1.4. Введение в язык SQL

Язык работы с базами данных должен предоставлять пользователям следующие возможности:

- создавать базу данных и таблицы с полным описанием их структуры;
- выполнять основные операции манипулирования данными (добавление, изменение, удаление данных);
- выполнять запросы, осуществляющие преобразование данных в необходимую информацию.

Создание базы данных обычно выполняется автоматически при установке самой СУБД, поэтому в данном пособии не рассматривается.

Для реализации этих функций SQL включает три группы средств:

- DDL (Data Definition Language) – язык определения данных;
- DML (Data Manipulation Language) – язык манипулирования данными;
- DCL (Data Control Language) – язык управления данными.

По стандарту ANSI подмножество команд DCL является частью DDL.

В командах SQL не различаются прописные и строчные буквы (за исключением строчных литералов). Символы и строки символов заключаются в одинарные кавычки, например, 'N', 'учебник'. Однострочные комментарии начинаются с двух минусов (--), многострочные заключаются в символы /* и */.

Каждая команда заканчивается символом ';'. Значения параметров команд, принятые по умолчанию, выделены подчеркиванием, например, *ALL*.

Примечание. Примем следующие обозначения для описания синтаксиса:

{ } – содержимое скобок рассматривается как единое целое для остальных символов;
| – заменяет слово ИЛИ;
[ ] – содержимое этих скобок является необязательным;
< > – содержимое этих скобок заменяется соответствующими ключевыми словами, литералами, идентификаторами или выражениями (в зависимости от контекста);
… – всё, что предшествует этим символам, может повторяться произвольное число раз;
.,.. – всё, что предшествует этим символам, может повторяться произвольное число раз, каждое вхождение отделяется запятой.

В соответствии со стандартов ISO идентификатор определяется как последовательность символов длиной не более 128, начинающаяся с буквы латинского алфавита и содержащая буквы латинского алфавита, цифры и знак подчеркивания (_). В большинстве СУБД накладываются более жёсткие ограничения на длину идентификатора.

Синтаксис команд и примеры, приведённые в данном пособии, соответствуют синтаксису СУБД Oracle 9i и выше.

1.4.1. Создание таблиц

Создание нового пустого отношения (таблицы) выполняется с помощью команды DDL **CREATE TABLE**. Упрощённый синтаксис этой команды:

```
CREATE TABLE <имя таблицы>
   ( <имя поля1>   <тип данных> [ (<размер>) ]
```

```
        [ [NOT]  NULL]    [ DEFAULT <выражение> ]
        [<ограничения целостности поля>…]
[, <имя поля2>   <тип данных>  [ (<размер>) ]
        [ [NOT]  NULL]    [ DEFAULT <выражение> ]
        [<ограничения целостности поля>] .,..]
 [, <ограничения целостности таблицы>] );
```

Расшифровка элементов описания приведена в табл. 1.

Таблица 1. Описание элементов команды CREATE TABLE

Элемент	Описание
<имя таблицы>	Имя таблицы, обычный идентификатор. Должно быть уникальным в рамках базы данных (подсхемы базы данных).
<имя поля>	Имя поля (столбца) таблицы, обычный идентификатор.
<тип данных>	Тип данных поля. Можно использовать один из следующих типов: – INTEGER, INT, SMALLINT – целые числа; – NUMERIC[(длина [, точность])], NUMBER[(длина [, точность])] – числа с фиксированной запятой; – FLOAT, REAL – вещественные числа; – CHAR[(длина)], VARCHAR(длина) – символьные строки фиксированной и переменной длины; – DATE, DATETIME – дата/время; – TIME – время (есть не во всех СУБД).
<размер>	Размер поля в символах (для текста и чисел).
<ограничения целостности>	– PRIMARY KEY – первичный ключ (обязательный, уникальный и единственный на таблицу); – UNIQUE – уникальность значения поля в пределах столбца таблицы; – CHECK (<условие>) – условие, которому должно удовлетворять значение поля; – REFERENCES <имя таблицы> [(<имя столбца>)] – внешний ключ.
<ограничения целостности таблицы>	– CHECK (<условие>) – условие, которому должны удовлетворять значения одного или нескольких полей; – FOREIGN KEY [(<список полей>)] REFERENCES <имя таблицы> [(<список полей>)] – внешний ключ; – PRIMARY KEY (<список полей>) – первичный ключ; – UNIQUE (<список полей>) – уникальное значение комбинации полей в пределах таблицы.

Если размер поля не указан, то принимается значение, принятое в данной СУБД по умолчанию для указанного типа. Для всех СУБД точность для числовых типов по умолчанию равна 0, размер поля типа CHAR по умолчанию равен 1, а для типа VARCHAR размер указывать обязательно.

Для типа DATE поддерживается арифметика дат, например:

(<текущая дата> + 1) – завтра

(<дата1> – <дата2>) – количество дней, прошедших между двумя датами;

(<текущая дата> + 1/24) – через час (для типа дата-время).

Получить текущую дату можно с помощью специальной функции, имя которой зависит от СУБД: *sysdate* – для Oracle; *now()* – для Access; *currdate()* – для MySQL; *getdate()* – для MS SQL Server и т.д.

Стандарт SQL включает понятие неопределённого или неизвестного значения – NULL-значения. Для обязательных полей устанавливается ограничение NOT NULL. Это означает, что при изменении значения этого поля или при добавлении новых записей таблицы значение этого поля не может быть неопределенным (NULL). Ограничение NOT NULL можно наложить на поле только один раз, иначе возникает ошибка.

Для любого поля с помощью конструкции DEFAULT <выражение> может быть задано значение по умолчанию. Оно используется в тех случаях, когда при добавлении данных в таблицу значение этого поля не указывается.

Для ограничений целостности можно задавать имена:

CONSTRAINT <имя> <ограничение целостности>

Примеры создания таблиц:

1. Таблица "Отделы" с полями "Номер отдела" (ПК), "Название отдела":

```
CREATE TABLE depart
  ( depno   NUMERIC(2)     CONSTRAINT pk_dep PRIMARY KEY,
    name    VARCHAR(80)    NOT NULL);
```

2. Таблица "Сотрудники" с полями "Номер отдела" (внешний ключ), "Табельный номер сотрудника" (ПК), "ФИО сотрудника", "Должность", "Оклад", "Дата рождения", "Телефон", "Дата поступления на работу":

```
CREATE TABLE emp
  (depno   NUMERIC(2)  CONSTRAINT ref_dep REFERENCES depart,
    tabno   CHAR(5)          CONSTRAINT pk_emp PRIMARY KEY,
    name    VARCHAR(50)      NOT NULL,
    post    VARCHAR(35)      NOT NULL,
    salary  NUMERIC(7,2)     NOT NULL
          CONSTRAINT check_salary CHECK (salary > 4600),
    born    DATE             NOT NULL,
    phone   CHAR(11),
    edate   DATE             DEFAULT trunc(sysdate));
```

Примечание: функция sysdate возвращает дату и время, поэтому следует с помощью функции trunc (сокращение от truncate) устанавливать время в 0 часов, 0 минут, 0 секунд.

3. Таблица "Дети сотрудников" с полями "Табельный номер родителя" (внешний ключ), "Имя ребенка", "Пол", "Дата рождения":

```
CREATE TABLE children
  ( tabno   CHAR(5) CONSTRAINT ref_emp REFERENCES emp(tabno),
    name    VARCHAR(50)    NOT NULL,
    sex     CHAR,
    born    DATE,
    CONSTRAINT pk_child PRIMARY KEY (tabno, name),
                          /* составной первичный ключ */
    CONSTRAINT check_sex CHECK (sex IN ('м', 'ж')));
```

Обратите внимание:

- общие ограничения целостности и составные ключи указываются через запятую после последнего поля;

- если внешний ключ ссылается на первичный ключ (ПК) другого отношения, имена полей ПК можно не указывать (см. создание таблицы *emp*);
- типы полей внешнего ключа должны совпадать с типами полей первичного (или уникального) ключа, на который он ссылается;
- если внешний ключ составной, список полей входящий в ключ, указывается после перечисления всех полей таблицы с ключевым словом FOREIGN KEY:

```
create table exam      -- "Расписание зачетов", основная таблица
   (   eclass    NUMERIC(3),          -- аудитория
       edate     DATE,                -- дата
       edisc     VARCHAR(60),         -- дисциплина
       UNIQUE (eclass, edate));
create table tab       -- "Зачеты", подчинённая таблица
   (   tid       NUMERIC(6)   PRIMARY KEY,
       tclass    NUMERIC(3),                  -- аудитория
       tdate     DATE,                        -- дата
       tgroup    CHAR(6),                     -- группа
       FOREIGN KEY (tclass, tdate) REFERENCES exam(eclass, edate));
```

1.4.2. Команды модификации данных

К командам модификации данных (DML) относятся добавление, удаление и изменение (обновление) записи (строки таблицы). При выполнении этих команд проверяются все установленные для таблицы ограничения целостности. В случае нарушения любого ограничения целостности или возникновения других проблем (переполнение памяти, например) команда DML не выполняется и выдаётся сообщение об ошибке. Если же команда выполнилась успешно, выдается информация о количестве обработанных строк.

INSERT – добавление записи в таблицу. Синтаксис команды:

```
INSERT INTO <имя таблицы> [(<имя поля>.,...)]
      VALUES (<список выражений>) | <запрос>;
```

Под <запросом> подразумевается команда SELECT (см. ниже), результаты работы которой добавляются в указанную таблицу.

В предложении VALUES указываются выражения, порождающие значения атрибутов новой записи таблицы. Выражение может включать вызовы функций, определенных в данной СУБД, константы, знак операций конкатенации строк (||) или знаки арифметических операций: –, +, *, /. Типы значений выражений должны соответствовать типам полей таблицы. Строки и даты должны заключаться в одинарные кавычки. Формат даты должен соответствовать тому, который установлен в СУБД по умолчанию.

Если значения устанавливаются не для всех полей или порядок значений не соответствует тому порядку полей, который был установлен при создании таблицы, то после имени таблицы в скобках приводится список полей в соответствии со списком значений.

В тех случаях, когда при добавлении записи значение какого-либо поля неизвестно, его можно не устанавливать, пропустив это поле в списке полей или указав для него значение NULL (но только для тех полей, на которые не наложено ограничение целостности NOT NULL).

Если в списке полей отсутствует какое-либо поле таблицы, то ему будет присвоено значение NULL или значение по умолчанию (DEFAULT), если оно определено в командах CREATE TABLE или ALTER TABLE.

Пример: Добавить в таблицу "Сотрудники" новую запись:

```
INSERT INTO emp (depno, tabno, name, post, salary, born, phone)
   VALUES(3, '00112', 'Попов В.Г.', 'экономист', 45400, '1979-12-23', '115-34-11');
1 строка создана.
```

В данном случае дата рождения вводится как строка '1979-12-23' в соответствии с форматом даты по умолчанию. А в качестве даты поступления сотрудника на работу будет установлена текущая дата, т.к. для поля edate определено значение по умолчанию DEFAULT и в команде INSERT значение не вводится.

Примечание: посмотреть формат даты по умолчанию в СУБД Oracle можно так:

```
select sysdate from dual;
```

Изменить формат даты в Oracle можно следующей командой:

```
alter session set nls_date_format = 'yyyy-mm-dd';
```

'yyyy' означает год (4 цифры), 'mm' – месяц, 'dd' – день; разделители могут быть любыми.

UPDATE – обновление данных в таблице. Синтаксис:

```
UPDATE <имя таблицы>
      SET {<имя поля> = <выражение>}.,..
      [WHERE  <условие>];
```

Команда обновления изменяет в указанной таблице значения указанных полей тех записей, которые удовлетворяют заданному условию отбора (WHERE <условие>). Если условие не указано, обновления применяются ко всем записям таблицы.

Пример: Изменить должность и зарплату сотрудника Попова В.Г., табельный номер '00112':

```
UPDATE emp
   SET   post = 'ст. экономист', salary = salary*1.1
   WHERE  tabno = '00112';
1 строка обновлена.
```

DELETE – удаление записей из таблицы. Синтаксис этой команды:

```
DELETE FROM <имя таблицы>
      [ WHERE  <условие> ];
```

Эта команда удаляет из указанной таблицы те записи, которые удовлетворяют заданному условию отбора (WHERE <условие>).

Внимание! Если не указывать условие выбора записей, то все записи таблицы будут удалены без предупреждения и без запроса на подтверждение!

Пример: Удалить запись о сотруднике Попове В.Г., табельный номер '00112':

```
DELETE FROM  emp
   WHERE  tabno = '00112';
1 строка удалена.
```

1.4.3. Извлечение данных из таблиц

Извлечение данных из таблиц БД выполняется с помощью команды **SELECT** (селекция). Эта команда не изменяет данные в БД.

Результатом выполнения команды SELECT является временная таблица, которая помещается в курсор (специальную область памяти СУБД) и обычно сразу выводится на экран. Упрощённый синтаксис этой команды:

```
SELECT   [ ALL | DISTINCT ]  { * | <список выбора> }
   FROM   {<имя таблицы>  [<алиас>] }.,..
   [ WHERE   <условие>]
   [ GROUP BY {<имя поля> | <выражение>}.,.. ]
   [ HAVING   <условие> ]
   [ UNION  [ALL]  SELECT …]
   [ ORDER BY  {<имя поля> | <целое> [ ASC | DESC ] }.,..];
```

Расшифровка элементов описания приведена в табл. 2. Порядок конструкций в команде SELECT не может быть изменён.

Таблица 2. Элементы команды SELECT

Элемент	Описание
<список выбора>	Список элементов, разделённых запятыми: <выражение> [AS] <алиас> Выражение может включать имена полей, знаки операций, вызовы функций и константы. Алиас – это название столбца результата.
<имя таблицы>	Имя или синоним имени таблицы или представления.
<алиас>	Временный синоним имени таблицы, определённый только внутри данного запроса.
<условие>	Условие, которое может быть истинным или ложным для каждой записи из таблицы (таблиц), определённых предложением FROM.
<имя поля>	Имя поля (столбца) таблицы.
<целое>	Число без десятичной точки. Номер элемента в <списке выбора>.

Список выбора (вывода) определяет схему результата (временной таблицы). Список выбора может быть модифицирован одним из двух ключевых слов:
DISTINCT – предикат удаления из результирующей таблицы повторов строк.
ALL – предикат, обратный к DISTINCT. Это значение используется по умолчанию, его можно не указывать.

Рассмотрим основные предложения (фразы) команды SELECT:
SELECT – после этого ключевого слова указывается **список выбора** – список выражений, которые будут образовывать результирующую таблицу. Вы-

ражению можно сопоставить временный синоним (алиас), который будет названием поля результирующей таблицы, например:

```
(sal*0.87+bonus) AS salary
```

Если надо вывести все поля из тех таблиц, к которым обращается данный запрос, можно указать символ *. В этом случае сначала будут выведены поля таблицы, стоящей первой в предложении FROM, затем – второй и т.д. Поля, относящиеся к одной таблице, будут выводиться в том порядке, в каком они были указаны при создании таблицы.

FROM – в этой части указывается имя таблицы (имена таблиц), из которой будут извлекаться данные.

WHERE – эта часть содержит условия выбора отдельных записей.

GROUP BY – объединяет в одну группу все записи с одинаковым значением указанного поля (комбинации полей). Каждой такой группе в результирующей таблице соответствует одна запись.

HAVING – позволяет указать условия выбора для групп записей.

ORDER BY – упорядочивает результирующие строки по значению одного или нескольких полей: ASC – по возрастанию, DESC – по убыванию.

Порядок выполнения операции SELECT такой:

1. Выбор из указанной во фразе FROM таблицы тех записей, которые удовлетворяют условию отбора (*WHERE*).
2. Группировка полученных записей (*GROUP BY*).
3. Выбор тех групп, которые удовлетворяют условию отбора групп (*HAVING*).
4. Сортировка записей в указанном порядке (*ORDER BY*).
5. Извлечение из записей полей, заданных в списке выбора, и формирование результирующей таблицы.

Если во фразе FROM указаны две и более таблицы, то эта последовательность действий выполняется для декартова произведения указанных таблиц.

Отношения для примеров приведены в таблицах 3-5.

Таблица 3. Отношение "Сотрудники" (Emp)

TabNo	*DepNo*	Name	Post	Salary	Born	Phone
988	*1*	Рюмин В.П.	начальник отдела	48500.0	01.02.1970	115-26-12
909	*1*	Серова Т.В.	вед. программист	48500.0	20.10.1981	115-91-19
829	*1*	Дурова А.В.	экономист	43500.0	03.10.1978	115-26-12
819	*1*	Тамм Л.В.	экономист	43500.0	13.11.1985	115-91-19
100	*2*	Волков Л.Д.	программист	46500.0	16.10.1982	null
110	*2*	Буров Г.О.	бухгалтер	42880.0	22.05.1975	115-46-32
023	*2*	Малова Л.А.	гл. бухгалтер	59240.0	24.11.1954	114-24-55
130	*2*	Лукина Н.Н.	бухгалтер	42880.0	12.07.1979	115-46-32
034	*3*	Перова К.В.	делопроизводитель	32000.0	24.04.1988	null
002	*3*	Сухова К.А.	начальник отдела	48500.0	08.06.1948	115-12-69
056	*5*	Павлов А.А.	директор	80000.0	05.05.1968	115-33-44
087	*5*	Котова И.М.	секретарь	35000.0	16.09.1990	115-33-65
088	*5*	Кроль А.П.	зам.директора	70000.0	18.04.1974	115-33-01

Таблица 4. Отношение "Отделы" (Depart)

DepNo	Name

Таблица 5.Отношение "Дети"(Children)

TabNo	**Name**	Born	Sex

2	Бухгалтерия
3	Отдел кадров
4	Отдел технического контроля
1	Плановый отдел
5	Администрация

988	Вадим	03.05.1995	м
110	Ольга	18.07.2001	ж
023	Илья	19.02.1987	м
023	Анна	26.12.1989	ж
909	Инна	25.01.2008	ж
909	Роман	21.11.2006	м
909	Антон	06.03.2009	м

Примеры:

1. Вывести все записи (строки) из таблицы "Отделы":

```
SELECT  *
    FROM depart;
```

DEPNO	NAME
2	Бухгалтерия
3	Отдел кадров
4	Отдел технического контроля
1	Плановый отдел
5	Администрация

2. Вывести список сотрудников с указанием должности и зарплаты за вычетом подоходного налога, упорядочить по отделам и фамилиям:

```
SELECT   depno, name, post, salary*0.87  AS  sal
    FROM  emp
    ORDER BY  depno, name;
```

DEPNO	NAME	POST	SAL
1	Дурова А.В.	экономист	37845.0
1	Рюмин В.П.	начальник отдела	42195.0
1	Серова Т.В.	вед. программист	42195.0
1	Тамм Л.В.	экономист	37845.0
2	Буров Г.О.	бухгалтер	37305.6
2	Волков Л.Д.	программист	40455.0
2	Лукина Н.Н.	бухгалтер	37305.6
2	Малова Л.А.	гл. бухгалтер	51538.8
3	Перова К.В.	делопроизводитель	27840.0
…	…	…	…

3. Вывести список должностей с окладом в порядке убывания оклада:

```
SELECT  DISTINCT  post, salary
    FROM  emp
    ORDER BY  salary  DESC;
```

POST	SALARY
директор	80000
зам.директора	70000
гл. бухгалтер	59240
вед. программист	48500
начальник отдела	48500
…	…
секретарь	35000

делопроизводитель	32000

1.4.4. Операторы и предикаты

Расширение возможностей команд языка SQL достигается за счёт применения различных **операторов, предикатов и функций**.

Операторы:

- символьные: || – конкатенация строк;
- арифметические: +, –, *, /;
- сравнения: =, >, <, >=, <=, <>;
- логические: AND, OR, NOT.

Примеры:

4. Составить список сотрудников второго и третьего отдела, имеющих зарплату выше 40000 рублей после уплаты подоходного налога 13%:

```
SELECT   depno, name, salary*0.87  AS  sal
   FROM   emp
   WHERE  salary*0.87>40000  AND  (depno=2  OR  depno=3)
   ORDER BY  name;
```

DEPNO	NAME	SALARY
2	Волков Л.Д.	40455.0
2	Малова Л.А.	51538.8
3	Сухов К.А.	42195.0

5. Составить список сотрудников первого отдела с указанием должности:

```
SELECT   post || ' ' || name  AS  ename
   FROM   emp
   WHERE  depno=1
   ORDER BY  1  DESC;
```

ENAME
экономист Тамм Л.В.
экономист Дурова А.В.
начальник отдела Рюмин В.П.
вед. программист Серова Т.В.

Обратите внимание на добавление пробела между полями: || ' ' ||.

Предикаты, используемые в командах:

- **IN** – предикат вхождения в список:

  <выражение> IN (<список значений>)

  – определяет множество значений, с которыми будет сравниваться значение <выражения>. Предикат считается истинным, если значение выражения равно хотя бы одному из элементов множества.

- **BETWEEN** – предикат нахождения в диапазоне:

  <выражение> BETWEEN <значение1> AND <значение2>

– определяет, входит ли значение <выражения> в указанные границы. Если значение выражения меньше, чем <значение1>, или больше, чем <значение2>, предикат возвращает "ложь".

- **LIKE** – предикат подобия:

  <выражение> LIKE '*образец*'

– используется для сравнения строк, применяется только к полям типа CHAR, VARCHAR. Возможно использование шаблонов: '_' – один любой символ и '%' – произвольное количество символов (в т.ч., ни одного);

- **IS NULL** – предикат неопределённого значения:

  <выражение> IS NULL

– определяет, установлено ли значение поля; возвращает истину, если не установлено. Другие предикаты и операторы сравнения возвращают неопределённый результат (NULL), если хотя бы один из операндов имеет значение NULL. Значение NULL интерпретируется как "ложь".

Все эти предикаты могут комбинироваться с операцией "не": NOT IN, NOT LIKE, NOT BETWEEN, IS NOT NULL.

Примеры:

6. Вывести список программистов и ведущих программистов:

```
SELECT depno, name, post
    FROM emp
    WHERE post like ('%программист%');
```

DEPNO	NAME	POST
1	Серова Т.В.	вед. программист
2	Волков Л.Д.	программист

7. Увеличить на 10% оклады начальникам отделов и программистам:

```
UPDATE emp
    SET salary=salary*1.1
    WHERE post LIKE 'нач%отдел%' OR post LIKE '%програм%';
4 строки обновлено.
```

8. Вывести список сотрудников старше 40 лет из 1-го и 3-го отделов:

```
SELECT depno, name, trunc(months_between(sysdate, born) / 12) AS age
  FROM emp
  WHERE depno IN (1, 3) AND
    trunc(months_between(sysdate, born)/12) > 40;
```

DEPNO	NAME	AGE
1	Рюмин В.П.	42
3	Сухова К.А.	63

Примечание: функция months_between() возвращает количество месяцев, прошедших между двумя датами, функция trunc() усекает полученное число до целого.

9. Список сотрудников, не имеющих телефонов:

```
SELECT  tabno, name, post
```

```
FROM  emp
WHERE   phone  IS NULL;
```

TABNO	NAME	POST
100	Волков Л.Д.	программист
034	Перова К.В.	делопроизводитель

10.Список сотрудников, родившихся в 80-е годы XX века:

```
SELECT   tabno, name, born, post
        FROM  emp
        WHERE  born  BETWEEN  '1980-01-01'  AND  '1989-12-31';
```

TABNO	NAME	BORN	POST
909	Серова Т.В.	1981-10-20	вед. программист
100	Волков Л.Д.	1982-10-16	программист
034	Перова К.В.	1988-04-24	делопроизводитель
819	Тамм Л.В.	1985-11-13	экономист

1.4.5. Функции агрегирования

Для подсчёта различных агрегированных значений (по группе записей) стандарт SQL включает т.н. функции агрегирования:

- COUNT(*) – определяет количество строк (записей) в результате.
- MAX(<поле>), MIN(<поле>) – определяет максимальное (минимальное) значение указанного поля в результирующем множестве.
- SUM(<поле>) – определяет арифметическую сумму значений указанного числового поля в результирующем множестве записей.
- AVG(<поле>) – определяет среднее арифметическое значений указанного числового поля в результирующем множестве записей.

Правила уточнения использования агрегирующих функций:
COUNT (<поле>) – подсчёт количества ненулевых значений поля;
COUNT (distinct <поле>) – подсчёт количества разных значений поля;
SUM (distinct <поле>) – суммирование разных значений поля;
AVG (distinct <поле>) – среднее арифметическое разных значений поля.

Примеры:

11.Посчитать количество сотрудников предприятия:

```
SELECT  count(*), ' человек(а)'
        FROM emp;
```

COUNT(*)	ЧЕЛОВЕК(А)
13	человек(а)

12.Определить минимальную и максимальную зарплату сотрудников:

```
SELECT  min(salary)  AS  minsal, max(salary)  AS  maxsal
        FROM  emp;
```

MINSAL	MAXSAL
32000	80000

13.Определить среднюю зарплату сотрудников 3-го отдела:

```
SELECT  avg(salary) AS avg3
        FROM emp
        WHERE depno=3;
```

AVG3
40250

Агрегирующие функции можно комбинировать с фразой GROUP BY: в этом случае подсчёт будет производиться для каждой группы записей с одинаковым значением комбинации полей, указанных в GROUP BY.

Примеры:

14.Посчитать количество сотрудников по отделам:

```
SELECT  depno, count(*), ' сотрудник(а)'
        FROM emp
        GROUP BY depno;
```

DEPNO	COUNT(*)	СОТРУДНИК(А)
1	4	сотрудник(а)
2	4	сотрудник(а)
3	2	сотрудник(а)
5	3	сотрудник(а)

15.Сумма зарплаты по отделам:

```
SELECT  depno, sum(salary) AS sumsal
        FROM emp
        GROUP BY depno;
```

DEPNO	SUMSAL
1	184000
2	191500
3	80500
5	185000

При использовании фразы GROUP BY существует правило, которого надо строго придерживаться: в списке выбора могут быть указаны **только** функции агрегирования, константы и поля, перечисленные в GROUP BY.

Если при использовании фразы GROUP BY необходимо вывести не все группы, а только те, которые удовлетворяют некоторому условию, то условие на результаты агрегирующих функций можно указать только в части HAVING. Это ограничение определяется порядком вычисления результатов команды SELECT: если указать условие на агрегирующую функцию в части WHERE, то на момент проверки этого условия значения агрегирующих функций ещё не подсчитано, поэтому его невозможно проверить. Ниже приведён пример использования фразы HAVING.

16.Вывести список отделов, в которых количество сотрудников больше трёх:

```
SELECT  depno as "Отдел", count(*) as "Количество сотрудников"
        FROM   emp
        GROUP BY  depno
        HAVING  count(*)>3;
```

Отдел	Количество сотрудников
1	4
2	4

1.4.6. Запрос SELECT на нескольких таблицах

Запрос SELECT на нескольких таблицах основан на декартовом произведении исходных таблиц. Если указать условие соответствия значений полей разных таблиц, то получится соединение таблиц. Для полей с одинаковыми названиями нужно указывать имя таблицы (или алиас) перед именем поля, разделяя их точкой (т.н. квалифицированная ссылка).

Примеры:

17. Запрос по двум таблицам. Вывести список сотрудников с детьми:

```
SELECT    e.name,  c.name AS child,  c.born
   FROM   emp  e,   children  c        /* e, c – алиасы таблиц*/
   WHERE  e.tabno = c.tabno            /* условие соединения */
   ORDER BY  e.name,  c.born;
```

NAME	CHILD	BORN
Буров Г.О.	Ольга	18.07.2001
Малова Л.А.	Илья	19.02.1987
Малова Л.А.	Анна	26.12.1989
Рюмин В.П.	Вадим	03.05.1995
Серова Т.В.	Роман	21.11.2006
Серова Т.В.	Инна	25.01.2008
Серова Т.В.	Антон	06.03.2009

18. Посчитать количество сотрудников по отделам:

```
SELECT  d.name, count(*), ' сотрудник(а)'
        FROM   emp  e,   depart  d
        WHERE  e.depno=d.depno
        GROUP BY  d.depno, d.name;
```

NAME	COUNT(*)	СОТРУДНИК(А)
Плановый отдел	4	сотрудник(а)
Бухгалтерия	4	сотрудник(а)
Отдел кадров	2	сотрудник(а)
Администрация	3	сотрудник(а)

Данные об отделе 4, в котором нет сотрудников, выведены не будут, т.к. для записи о 4-м отделе не выполняется условие соединения (e.depno=d.depno).

1.4.7. Подзапросы

Подзапросом называется запрос SELECT, который находится внутри другой команды SQL. Подзапросы можно разделить на следующие группы в зависимости от возвращаемых результатов:

- Скалярные – это подзапросы, возвращающие единственное значение.

- Векторные – подзапросы, возвращающие от 0 до нескольких однотипных элементов (список элементов).
- Табличные – это подзапросы, возвращающие таблицу.

Подзапросы бывают коррелированные и некоррелированные. **Коррелированные** подзапросы содержат ссылки на значения полей в запросе верхнего уровня, а некоррелированные – не содержат. Некоррелированный подзапрос вычисляется один раз для запроса верхнего уровня, а коррелированный – для каждой строки запроса верхнего уровня.

Сначала рассмотрим использование подзапросов в команде SELECT. Подзапросы могут располагаться в разных частях этой команды:
– в части FROM – табличные некоррелированные;
– в части WHERE – любые;
– в части HAVING – любые;
– в части SELECT – скалярные.

Подзапрос всегда стоит справа от оператора сравнения или предиката. Следующие операторы используются для модификации операторов сравнения:

- ALL – оператор, эквивалентный понятию "все". Например:
  > ALL (< ALL) – больше (меньше) каждого значения элементов результирующего множества.
- ANY (SOME) – оператор, эквивалентный понятию "любой".
  = ANY – равно одному из значений элементов результирующего множества (эквивалентно использованию предиката IN).
  > ANY (< ANY) – больше (меньше) любого значения элементов результирующего множества.
- EXISTS – оператор, эквивалентный понятию "существует". Может использоваться с логическим оператором NOT.

Если список, модифицированный оператором ALL, содержит NULL-значение, то результирующий запрос будет пуст, т.к. нельзя сравнить никакое значение с NULL-значением.

Выражение <>ANY(…) не эквивалентно NOT IN: оно выполняется всегда, кроме случаев сравнения со списком NULL-значений.

Примеры использования подзапросов в части WHERE:

19. Выдать список сотрудников, имеющих детей:

а) с помощью операции соединения таблиц:

```
SELECT  e.*
   FROM  emp e, children c
   WHERE e.tabno=c.tabno;
```

б) с помощью некоррелированного векторного подзапроса:

```
SELECT  *
   FROM  emp
   WHERE  tabno  IN  (SELECT  tabno FROM children);
```

в) с помощью коррелированного табличного подзапроса:

```
SELECT  *
```

```
FROM emp e
WHERE EXISTS (SELECT * FROM children c WHERE e.tabno=c.tabno);
```

Оператор EXISTS возвращает "истину", если подзапрос выбирает хотя бы одну строку, и "ложь", если результат подзапроса пуст.

TABNO	DEPNO	NAME	POST	SALARY	BORN	PHONE
988	1	Рюмин В.П.	начальник отдела	48500.0	01.02.1970	115-26-12
909	1	Серова Т.В.	вед. программист	48500.0	20.10.1981	115-91-19
110	2	Буров Г.О.	бухгалтер	42880.0	22.05.1975	115-46-32
023	2	Малова Л.А.	гл. бухгалтер	59240.0	24.11.1954	114-24-55

20. Выдать список сотрудников, оклад которых выше среднего на предприятии (некоррелированный скалярный подзапрос):

```
SELECT depno, name, salary
    FROM emp
    WHERE salary > ANY(SELECT avg(salary) FROM emp);
```

DEPNO	NAME	POST	SALARY
2	Малова Л.А.	гл. бухгалтер	59240
5	Павлов А.А.	директор	80000
5	Кроль А.П.	зам. директора	70000

Примеры использования подзапросов в части FROM:

21. Выдать список сотрудников, имеющих оклады выше среднего по отделу:

а) с помощью подзапроса в части WHERE:

```
SELECT depno, name, post, salary
    FROM emp e
    WHERE salary > (SELECT avg(salary) FROM emp m
                        WHERE m.depno=e.depno);
```

б) с помощью подзапроса в части FROM:

```
SELECT e.depno, name, post, salary
    FROM emp e,
    (SELECT depno, avg(salary) avgsal FROM emp GROUP BY depno) m
        WHERE e.depno=m.depno AND e.salary>avgsal;
```

Второй вариант будет работать быстрее, т.к. подзапрос в части FROM вычисляется один раз, а коррелированный подзапрос в части WHERE вычисляется для каждой строки запроса верхнего уровня (в нашем случае – для каждого сотрудника).

DEPNO	NAME	POST	SALARY
1	Рюмин В.П.	начальник отдела	48500.0
1	Серова Т.В.	вед. программист	48500.0
2	Малова Л.А.	гл. бухгалтер	59240.0
3	Сухова К.А.	начальник отдела	48500.0
5	Павлов А.А.	директор	80000.0
5	Кроль А.П.	зам.директора	70000.0

Пример использования подзапросов в части HAVING:

22. Выдать список отделов, в которых средние оклады ниже среднего оклада по предприятию:

```
SELECT  depno,  avg(salary)
   FROM  emp
   GROUP BY  depno
   HAVING  avg(salary) < (SELECT  avg(salary)  FROM  emp);
```

DEPNO	AVG(SALARY)
1	46000
2	47875
3	40250

Предложение UNION позволяет объединять результаты нескольких запросов SELECT для реализации соответствующей операции реляционной алгебры. Результаты этих запросов должны быть построены по одной схеме. Предложение ORDER BY может встречаться в таком запросе один раз – в конце последнего предложения SELECT.

Пример использования операции UNION:

23. Посчитать количество сотрудников по всем отделам (включая те отделы, в которых нет сотрудников):

```
SELECT  depno,  count(*), ' сотрудник(а)'
      FROM  emp
      GROUP BY  depno
   UNION
      SELECT  depno,  0,  ' сотрудников'
      FROM  depart
      WHERE  depno  NOT  IN  (SELECT  depno  FROM  emp)
   ORDER BY 1; /*  упорядочение по первому столбцу */
```

NAME	COUNT(*)	СОТРУДНИК(А)
1	4	сотрудник(а)
2	4	сотрудник(а)
3	2	сотрудник(а)
4	0	сотрудников
5	3	сотрудник(а)

А с помощью подзапроса в части SELECT можно запрос из примера 22 написать гораздо короче. (К сожалению, использование подзапроса в части SELECT поддерживается не всеми СУБД).

Пример использования подзапросов в части SELECT:

24. Подсчёт количества сотрудников по всем отделам (включая те отделы, в которых нет сотрудников):

```
SELECT  depno, (SELECT  count(*)  FROM  emp  e
                     WHERE  e.depno=d.depno)  AS  cnt
   FROM  depart  d
   ORDER BY 1; /*  упорядочение по первому столбцу */
```

1.4.8. Самосоединение

В команде SELECT можно обратиться к одной и той же таблице несколько раз. При этом для каждой таблицы необходимо задать свой алиас, чтобы можно было обращаться к полям этих таблиц. Система будет выполнять такой запрос на основе декартова произведения таблиц, поэтому необходимо указывать условие соединения. А для того чтобы исключить соединение записи таблицы с самой собой в запросе на самосоединение необходимо также указывать условие типа "не равно" (<>, >, <).

Пример использования самосоединения:

25. Вывести список детей сотрудников, у которых есть младшие братья или сёстры:

```
SELECT e.name, c1.name AS child1, c1.born AS born1,
       c2.name AS child2, c2.born AS born2
FROM children c1, children c2, emp e
WHERE c1.tabno=e.tabno          -- первое условие соединения
      AND c1.tabno=c2.tabno     -- второе условие соединения
      AND c1.born<c2.born       -- условие исключения
      ORDER BY 1, 3;
```

NAME	CHILD1	BORN1	CHILD2	BORN2
Малова Л.А.	Илья	19.02.1987	Анна	26.12.1989
Серова Т.В.	Роман	21.11.2006	Инна	25.01.2008
Серова Т.В.	Роман	21.11.2006	Антон	06.03.2009
Серова Т.В.	Инна	25.01.2008	Антон	06.03.2009

1.4.9. Замечания по использованию NULL-значений

Понятие неопределённого значения (NULL-значения) включено в стандарт SQL. Это значение не зависит от типа данных и может быть присвоено полю любого типа.

Значение NULL несравнимо ни с каким другим значением, даже со значением NULL. То есть при сравнении двух неопределённых значений A и B (A IS NULL и B IS NULL) условие (A=B) даст NULL, и это будет интерпретировано как "ложь". Таким образом, в SQL применяется трёхзначная логика ("да", "нет", "не знаю"), о чём надо помнить при формировании условий сравнения.

Пример влияния NULL-значений на результат:

26. Вывести список сотрудников, причём сначала – тех, у которых номер телефона начинается с '114', а затем – всех остальных:

```
SELECT e.*
   FROM emp e
      WHERE phone LIKE '114%'
UNION ALL
SELECT e.*
   FROM emp e
```

```
WHERE phone not LIKE '114%';
```

В результате выполнения данного запроса те сотрудники, у которых нет телефона (phone IS NULL), выведены не будут, хотя с точки зрения привычной двузначной логики условие (phone LIKE '114%' UNION ALL phone NOT LIKE '114%') даёт полное множество событий.

Тем не менее, из правила о несравнимости NULL-значений ни с какими другими значениями есть исключения:

- предложение GROUP BY объединяет все NULL-значения в одну группу,
- предикат DISTINCT <имя поля> оставляет только одно значение NULL,
- функция AVG не учитывает NULL-значения, и сумма значений поля делится на количество ненулевых (IS NOT NULL) значений.

1.4.10. Оператор CASE

Многие СУБД поддерживают оператор CASE. Этот оператор может быть использован в одной из двух синтаксических форм записи:

1-я форма:

```
CASE <проверяемое выражение>
   WHEN <сравниваемое выражение 1> THEN <возвращаемое значение 1>
   […
   WHEN <сравниваемое выражение N> THEN <возвращаемое значение N>]
   [ELSE <возвращаемое значение>]
END
```

2-я форма:

```
CASE
   WHEN <предикат 1> THEN <возвращаемое значение 1>
   […
   WHEN <предикат N> THEN <возвращаемое значение N>]
   [ELSE <возвращаемое значение>]
END
```

Все предложения WHEN должны иметь одинаковую синтаксическую форму, то есть нельзя смешивать первую и вторую формы. При использовании первой синтаксической формы условие WHEN удовлетворяется, как только значение проверяемого выражения станет равным значению выражения, указанного в предложении WHEN. При использовании второй синтаксической формы условие WHEN удовлетворяется, как только предикат принимает значение TRUE. При удовлетворении условия оператор CASE возвращает значение, указанное в соответствующем предложении THEN. Если ни одно из условий WHEN не выполнилось, то будет использовано значение, указанное в предложении ELSE. При отсутствии ELSE, будет возвращено NULL-значение. Если удовлетворены несколько условий, то будет возвращено значение предложения THEN первого из них, так как остальные просто не будут проверяться.

Примеры использования операции CASE:

27.Вывести список сотрудников с указанием количества сыновей и дочерей:

```
SELECT  e.name,
```

```
count(case when sex='м' then 1 else null end) sons,
count(case when sex='ж' then 1 else null end) daughters
FROM   emp e,  children c
   WHERE   e.tabno=c.tabno
   GROUP BY  e.name;
```

NAME	SONS	DAUGHTERS
Буров Г.О.	0	1
Малова Л.А.	1	1
Рюмин В.П.	1	0
Серова Т.В.	2	1

Обратите внимание: бездетные сотрудники в список не вошли.

28. Вывести список сотрудников с указанием тех, у кого самая высокая и самая низкая зарплата на предприятии:

```
SELECT  e.name, salary,
   (case  salary
      when (select max(salary) from emp) then 'самая высокая'
      when (select min(salary) from emp) then 'самая низкая'
      else 'средняя'
   end) as note
FROM   emp e;
```

NAME	SALARY	NOTE
Рюмин В.П.	48500	средняя
Серова Т.В.	48500	средняя
Волков Л.Д.	46500	средняя
Перова К.В.	32000	самая низкая
Буров Г.О.	42880	средняя
Малова Л.А.	59240	средняя
Сухова К.А.	48500	средняя
Лукина Н.Н.	42880	средняя
Дурова А.В.	43500	средняя
Тамм Л.В.	43500	средняя
Павлов А.А.	80000	самая высокая
Котова И.М.	35000	средняя
Кроль А.П.	70000	средняя

1.4.11. Работа с представлениями

Представление (view, обзор) – это хранимый запрос, создаваемый на основе команды *SELECT*. Представление реально не содержит данных. Запрос, определяющий представление, выполняется тогда, когда к представлению происходит обращение с другим запросом, например, SELECT, UPDATE и т.д.

Назначение представлений:

1. Хранение сложных запросов.
2. Представление данных в виде, удобном пользователю.
3. Сокрытие конфиденциальной информации.
4. Предоставление дифференцированного доступа к данным.

Создание представления выполняется командой **CREATE VIEW**:

```
CREATE [ OR REPLACE ] VIEW <имя представления>
      [ (<список имён столбцов>) ]
   AS <запрос> [ WITH CHECK OPTION ];
```

Запрос (команда SELECT), на основании которого создаётся представление, называется **определяющим запросом**, а таблицы, к которым происходит обращение в определяющем запросе – **базовыми таблицами**. Определяющий запрос по стандарту SQL не может включать предложение ORDER BY.

Если не указывать имена столбцов, то они получат названия по именам, перечисленным в списке выбора определяющего запроса. Указывать имена столбцов представления обязательно, если список выбора содержит агрегирующие функции или столбцы с одинаковыми именами из разных таблиц.

Примеры:

29. Создать представление "Сотрудники с детьми" (для удобного представления данных о детях сотрудников):

```
CREATE VIEW emp_child(depno, name, child, sex, born)
   AS SELECT e.depno, e.name, c.name, c.sex, c.born
        FROM emp e, children c
        WHERE e.tabno = c.tabno;

SELECT * FROM emp_child;
```

DEPNO	NAME	CHILD	SEX	BORN
2	Буров Г.О.	Ольга	ж	18.07.2001
2	Малова Л.А.	Илья	м	19.02.1987
2	Малова Л.А.	Анна	ж	26.12.1989
…	…	…	…	…
1	Серова Т.В.	Антон	м	06.03.2009

30. Создать представление "Сотрудники 2-го отдела" (для предоставления полного доступа к данным о сотрудниках 2-го отдела начальнику этого отдела):

```
CREATE VIEW emp2
   AS SELECT *
        FROM emp
        WHERE depno = 2;

SELECT * FROM emp2;
```

TABNO	DEPNO	NAME	POST	SALARY	BORN	PHONE
110	2	Буров Г.О.	бухгалтер	42880	22.05.75	115-46-32
100	2	Волков Л.Д.	программист	46500	16.10.82	
130	2	Лукина Н.Н.	бухгалтер	42880	12.07.79	115-46-32
023	2	Малова Л.А.	гл. бухгалтер	59240	24.11.54	114-24-55

31. Создать представление "Сотрудники" (без данных о зарплате, для сокрытия конфиденциальной информации):

```
CREATE VIEW employees
   AS SELECT tabno, depno, name, post, born, phone
```

```
FROM emp;
```

Представление может быть обновляемым и не обновляемым. Обновляемым является представление, при обращении к которому можно обновить базовую таблицу.

Пример обновления базовой таблицы ***emp*** через представление ***emp2***:

```
UPDATE  emp2
   SET  salary = 48000
   WHERE  tabno = '100';
```

Изменения будут произведены в базовой таблице и отразятся в представлении.

```
SELECT  *  FROM  emp2;
```

TABNO	DEPNO	NAME	POST	SALARY	BORN	PHONE
110	2	Буров Г.О.	бухгалтер	42880	22.05.75	115-46-32
100	2	Волков Л.Д.	программист	**48000**	16.10.82	
130	2	Лукина Н.Н.	бухгалтер	42880	12.07.79	115-46-32
023	2	Малова Л.А.	гл. бухгалтер	59240	24.11.54	114-24-55

Вносимые изменения могут выйти за рамки определяющего запроса и поэтому не будут видны через представление. Если необходимо защитить данные от такого вмешательства, то нужно в команде создания представления указать ключевые слова WITH CHECK OPTION: тогда система отвергнет изменения, выходящие за рамки определяющего запроса.

По стандарту SQL-2 представление не является обновляемым, если определяющий запрос:

1. содержит ключевое слово *DISTINCT*;
2. содержит множественные операции (*UNION* и др.);
3. содержит предложение *GROUP BY*;
4. ссылается на другое необновляемое представление;
5. содержит вычисляемые выражения в списке выбора;
6. выбирает данные более чем из одной таблицы.

1.4.12. Удаление объектов базы данных

Удаление объектов БД выполняется с помощью команды **DROP**.

- **DROP TABLE** – удаление таблицы:

  DROP TABLE <имя таблицы> [RESTRICT | CASCADE];

  Таблица будет удалена без дополнительного запроса на подтверждение вместе с данными и некоторыми другими объектами, существование которых зависит от наличия таблицы (индексы, триггеры и проч.). При указании *CASCADE* вместе с таблицей каскадно удаляются все зависящие от неё объекты БД (другие таблицы). Если указать *RESTRICT*, то при наличии зависимых от удаляемой таблицы объектов операция будет отменена.

Примечание: в СУБД Oracle необходимо указывать CASCADE CONSTRAINTS, т.к. каскадно удаляются только ограничения целостности, но не другие таблицы.