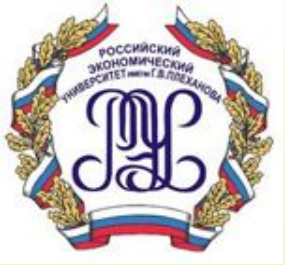


Тема 10.
Язык запросов SQL

Команда SELECT

Вопросы лекции:

1. Общий синтаксис команды SELECT
2. Общий алгоритм выполнения операции SELECT
3. Формирование списка вывода (проекция)
4. Использование псевдонимов
5. Упорядочение результата
6. Выбор данных из таблицы (селекция)
7. Предикаты формирования условия
8. Агрегирующие функции
9. Операции реляционной алгебры на SQL
0. Подзапросы
1. Представления
2. Оператор CASE
3. Курсоры
4. Запросы на объединение

http://www.sql.ru/docs

http://www.sql-ex.ru

Общий синтаксис команды SELECT

Формат запроса с использованием оператора **SELECT:**

SELECT *список полей* **FROM** *список таблиц* **WHERE** *условия…*

Таблица «T»	Запрос	Результат
C1 C2 1 a 2 b	`SELECT * FROM T;`	C1 C2 1 a 2 b
C1 C2 1 a 2 b	`SELECT C1 FROM T;`	C1 1 2
C1 C2 1 a 2 b	`SELECT * FROM T WHERE C1 = 1;`	C1 C2 1 a
C1 C2 1 a 2 b	`SELECT * FROM T ORDER BY C1 DESC;`	C1 C2 2 b 1 a

Команда SELECT – выборка данных

Общий синтаксис:

SELECT *список полей* **FROM** *список таблиц* **WHERE** *условия…*

SELECT [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;
```

Общий алгоритм выполнения операции SELECT

* 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

Формирование списка вывода (проекция)

Формирование списка вывода (проекция)

Общий синтаксис списка вывода:

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

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

1. 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;

Установить другой формат вывода даты:

alter session set nls_date_format = 'dd/mm/yyyy';

Формирование списка вывода (проекция)

1. Вывести должности и оклады сотрудников:

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

2. Вывести должности и оклады сотрудников **без повторов**:

```
select  DISTINCT  post,  salary
      from  emp;
```

3. Вывести отделы и должности сотрудников **без повторов**:

```
select  DISTINCT  depno,  post
      from  emp;
```

4. Задание: вывести список сотрудников с указанием ФИО, даты рождения и адреса.

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

Использование псевдонимов

SQL Alias

SQL Alias – псевдонимы могут быть использоваться для переименования таблиц и колонок.

Существует возможность задавать таблицам или столбцам другие имена, используя для этого псевдоним. Это может быть полезным, если у нас очень длинные или сложные имена таблиц или столбцов.

Псевдоним может быть каким угодно, но обычно это короткие имена.

SQL Alias

Синтаксис псевдонимов для таблиц SQL

```
SELECT column_name(s) FROM table_name AS alias_name;
```

Синтаксис псевдонимов для столбцов SQL

```
SELECT column_name AS alias_name FROM table_name;
```

SQL Alias

Пример

Предположим, мы имеем одну таблицу под названием "Persons", а другую таблицу под названием "Product_Orders". Первой мы присвоим псевдоним "p“, второй – "po".
Получим список всех заказов, которые имеет "Ola Hansen".

Запрос с использованием псевдонимов

```
SELECT
po.OrderID,
p.LastName,
p.FirstName
FROM
Persons AS p,
Product_Orders AS
po WHERE
p.LastName='Hansen'
AND
p.FirstName='Ola'
```

Запрос без использования псевдонимов

```
SELECT
Product_Orders.OrderID,
Persons.LastName,
Persons.FirstName
FROM
Persons, Product_Orders
WHERE
Persons.LastName='Hansen'
AND
Persons.FirstName='Ola'
```

Упорядочение результата

Упорядочение результата

1. Вывести данные из таблицы Проекты в порядке даты начала проекта:

```
select *
    from Project
    order by  dbegin;
```

2. Упорядочить список сотрудников по отделам и по ФИО:

```
select depno, name, post
    from emp
    order by  depno,  name;  -- order by 1,2;
```

3. Упорядочить сотрудников по зарплате (от большей к меньшей):

```
select name  'ФИО',  post  'Должность',  salary  'Зарплата'
    from emp
    order by  3  DESC;
```

4. Упорядочить данные об отделах, должностях и зарплатах:

```
select depno  'Номер отдела',  post  'Должность',  salary
'Зарплата'
    from emp
    order by  1,  3  DESC,  2;
```

Выбор данных из таблицы (селекция)

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,

UNKNOWN – неизвестное, неопределённое значение (интерпретируется как ложь).

Условие формируется путём применения различных операторов и предикатов.

Операторы сравнения:

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

1. Вывести список сотрудников 2-го отдела:

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

2. Вывести список текущих проектов:

```
select *  from  project
      where  dend > curdate();
```

-- curdate() – функция, возвращающая текущую дату

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

Примеры использования логических операторов

1. Вывести список сотрудников 2-го отдела с зарплатой больше 30000 рублей:

```
select * from emp
    where depno = 2 AND salary > 30000 ;
```

2. Вывести список сотрудников-мужчин, родившихся после 1979 года:

```
select * from emp
    where born > '31/12/1979' AND sex = 'м';
```

3. Вывести список сотрудников 2-го и 5-го отделов:

```
select * from emp
    where depno=2 OR depno = 5;
```

4. Вывести список сотрудников 2-го и 5-го отделов в зарплатой не менее 30000:

```
select * from emp
    where (depno=2 OR depno = 5) AND salary >= 30000 ;
```

5. Вывести список всех сотрудников, кроме сотрудников 2-го и 5-го:

```
select * from emp
    where NOT (depno=2 OR depno = 5);
```

Примеры использования логических операторов

Вывести список текущих проектов стоимостью более 2 млн. рублей.

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

Вывести список сотрудников, работающих в должностях 'инженер' и 'ведущий инженер'.

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

Вывести список сотрудников, работающих в должности 'охранник', с зарплатой более 20000 рублей.

```
select  *
 from  emp
 where  post = 'охранник'  AND  salary > 20000;
```

Выбор данных из таблицы (селекция)

Предикаты формирования условия

Предикат –

любое выражение, результатом которого являются значения TRUE, FALSE или UNKNOWN. Предикаты используются в условиях поиска предложений WHERE и HAVING в условиях соединения предложений FROM и других конструкциях, где требуется логическое значение.

Предикаты формирования условия

Предикат вхождения в список значений:

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

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

Примеры:

1. **Список сотрудников отделов 5, 8 и 9:**

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

2. **Список сотрудников, работающих в должностях 'инженер' и 'ведущий инженер' :**

```
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:

1. **Список всех сотрудников со 2-го по 5-й отделы:**

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

2. **Список сотрудников с чистой зарплатой от 20 до 30 тысяч рублей:**

```
select  *
    from  emp
    where  salary*0.87  BETWEEN  20000 AND  30000;
```

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:

Список всех сотрудников-экономистов:

```
select *  from emp
    where post  LIKE  '%экономист%' ;
```

Список всех инженеров-специалистов (кроме просто инженеров):

```
 select  *  from  emp
    where  post  LIKE  'инженер_%' ;
```

Экранировать специальное значение символов '_' и '%' можно так:

```
where  <строка>  LIKE  '_#%%'  ESCAPE  '#';
```

Символ экранирования (escape) может быть любым. В примере первый символ % будет искаться как символ, а второй имеет специальное значение.

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:

Список всех сотрудников, у которых нет телефона (номер телефона неопределен):

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

Список все проекты, у которых определена стоимость:

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

Примеры использования предикатов

Вывести список сотрудников, которых зовут 'ЮРИЙ'.

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

Вывести список проектов стоимостью от 1 до 2 млн. рублей.

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

Вывести список сотрудников, которые являются начальниками отделов.

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

Агрегирующие функции

Агрегирование и групповые функции COUNT,SUM,AVG,MAX,MIN

Агрегирующие функции позволяют получать из таблицы **сводную (агрегированную)** информацию, выполняя операции над группой строк таблицы. Для задания в SELECT запросе агрегирующих операций используются следующие ключевые слова:

COUNT определяет количество строк или значений поля, выбранных посредством запроса и не являющихся NULL-значениями;

SUM вычисляет арифметическую сумму всех выбранных значений данного поля;

AVG вычисляет среднее значение для всех выбранных значений данного поля;

MAX вычисляет наибольшее из всех выбранных значений данного поля;

MIN вычисляет наименьшее из всех выбранных значений данного поля.

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 – определяет максимальное (минимальное) значение указанного поля в результирующем множестве. Применяется к полям любого типа.

SUM – определяет арифметическую сумму значений указанного числового поля в результирующем множестве записей.

AVG – определяет среднее арифметическое значений указанного числового поля в результирующем множестве записей. Не учитывает *NULL*-значения, и сумма значений поля делится на количество определённых значений.

Примеры использования функции COUNT

1. Вывести количество сотрудников:

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

2. Вывести количество сотрудников с телефонами:

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

3. Вывести количество разных должностей сотрудников:

```
 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. Вывести максимальную и минимальную стоимость проектов:

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

2. Вывести сумму зарплаты сотрудников 8-го отдела:

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

3. Вывести среднюю зарплату сотрудниц предприятия:

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

4. Вывести даты начала работы над первым проектом и завершения работы над последним проектом:

```
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. Вывести минимальную и максимальную зарплату в каждом отделе:

```
select depno, MIN(salary) minsal, MAX(salary) maxsal
    from emp
    group by depno;
```

2. Вывести количество разных должностей в каждом отделе:

```
select depno, COUNT(distinct post) cnt
    from emp
    group by depno;
```

3. Посчитать сумму зарплат в каждом отделе:

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

4. Посчитать среднюю зарплату по каждой должности:

```
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.

Если включить в список выбора поля, не указанные в *GROUP BY*, то СУБД не будет выполнять такой запрос и выдаст ошибку "нарушение условия группирования" (not a GROUP BY expression).

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. Сумма зарплаты по отделам и по должностям:

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

2. Количество мужчин и женщин по отделам:

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

3. Информация 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 (но не во фразе WHERE).

Пример. Список отделов, в которых работает больше пяти человек:

```
select  depno,  count(*), 'человек(а)'
   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 30000 рублей.

```
select  depno,  avg(salary)
   from  emp
   group by  depno
   having  avg(salary) > 30000;
```

Операции реляционной алгебры на SQL

Операции реляционной алгебры

Унарные операции:

- **селекция** – выбор из таблицы подмножества строк по условию.

  Например, список сотрудников 5-го отдела:

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

- **проекция** – выбор из таблицы подмножества столбцов.

  Например, сведения о должности и зарплате сотрудников:

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

Бинарные операции реляционной алгебры

Бинарные операции РА:

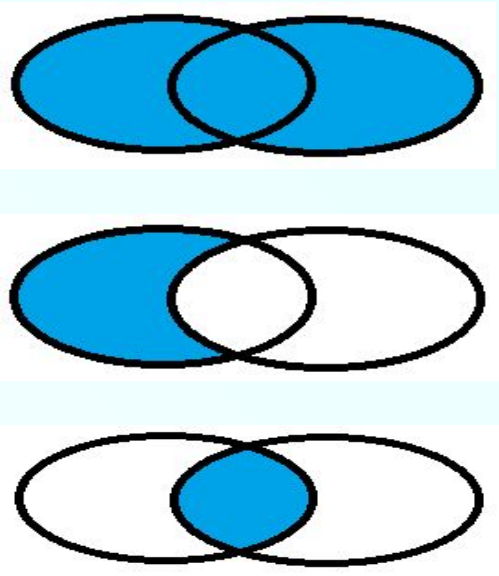
- **разносхемные** – применяются к любым двум отношениям.
- **односхемные** – применяются к односхемным отношениям. Исходные отношения должны иметь одинаковое количество столбцов одинаковых (или сравнимых) типов. ***Сравнимыми*** считаются типы, относящиеся к одному и тому же семейству данных (в таблице полужирным шрифтом выделены базовые типы).

Семейства типов данных MySQL:

Числовые:	**Символьные:**	**Календарные:**
DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, **NUMBER,** NUMERIC, REAL, SMALLINT	CHAR, CHARACTER, LONG, LONG RAW RAW, ROWID, STRING, VARCHAR, **VARCHAR2**	**DATE**

Бинарные односхемные операции РА

- ✔ **Объединение** двух односхемных отношений содержит все строки исходных отношений без повторов.
- ✔ **Разность** двух односхемных отношений содержит все строки первого отношения, не входящие во второе отношение (без повторов).
- ✔ **Пересечение** двух односхемных отношений содержит все строки, входящие и в первое, и во второе отношения (без повторов).



Добавим в нашу БД проектной организации таблицу "Архив должностей":

```
create table  archive (
      tabno    number(6) REFERENCES  emp,    -- ссылка на сотрудника
      name    varchar2(100)  not null,     -- ФИО сотрудника
      dbegin  date  not null,            -- начало работы в должности
      post      varchar(50)  not null          -- должность
);
```

Операция объединения

Объединение реализуется с помощью специального ключевого слова **UNION** (или **UNION ALL**, если не нужно удалять повторы).

Примеры:

- Список сотрудников с телефонами или адресами (если нет телефона):

```
select depno, name, PHONE
    from emp where phone is not null
UNION ALL
select depno, name, ADR
    from emp where phone is null;
```

- Список сотрудников со всеми переводами с одной должности на другую:

```
select tabno, name, edate, post
    from emp
UNION ALL
select tabno, name, dbegin, post
    from archive
   order by 1, 3;
```

Разность отношений

Разность в **Oracle** реализуется с помощью специального ключевого слова **MINUS**.

Примеры:

- Список сотрудников 5-го и 8-го отделов, которые не являются инженерами:

```
select * from emp
    where depno IN (5, 8)
MINUS
select * from emp
    where post LIKE '%инженер%'
order by depno;
```

- Список сотрудников, которые не переводились на другие должности:

```
select tabno, name
    from emp
MINUS
select tabno, name
    from archive;
```

Пересечение отношений

Пересечение в **Oracle** реализуется с помощью специального ключевого слова **INTERSECT**.

Примеры:

- Список сотрудников 5-го и 8-го отделов, которые являются инженерами:

```
select * from emp
    where depno IN (5, 8)
INTERSECT
select * from emp
    where post LIKE '%инженер%'
order by depno;
```

- Список сотрудников, которые переводились на другие должности:

```
select tabno, name
    from emp
INTERSECT
select tabno, name
    from archive;
```

Применение односхемных операций РА

Задание 1: вывести список должностей, которые занимают (или занимали) сотрудники.

```
select  post  from  emp
UNION
select  post  from  archive;
```

Задание 2: вывести список должностей, на которые переназначены другие сотрудники.

```
select  post  from  emp
INTERSECT
select  post  from  archive;
```

Задание 3: вывести список должностей, которые в настоящее время не занимает ни один сотрудник.

```
select   post  from  archive
MINUS
select  post  from  emp;
```

Разносхемные операции РА

Декартово произведение (ДП): операция над двумя произвольными (возможно, разносхемными) отношениями. Результат ДП – все комбинации строк исходных отношений. Пример:

"Студенты"

Группа	*ФИО*
СТ-4/09	Рогов В.П.
СТ-4/09	Белова О.Г.

"Предметы"

Предмет
Базы данных
Сетевые технологии

"Оценки"

Оценка
удовл.
хор.
отл.

Декартово произведение: "Оценки студентов"

Группа	*ФИО*	*Предмет*	*Оценка*
СТ-4/09	Рогов В.П.	Базы данных	удовл.
СТ-4/09	Рогов В.П.	Базы данных	хор.
СТ-4/09	Рогов В.П.	Базы данных	отл.
СТ-4/09	Рогов В.П.	Сетевые технологии	удовл.
СТ-4/09	Рогов В.П.	Сетевые технологии	хор.
СТ-4/09	Рогов В.П.	Сетевые технологии	отл.
СТ-4/09	Белова О.Г.	Базы данных	удовл.
СТ-4/09	Белова О.Г.	Базы данных	хор.
СТ-4/09	Белова О.Г.	Базы данных	отл.
СТ-4/09	Белова О.Г.	Сетевые технологии	удовл.
СТ-4/09	Белова О.Г.	Сетевые технологии	хор.
СТ-4/09	Белова О.Г.	Сетевые технологии	отл.

Разносхемные операции РА

Пример декартова произведения реальных таблиц:

```
select  *
    from  depart,  emp;
```

Если в части FROM указываются 2 и более таблицы, то СУБД по умолчанию строит их декартово произведение.

Другая разносхемная операция – соединение: селекция от декартова произведения.

Примеры.

1. Список отделов и их сотрудников:

```
select  *
    from  depart, emp
    where  emp.depno = depart.did;
```

2. Список проектов и их участников:

```
select  *
    from  project,  emp,  job
    where  emp.tabno = job.tabno
       and  job.pro = project.pro;
```

Применение операции соединения

Задание 1: вывести сотрудников с указанием ролей, которые они исполняют в проектах.

```
select  e.name,  j.rel
    from  emp  e,  job  j
    where  e.tabNo = j.tabNo;
```

Задание 2: вывести список проектов с указанием их руководителей.

```
select  p.title,  e.name
     from  emp  e,   job  j,   project  p
     where e.tabno = j.tabno
        and  j.pro = p.pro
        and  j.rel = 'руководитель';
```

Применение операции соединения

Задание 3: вывести список сотрудников с указанием количества проектов, в которых они участвуют.

```
select  name,  count(*)
    from  emp,  job
    where emp.tabno=job.tabno
    group by  emp.tabno,  emp.name;
```

Задание 4: вывести список проектов, в которых участвует более 5 сотрудников.

```
select  p.title,  count(*)
    from  job  j,  project  p
    where p.pro = j.pro
    group by  p.pro,  p.title
   having  count(*) > 5;
```

Самосоединение

В команде SELECT можно обратиться к одной и той же таблице несколько раз. При этом для каждой таблицы необходимо задать свой алиас (псевдоним), 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:

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;
```

Данные таблицы Сотрудники

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

* Данные таблицы Дети сотрудников

TabNo	**Name**	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

Результат самосоединения

TabNo	**Name**	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

Подзапросы

Подзапрос – это запрос SELECT, расположенный внутри другой команды.

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

✔**скалярные** – запросы, возвращающие единственное значение (начинаются с немодифицированного оператора сравнения);

✔**векторные** – запросы, возвращающие от 0 до нескольких элементов (начинаются с оператора *IN* или модифицированного оператора сравнения);

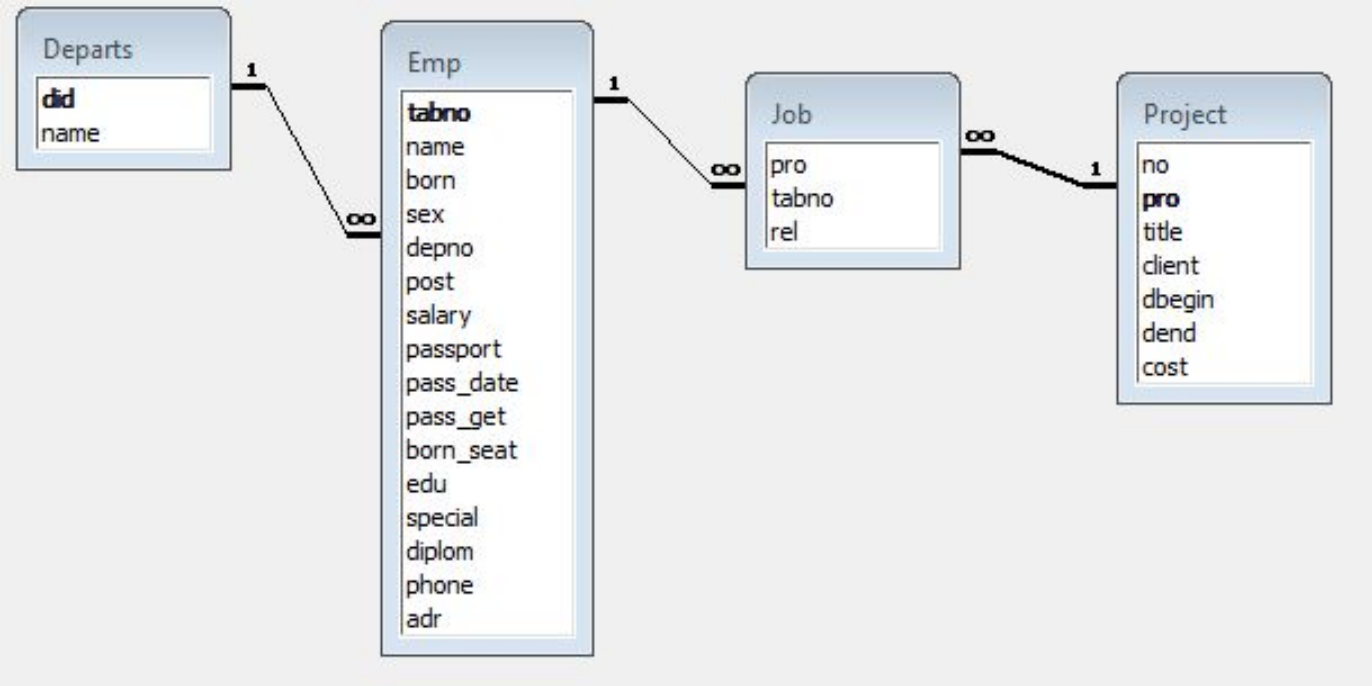
✔**табличные** – запросы, возвращающие таблицу (обычно, запросы на существование, начинаются с оператора *EXISTS*).

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	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

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 distinct 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. Вывести список сотрудников, у которых зарплата выше, чем средняя по предприятию:

```
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. Вывести список сотрудников, у которых зарплата выше, чем средняя по каждому отделу предприятия:

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

Оператор **ALL** считает условие верным, если каждое значение, выбранное подзапросом, удовлетворяет условию внешнего запроса.

Примеры использования подзапросов в части WHERE

Выдать список сотрудников, имеющих детей:

а) **с помощью операции соединения таблиц:**

```
SELECT  distinct 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** берет подзапрос, как аргумент, и оценивает его как верный, если подзапрос возвращает какие-либо записи и неверный, если тот не делает этого.

Расположение подзапросов в команде select

- Подзапрос в части **FROM.**

Например, выведем список сотрудников, у которых зарплата выше, чем средняя в отделе, в котором работает данный сотрудник, через коррелированный подзапрос:

```
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, выведем список отделов, в которых средняя зарплата ниже, чем средняя по предприятию:

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

- Подзапрос в части **SELECT.**
  Например, выведем список сотрудников с указанием количества проектов, в которых они участвуют:

```
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

Представления

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:

- ✔ Хранение сложных запросов.
- ✔ Представление данных в виде, удобном пользователю.
- ✔ Сокрытие конфиденциальной информации.
- ✔ Предоставление дифференцированного

Представления

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.

Данные таблицы Сотрудники

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

* Данные таблицы Дети сотрудников

TabNo	**Name**	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

Представления: пример

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;
```

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

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;
```

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

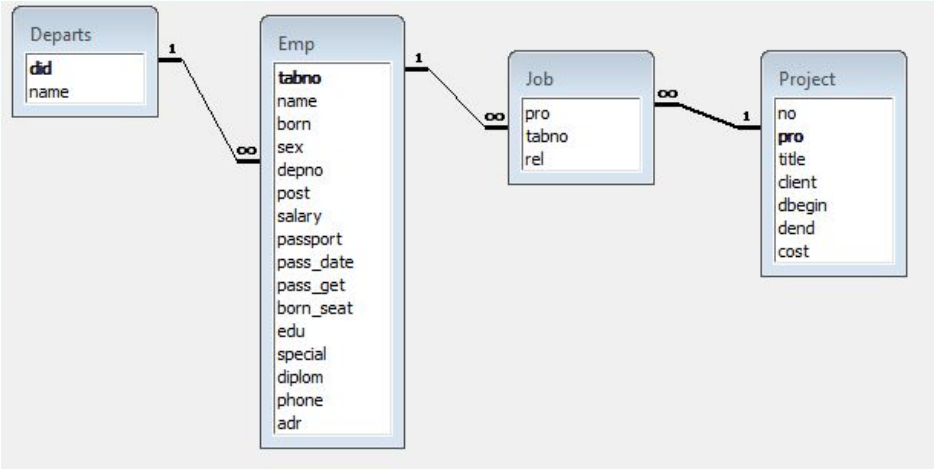
Создать представление "Сотрудники" (без данных о зарплате, для сокрытия конфиденциальной информации):

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

Представления: примеры

Создать представление "Статистика по проектам" (для хранения сложных запросов): название проекта, ФИО руководителя, количество исполнителей, количество консультантов.

```
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

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.

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

Обновляемые представления

- 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:
  - ✓ 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;

Оператор CASE

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
```

Оператор CASE

1-я форма - пример:

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
2	1	1	2011-01-12	2011-02-12	N
3	3	3	2012-05-17	2012-07-17	Y
42	1	2	2012-06-11	2012-08-11	N
57	4	5	2012-06-11	2012-08-11	N
61	1	7	2014-08-03	2014-10-03	N
62	3	5	2014-08-03	2014-10-03	Y
86	3	1	2014-08-03	2014-09-03	Y
91	4	1	2015-10-07	2015-03-07	Y
95	1	4	2015-10-07	2015-11-07	N
99	4	4	2015-10-08	2025-11-08	Y
100	1	3	2011-01-12	2011-02-12	N

Задача. Показать, сколько книг было возвращено и не возвращено в библиотеку

Оператор CASE

1-я форма - пример:

```
/* Показать, сколько книг было возвращено и не возвращено
   в библиотеку*/

SELECT (CASE sb_is_active
            WHEN  'Y' THEN 'Not returned'
            ELSE 'Returned'
        END)
        AS "status", COUNT(sb_id) AS books FROM subscriptions
            GROUP BY
                (CASE sb_is_active
                    WHEN  'Y' THEN 'Not returned'
                    ELSE 'Returned'
                END)
        ORDER BY "status" DESC;
```

Результат:

Returned	6
Not returned	5

Оператор CASE

2-я форма:

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

Оператор CASE

2-я форма пример:

```
/* Показать, сколько книг было возвращено и не возвращено
   в библиотеку*/

SELECT (CASE
            WHEN sb_is_active = 'Y' THEN 'Not returned'
            ELSE 'Returned'
        END)
        AS "status", COUNT(sb_id) AS books FROM subscriptions
            GROUP BY
                (CASE
                    WHEN sb_is_active = 'Y' THEN 'Not returned'
                    ELSE 'Returned'
                END)
        ORDER BY "status" DESC;
```

Результат:

status	books
Returned	6
Not returned	5

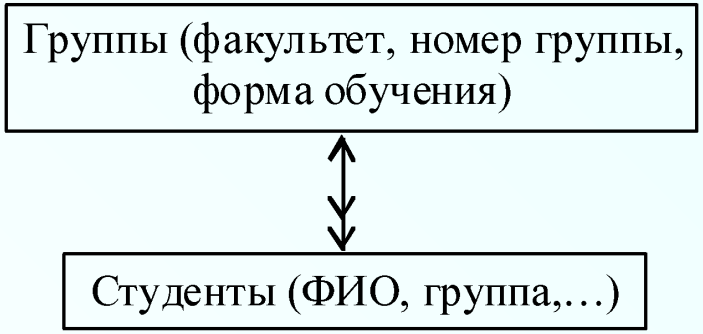
Особенности использования CASE

- ✔ 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 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

Курсоры

* Понятие курсора. Работа с курсорами

Курсор *– используемая в рамках SQL, встроенного в процедурный язык, возможность для позаписного доступа к таблицам из БД.*

Работу с курсором можно разделить на несколько четко выраженных стадий.

- ✔ Прежде чем курсор может быть использован, его следует объявить (определить). В ходе этого процесса выборка данных не производится, просто определяется оператор SELECT, который будет использован, и некоторые опции курсора.
- ✔ После объявления курсор может быть открыт для использования. В ходе этого процесса уже производится выборка данных согласно предварительно определенному оператору SELECT.
- ✔ После того как курсор заполнен данными, могут быть извлечены (выбраны) отдельные необходимые строки.
- ✔ После того как это сделано, курсор должен быть закрыт и, возможно, должны быть освобождены ресурсы, которые он занимал (в зависимости от СУБД).

После того как курсор объявлен, его можно открывать и закрывать столь часто, сколько необходимо. Если курсор открыт, операция выборки может выполняться так часто, как необходимо.

* Понятие курсора. Работа с курсорами

Курсоры создаются с помощью оператора DECLARE, синтаксис которого различен для разных СУБД.

Оператор DECLARE дает курсору имя и принимает оператор SELECT, дополненный при необходимости предложением WHERE и другими.

Чтобы показать как это работает создадим курсор, который будет делать выборку всех клиентов, не имеющих адресов электронной почты, в виде части приложения, позволяющего служащему вводить недостающие адреса.

Версия для Oracle:

```
DECLARE CURSOR CustCursor
IS
SELECT * FROM Customers
WHERE cust_email IS NULL;
```

* Понятие курсора. Работа с курсорами

Теперь, после того как курсор определен, его можно открыть.

Курсоры открываются с помощью оператора OPEN CURSOR, синтаксис которого настолько прост, что его поддерживают большинство СУБД:

OPEN CURSOR CustCursor

При обработке оператора **OPEN CURSOR** выполняется запрос, и выборка данных сохраняется для последующих просмотра и прокрутки.

Теперь доступ к данным этого курсора может быть получен с помощью оператора **FETCH.**

Оператор **FETCH** указывает строки, которые должны быть выбраны, откуда они должны быть выбраны и где их следует сохранить (имя переменной, например).

* Понятие курсора. Работа с курсорами

В первом примере используется синтаксис Oracle для выборки одной строки курсора (первой).

```
DECLARE TYPE CustCursor IS
REF CURSOR RETURN Customers%ROWTYPE;
DECLARE CustRecord Customers%ROWTYPE;
BEGIN;
OPEN CustCursor;
FETCH CustCursor INTO CustRecord;
CLOSE CustCursor;
END;
```

В данном примере оператор **FETCH** используется для выборки текущей строки (автоматически он начнет с первой строки) в переменную, объявленную с именем ***CustRecord***. С выбранными данными ничего не делается.

* Понятие курсора. Работа с курсорами

В следующем примере (в нем вновь используется синтаксис Oracle) выбранные данные подвергаются циклической обработке от первой строки до последней:

```
DECLARE TYPE CustCursor IS
REF CURSOR  RETURN Customers%ROWTYPE;
DECLARE CustRecord Customers%ROWTYPE;
BEGIN;
OPEN CustCursor;
LOOP
FETCH CustCursor INTO CustRecord;
EXIT WHEN CustCursor%NOTFOUND;
END LOOP;
CLOSE CustCursor;
END;
```

Аналогично предыдущему примеру, здесь используется оператор **FETCH** для выборки текущей строки в переменную, объявленную с именем ***CustRecord.*** Однако в отличие от предыдущего примера, здесь оператор **FETCH** находится внутри цикла **LOOP**, так что он выполняется снова и снова.

* Понятие курсора. Работа с курсорами

Код **EXIT WHEN CustCursor%NOTFOUND** указывает, что этот процесс должен быть завершен (выход из цикла), когда больше не останется строк для выборки. В этом примере для простоты также не выполняется никакой обработки, тогда как в реальный программный код следовало бы включить операторы анализа и обработки данных.

Как следует из предыдущих примеров, после использования курсоров их нужно закрывать.

Вот соответствующий синтаксис для СУБД Oracle:

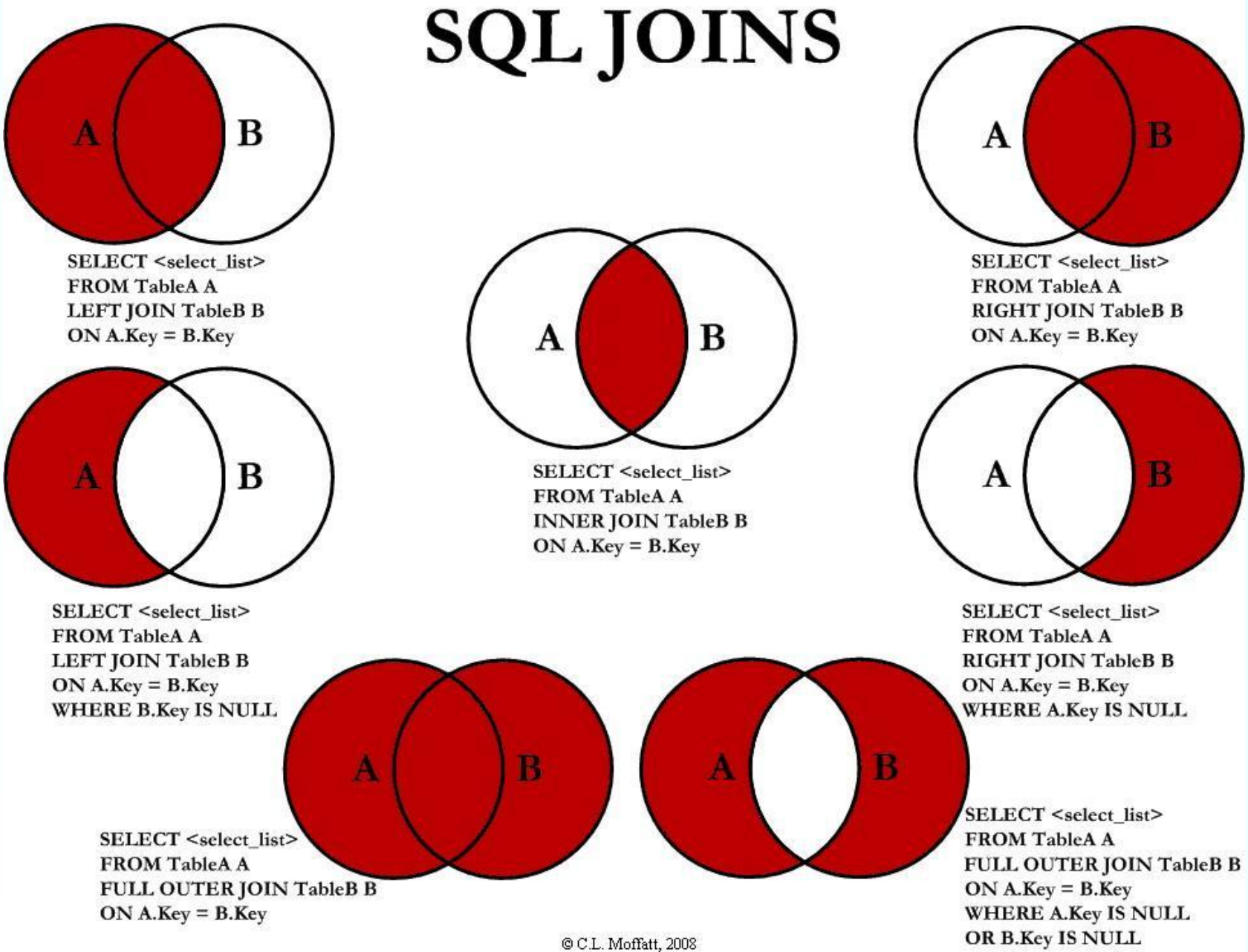
CLOSE CustCursor;

Для закрытия курсора используется оператор **CLOSE**;

После того как курсор закрыт, его нельзя использовать, не открыв перед этим вновь. Однако его не нужно объявлять заново при повторном использовании, достаточно оператора **OPEN**.

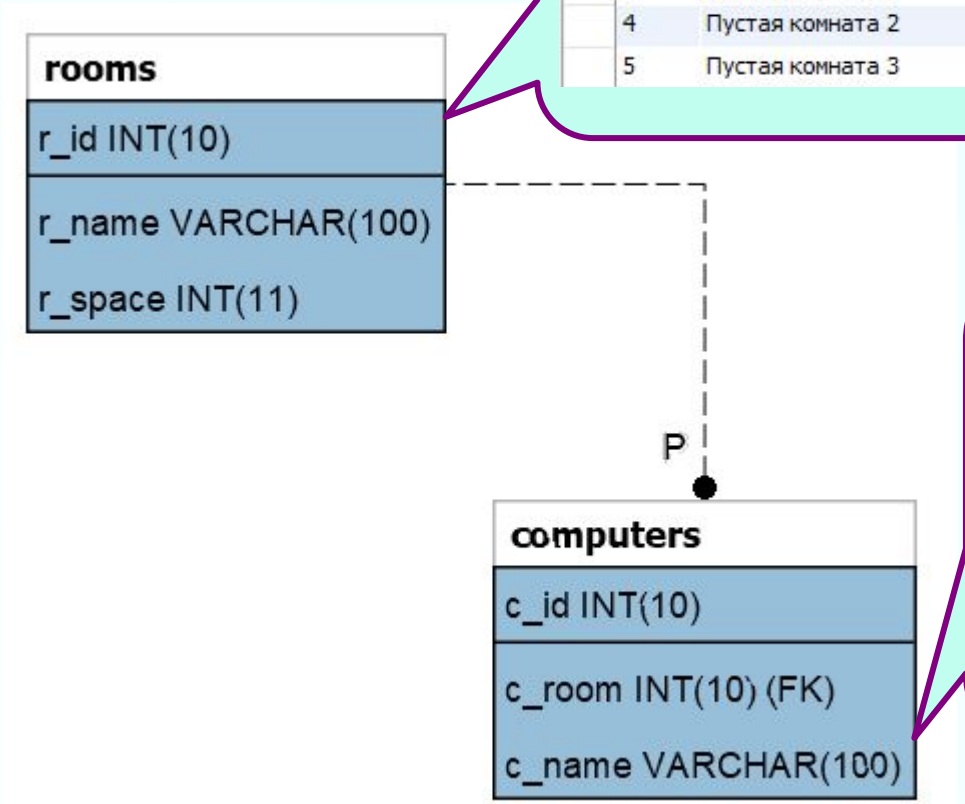
Запросы на объединение

Запросы на объединение



Пример

r_id	r_name	r_space
1	Комната с двумя компьютерами	5
2	Комната с тремя компьютерами	5
3	Пустая комната 1	2
4	Пустая комната 2	2
5	Пустая комната 3	2

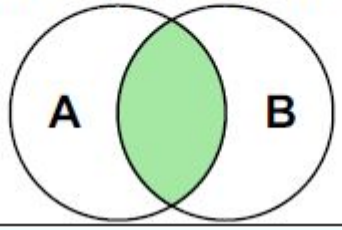


c_id	c_room	c_name
1	1	Компьютер A в комнате 1
2	1	Компьютер B в комнате 1
3	2	Компьютер A в комнате 2
4	2	Компьютер B в комнате 2
5	2	Компьютер C в комнате 2
6	NULL	Свободный компьютер A
7	NULL	Свободный компьютер B
8	NULL	Свободный компьютер C

В качестве примера рассмотрим базу, содержащую сведения о размещении компьютеров. Связь между таблицами реализуется с использованием внешнего ключа в таблице **computers** (c_room), ссылающегося на первичный ключ таблицы **rooms** (r_id).

Внутреннее объединение

Вид объединения	Графическое представление	Псевдокод запроса

Внутреннее объединение		`SELECT поля`
		`FROM A INNER JOIN B`
		`ON A.поле = B.поле`

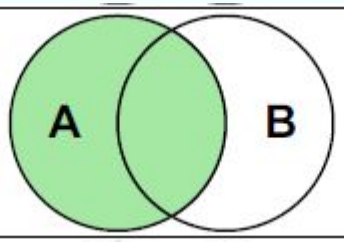
Логика внутреннего объединения состоит в том, чтобы подобрать из двух таблиц пары записей, у которых совпадает значение поля, по которому происходит объединение.

```
-- Показать информацию о том, как компьютеры распределены по комнатам

SELECT  r_name, r_id, c_room, c_id,  c_name FROM rooms
        JOIN computers ON r_id = c_room;
```

	r_name	r_id	c_room	c_id	c_name
▶	Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
	Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
	Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
	Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
	Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2

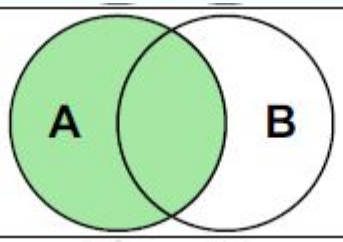
Левое внешнее объединение

Вид объединения	Графическое представление	Псевдокод запроса
Левое внешнее объединение		SELECT поля FROM A LEFT OUTER JOIN B ON A.поле = B.поле

В случае левого внешнего объединения СУБД извлекает все записи из левой таблицы и пытается найти им пару из правой таблицы. Если пары не находится, соответствующая часть записи в итоговой таблице заполняется NULL-значениями.

Левое внешнее объединение

Вид объединения	Графическое представление	Псевдокод запроса

Левое внешнее объединение		`SELECT поля`
		`FROM A LEFT OUTER JOIN B`
		`ON A.поле = B.поле`

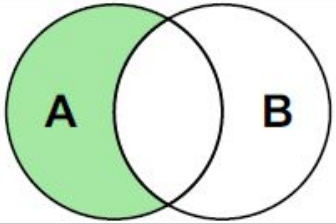

```
-- Показать все комнаты с поставленными в них компьютерами

SELECT  r_name, r_id, c_room, c_id,  c_name FROM rooms
        LEFT JOIN computers ON r_id = c_room;
```

r_name	r_id	c_room	c_id	c_name
Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2
Пустая комната 1	3	NULL	NULL	NULL
Пустая комната 2	4	NULL	NULL	NULL
Пустая комната 3	5	NULL	NULL	NULL

В этом решении нужно показать все записи из таблицы **rooms** — как те, для которых есть соответствие в таблице **computers**, так и те, для которых такого соответствия нет.

Левое внешнее объединение с исключением

Вид объединения	Графическое представление	Псевдокод запроса
Левое внешнее объединение с исключением		SELECT поля FROM A LEFT OUTER JOIN B ON A.поле = B.поле WHERE B.поле IS NULL


```
-- Показать все показать все пустые комнаты

SELECT  r_name, r_id, c_room, c_id,  c_name FROM rooms
        LEFT JOIN computers ON r_id = c_room
                WHERE c_room IS NULL;
```

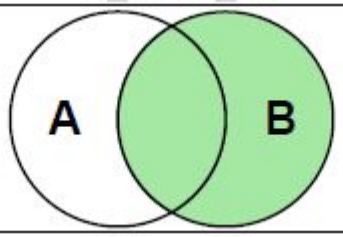
	r_name	r_id	c_room	c_id	c_name
▶	Пустая комната 1	3	NULL	NULL	NULL
	Пустая комната 2	4	NULL	NULL	NULL
	Пустая комната 3	5	NULL	NULL	NULL

Используем левое внешнее объединение с исключением, т.е. выберем только те записи из таблицы rooms, для которых нет соответствия в таблице computers.

Благодаря условию в 3-й строке запроса в конечную выборку проходят только строки со значением NULL в поле c_room.

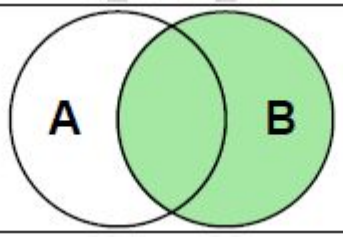
Правое внешнее объединение

Вид объединения	Графическое представление	Псевдокод запроса

Правое внешнее объединение		`SELECT поля`
		`FROM A RIGHT OUTER JOIN B`
		`ON A.поле = B.поле`

В случае правого внешнего объединения СУБД извлекает все записи из правой таблицы и пытается найти им пару из левой таблицы. Если пары не находится, соответствующая часть записи в итоговой таблице заполняется NULL-значениями.

Правое внешнее объединение

Вид объединения	Графическое представление	Псевдокод запроса
Правое внешнее объединение		SELECT поля FROM A RIGHT OUTER JOIN B ON A.поле = B.поле

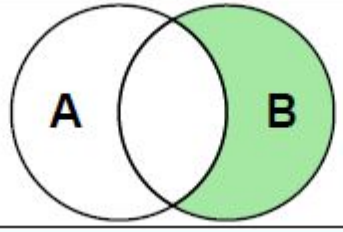
```
-- Показать все компьютеры с информацией о том, в каких они расположены комнатах

SELECT  r_name, r_id, c_room, c_id,  c_name FROM rooms
        RIGHT JOIN computers ON r_id = c_room;
```

r_name	r_id	c_room	c_id	c_name
Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2
NULL	NULL	NULL	6	Свободный компьютер A
NULL	NULL	NULL	7	Свободный компьютер B
NULL	NULL	NULL	8	Свободный компьютер C

Показаны все записи из таблицы **computers** вне зависимости от того, есть ли им соответствие из таблицы **rooms**.

Правое внешнее объединение с исключением

Вид объединения	Графическое представление	Псевдокод запроса
Правое внешнее объединение с исключением		SELECT поля FROM A RIGHT OUTER JOIN B ON A.поле = B.поле WHERE B.поле IS NULL

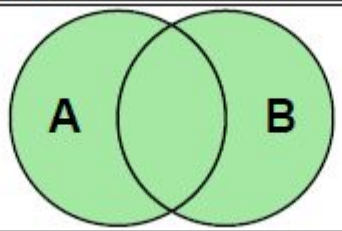

```
-- Показать все свободные компьютеры

SELECT  r_name, r_id, c_room, c_id,  c_name FROM rooms
        RIGHT JOIN computers ON r_id = c_room
                WHERE r_id IS NULL;
```

r_name	r_id	c_room	c_id	c_name
NULL	NULL	NULL	6	Свободный компьютер A
NULL	NULL	NULL	7	Свободный компьютер B
NULL	NULL	NULL	8	Свободный компьютер C

Выбираются только те записи из таблицы computers, для которых нет соответствия в таблице rooms.

Полное внешнее объединение

Вид объединения	Графическое представление	Псевдокод запроса
Полное внешнее объединение		SELECT поля FROM A FULL OUTER JOIN B ON A.поле = B.поле

При выполнении полного внешнего объединения СУБД извлекает все записи из обеих таблиц и ищет их пары. Там, где пары находятся, в итоговой выборке получается строка с данными из обеих таблиц. Там, где пары нет, недостающие данные заполняются NULL-значениями.

Полное внешнее объединение

```
/* Показать всю информацию о том, как компьютеры размещены по комнатам
(включая пустые комнаты и свободные компьютеры)*/

SELECT  r_name, r_id, c_room, c_id,  c_name FROM rooms
        FULL JOIN computers ON r_id = c_room;
```

r_name	r_id	c_room	c_id	c_name
Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2
Пустая комната 1	3	NULL	NULL	NULL
Пустая комната 2	4	NULL	NULL	NULL
Пустая комната 3	5	NULL	NULL	NULL
NULL	NULL	NULL	6	Свободный компьютер A
NULL	NULL	NULL	7	Свободный компьютер B
NULL	NULL	NULL	8	Свободный компьютер C

Показаны все записи из таблицы **rooms** вне зависимости от наличия соответствия в таблице **computers**, а также все записи из таблицы **computers** вне зависимости от наличия соответствия в таблице **rooms**.

Полное внешнее объединение

СУБД MySQL не поддерживает полное внешнее объединение, потому использование там FULL JOIN даёт неверный результат.

```
SELECT  r_name, r_id, c_room, c_id,  c_name FROM rooms
        FULL JOIN computers ON r_id = c_room;
```

	r_name	r_id	c_room	c_id	c_name
▶	Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
	Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
	Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
	Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
	Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2

Полное внешнее объединение

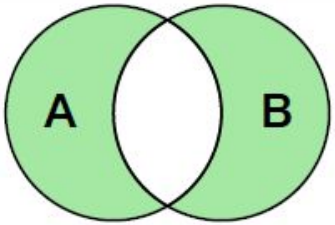
Возможным решением этой задачи для MySQL является объединение выборок с левым и правым внешними объединениями с помощью конструкции UNION.

```
/* Показать всю информацию о том, как компьютеры размещены по комнатам
(включая пустые комнаты и свободные компьютеры)*/

SELECT r_id, r_name, c_id, c_room, c_name FROM rooms
    LEFT JOIN computers ON r_id = c_room
UNION
SELECT r_id, r_name, c_id, c_room, c_name FROM rooms
    RIGHT JOIN computers ON r_id = c_room;
```

r_id	r_name	c_id	c_room	c_name
1	Комната с двумя компьютерами	1	1	Компьютер A в комнате 1
1	Комната с двумя компьютерами	2	1	Компьютер B в комнате 1
2	Комната с тремя компьютерами	3	2	Компьютер A в комнате 2
2	Комната с тремя компьютерами	4	2	Компьютер B в комнате 2
2	Комната с тремя компьютерами	5	2	Компьютер C в комнате 2
3	Пустая комната 1	NULL	NULL	NULL
4	Пустая комната 2	NULL	NULL	NULL
5	Пустая комната 3	NULL	NULL	NULL
NULL	NULL	6	NULL	Свободный компьютер A
NULL	NULL	7	NULL	Свободный компьютер B
NULL	NULL	8	NULL	Свободный компьютер C

Полное внешнее объединение с исключением

Вид объединения	Графическое представление	Псевдокод запроса
Полное внешнее объединение с исключением		SELECT поля FROM A FULL OUTER JOIN B ON A.поле = B.поле WHERE A.поле IS NULL OR B.поле IS NULL


```
/* Показать информацию по всем пустым комнатам и свободным компьютерам*/

SELECT  r_name, r_id, c_room, c_id,  c_name FROM rooms
        FULL JOIN computers ON r_id = c_room
            WHERE r_id IS NULL OR c_id IS NULL;
```

r_id	r_name	c_id	c_room	c_name
3	Пустая комната 1	NULL	NULL	NULL
4	Пустая комната 2	NULL	NULL	NULL
5	Пустая комната 3	NULL	NULL	NULL
NULL	NULL	6	NULL	Свободный компьютер A
NULL	NULL	7	NULL	Свободный компьютер B
NULL	NULL	8	NULL	Свободный компьютер C

Показаны все записи из таблицы **rooms**, для которых нет соответствия в таблице **computers**, а также все записи из таблицы **computers**, для которых нет соответствия в таблице **rooms**.

Полное внешнее объединение с исключением

СУБД MySQL не поддерживает полное внешнее объединение, потому использование там FULL JOIN даёт неверный результат.

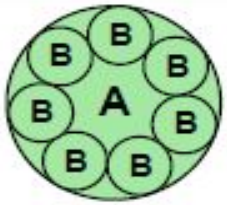
Возможным решением этой задачи для MySQL является объединение выборок с левым и правым внешними объединениями с исключениями с помощью конструкции UNION.

```
-- Показать информацию по всем пустым комнатам и свободным компьютерам

SELECT r_id, r_name, c_id, c_room, c_name FROM rooms
    LEFT JOIN computers ON r_id = c_room
        WHERE c_id IS NULL
UNION
SELECT r_id, r_name, c_id, c_room, c_name FROM rooms
    RIGHT JOIN computers ON r_id = c_room
        WHERE r_id IS NULL;
```

r_id	r_name	c_id	c_room	c_name
3	Пустая комната 1	NULL	NULL	NULL
4	Пустая комната 2	NULL	NULL	NULL
5	Пустая комната 3	NULL	NULL	NULL
NULL	NULL	6	NULL	Свободный компьютер A
NULL	NULL	7	NULL	Свободный компьютер B
NULL	NULL	8	NULL	Свободный компьютер C

Перекрёстное объединение (декартово произведение)

Вид объединения	Графическое представление	Псевдокод запроса
Перекрёстное объединение (декартово произведение): попарная комбинация всех записей		SELECT поля FROM A CROSS JOIN B или SELECT поля FROM A, B


```
-- Показать возможные варианты расстановки компьютеров по комнатам
```

Вариант 1: без ключевого слова JOIN

```
SELECT r_name, r_id, c_room, c_id, c_name
    FROM rooms, computers;
```

Вариант 2: С ключевым словом JOIN

```
SELECT r_name, r_id, c_room, c_id, c_name
    FROM rooms CROSS JOIN computers;
```

При выполнении перекрёстного объединения (декартового произведения) СУБД каждой записи из левой таблицы ставит в соответствие все записи из правой таблицы. Иными словами, СУБД находит все возможные попарные комбинации записей из обеих таблиц.

Перекрёстное объединение (декартово произведение)

r_name	r_id	c_room	c_id	c_name
Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
Комната с тремя компьютерами	2	1	1	Компьютер A в комнате 1
Пустая комната 1	3	1	1	Компьютер A в комнате 1
Пустая комната 2	4	1	1	Компьютер A в комнате 1
Пустая комната 3	5	1	1	Компьютер A в комнате 1
Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
Комната с тремя компьютерами	2	1	2	Компьютер B в комнате 1
Пустая комната 1	3	1	2	Компьютер B в комнате 1
Пустая комната 2	4	1	2	Компьютер B в комнате 1
Пустая комната 3	5	1	2	Компьютер B в комнате 1
Комната с двумя компьютерами	1	2	3	Компьютер A в комнате 2
Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
Пустая комната 1	3	2	3	Компьютер A в комнате 2
Пустая комната 2	4	2	3	Компьютер A в комнате 2
Пустая комната 3	5	2	3	Компьютер A в комнате 2
Комната с двумя компьютерами	1	2	4	Компьютер B в комнате 2
Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
Пустая комната 1	3	2	4	Компьютер B в комнате 2
Пустая комната 2	4	2	4	Компьютер B в комнате 2
Пустая комната 3	5	2	4	Компьютер B в комнате 2
Комната с двумя компьютерами	1	2	5	Компьютер C в комнате 2
Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2
Пустая комната 1	3	2	5	Компьютер C в комнате 2
Пустая комната 2	4	2	5	Компьютер C в комнате 2
Пустая комната 3	5	2	5	Компьютер C в комнате 2
Комната с двумя компьютерами	1	NULL	6	Свободный компьютер A
Комната с тремя компьютерами	2	NULL	6	Свободный компьютер A
Пустая комната 1	3	NULL	6	Свободный компьютер A
Пустая комната 2	4	NULL	6	Свободный компьютер A
Пустая комната 3	5	NULL	6	Свободный компьютер A
Комната с двумя компьютерами	1	NULL	7	Свободный компьютер B
Комната с тремя компьютерами	2	NULL	7	Свободный компьютер B
Пустая комната 1	3	NULL	7	Свободный компьютер B
Пустая комната 2	4	NULL	7	Свободный компьютер B
Пустая комната 3	5	NULL	7	Свободный компьютер B
Комната с двумя компьютерами	1	NULL	8	Свободный компьютер C
Комната с тремя компьютерами	2	NULL	8	Свободный компьютер C
Пустая комната 1	3	NULL	8	Свободный компьютер C
Пустая комната 2	4	NULL	8	Свободный компьютер C
Пустая комната 3	5	NULL	8	Свободный компьютер C

Показаны возможные варианты расстановки компьютеров по комнатам

Спасибо за внимание!