

# Язык запросов SQL. Введение

A decorative graphic consisting of several horizontal lines in shades of teal and light blue, extending across the width of the slide below the title.

# SQL – Structured Query Language

- **SQL** – это структурированный язык запросов к реляционным базам данных (БД).
- SQL – декларативный язык, основанный на операциях реляционной алгебры.
- Стандарты SQL, определённые Американским национальным институтом стандартов (ANSI):
  - ✓ SQL-1 (SQL/89) – первый вариант стандарта.
  - ✓ **SQL-2 (SQL/92) – основной расширенный стандарт.**
  - ✓ SQL-3 (SQL/1999, SQL/2003) – относится к объектно-реляционной модели данных.
- Подмножества языка SQL:
  - ✓ **DDL** (Data Definition Language) – команды создания/изменения/удаления объектов базы данных (*create/alter/drop*);
  - ✓ **DML** (Data Manipulation Language) – команды добавления/модификации/удаления данных (*insert/update/delete*), а также команда извлечения данных *select*;
  - ✓ **DCL** (Data Control Language) – команды управления данными (установка/снятие ограничений целостности). Входит в подмножество DDL.

# Команды DDL

CREATE – создание объекта.

ALTER – изменения структуры объекта.

DROP – удаление объекта.

Общий вид синтаксиса команд DDL:

create

alter *тип\_объекта имя\_объекта*  
*[параметры];*

drop

# Создание таблиц

```
CREATE TABLE [имя_схемы.]имя_таблицы  
  ( имя_поля тип_данных [(размер)] [NOT NULL]  
    [DEFAULT выражение]  
    [ограничения_целостности_поля...]  
    ,...  
    [, ограничения_целостности_таблицы ,...]  
  )  
  [ параметры ];
```

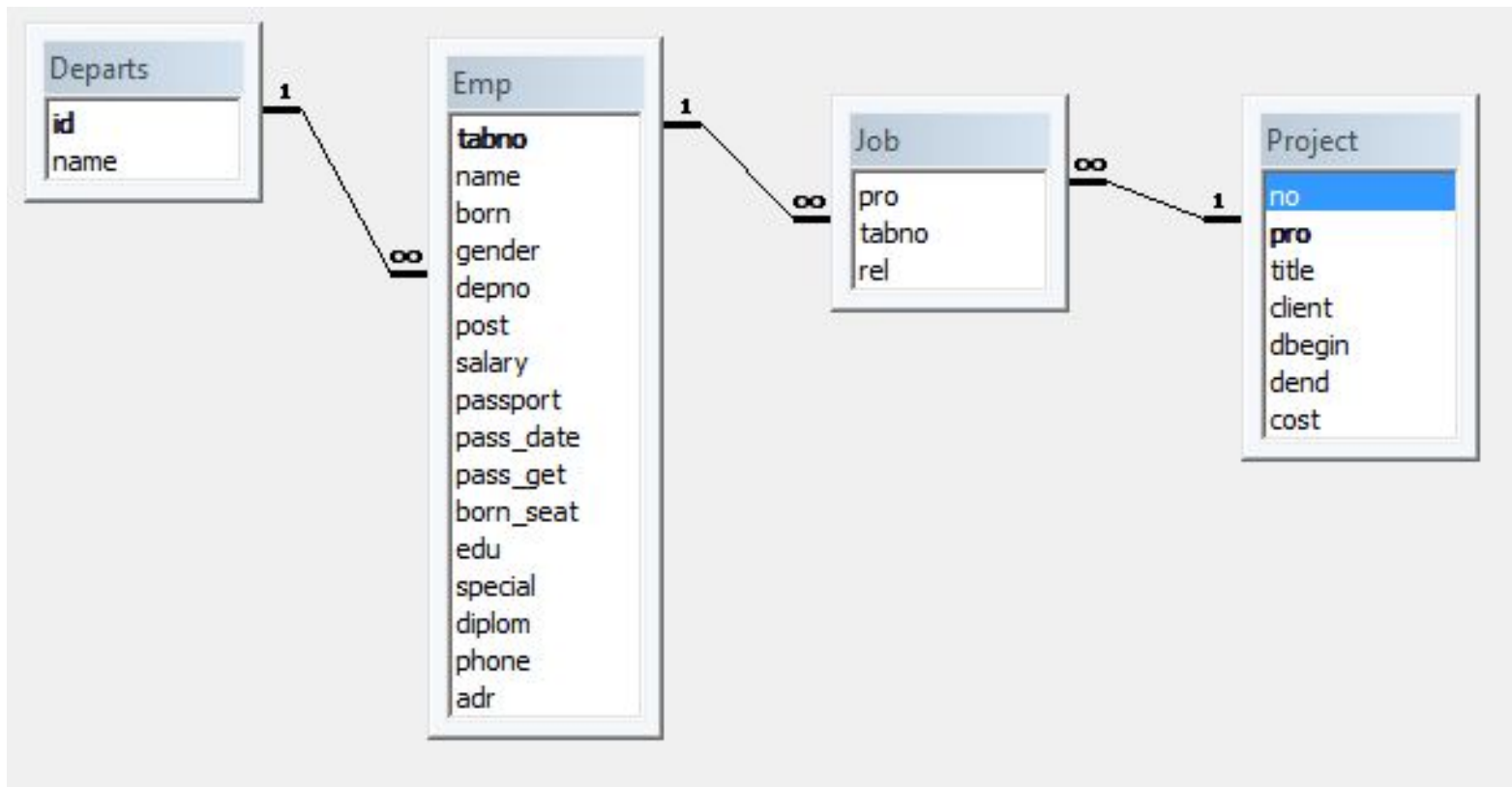
ограничения\_целостности (ОЦ):

```
[CONSTRAINT имя_ОЦ ] название_ОЦ [параметры]
```

# Типы данных

- Символьные типы:
  - ✓ **CHAR** [(длина)] – строка фиксированной длины.  
Длина по умолчанию – 1, максимальная длина 2000 б.  
Строка дописывается до указанной длины пробелами.
  - ✓ **VARCHAR2** (длина) – строка переменной длины.  
Максимальная длина 4000 б. Хранятся только значащие символы.
- Числовой тип:
  - ✓ **NUMBER** [(точность[, масштаб])] – используется для представления чисел с заданной точностью.  
Точность по умолчанию 38, масштаб по умолчанию – 0.  
number(4) – числа от -999 до 9999  
number(8,2) – числа от -99999.99 до 999999.99
- **DATE** – дата и время с точностью до секунды. Занимает 7 байт.
  - ✓ sysdate – функция получения текущих даты и времени.
  - ✓ Тип date поддерживает арифметику дат:  
sysdate+1 – завтра  
(дата1 – дата2) – количество дней, прошедших между двумя датами  
(sysdate – 0.5) – 12 часов назад

# Пример БД: проектная организация



Departs – отделы,

Project – проекты,

Emp – сотрудники,

Job – участие в проектах.

# Пример БД: проектная организация

**Emp** – сотрудники:

**tabno** – табельный номер сотрудника, первичный ключ;

**name** – ФИО сотрудника, обязательное поле;

**born** – дата рождения сотрудника, обязательное поле;

**gender** – пол сотрудника, обязательное поле;

**depno** – номер отдела, обязательное поле, внешний ключ;

**post** – должность сотрудника;

**salary** – оклад, больше МРОТ;

**passport** – серия и номер паспорта, уникальный обязательный атрибут;

**pass\_date** – дата выдачи паспорта, обязательное поле;

**pass\_get** – кем выдан паспорт, обязательное поле;

**born\_seat** – место рождения сотрудника;

**edu** – образование сотрудника;

**special** – специальность по образованию;

**diplom** – номер диплома;

**phone** – телефоны сотрудника;

**adr** – адрес сотрудника;

**edate** – дата вступления в должность, обязательное поле.

# Пример БД: проектная организация

**Departs** – отделы:

**did** – номер отдела, первичный ключ;

**name** – название отдела, обязательное поле.

**Project** – проекты:

**No** – номер проекта, первичный ключ;

**title** – название проекта, обязательное поле;

**pro** – краткое название проекта, обязательное уникальное поле;

**client** – заказчик, обязательное поле;

**dbegin** – дата начала выполнения проекта, обязательное поле;

**dend** – дата завершения проекта, обязательное поле;

**cost** – стоимость проекта, обязательное поле.

**Job** – участие в проектах:

**pro** – краткое название проекта, внешний ключ;

**tabNo** – номер сотрудника, участвующего в проекте, внешний ключ;

**rel** – роль сотрудника в проекте; может принимать одно из трех значений: 'исполнитель', 'руководитель', 'консультант'.

Первичный ключ – комбинация полей **pro** и **tabNo**.



# Создание таблиц БД проектной организации

## Таблица «Отделы» (Depart):

```
create table depart (did number(4) constraint pk_depart PRIMARY KEY,
                    name varchar2(100) not null
);
```

## Таблица «Сотрудники» (Emp):

```
create table emp ( tabno number(6) constraint pk_emp PRIMARY KEY,
                  name varchar2(100) not null,
                  born date not null,
                  gender char not null,
                  depno number(4) not null constraint fk_depart REFERENCES depart,
                  post varchar(50) not null,
                  salary number(8,2) not null constraint check_sal check (salary > 4630),
                  passport char(10) not null constraint passp_uniq UNIQUE,
                  pass_date date not null, pass_get varchar2(100) not null,
                  born_seat varchar2(100), edu varchar2(30),
                  special varchar2(100), diplom varchar2(40),
                  phone varchar2(30), adr varchar2(80),
                  edate date not null default trunc(sysdate),
                  chief number(6) constraint fk_emp REFERENCES emp
);
```

# Создание таблиц БД проектной организации

## Таблица «Проекты» (Project):

```
create table project (No number(5) constraint pk_project primary key,
    title varchar2(200) not null,
    pro varchar(15) not null constraint pro_uniq unique,
    client varchar(100) not null,
    dbegin date not null,
    dend date not null,
    cost number(9)
);
```

## Таблица «Участие в проектах» (Job):

```
create table job ( pro varchar(15) not null references project (abbr),
    tabNo number(6) not null references emp,
    rel varchar(20) default 'исполнитель',
    primary key (tabno, pro),
    check ( rel IN ('исполнитель', 'руководитель', 'консультант') )
);
```

# Подмножество команд DML

- **INSERT** – добавление строк в таблицу.
  - ✓ Добавляет одну или несколько строк в указанную таблицу.
- **UPDATE** – изменение данных.
  - ✓ Изменяет значения одного или нескольких полей в записях указанной таблицы.
  - ✓ Можно указать условие, по которому выбираются обновляемые строки.
  - ✓ Если условие не указано, обновляются все строки таблицы.
  - ✓ Если ни одна строка не удовлетворяет условию, ни одна строка не будет обновлена.
- **DELETE** – удаление строк из таблицы.
  - ✓ Удаляет одну или несколько строк из таблицы.
  - ✓ Можно указать условие, по которому выбираются удаляемые строки.
  - ✓ Если условие не указано, удаляются все строки таблицы.
  - ✓ Если ни одна строка не удовлетворяет условию, ни одна строка не будет удалена.

# Добавление данных

**INSERT** – добавление строк в таблицу:

```
INSERT INTO имя_таблицы [(список_полей_таблицы)]  
  { VALUES (список_выражений) | запрос };
```

Примеры:

-- Добавить в таблицу "Отделы" новую запись (все поля):

```
insert into depart  
  values(7, 'Договорной отдел');
```

-- Добавить в таблицу "Сотрудники" новую запись (не все поля):

```
insert into emp (tabno, name, born, gender, depno, passport, pass_date_pass_get,  
  post, salary, phone)  
  values( 301, 'САВИН АНДРЕЙ ПАВЛОВИЧ', to_date('11.07.1969',  
  'dd.mm.yyyy'),  
  'М', 5, '4405092876', to_date('15.02.1999', 'dd.mm.yyyy'),  
  'ОВД "Митино" г.Москвы', 'программист', 38050, '121-34-11');
```

**Замечание:** значение по умолчанию используется только тогда, когда значение поля не вводится в явном виде.

# Изменение данных

**UPDATE** – изменение данных:

**UPDATE** *имя\_таблицы*

**SET** *имя\_поля1* = *выражение1* [, *имя\_поля2* = *выражение2*,...]  
[**WHERE** *условие*];

Примеры:

-- Изменить статус сотрудника Бобкова Л.П., табельный номер 74, по отношению к проекту 30. "Система автоматизированного управления предприятием":

**update job**

**set rel = 'консультант'**

**where tabno = 74 and pro = 30;**

-- Перевести сотрудника Жаринова А.В., табельный номер 68, на должность ведущего программиста и повысить оклад на три тысячи рублей:

**update emp**

**set post = 'ведущий программист', salary = salary+3000**

**where tabno = 68;**

# Удаление данных

**DELETE** – удаление строк из таблицы:

```
DELETE FROM имя_таблицы  
[ WHERE условие ];
```

**Примеры.**

-- Удалить сведения о том, что сотрудник Афонасьев В.Н., табельный номер 147, участвует в проектах:

```
delete from job  
where tabno=147;
```

-- Удалить сведения о сотруднике Афонасьеве В.Н., табельный номер 147:

```
delete from emp  
where tabno = 147;
```

**Замечание:** отменить удаление данных можно командой  
ROLLBACK;

Язык запросов SQL.

Команда SELECT

# Команда SELECT – выборка данных

Общий синтаксис:

```
SELECT [{ ALL | DISTINCT }] { список_вывода | * }  
  FROM имя_таблицы1 [ алиас1 ] [, имя_таблицы2 [ алиас2 ],...]  
  [ WHERE     условие_отбора_записей ]  
  [ GROUP BY { имя_поля | выражение },... ]  
  [ HAVING     условие_отбора_групп ]  
  [ UNION [ALL] SELECT ... ]  
  [ ORDER BY имя_поля1 | целое [ ASC | DESC ]  
    [, имя_поля2 | целое [ ASC | DESC ],...]];
```

Примеры:

```
select * from departs;  
select name, post from emp;
```



# Формирование списка вывода (проекция)

Общий синтаксис списка вывода:

[{all | distinct}] { \* | *выражение1* [*алиас1*] [, *выражение2* [*алиас2*] ,...]}

Список вывода находится между ключевыми словами **SELECT** и **FROM**.

- Вывести все поля всех записей из таблицы Проекты (Project):  
**select \* from project;**
- 2. Вывести список сотрудников с указанием их должности и № отдела:  
**select depno, name, post  
from emp;**
- 3. Вывести список сотрудников с указанием их должности и зарплаты:  
**select name 'ФИО', post 'Должность', salary\*0.87 'Зарплата'  
from emp;**

# Формирование списка вывода (проекция)

1. **select post, salary  
from emp;**
2. **select DISTINCT post, salary  
from emp;**
3. **select DISTINCT depno, post  
from emp;**

```
select name 'ФИО', born 'Дата рождения', adr 'Адрес'  
from emp;
```

# Упорядочение резултата

1. **select \***  
**from Project**  
**order by dbegin;**
2. **select depno, name, post**  
**from emp**  
**order by depno, name; -- order by 1,2;**
3. **select name 'ФИО', post 'Должност', salary 'Зарплата'**  
**from emp**  
**order by 3 DESC;**
4. **select depno 'Номер отдела', post 'Должност', salary 'Зарплата'**  
**from emp**  
**order by 1, 3 DESC, 2;**

# Выбор данных из таблицы (селекция)

**WHERE** – содержит условия выбора отдельных записей. Условие является логическим выражением и может принимать одно из 3-х значений:

- TRUE – истина,
- FALSE – ложь,
- NULL – неизвестное, неопределённое значение (интерпретируется как ложь).

Условие формируется путём применения различных операторов и предикатов.

## **Операторы сравнения:**

= равно, <>, != не равно, > больше,  
>= больше или равно, <= меньше или равно, < меньше.

1. Вывести список сотрудников 2-го отдела:

```
select * from emp  
  where depno = 2;
```

# Логические операторы

Для формирования условий используются следующие логические операторы:

AND – логическое произведение (И),

OR – логическая сумма (ИЛИ),

NOT – отрицание (НЕ).

Операция И:

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

Операция ИЛИ:

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

Операция НЕ:

a	NOT a
0	1
1	0

# Выбор данных из таблицы по условию

1. **select \* from emp  
where depno = 2 AND salary > 3000 ;**
2. **select \* from emp  
where born > '31/12/1979' AND sex = 'м';**
3. **select \* from emp  
where depno=2 OR depno = 5;**
4. **select \* from emp  
where (depno=2 OR depno = 5) AND salary >= 3000 ;**
5. **select \* from emp  
where NOT (depno=2 OR depno = 5);**

# Выбор данных из таблицы по условию

Задание 1

```
select *  
  from project  
  where dend > sysdate AND cost > 2000000;
```

Задание 2

```
select *  
  from emp  
  where post = 'инженер' OR post = 'ведущий инженер' ;
```

Задание 3:

```
select *  
  from emp  
  where post = 'охранник' AND salary > 2000;
```

# Предикаты формирования условия

## Предикат вхождения в список значений:

*имя\_поля* IN ( *значение1* [, *значение2*,... ] )

*выражение* IN ( *значение1* [, *значение2*,... ] )

## Примеры:

```
select *  
  from emp  
  where depno IN ( 5, 8, 9 );
```

- ```
select *  
  from emp  
  where post IN ( 'инженер', 'ведущий инженер' );
```



# Предикаты формирования условия

## Предикат вхождения в диапазон:

*имя\_поля BETWEEN минимальное\_значение AND максимальное\_значение*  
*выражение BETWEEN минимальное\_значение AND максимальное\_значение*

Минимальное значение должно быть меньше либо равно максимальному.

## Примеры:

```
select *  
  from emp  
  where depno BETWEEN 2 AND 5 ;
```

- ```
select *  
  from emp  
  where salary*0.87 BETWEEN 2000 AND 3000;
```

# Предикаты формирования условия

**Предикат поиска подстроки: *имя\_поля* LIKE '*шаблон*'**

Этот предикат применяется только к полям типа CHAR и VARCHAR.

Возможно использование шаблонов:

'\_' – один любой символ,

'%' – произвольное количество любых символов (в т.ч., ни одного).

## Примеры:

**1. select \* from emp  
where post LIKE '%экономист%' ;**

**2. select \* from emp  
where post LIKE 'инженер\_%' ;**

# Предикаты формирования условия

## **Предикат поиска неопределенного значения:**

*значение* **IS [NOT] NULL**

Если значения является неопределенным (NULL), то предикат IS NULL выдаст истину, а предикат IS NOT NULL – ложь.

### Примеры:

```
select *  
  from emp  
 where phone IS NULL ;
```

```
select *  
  from project  
 where cost IS NOT NULL ;
```

# Использование предикатов

Задание 1:

```
select *  
  from emp  
  where name LIKE '%ЮРИЙ%';
```

Задание 2:

```
select *  
  from project  
  where cost BETWEEN 1000000 AND 2000000;
```

Задание 3:

```
select *  
  from emp  
  where post LIKE 'нач%отдел%';
```

# Базы данных

Язык запросов SQL.  
Команда SELECT (продолжение)

# Агрегирующие функции

**COUNT** – подсчёт количества строк (значений). Применяется к записям и полям любого типа. Имеет 3 формата вызова:

- **count (\*)** – количество строк результата;
- **count (имя\_поля)** – количество значений указанного поля, не являющихся *NULL*-значениями.
- **count (distinct имя\_поля)** – количество разных не-NULL значений указанного поля.

**MAX, MIN**

**SUM**

**AVG**

# Примеры использования функции COUNT

1. 

```
select count(*)  
  from emp;
```
2. 

```
select count( phone )  
  from emp;
```
3. 

```
select count (DISTINCT post)  
  from emp;
```
4. Задание: вывести количество сотрудников 6-го отдела.

```
select count(*)  
  from emp  
  where depno = 6;
```

# Примеры использования агрегирующих функций

1. 

```
select max(cost) "Максимальная цена", min(cost) "Минимальная цена"  
from project;
```
2. 

```
select sum(salary)  
from emp  
where depno = 8;
```
3. 

```
select avg(salary)  
from emp  
where sex = 'Ж';
```
4. 

```
select min(dbegin), max(dend)  
from project;
```



# Группировка данных: предложение GROUP BY

Агрегирующие функции обычно используются совместно с предложением **GROUP BY**.

Например, следующая команда считает количество сотрудников по отделам:

```
select depno, count(*)  
  from emp  
  group by depno;
```

depno	name	...
1	Белов С.В.	
1	Иванова К.Е.	
1	Седов О.Л.	
2	Волков Н.Е.	
2	Рогов И.Л.	
3	Санина В.П.	
3	ДЫМОВА С.Т.	
3	Павлов К.Д.	
3	Орлов Т.Ф.	

depno	count(*)
1	3
2	2
3	4

## Примеры использования GROUP BY

1. **select depno, MIN(salary) minsal, MAX(salary) maxsal  
from emp  
group by depno;**

2. **select depno, COUNT(distinct post) cnt  
from emp  
group by depno;**

**select depno, SUM(salary) allsal  
from emp  
group by depno;**

4. **select post, AVG(salary) avgsal  
from emp  
group by post;**

# Использование GROUP BY

Правило использования *GROUP BY* :

В списке вывода при использовании *GROUP BY* могут быть указаны только функции агрегирования, константы и поля, перечисленные в *GROUP BY*.

Например, **нельзя** получить сведения о том, у каких сотрудников самая высокая зарплата в своём отделе с помощью такого запроса:

```
select depno, name, max(salary) as max_sal  
from emp  
group by depno;
```

**Этот запрос синтаксически неверен!**

depno	name	salary
1	Белов С.В.	58000
1	Иванова К.Е.	28000
1	Седов О.Л.	41000
2	Волков Н.Е.	40000
2	Рогов И.Л.	32000
3	Санина В.П.	47000
3	Дымова С.Т.	29000
3	Павлов К.Д.	47000
3	Орлов Т.Ф.	30000

# Группировка по нескольким полям

1. `select depno, post, count(*), sum(salary)  
from emp  
group by depno, post;`

2. `select depno, sex, count(*)  
from emp  
group by depno, sex;`

**Задание:** вывести информацию о зарплате и количестве сотрудников, которые получают такую зарплату.

```
select salary, count(*)  
from emp  
group by salary;
```

# Использование фразы HAVING

Если необходимо вывести не все записи, полученные в результате группировки (GROUP BY), то условие на группы можно указать во фразе HAVING.

Пример:

```
select depno, count(*), 'человек(a)'  
  from emp  
  group by depno  
  having count(*)>5;
```

**Правило: нельзя указывать агрегирующие функции в части WHERE – это синтаксическая ошибка!**

**Задание:** вывести список отделов, в которых средняя зарплата больше 3000 грн.

```
select depno, avg(salary)  
  from emp  
  group by depno  
  having avg(salary) > 3000;
```

# Операции реляционной алгебры

## Унарные операции:

### □ селекция

Например:

```
select *  
  from emp  
  where depno = 5;
```

### □ проекция

Например:

```
select distinct name, post, salary  
  from emp;
```

# Общий алгоритм выполнения операции *SELECT*

1. Выбор записей из указанной таблицы (*from*).
2. Проверка для каждой записи условия отбора (*where*).
3. Группировка полученных в результате отбора записей (*group by*) и вычисление для этих групп значений агрегирующих функций.
4. Выбор тех групп, которые удовлетворяют условию отбора групп (*having*).
5. Сортировка полученных записей в указанном порядке (*order by*).
6. Извлечение из полученных записей тех полей, которые заданы в списке вывода, и формирование результирующего отношения.

Если в части FROM указывается 2 и более таблицы, то приведенный алгоритм выполняется для декартова произведения этих таблиц.

# Базы данных

Команда `SELECT`.  
Дополнительные возможности



# Самосоединение

В команде SELECT можно обратиться к одной и той же таблице несколько раз. А для того чтобы исключить соединение записи таблицы с самой собой в запросе на самосоединение необходимо также указывать условие типа "не равно" ( $\neq$ ,  $>$ ,  $<$ ).

## Пример использования самосоединения:

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;
```

# Результат самосоединения

<i>TabNo</i>	<b>Name</b>	Born	Sex
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

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

# Подзапросы

**Подзапрос** – это запрос SELECT, расположенный внутри другой команды.

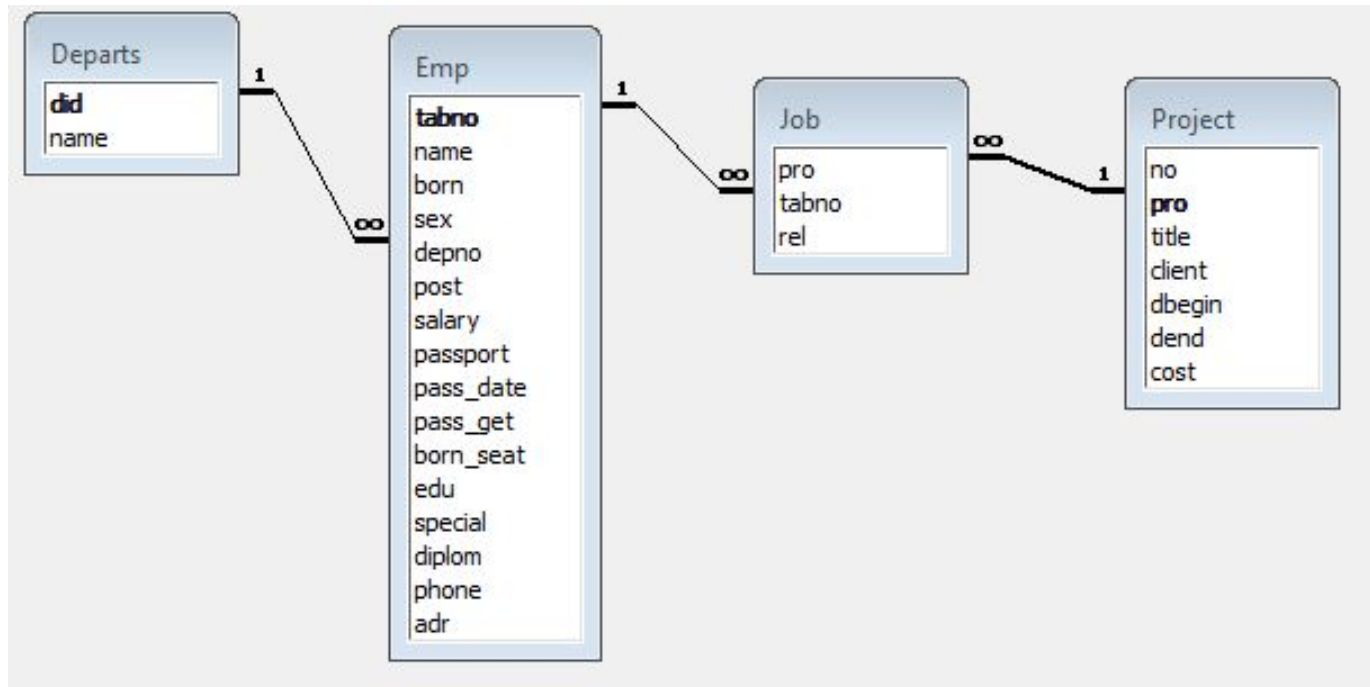
Подзапросы можно разделить на следующие группы в зависимости от возвращаемых результатов:

- ✓ **скалярные**
- ✓ **векторные**
- ✓ **табличные**

Подзапросы бывают:

- ✓ **некоррелированные** – не содержат ссылки на запрос верхнего уровня; вычисляются один раз для запроса верхнего уровня;
- ✓ **коррелированные** – содержат условия, зависящие от значений полей в основном запросе; вычисляются для каждой строки запроса верхнего уровня.

# Пример БД: проектная организация



Departs – отделы,

Project – проекты,

Emp – сотрудники,

Job – участие в проектах.

## Данные таблицы Emp (сотрудники)

TabNo	DepNo	Name	Post	Salary	Born	Phone
988	1	Рюмин В.П.	начальник отдела	4850.00	01.02.1970	115-26-12
909	1	Серова Т.В.	вед. программист	4850.00	20.10.1981	115-91-19
829	1	Дурова А.В.	экономист	4350.00	03.10.1978	115-26-12
819	1	Тамм Л.В.	экономист	4350.00	13.11.1985	115-91-19
100	2	Волков Л.Д.	программист	4650.00	16.10.1982	null
110	2	Буров Г.О.	бухгалтер	4288.00	22.05.1975	115-46-32
023	2	Малова Л.А.	гл. бухгалтер	5924.00	24.11.1954	114-24-55
130	2	Лукина Н.Н.	бухгалтер	4288.00	12.07.1979	115-46-32
034	3	Перова К.В.	делопроизводитель	3200.00	24.04.1988	null
002	3	Сухова К.А.	начальник отдела	4850.00	08.06.1948	115-12-69
056	5	Павлов А.А.	директор	8000.00	05.05.1968	115-33-44
087	5	Котова И.М.	секретарь	3500.00	16.09.1990	115-33-65
088	5	Кроль А.П.	зам.директора	7000.00	18.04.1974	115-33-01

# Расположение подзапросов в командах DML

В команде **INSERT**:

- Вместо **VALUES**, например, добавление данных из одной таблицы в другую:  
insert into emp select \* from new\_emp;

В команде **UPDATE**:

- в части **WHERE** для вычисления условий, например, повышение зарплаты на 10% всем участникам проектов:

```
update emp set salary = salary*1.1  
    where tabNo IN (select tabNo from job);
```

- в части **SET** для вычисления значений полей, например, повышение зарплаты на 10% за каждое участие сотрудника в проекте:

```
update emp e set salary = salary*(1+(select count(*)/10 from job j  
    where j.tabNo = e.tabNo) );
```

В команде **DELETE**:

- в части **WHERE** для вычисления условий, например, удаление сведений об участии в закончившихся проектах:

```
delete from job  
    where pro IN (select pro from project where dend < sysdate);
```

# Расположение подзапросов в команде select

- Чаще всего подзапрос располагается в части **WHERE**.

Пример 1:

```
select * from emp  
where salary > (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

Пример 2. :

```
select * from emp  
where salary > ALL (select avg(salary) from emp group by depno);
```

# Примеры использования подзапросов в части WHERE

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);
```



# Расположение подзапросов в команде select

□ Подзапрос в части **FROM**.

Например,

```
select * from emp e
      where salary > (select avg(salary) from emp m
                      where m.depno = e.depno);
```

Это работает долго, т.к. коррелированный подзапрос вычисляется для каждой строки основного запроса. Можно ускорить выполнение данного запроса:

```
select *
      from emp e,
      (select depno, avg(salary) sal
       from emp
       group by depno) m -- подзапрос вычисляется 1 раз
      where m.depno = e.depno
            and salary > sal;
```

# Расположение подзапросов в команде select

- Подзапрос в части **HAVING**.

Например,

```
select depno, avg(salary) sal  
  from emp  
  group by depno  
  having avg(salary) < (select avg(salary) from emp);
```

- Подзапрос в части **SELECT**.

Например,

```
select depno, name,  
       (select count(*) from job j where j.tabno = e.tabno) cnt  
  from emp e;
```

Этот запрос выведет даже тех сотрудников, которые не участвуют в проектах (для них **cnt** будет равен 0).

# Представления

Представление (view, обзор) – это хранимый запрос, создаваемый на основе команды *SELECT*.

Назначение представлений:

- Хранение сложных запросов.
- Представление данных в виде, удобном пользователю.
- Соккрытие конфиденциальной информации.
- 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**.

## Представления: пример

```
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

## Представления: пример

```
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

# Представления: примеры

1. CREATE VIEW employees  
AS SELECT tabno, depno, name, post, born, phone  
FROM emp;
2. CREATE VIEW pro\_stat  
AS SELECT title, e.name,  
(select count(\*) from job j where j.pro=p.pro and rel='исполнитель') jobs,  
(select count(\*) from job j where j.pro=p.pro and rel='консультант') consult  
FROM emp e, project p, job j  
where e.tabno=j.tabno and j.pro=p.pro  
and j.rel='руководитель';

# Обновляемые представления

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

```
UPDATE emp2
```

```
SET salary = 4800.00
```

```
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	4288.00	22.05.75	115-46-32
100	2	Волков Л.Д.	программист	<b>4800.00</b>	16.10.82	
130	2	Лукина Н.Н.	бухгалтер	4288.00	12.07.79	115-46-32
023	2	Малова Л.А.	гл. бухгалтер	5924.00	24.11.54	114-24-55

# Обновляемые представления

Вносимые изменения могут выйти за рамки определяющего запроса и поэтому не будут видны через представление. Необходимо 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:

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



# Оператор CASE

Оператор **CASE** 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>

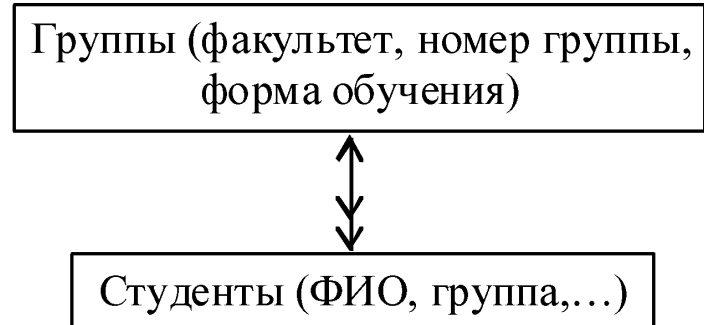
    [ELSE <возвращаемое значение>]

END

# Примеры использования оператора CASE

1) Посчитать количество студентов дневной и вечерней формы обучения:

```
create view students_number  
(DEPARTMENT, YEAR, DAY_FORM, EVENING_FORM) as  
select gr.department, gr.year,  
       count(case when gr.study='ДНЕВНАЯ' then 1 else null end) form1,  
       count(case when gr.study='ВЕЧЕРНЯЯ' then 1 else null end) form2  
from groups gr, students st  
where gr.group_code = st.group_code  
group by gr.department, gr.year, gr.study  
order by gr.department, gr.year asc;
```



# Примеры использования оператора CASE

2) Вывести все имеющиеся модели ПК с указанием цены. Отметить самые дорогие и самые дешевые модели.

```
SELECT DISTINCT model, price,  
CASE price  
WHEN (SELECT MAX(price) FROM  
PC)  
THEN 'Самый дорогой'  
WHEN (SELECT MIN(price) FROM PC)  
THEN 'Самый дешевый'  
ELSE 'Средняя цена'  
END comment  
FROM PC  
ORDER BY price;
```

model	price	comment
1232	350.0	Самый дешевый
1260	350.0	Самый дешевый
1232	400.0	Средняя цена
1232	600.0	Средняя цена
1233	600.0	Средняя цена
1121	850.0	Средняя цена
1233	950.0	Средняя цена
1233	980.0	Самый дорогой