удалет в месте с таблицей все ссылающиеся на нее ограничения целостности.

|  |
| --- |
| DROP TABLE dolj CASCADE CONSTRAINTS PURGE; |

В следующем примере мы произодим удаление ранее созданного нами представления.

|  |
| --- |
| DROP VIEW v\_fiz\_lic; |

В этом примере производим удаление последовательности с именем «my\_seq».

|  |
| --- |
| DROP SEQUENCE my\_seq; |

Пересоздадим удаленные объекты

|  |
| --- |
| CREATE TABLE dolj  (kod NUMBER(10) NOT NULL PRIMARY KEY,  naimen VARCHAR2(50));  ALTER TABLE s\_fiz\_lic3 ADD CONSTRAINT fk\_dolj FOREIGN KEY (kod\_dolj) REFERENCES dolj (KOD); |

### ЯЗЫК МАНИПУЛИРОВАНИЯ ДАННЫМИ (DML - Data Manipulation Language )

Язык манипулирования данными (**DML**) включает операторы, управляющие содержанием таблиц базы данных и извлекающими информацию из этих таблиц.

**DML** включает в себя четыре основные команды:

* **INSERT** – вставляет данные в таблицу;
* **UPDATE** – обновляет данные таблицы;
* **DELETE** – удаляет данные из таблицы;
* **SELECT** – возвращает (выбирает) данные из объекта базы данных.

***КОМАНДА INSERT***

Оператор **INSERT** предназначен для добавления строк в таблицу или представление данных.

**Синтаксис команды:**

INSERTINTO имя\_таблицы/представления (столбец1, столбец2,...)

VALUES(значение1, значение2,...);

INSERT INTO имя\_таблицы/представления (столбец1, столбец2,...) SELECT...;

Имена в списке столбцов могут быть перечислены в любом порядке. В столбцы, не указанные в списке, заносится пустое значение. Все столбцы с признаком NOT NULL должны быть указанны, и иметь предназначающиеся для них значения. В предложении VALUES перечисляются конкретные значения столбцов в добавляемой строке. Каждый указанный столбец должен иметь соответствующее ему значение в предложении VALUES. Типы данных значения и столбца должен быть совместимы или преобразуемы. Значения типа CHAR, VARCHAR2 и DATE надо заключать в апострафы (‘абв’). Чтобы добавить строки из другой таблицы, следует использовать подзапрос. Оператор SELECT в этом подзапросе должен извлекать значения для каждого перечисленного столбца.

Рассмотрим примеры вставки данных в таблицы. В первом примере мы производим вставку данных во всех столбцы таблицы и в том порядке, в каком она была создана, поэтому здесь мы можем опустить спецификацию столбцов для вставки.

|  |
| --- |
| INSERT INTO s\_fiz\_lic VALUES (1,'Иванов','Иван','Иванович','ПВС Западного округа','3310 124568');  INSERT INTO s\_fiz\_lic VALUES (2,'Петров','Петр','Петрович', 'ПВС Центрального округа','3250 145868');  INSERT INTO s\_fiz\_lic VALUES (3,'Сидоров','Иван','Иванович', 'ПВС Западного округа','7810 124879');  INSERT INTO dolj VALUES (1, ‘Слесарь’);  INSERT INTO dolj VALUES (2, ‘Программист’);  INSERT INTO dolj VALUES (3, ‘Бухгалтер’); |

В следующем примере в последние два столбца вставляем NULL значения, для того чтобы указать, что у нас отсутствуют данные.

|  |
| --- |
| INSERT INTO s\_fiz\_lic VALUES (4, ’Иванов’, ’Иван’,’Иванович’,NULL,NULL); |

В данном примере мы производим аналогичную вставку, но со спецификацией столбцов, которая определяет в каком порядке вставлять данные.

|  |
| --- |
| INSERT INTO s\_fiz\_lic (kod,im,otch,fam, p\_vidan,p\_ser\_nom) VALUES (5, 'Иван', 'Иванович', 'Иванов', 'ПВС Западного округа', '3310 124568'); |

Рассмотрим пример вставки данных с использованием запроса. В этом примере мы создаем дублирующие данные таблицы «s\_fiz\_lic», при этом для всех выбранных строк из таблицы, мы увеличиваем значение поля «kod» на 5. В рельзультате в таблице мы получим дубликаты строк физизческих лиц, но с разными значениями в поле «kod». В последней строке мы производим фиксацию транзакции.

|  |
| --- |
| INSERT INTO s\_fiz\_lic (SELECT kod+5,im,otch,fam, p\_vidan,p\_ser\_nom FROM s\_fiz\_lic);  COMMIT; |

В следующем примере мы вставляем данные из таблицы «s\_fiz\_lic» в таблицу «s\_fiz\_lic3».

|  |
| --- |
| INSERT INTO s\_fiz\_lic3 (kod, fio) (SELECT kod, UPPER(fam||’ ’||im||’ ’||otch) FROM s\_fiz\_lic);  COMMIT; |

***КОМАНДА UPDATE***

Оператор **UPDATE** заменяет значения одного или нескольких указанных столбцов на значения выражений или результат запроса.

**Синтаксис команды:**

UPDATE таблица/имя\_предст\_данных SET

столбец\_имя = выражение

столбец\_имя = (SELECT\_с\_одним\_результатом)

(столбец\_имя, столбец\_имя,...) = оператор\_SELECT

WHERE\_предложение;

Оператор SELECT в этом запросе должен возвращать как минимум одну строку и обеспечивать значения для каждого столбца, стоящего слева от знака =. Этот оператор SELECT не может содержать фразы INTO. Для определения набора строк, подлежащих обновлению, используется предложение WHERE. В нем указываются условия, которым должна отвечать обновляемая строка. Если предложение WHERE опустить, то будут обновлены все строки.

Рассмотрим примеры обновления данных. В первом примере строке с кодом 5, мы устанавливаем новые значения поля «fam», «im», «otch».

|  |
| --- |
| UPDATE s\_fiz\_lic SET Fam=‘Сидоров’, im=’Петр’, otch=’Иванович’ WHERE kod=5; |

В следующем примере мы соединяем значения полей «fam», «im», «otch» в верхнем регистре строки с кодом 5 из таблицы «s\_fiz\_lic3» и присваиваем его полю «fio» таблицы «s\_fiz\_lic» с кодом 4.

|  |
| --- |
| UPDATE s\_fiz\_lic3 SET fio=(SELECT UPPER(fam||’ ’||im||’ ’||otch) FROM s\_fiz\_lic WHERE kod=4) WHERE kod=5; |

Обновим таблицу «s\_fiz\_lic3» проставив код должности из таблицы «dolj».

|  |
| --- |
| UPDATE s\_fiz\_lic3 SET kod\_dolj=1 WHERE kod IN (1,2,5,6);  UPDATE s\_fiz\_lic3 SET kod\_dolj=2 WHERE kod IN (3,7);  UPDATE s\_fiz\_lic3 SET kod\_dolj=3 WHERE kod IN (8);  COMMIT; |

***КОМАНДА DELETE***

Оператор **DELETE** производит удаление строк таблицы. Для определения набора строк, подлежащих удалению, используется предложение WHERE. В нем указываются условия, которым должна отвечать удаляемая строка. Если предложение WHERE опустить, то будут удалены все строки.

**Синтаксис команды:**

DELETEFROM таблица/имя\_предст\_данных WHERE\_предложение;

Рассмотрим примеры удаления данных. В первом примере мы производим удаление строки из таблицы с кодом 5.

|  |
| --- |
| DELETE FROM s\_fiz\_lic WHERE kod=5; |

В следующем примере выполянется удаление всех строк таблицы «s\_fiz\_lic».

|  |
| --- |
| DELETE FROM s\_fiz\_lic; |

***КОМАНДА SELECT***

Команда **SELECT** извлекает данные из столбцов одной или нескольких таблиц. Команда SELECT сам по себе является запросом. Если он используется как предложение внутри другого оператора, то он называется подзапросом. В операторе SELECT обязательно должно присутствовать предложение FROM. Остальные предложения не являются необходимыми.

Предложение SELECT может использоваться как:

* самостоятельная команда на получение и вывод строк таблицы, сформированной из столбцов и строк одной или нескольких таблиц (представлений);
* фраза выбора в командах CREAT VIEW, DECLARE CURSOR или INSERT;
* средство присвоения глобальным переменным значений из строк сформированной таблицы (INTO-фраза).

**Синтаксис команды:**

**SELECT** alias.столбец\_1, alias.столбец\_2 ….,

функция(столбец\_3), (столбец\_1+ столбец\_2) as выражение

**FROM** таблица\_1 alias, таблица\_2 alias, (запрос\_1) alias

**WHERE** условие\_1 [OR [NOT]| AND [NOT]] условие\_2

**GROUP BY** Групповой\_столбец\_1, групповой\_столбец2

**HAVING** условие\_по\_групповому\_столбцу

**ORDER BY** столбец\_1 **ASC**|**DESC**

После указания перечня выбираемых столбцов и операций над ними, указывается служебное слово FROM – оно определяет ресурс данным, т.е. откуда будут выбираться данные, в качестве ресурса данных могут выступать таблицы, представления, запросы.

Таблице или столбцу можно присвоить альтернативное имя (алиас - **alias**), действие которого будет действительно только в пределах оператора, в котором оно определено. Если альтернативное имя стоит после имени столбца в списке оператора SELECT, то оно будет использоваться вместо настоящего имени как заголовок для данного столбца. Использование имен столбцов и алиаса приведен в примере ниже.

|  |
| --- |
| SELECT f.kod “Код”, f.fio “Ф.И.О.” FROM s\_fiz\_lic3 f; |

Альтернативное имя таблицы можно использовать при соединении таблицы с самой собой в соотносящемся запросе. При использовании таблиц с одинаковыми именами полей требуется указание имени таблицы перед именем колонки таблицы.

Для выбора данных по условию применяют предложение **WHERE**. После его указания определяется перечень условий, разделяемые между собой логикой – элементы AND, NOT AND, OR, NOT OR. Условием в языке SQL называется сочетание одного или нескольких выражений и логических операторов, вырабатывающих значение TRUE (истина) или FALSE (ложь). Каждое условие можно представить в виде операторов сравнения (см. таблица).

|  |  |
| --- | --- |
| **Оператор** | **Значение/действие в SQL** |
| = | равно |
| != или <> | не равно |
| >= | больше или равно |
| <= | меньше или равно |
| IN | равен любому элементу в |
| NOT IN | не равен любому элементу в |
| ANY | Сравнивает с любым из значений в списке. Употребляется после =, !=, >, <, <=, >=. |
| ALL | Сравнивает с каждым значением в списке. Употребляется после =, !=, >, <, <=, >=. |
| BETWEEN | больше или равно значения\_1 и меньше или равно значение\_2 |
| NOT | не больше и не равно |
| EXISTS | Истина, если подзапрос извлекает хотя бы одну строку |
| IS NULL | Истина, значение есть NULL. Проверки типа x=NULL – являются неправильными. |
| IS NOT NULL | Истина, если значение не пустое |

Рассмотрим примеры с использованием операторов сравнения и прекдиката WHERE. В первом примере мы выбираем все столбцы таблицы «Должности» и ту должность, поле «kod», которой равно 1. Для вывода всех столбцов в запросе используется символ «\*».

|  |
| --- |
| SELECT \* FROM dolj WHERE kod=1; |

Покажем все должности, у которых значение поле «kod» больше 2 и не равно 5.

|  |
| --- |
| SELECT \* FROM dolj WHERE kod>2 AND kod!=5; |

Выберем должности значение поля «kod» у которых находится между 2 и 4 включительно.

|  |
| --- |
| SELECT \* FROM dolj WHERE kod between 2 AND 4; |

Покажем всeх людей, у которых поле «kod\_dolj» не пустое.

|  |
| --- |
| SELECT kod, fio, kod\_dolj FROM s\_fiz\_lic3 WHERE kod\_dolj IS NOT NULL; |

Покажем должности входящие в определенный перечень.

|  |
| --- |
| SELECT naimen FROM dolj WHERE kod IN (1,2,3); |

Для группировки результирующего запроса по столбцам используется выражение **GROUP BY,** после которого указывается список столбцов, по которым будет группироваться таблица, для вычисляемых столбцов применяются групповые функции SUM, AVG, MIN, MAX, COUNT при определении списка выбираемых столбцов.

Рассмотрим запрос, который показывает количество людей каждой должности в таблице «s\_fiz\_lic», где поле «kod\_dolj» имеет не пустое значение.

|  |
| --- |
| SELECT kod\_dolj, COUNT(\*) kolvo FROM s\_fiz\_lic3 WHERE kod\_dolj IS NOT NULL GROUP BY kod\_dolj; |

Для получения данных по условию для сгруппированной таблицы применяется элемент **HAVING,** после чего указывается условие для сгруппированной таблицы. Условие определяется аналогично условиям **WHERE**. В нижеприведенном примере, мы оставили те группы должностей, у которых количество физических лиц больше одного.

|  |
| --- |
| SELECT kod\_dolj, COUNT(\*) kolvo FROM s\_fiz\_lic3 WHERE kod\_dolj IS NOT NULL GROUP BY kod\_dolj HAVING COUNT(\*)>1; |

Для сортировки полученных данных в запросе применяется **ORDER BY.**  После указания выражения перечисляются столбцы, покоторым будет сортировка, и тип сортировки по возрастанию (**ASC**) или убыванию (**DESC**). По умолчанию сортировка по возрастанию.

Отсортируем по убыванию физических лиц по полю «fio».

|  |
| --- |
| SELECT kod, fio FROM s\_fiz\_lic3 ORDER BY fio DESC; |

Произведем сортировку по двум столбца – «fio» и «kod».

|  |
| --- |
| SELECT kod, fio FROM s\_fiz\_lic3 ORDER BY 2,1 ASC; |

Для выбора всех или уникальных записей применяют выражения [**DISTINCT** | **UNIQUE** | **ALL**] после слово SELECT, затем перечисляются столбцы, по которым будет отслеживаться уникальность данных.

**Синтаксис команды:**

**SELECT** [**DISTINCT | UNIQUE | ALL** ] alias.столбец\_1, alias.столбец\_2

**FROM** таблица\_1 alias

Выражения DISTINCT или UNIQUE позволяют осуществить выбор только уникальных записей. Выражение ALL – выбор всех записей из таблицы.

Для получения объединения, пересечения и разности запросов используется служебные слова **UNION**, **UNION ALL**, **MINUS**, **INTERSECT**:

**Синтаксис команды:**

SELECT запрос1

[UNION | UNION ALL | MINUS | INTERSECT]

SELECT запрос2

UNION – объединяет не дублирующиеся данные нескольких запросов

UNION ALL – объединяет все данные

MINUS – производит вычитание данных из запроса1 запрос2

INTERSECT – показывает одинаковые данные имеющиеся в запросах

Оператор **UNION** объединяет выходные строки каждого из запросов в один результирующий набор. Если определен параметр **ALL**, то сохраняются все дубликаты выходных строк, в противном случае в результирующем наборе остаются только уникальные строки. Заметим, что можно связывать вместе любое число запросов. Кроме того, с помощью скобок можно менять порядок объединения.

При этом должны выполняться следующие условия:

* Количество выходных столбцов каждого из запросов должно быть одинаковым.
* Выходные столбцы каждого из запросов должны быть сравнимыми между собой (в порядке их следования) по типам данных.
* В результирующем наборе используются имена столбцов, заданные в первом запросе.
* Предложение ORDER BY применяется к результату соединения, поэтому оно может быть указано только в конце составного запроса.

Для выполнения операций **пересечения** и **разности** запросов. Этими предложениями являются **INTERSECT** (пересечение) и **MINUS** (разность), которые работают аналогично предложению **UNION**. В результирующий набор попадают только те строки, которые присутствуют в обоих запросах (INTERSECT) или только те строки первого запроса, которые отсутствуют во втором (MINUS).

Рассмотрим примеры. В первом примере получаем объединение запросов с дубликатами строк.

|  |
| --- |
| SELECT kod, fio FROM s\_fiz\_lic3  UNION ALL  SELECT kod, fio FROM s\_fiz\_lic3; |

Объединяем запросы с устранением дубликатов.

|  |
| --- |
| SELECT kod, fio FROM s\_fiz\_lic3  UNION  SELECT kod, fio FROM s\_fiz\_lic3; |

### ЯЗЫК УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ (TCL - Transaction Control Language)

Транзакцией в SQL называется логически неделимая последовательность операторов, рассматриваемая как единое целое. Результаты выполнения операторов, входящих в транзакцию, могут быть либо сохранены в БД при помощи оператора COMMIT, либо полностью аннулированы оператором ROLLBACK до точки сохранения. Транзакция начинается с 1-го выполняемого оператора, либо с 1-го оператора после COMMIT или ROLLBACK. Транзакция заканчивается при выполнении операторов COMMIT или ROLLBACK.

**Операторы управления транзакциями:**

COMMIT - фиксация транзакции

ROLLBACK – откат транзакции

SAVEPOINT – установки точки отката транзакции

Рассмотрим пример фиксации транзакции командой COMMIT. В уазанном примере мы производим удаление строки из таблицы «s\_fiz\_lic3» с кодом 5. После чего производим фиксацию выполненной операции.

|  |
| --- |
| DELETE FROM s\_fiz\_lic3 WHERE kod=5;  COMMIT; |

Следующий пример показывает принцип использования команды отката транзакции ROLLBACK. В данном примере мы удаляем строку, так как мы не зафиксировали операцию мы можем ее откатить, для этого применяем команду ROLLBACK.

|  |
| --- |
| DELETE FROM s\_fiz\_lic3 WHERE kod=4;  ROLLBACK; |

Теперь рассмотрим более сложный пример управления транзакциями с применением точек отката транзации. При выполнении группы операций можно проименовать эти группы операций установив точки отката транзакций с помощью команды SAVEPOINT. Точки отката транзакций предназначены, для того чтобы, имелась возможность откатить транзакцию до определенного момента в группе операций. Рассмотрим нижеприведенный пример. В данном примере удаляем строку с кодом 5, затем устанавливаем точку отката транзакции с именем example, затем производим удаление строки с кодом 4 и производим откат до точки отката example, в результате операция удаления строки с кодом 4 будет аннулирована, затем мы производим фиксацию транзакции. В результате выполнения команд будет выполнено только одно удаление строки с кодом 5.

|  |
| --- |
| DELETE FROM s\_fiz\_lic3 WHERE kod=1;  SAVEPOINT example;  DELETE FROM s\_fiz\_lic3 WHERE kod=4;  ROLLBACK WORK TO example;  COMMIT; |

### ЯЗЫК УПРАВЛЕНИЯ ДОСТУПА К ДАННЫМ (DCL - Data Control Language)

Операторы Data Control Language, иногда называемые операторами Access Control Language, применяются для осуществления административных функций, присваивающих или отменяющих право (привилегию) использовать базу данных, таблицы в базе данных, а также выполнять те или иные операторы SQL над объектами БД.

**Операторы управления транзакциями:**

GRANT – назначение права (привилегии, роли) пользователю на объект

REVOKE – отзыв права (привилегииь, роли) у пользователя на объект

C помощью этих операторов производится назначение прав пользователю на выполнение операций над объектом. Каждый объект имеет определенный перечень прав. Нижеприведенный общий перчень прав, которые могут быть назначены пользователю: SELECT, ALTER, INDEX, DELETE, UPDATE, EXECUTE, DEBUG, REFERENCES, READ, WRITE. Надо отметить, что кроме указанных прав (привилегий), пользователю могут быть назначены роли (ROLE), роли – представляют собой совокупность привилегий. Роли могут как системными, так и определенные пользователем – пользовательские.

В нижеприведенном примере мы производим наделение правами пользователя SYSTEM на осуществление операций выборки, изменения таблицы и удаления данных из нее, а следующий командой мы производим отзыв права удаления из таблицы «s\_fiz\_lic».

|  |
| --- |
| GRANT SELECT, ALTER, DELETE ON s\_fiz\_lic TO system;  REVOKE DELETE ON s\_fiz\_lic FROM system; |

В данном примере служебным словом ALL мы производим наделение всеми правами пользователя SYSTEM на осуществление операций над табдицей «s\_fiz\_lic»

|  |
| --- |
| GRANT ALL ON s\_fiz\_lic TO system; |