

# Введение в моделирование данных, базы данных и SQL

Лекции 1,2, 6,7

# SQL SELECT

# Select-From-Where

- Основная форма запроса имеет вид:

**SELECT**    (*ВЫБРАТЬ*) требуемые атрибуты

**FROM**      (*ИЗ*) одной или более таблиц

**WHERE**    (*ГДЕ*) условие на записи таблицы

# БД примеров

- Большинство SQL запросов будет использовать БД со следующими заголовками отношений.

**Beers**(name, manf)

**Bars**(name, addr, license)

**Drinkers**(name, addr, phone)

**Likes**(drinker, beer)

**Sells**(bar, beer, price)

**Frequents**(drinker, bar)

– Подчеркнуты атрибуты первичного ключа

- Запросы с одной таблицей

# Пример

- Используя Beers(name, manf), узнаем **какие виды пива производит «Guinness Brewing Worldwide»?**

```
SELECT name
```

```
FROM Beers
```

```
WHERE manf =
```

```
    'Guinness Brewing Worldwide' ;
```

# Результат запроса

name
'Guinness Extra Stout'
'Guinness Foreign Extra Stout'
'Harp'
'Kilkenny'

Ответ - отношение с одним атрибутом name и записи с именами пива производимого «Guinness Brewing Worldwide», такими как «Guinness».

# Соглашения (регулярные выражения)

- $[A]$  – обязательно есть  $A$ , может отсутствовать  $A$  или отсутствует

- $(\mathbf{A})$ ,  $(\mathbf{A})^+$  – непустая последовательность  $\mathbf{A}$   
 $\mathbf{A}$  или  $\mathbf{A} \mathbf{A}$   
 $\mathbf{A} \mathbf{A} \mathbf{A}$  или  $\mathbf{A} \mathbf{A} \mathbf{A} \mathbf{A} \dots$

- $\{\mathbf{A}\} \dots, (\mathbf{A})^*, [(\mathbf{A})^+ ] -$   
возможно пустая последовательность  $\mathbf{A}$ 

$\mathbf{A}$	или	<i>отсутствует</i>
$\mathbf{A}$	или	$\mathbf{A} \mathbf{A}$
$\mathbf{A} \mathbf{A} \mathbf{A}$	или	$\mathbf{A} \mathbf{A} \mathbf{A} \mathbf{A} \dots$

- **A | B | C | D** – одно из **A, B, C, D**  
**A** или **B** или  
**C** или **D**

Эти варианты заключаются в нужные скобки - {}, (), []

# Структура Select - a

```
SELECT [all | distinct]  
  { * | { [table.] c | expr [alias] }  
    { , [table.] c | expr [alias] } ... }  
FROM table [alias] { , table [alias] } ...  
[WHERE condition]  
[GROUP BY expr { , expr } ...]  
[HAVING condition]  
[ { UNION | INTERSECT | EXCEPT }  
  SELECT ... ]  
[ORDER BY { expr | position } [ASC | DESC ]  
  { , expr | position } [ASC | DESC ] } ... ]
```



# SELECT

**SELECT** [ **ALL** | **DISTINCT** ]

( \* | элемент\_SELECT {, элемент\_SELECT } ... )

**FROM** *таблица* [ псевдоним ]  
{, *таблица* [ псевдоним ] } ...

**WHERE**

[ **NOT** ] WHERE\_условие  
{ [ **AND**|**OR** ] [ **NOT** ] WHERE\_условие } ...

[ **GROUP BY**  
фраза [ **HAVING** фраза ] ]

*таблица* – это  
базовая\_таблица | представление | (подзапрос)

# Элемент\_SELECT

[*таблица.*]\* | значение | SQL\_функция

## Значение – это

[*таблица.*]столбец | (выражение) |  
константа | переменная | SQL\_функция

## Выражение– это

выражение с и

унарными операциями + , - и

бинарными операциями + , - , \* , \*\*

# Фраза WHERE

## WHERE\_условие:

- значение { = | <> | < | <= | > | >= }  
          { значение | ( подзапрос ) }
- значение [ **NOT** ] **BETWEEN**  
          значение\_1 **AND** значение\_2
- значение [ **NOT** ] **IN**  
          { ( константа [ , константа ] ... ) |  
          ( **подзапрос** ) }
- [ *таблица.* ] столбец [ **NOT** ] **LIKE**  
          '*строка\_символов*' [ **ESCAPE** '*символ*' ]
- значение **IS** [ **NOT** ] **NULL**
- **EXISTS** ( **подзапрос** )

# Смысл запроса с одной таблицей

- Используя таблицу, указанную в **FROM** части
- Осуществить **отбор** записей, следуя указаниям **WHERE** части
- Применить операцию **проекции** в соответствии с требованиями **SELECT** части
  - выбрать данные указанных столбцов,
  - выполнить необходимое их преобразование

# Операционная семантика

- Последовательно перебираем записи таблицы, указанной в **FROM** части
- Проверяем, удовлетворяет ли текущая запись условиям **WHERE** части
- Если да, в соответствии с требованиями **SELECT** части **выбираем значения атрибутов, вычисляем выражения, используя элементы текущей записи**

# ЦИКЛ

**SELECT**  $a_1, a_2, \dots, a_k$   
**FROM**  $R_1$  as  $x_1$   
**WHERE** Conditions

1. ЦИКЛ:

```
Answer = {}  
for  $x_1$  in  $R_1$  do  
    if Conditions  
        then Answer = Answer  $\cup \{(a_1, \dots, a_k)\}$   
return Answer
```

## \* в SELECT части

- Если в **FROM** части указано одно отношение, то символ \* в **SELECT** части означает - “все атрибуты этого отношения”
- Пример для отношения Beers(name, manf):

**SELECT** \*

**FROM** Beers

**WHERE** manf =

'Guinness Brewing Worldwide' ;

# Результат запроса

name	manf
'Guinness Extra Stout'	'Guinness Brewing Worldwide'
'Guinness Foreign Extra Stout'	'Guinness Brewing Worldwide'
'Harp'	'Guinness Brewing Worldwide'
'Kilkenny'	'Guinness Brewing Worldwide'

Результат содержит все атрибуты Beers записей пива производимого «Guinness Brewing Worldwide».



# Еще пример

Company(sticker, name, country, stockPrice)

Все компании UK, имеющие акции дороже > 50:

```
SELECT *  
FROM Company  
WHERE country="UK" AND stockPrice > 50
```

Заголовок результирующего отношения:

R(sticker, name, country, stockPrice)

# Переименование атрибутов

- Если необходимо, чтобы столбец результата имел другое имя, то используя

**AS** <новое имя> ,

можно переименовать столбец.

- Пример для Beers(name, manf):

```
SELECT name AS beer, manf
```

```
FROM Beers
```

```
WHERE manf =
```

```
'Guinness Brewing Worldwide'
```

# Результат запроса

beer	manf
'Guinness Extra Stout'	'Guinness Brewing Worldwide'
'Guinness Foreign Extra Stout'	'Guinness Brewing Worldwide'
'Harp'	'Guinness Brewing Worldwide'
'Kilkenny'	'Guinness Brewing Worldwide'

# Выражения в SELECT части

- Любое имеющее смысл выражение можно указать как элемент **SELECT** части (атрибут результирующего отношения, столбец таблицы)

- Пример для Sells(bar, beer, price):

```
SELECT bar, beer,
```

```
    price * 28.5 AS priceInRub
```

```
FROM Sells;
```

# Результат запроса

bar	beer	priceInRub
Удача	Kilkenny	180
Встреча	Miller	70
...	...	...

# Константные выражения

- Используя Likes(drinker, beer):

```
SELECT drinker,  
       'любит Харп' AS whoLikesHarp  
FROM Likes  
WHERE beer = 'Harp';
```

# Результат запроса

drinker	<b>whoLikesHarp</b>
‘Алекс’	‘любит Харп’
‘Тим’	‘любит Харп’
...	...

# Сложные выражения в WHERE части

- В Sells(bar, beer, price) найти **стоимость** пива «Харп» в баре «Встреча» :

```
SELECT price
FROM Sells
WHERE bar = 'Встреча' AND
      beer = 'Harp' ;
```



# Элементы условий

Что можно использовать в WHERE части:

имена атрибутов отношений

операторы сравнения: =, <>, <, >, <=, >=

арифметические операторы: stockprice\*2

строковые операторы (“||” конкатенация).

лексикографический порядок при сравнении строк

дополнение строк пробелами до равной длины

сопоставление с шаблоном: s LIKE p

операторы/функции над датами, временными значениями

# Важные моменты

- Две одиночных кавычки в строке представляют саму кавычку (апостроф - ').
- Условия в WHERE части могут использовать операции AND, OR, NOT, и скобки в соответствии с обычным способом построения логических выражений.
- SQL не чувствителен к регистру букв. Прописные и строчные буквы суть одно и тоже, пока не заключены в двойные кавычки (").

# Шаблоны

- WHERE часть может включать условия, в которых строки сопоставляются с шаблонами, чтобы обнаружить соответствия.
  - `<Attribute> LIKE <pattern>` или  
`<Attribute> NOT LIKE <pattern>`
- Шаблон это строка в кавычках с метасимволами
  - `%` = “любая цепочка символов”
  - `_` = “любой символ”

# Пример

- В Drinkers(name, addr, phone) найти с любителей пива с тф. номером от «МТС» или «Мегафон» - 916 или 926 :

```
SELECT name  
FROM Drinkers  
WHERE phone LIKE ' 89_6%' ;
```

**P** -> 89**D**6**S**

**D** -> *цифра* /

**S** -> *цифра* **S** /

# Пример

Company(sticker, name, address, country, stockPrice)

Найти все компании из UK, чей адрес содержит  
'London':

```
SELECT *  
FROM Company  
WHERE country='UK' AND  
address LIKE '% London %'
```

**NULL значения**

# NULL значения

- Кортежи SQL отношений могут содержать **NULL** в качестве значения.
- Смысл зависит от контекста. Два общих случая:
  - *Пропущено/Неизвестно*: знаем бар «Встреча» имеет адрес, но не знаем какой.
  - *Неприменимо* : значение атрибута *spouse* (супруга) для холостого мужчины.
- Если  $x = \text{NULL}$ , то  $4 * (3 - x) / 7$  - **NULL**

# Сравнение NULL со значениями

- Логические условия в SQL относятся к трехзначной логике :
  - **TRUE, FALSE, UNKNOWN** (*неопределенно*).
- Когда некоторое значение сравнивается с **NULL**, значение результата - **UNKNOWN**.
- Если  $x = \text{NULL}$ , то  $(x = \text{'Встреча'})$  – **UNKNOWN**
- Результат запроса включает кортеж, если значение выражения **WHERE** части
  - **TRUE** (ни **FALSE**, ни **UNKNOWN**).



# Пример

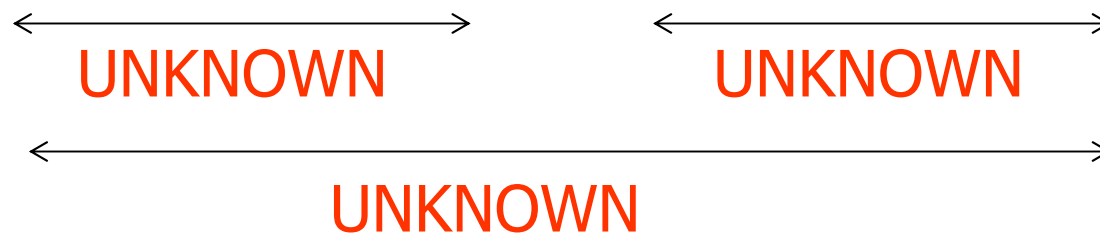
- Из отношения Sells:

bar	beer	price
Встреча	Харп	NULL

**SELECT** bar

**FROM** Sells

**WHERE** price < 20.0 **OR** price >= 20.0;



Бар «Встреча» не попадает в результат.

Что делать?

# Проверка на Null

Можно явно проверить на наличие NULL значения:

- $x$  IS NULL
- $x$  IS NOT NULL

```
SELECT bar
FROM    Sells
WHERE   price < 20.0 OR price >= 20.0
                OR price IS NULL
```

Включены все бары.

# Трехзначная логика

- Чтобы понять, что представляют AND, OR, и NOT в трехзначной логике можно считать, что
- **TRUE = 1, FALSE = 0, UNKNOWN =  $\frac{1}{2}$ .**
- **$x \text{ AND } y = \min(x, y)$**   
 **$x \text{ OR } y = \max(x, y)$**   
 **$\text{NOT}(x) = 1-x$**
- Пример:

$$\begin{aligned} &\text{TRUE AND (FALSE OR NOT(UNKNOWN))} \\ &= \\ &\text{MIN(1, MAX(0, (1 - } \frac{1}{2} \text{ )))} = \\ &\text{MIN(1, MAX(0, } \frac{1}{2} \text{ ))} = \text{MIN(1, } \frac{1}{2} \text{ )} = \frac{1}{2}. \end{aligned}$$

## двухзначная логика $\neq$ трехзначной логике

- Некоторые общие законы, например, коммутативность **AND**, сохраняется в трехзначной логике.
- Но некоторые нет, например, :  
“закон отсутствия середины,”
  - $p \text{ OR NOT } p = \text{TRUE}$ .
  - Если  $p = \text{UNKNOWN}$ , то получаем
  - $\text{MAX}(1/2, (1 - 1/2)) = 1/2 \quad (\neq 1)$

# Запросы с несколькими отношениями

# Запросы с несколькими отношениями

- Часто требуемые запросы должны комбинировать данные из более одного отношения
- Можно обратиться к нескольким отношениям в одном запросе указав их всех в FROM части
- Чтобы различить одноименные атрибуты разных отношений, следует указывать отношение
  - **<отношение>.<атрибут>**

# Пример

- Используя отношения Likes(drinker, beer) и Frequents(drinker, bar), найти сорта пива, предпочитаемые посетителями бара «Встреча»

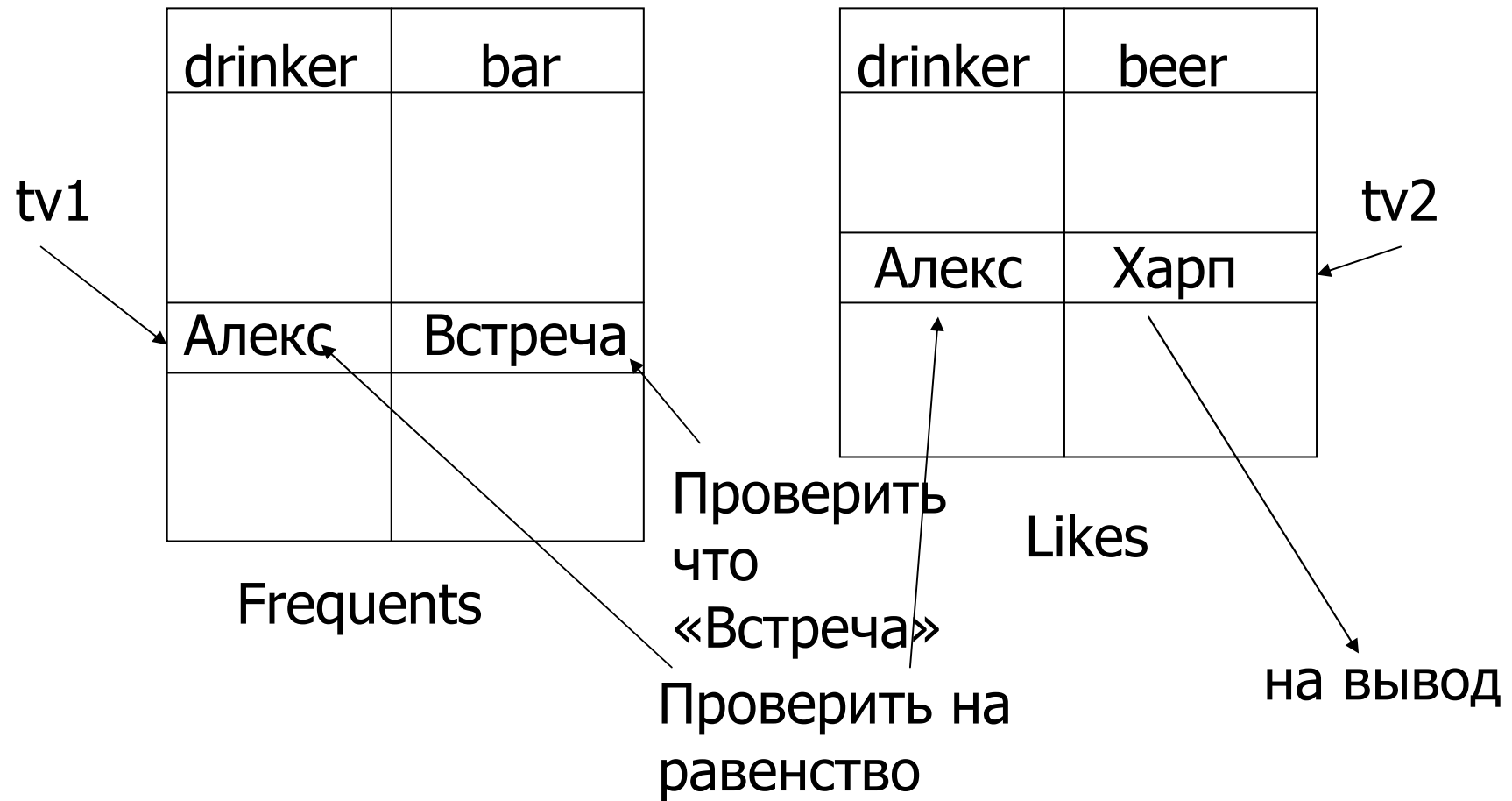
```
SELECT beer
```

```
FROM Likes, Frequents
```

```
WHERE bar = 'Встреча' AND
```

```
Frequents.drinker = Likes.drinker;
```

# Пример





# Еще пример

Product (pname, price, category, maker)

Purchase (buyer, seller, store, product)

Company (cname, stockPrice, country)

Person(pname, phoneNumber, city)

Найти персон, живущих в Долгопрудном, покупающих компьютеры, указать наименования магазинов, в которых делаются эти покупки

```
SELECT  pname, store
FROM    Person, Purchase
WHERE   pname=buyer AND city='Долгопрудный'
          AND product='компьютер'
```

# Одноименные атрибуты

Найти имена персон покупающих телефоны:

Product (name, price, category, maker)

Purchase (buyer, seller, store, product)

Person(name, phoneNumber, city)

```
SELECT Person.name
FROM    Person, Purchase, Product
WHERE   Person.name=Purchase.buyer
          AND product=Product.name
          AND Product.category='телефон'
```

# Псевдонимы

- Иногда в запросах нужно использовать две копии одного и того же отношения
- Чтобы различать копии, в FROM части после имени копии указывают «псевдоним» (алиас, alias).
- Можно переименовывать отношения для лучшей «читаемости» запросов

# Пример

- В Beers(name, manf) найти все пары наименований пива одного и того же производителя.
  - не выводить пары вида (Харп, Харп).
  - выводить пары в алфавитном порядке, например, (Миллер, Харп), не (Харп, Миллер).

```
SELECT b1.name, b2.name  
FROM Beers b1, Beers b2  
WHERE b1.manf = b2.manf AND  
        b1.name < b2.name;
```

# Еще пример

Найти пары компаний, производящих товары одной и той же категории

```
SELECT product1 maker, product2 maker  
FROM Product as product1, Product as product2  
WHERE product1.category=product2.category  
AND product1.maker <> product2.maker
```

Product ( name, price, category, maker)

# Псевдонимы

Псевдонимы вводятся компилятором языка SQL автоматически

`Product ( name, price, category, maker)`

```
SELECT name  
FROM Product  
WHERE price > 100
```

Становится:

```
SELECT Product.name  
FROM Product as Product  
WHERE Product.price > 100
```

Не срабатывает, когда Product встречается более одного раза:

В этом случае пользователь должен явно определить псевдоним.

# Формальная семантика запросов с несколькими отношениями

- Почти тоже самое, что для запросов с одним отношением:
  1. Начиная с произведения всех отношений, указанных в **FROM** части.
  2. применить условия отбора из **WHERE** части
  3. выполнить проекцию на список атрибутов и выражений в **SELECT** части

## Операционная семантика запросов с несколькими отношениями

- Последовательно перебираем записи всех таблиц, указанных в **FROM** части
  - Кортеж-переменная для каждого отношения
  - Кортеж-переменные перебирают все комбинации кортежей (вложенные циклы)
- Проверяем, удовлетворяют ли набор текущих записей условиям **WHERE** части
- Если да, в соответствии с требованиями **SELECT** части выбираем значения атрибутов, вычисляем выражения, используя элементы текущих записей



# Вложенные циклы

**SELECT**  $a_1, a_2, \dots, a_k$   
**FROM**  $R_1$  as  $x_1, R_2$  as  $x_2, \dots, R_n$  as  $x_n$   
**WHERE** Conditions

## 1. Вложенные циклы:

```
Answer = {}  
for  $x_1$  in  $R_1$  do  
    for  $x_2$  in  $R_2$  do  
        .....  
        for  $x_n$  in  $R_n$  do  
            if Conditions  
                then Answer = Answer  $\cup \{(a_1, \dots, a_k)\}$   
return Answer
```

# Параллельное вычисление

**SELECT**  $a_1, a_2, \dots, a_k$   
**FROM**  $R_1$  as  $x_1, R_2$  as  $x_2, \dots, R_n$  as  $x_n$   
**WHERE** Conditions

## 2. Параллельное вычисление

```
Answer = {}  
for all assignments  $x_1$  in  $R_1, \dots, x_n$  in  $R_n$  do  
    if Conditions then Answer = Answer  $\cup \{(a_1, \dots, a_k)\}$   
return Answer
```

Не предполагает какого-либо порядка

# В реляционной алгебре

**SELECT**  $a_1, a_2, \dots, a_k$   
**FROM**  $R_1 \text{ as } x_1, R_2 \text{ as } x_2, \dots, R_n \text{ as } x_n$   
**WHERE** Conditions

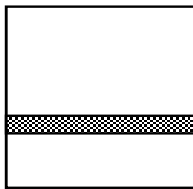
3. Трансляция в реляционную алгебру:

$\Pi_{a_1, \dots, a_k} ( \sigma_{\text{Conditions}} (R_1 \times R_2 \times \dots \times R_n) )$

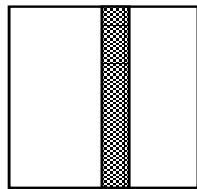
<b>Select-From-Where</b>	<b>=</b>	<b>Select-Project-Join</b>
запросы		выражения

# Реляционные операции

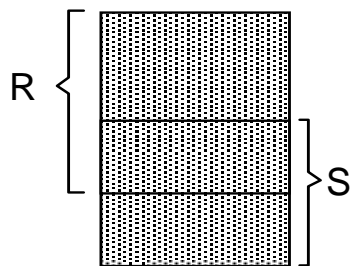
Select  $\sigma$



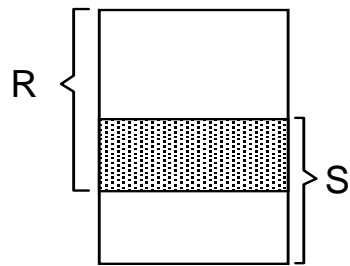
Project  $\pi$



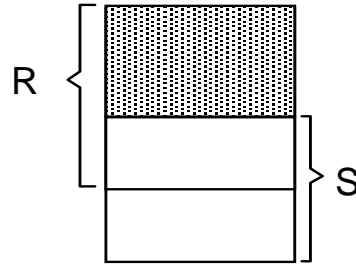
Union



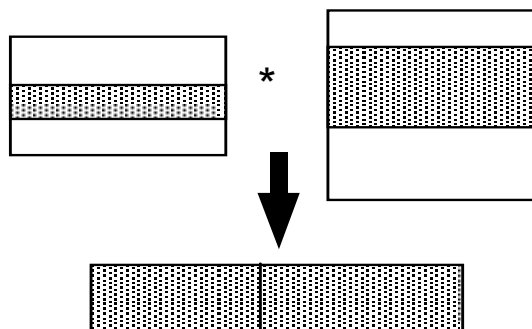
Intersection



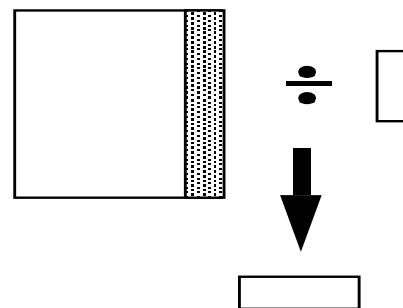
Difference



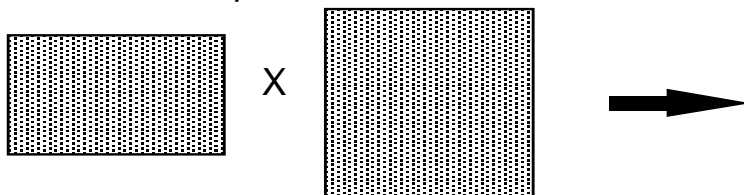
Join  $\bowtie$



Division



Cartesian product



# Подзапросы

# Подзапросы

- **SELECT-FROM-WHERE** оператор в круглых скобках (*подзапрос*) может использоваться в качестве операнда в ряде мест, включая **FROM** и **WHERE** части.
- Пример: на место отношения в **FROM** части, можно поместить другой запрос и выполнить запрос к его результату.
  - Лучше использовать псевдонимы для именования кортежей результата

# Подзапросы, возвращающие одну запись

- Если гарантируется, что запрос возвращает одну запись, то подзапрос может использоваться в качестве значения
  - обычно такие записи имеют одно поле
  - обычно единственность записи гарантируется первичным ключом.
  - если запрос не возвращает записей или возвращает больше одной, то возникает **ошибка** времени исполнения.

# Пример

- В Sells(bar, beer, price) найти бары предлагающие «Миллер» по той же цене, что «Встреча» продает «Харп».
- Два запроса обеспечивают ответ:
  1. найти стоимость «Харп» в баре «Встреча»
  2. найти бары продающие «Миллер», по этой цене



# Запрос + Подзапрос

**SELECT** bar

**FROM** Sells

**WHERE** beer = 'Миллер' **AND**


price = (SELECT price

**FROM** Sells

**WHERE** bar = 'Встреча'

**AND** beer = 'Харп');

Цена продажи  
«Харп» в баре  
«Встреча»



Логические операторы  
**IN, EXISTS,**  
**ANY, ALL**

# Оператор IN

- **<кортеж> IN <отношение>**  
истинно, если кортеж содержится в отношении.
- **<кортеж> NOT IN <отношение>**  
если кортеж не содержится в отношении
- очень часто <отношение> – это подзапрос
- IN-выражения могут использоваться в WHERE части

# Пример

- В Beers(name, manf) и Likes(drinker, beer) найти наименование и производителя сортов пива, предпочитаемых Алексом.

**SELECT \***

**FROM Beers**

**WHERE name IN**

**(SELECT beer FROM Likes  
WHERE drinker = 'Алекс');**

Набор  
сортов  
любимых  
Алексом



# Оператор Exists

- **EXISTS**( <отношение> ) истинен, тогда и только тогда, когда <отношение> не пусто
- **EXISTS** логический оператор, может входить в **WHERE** часть
- Пример: В Beers(name, manf) найти сорта пива, уникальные для их производителя (им производится только этот сорт пива)

# Пример запроса с EXISTS

**SELECT** name

**FROM** Beers **b1**

**WHERE NOT EXISTS**

**SELECT \***

**FROM** Beers

**WHERE** manf = **b1**.manf **AND**  
name <> **b1**.name);

Области видимости: manf относится  
к непосредственно охватывающему  
FROM с отношением, имеющем этот  
атрибут

Выбирает сорта  
пива с тем же  
производителем,  
что и b1, но с  
другим сортом

*Ссылка на b1 –  
запросы как  
вложенные циклы.*

# Оператор ANY

- $x = \text{ANY}(\langle \text{отношение} \rangle)$  логическое условие, означающее, что  $x$  равен хотя бы одному кортежу отношения
- вместо  $=$  может стоять любой оператор сравнения.
  - Пример :  $x \geq \text{ANY}(\langle \text{отношение} \rangle)$  означает, что  $x$  не меньше, чем все кортежи отношения
- В кортежах отношения **ДОЛЖЕН** быть только один атрибут

# Оператор ALL

- $x \neq \mathbf{ALL}(\langle \text{отношение} \rangle)$  истинен, тогда и только тогда, когда для каждого кортежа  $t$  отношения,  $x$  не равен  $t$ .
  - то есть  $x$  не входит в отношение
- вместо  $\neq$  может стоять любой оператор сравнения
  - Пример:  $x \geq \mathbf{ALL}(\langle \text{отношение} \rangle)$  означает, что в отношении нет кортежей больших, чем  $x$
- В кортежах отношения **ДОЛЖЕН** быть только один атрибут



# Пример

- В Sells(bar, beer, price) найти самые дорогие сорт(а) пива

**SELECT** beer

**FROM** Sells

**WHERE** price  $\geq$  **ALL**(  
SELECT price  
FROM Sells);

Цена пива для охватывающего Sells не должна быть меньше, чем какая-либо цена пива

# Отношения как мультимножества группировка и агрегирование

# Реляционная алгебра и операции над мультимножествами

- Объединение:  $\{a,b,b,c\} \cup \{a,b,b,b,e,f,f\} = \{a,a,b,b,b,b,c,e,f,f\}$ 
  - РА: все кортежи, входящие хотя бы в одно из отношений-операндов
- Разность:  $\{a,b,b,b,c,c\} - \{b,c,c,c,d\} = \{a,b,b\}$ 
  - РА: все кортежи, входящие в первое отношение, такие, что ни один из них не входит во второе.
- Пересечение:  $\{a,b,b,b,c,c\} \cap \{b,b,c,c,c,c,d\} = \{b,b,c,c\}$ 
  - РА: все кортежи, входящие в оба отношения-операнда

# Семантика мультимножеств в запросах

- SELECT-FROM-WHERE оператор следует семантике мультимножеств:
  - Выборка (ограничение): не удаляет дубликаты
    - РА: все кортежи отношения-операнда, удовлетворяющее условию
  - Проекция: не удаляет дубликаты
    - РА: взятие соответствующих полей из кортежей отношения
  - Декартово произведение, соединение: не удаляет дубликаты
    - РА: конкатенация кортежей отношений, удовлетворяющих условию

# Union, Intersect и Except

- Объединение, пересечение и разность отношений выражается следующим образом, выполняется над подзапросами:
  - ( подзапрос ) UNION (подзапрос )
  - (подзапрос ) INTERSECT (подзапрос )
  - (подзапрос ) EXCEPT (подзапрос )
    - MINUS

# Пример

- Из отношений Likes(drinker, beer), Sells(bar, beer, price) и Frequent(drinker, bar) найти персон и сорта пива такие что:
  1. Персона любит пиво и
  2. Персона посещает хотя бы один бар, продающий пиво

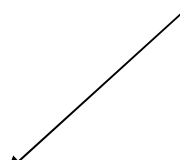
# Решение

(SELECT \* FROM Likes)

INTERSECT

```
(SELECT drinker, beer
FROM Sells, Frequents
WHERE Frequents.bar = Sells.bar
)
```

Персона посещает  
хотя бы один бар,  
продающий пиво



# Семантика **множеств** для операций union, intersect и except

- Хотя SELECT-FROM-WHERE оператор следует семантике мультимножеств, операции union, intersect и except исходит из семантики множеств.
  - То есть дубликаты удаляются в результата операции



# Основание: эффективность

- При реализации операции проекции проще избежать удаление дубликатов
  - по кортежная обработка
- При реализации операции intersect и exsert требуется в начале отсортировать отношения
  - как результат в ходе этого можно устранить дубликаты

# Управление удалением дубликатов

- Чтобы результат был множеством, в подзапросах используем **SELECT DISTINCT . . .**
- Чтобы результат был мультимножеством (содержал дубликаты), используем квалификатор **ALL**:  
**. . . UNION ALL . . .**

# Пример: ALL

- Используя отношения `Frequents(drinker, bar)` и `Likes(drinker, beer)`:

```
(SELECT drinker FROM Frequents)
```

```
EXCEPT ALL
```

```
(SELECT drinker FROM Likes)
```

- Любители пива, посещающие больше число баров, чем число любимых сортов пива, где количество дубликатов равно разнице этих чисел

# Агрегатные функции

- Можно применить операции SUM, AVG, COUNT, MIN и MAX к столбцам в SELECT части, чтобы получить соответствующее агрегатное значение столбца.
- COUNT(\*) подсчитывает число записей.

# Пример

- На основе Sells(bar, beer, price) получить среднюю стоимость пива Bud:

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Bud'
```

## Удаление дубликатов при «агрегации»

- DISTINCT внутри агрегатной функции вызывает удаление дубликатов до выполнения агрегатной операции
- Пример: подсчитать количество различных цен на пиво Bud:

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Bud'
```


## NULL игнорируется при «агрегации»

- NULL не оказывает влияния на вычислений функции SUM, AVG, COUNT, MIN и MAX.
- если в столбце нет не-NULL значений, то результат агрегации NULL.

# Пример


```
SELECT count(*)  
FROM Sells  
WHERE beer = 'Bud'
```

Число баров,  
продающих Bud.



```
SELECT count(price)  
FROM Sells  
WHERE beer = 'Bud'
```

Число баров,  
продающих Bud, но  
по известной цене





# Select-From-Where

- Основная форма запроса имеет вид:

**SELECT**    (*ВЫБРАТЬ*) требуемые атрибуты

**FROM**      (*ИЗ*) одной или более таблиц

**WHERE**    (*ГДЕ*) условие на записи таблицы

**GROUP BY** (*ГРУППИРУЯ*) атрибуты

**HAVING** (*ТАКИЕ ЧТО*)

условие на группы записей

# SELECT

**SELECT** **[ALL | DISTINCT]**

{ \* | элемент\_SELECT [,элемент\_SELECT] ... }

**FROM** {базовая\_таблица | представление}...

**WHERE** **[NOT]** условие **[AND|OR][NOT]** условие...

**GROUP BY** [таблица.]столбец[, [таблица.]столбец]

...

**[HAVING [NOT] условие [AND|OR][NOT] условие]...**

**]**

# Группировка

- После SELECT-FROM-WHERE выражения можно указать конструкцию GROUP BY и список столбцов
- Результирующее отношение SELECT-FROM-WHERE разбивается на группы кортежей, сгруппированных в соответствии со значениями этих столбцов, любая агрегатная функция вычисляется только на множестве значений группы.

# Пример

- По Sells(bar, beer, price) найти среднюю цену каждого вида пива:

```
SELECT beer, AVG(price)  
FROM Sells  
GROUP BY beer
```

# Пример

- По Sells(bar, beer, price) и Frequents(drinker, bar) для каждого любителя пива найти среднюю (по посещаемым барам) цену пива Bud:

```
SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = 'Bud' AND
      Frequents.bar = Sells.bar
GROUP BY drinker
```

вычисляем  
цену Bud  
по барам,  
затем группируем  
по любителям.

# Ограничение элементы SELECT части при группировке

- При группировке все элементы SELECT части должны быть:
  1. агрегатными значениями или
  2. группирующими столбцами из списка GROUP BY

# Конструкция HAVING

- После конструкции GROUP BY можно указывать конструкцию HAVING <условие>
- В этом случае условие применяется к каждой группе, группы не удовлетворяющие условию удаляются из результата

# Требования к условиям в HAVING

- Условия могут ссылаться на любой элемент FROM части
- В условии можно указывать те столбцы таблиц, имеющие смысл в группе:
  1. агрегатные значения или
  2. группирующие столбцы из GROUP BY



# Пример

**Purchase(product, date, price, quantity)**

**Продукты с числом продаж не менее 30**

```
SELECT      product, Sum(price * quantity)
FROM        Purchase
WHERE       date > “9/1/2004”
GROUP BY   product
HAVING     Sum(quantity) > 30
```

# Пример

- Используя Sells(bar, beer, price) и Beers(name, manf) найти среднюю цену пива, которое либо продается хотя бы тремя барами либо производится 'Guinness Brewing Worldwide'.

# Решение

Группы сортов пиво, которое  
продается хотя бы тремя  
барами либо производится  
'Guinness Brewing Worldwide'

пиво, которое  
производится  
'Guinness BW'

```
SELECT beer, AVG(price)
```

```
FROM Sells
```

```
GROUP BY beer
```

```
HAVING COUNT(bar) >= 3 OR
```

```
beer IN (SELECT name
```

```
FROM Beers
```

```
WHERE manf = 'Guinness BW')
```

# Общая форма агрегатных и группирующих конструкций

```
SELECT  S  
FROM    R1,...,Rn  
WHERE   C1  
GROUP BY a1,...,ak  
HAVING  C2
```

S = может содержать атрибуты  $a_1, \dots, a_k$  и/или агрегатные функции, но никакие другие атрибуты

C1 = любое условие на атрибуты in  $R_1, \dots, R_n$

C2 = любое условие с агрегатными выражениями

# Общий алгоритм реализации агрегатных и группирующих конструкций

```
SELECT  S  
FROM    R1,...,Rn  
WHERE   C1  
GROUP BY a1,...,ak  
HAVING  C2
```

1. Вычислить FROM-WHERE, получить таблицу со всеми атрибутами  $R_1, \dots, R_n$
2. Сгруппировать по атрибутам  $a_1, \dots, a_k$
3. Вычислить агрегатное выражение C2 и оставить только группы удовлетворяющие C2
4. Вычислить агрегатные выражения в S и вернуть результат

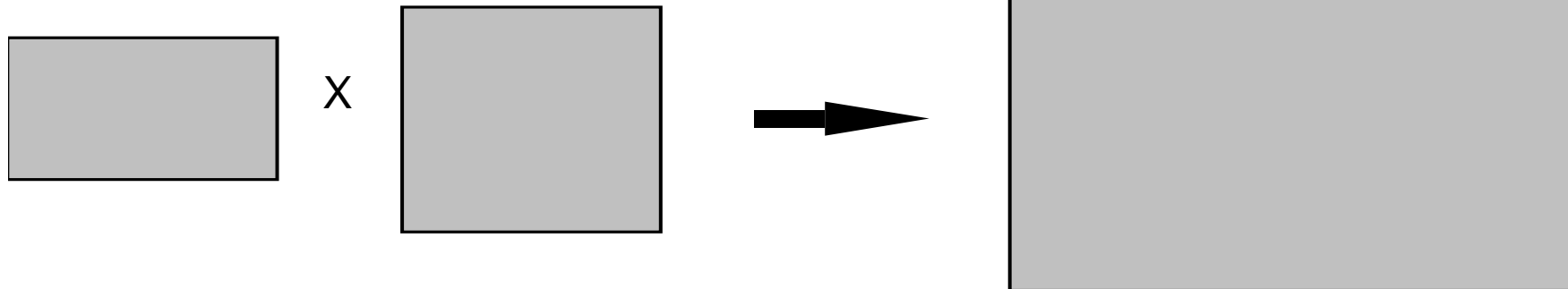
**Соединения**

# Соединения (Join)

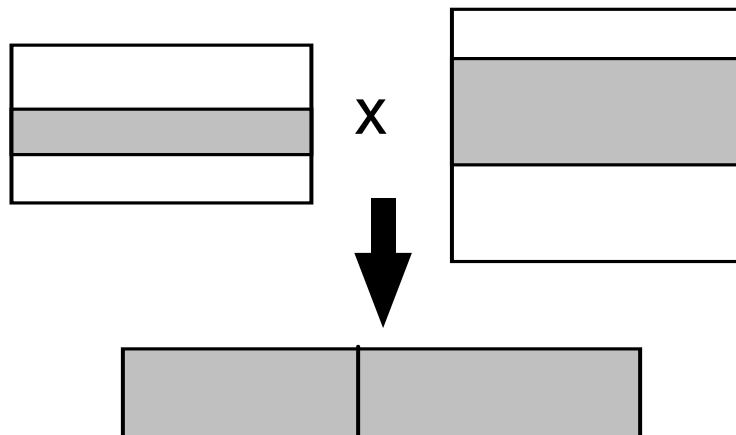
- SQL предоставляет ряд выражений подобных операции соединения реляционной алгебры
  - но следует семантике мультимножеств, а не семантики множеств.
- Эти выражения могут быть самостоятельными запросами или использоваться в позиции отношений FROM части

# Реляционные операции

**Cartesian product**



**Join x**





# Перекрестное соединение

- R **CROSS JOIN** S
  - нет условия
- CROSS JOIN возвращает декартово произведение двух таблиц
  - **SELECT \* FROM** Frequents **CROSS JOIN** Sells
  - CROSS от cross-product, синоним Cartesian product.
- можно убрать CROSS JOIN
  - для получения декартова произведения нескольких таблиц надо указать
    - во фразе FROM перечень перемножаемых таблиц,
    - во фразе SELECT – их столбцы, а
    - во фразе WHERE не указывать их сопоставление.
  - **SELECT \* FROM** Frequents, Sells

# Естественное [экви]соединение

- **R NATURAL [<тип соединения>] JOIN S**
  - условие не указывается
- по всем столбцам таблиц R и S, имеющим одинаковые имена по равенству их значений
- в результирующую таблицу одинаковые столбцы вставляются только один раз.
- устраняет необходимость включения ON предиката в JOIN выражение

– **Likes NATURAL JOIN Sells**

**Likes(drinker, beer)** и **Sells(bar, beer, price)**

# Соединение посредством предиката

- **R** [<тип соединения>] **JOIN S ON** <условие>
  - соединение посредством указанного предиката <условие> соединяет строки таблиц R и S
  - соединение использующее <условие> для отбора кортежей.
- Drinkers(**name**, addr) и Frequent(**drinker**, bar):

Drinkers **JOIN** Frequent **ON** name = drinker

- получаем  $(d, a, d, b)$  четверки такие, что любитель  $d$  живет по адресу  $a$  и посещает бар  $b$ .

# Тип соединения

<тип соединения> ::=

**INNER** | { **LEFT** | **RIGHT** | **FULL** } **OUTER**

- **INNER**

- "внутреннее" соединение - в таблицах R и S соединяются только те строки, для которых найдено совпадение.

- используется по умолчанию, когда тип соединения явно не задан

- { **LEFT** | **RIGHT** | **FULL** } **OUTER**

- "левое, правое, полное **внешние**" соединения

# Внутренние соединения

Соединение отношений:

Product(name, category) и Purchase(prodName, store)  
**сравниваем в WHERE фразе:**

**SELECT** Product.name, Purchase.store

**FROM** Product, Purchase

**WHERE** Product.name = Purchase.prodName

**то же самое**

**SELECT** Product.name, Purchase.store

**FROM** Product **JOIN** Purchase

**ON** Product.name = Purchase.prodName

**или явно указываем еще и INNER**

**SELECT** Product.name, Purchase.store

**FROM** Product **INNER JOIN** Purchase

**ON** Product.name = Purchase.prodName

# Необходимость внешних соединений

Product(name, category) и Purchase(prodName, store)

Запросы с соединениями:

```
SELECT Product.name, Purchase.store
```

```
FROM   Product JOIN Purchase ON
```

```
        Product.name = Purchase.prodName
```

и

```
SELECT Product.name, Purchase.store
```

```
FROM   Product, Purchase
```

```
WHERE  Product.name = Purchase.prodName
```

Не имеют в результате не продававшихся продуктов

## Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

## Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

# NULL значения и внешние запросы

Левое внешнее соединение (left outer joins) отношений:

Product(name, category)

Purchase(prodName, store)

SELECT Product.name, Purchase.store

FROM Product LEFT OUTER JOIN Purchase ON

Product.name = Purchase.prodName



## Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

## Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	-

# Внешние соединения

R [NATURAL]  
[{LEFT | RIGHT | FULL}] OUTER JOIN

S [ON <условие>]

— основа выражений внешних соединений.

- Его модификации:

1. Необязательный **NATURAL** перед **OUTER**.
2. Необязательные **ON** <условие> после **JOIN**.
3. Необязательные **LEFT**, **RIGHT** или **FULL** перед **OUTER**.
  - **LEFT** = добавляет только кортежи **R**
  - **RIGHT** = добавляет только кортежи **S**.
  - **FULL** = добавляет кортежи обоих отношений (действие по умолчанию).

# Внешние запросы

- **Левое** внешнее соединение (R **LEFT OUTER JOIN** S ...):
  - включает в результат кортеж **левого** операнда, даже если он **не** соответствует условия
    - для строк из таблицы R, для которых не найдено соответствия в таблице S, в столбцы, извлекаемые из таблицы S, заносятся значения NULL.
- **Правое** внешнее соединение (R **RIGHT OUTER JOIN** S ...):
  - включает в результат кортеж **правого** операнда, даже если он **не** соответствует условия
    - для строк из таблицы S, для которых не найдено соответствия в таблице R, в столбцы, извлекаемые из таблицы R, заносятся значения NULL.
- **Полное** внешнее соединение (R **FULL OUTER JOIN** S ...):
  - Включает кортежи **обоих** операндов, даже если они **не** соответствуют условию
    - для совпадающих строк поля заполняются реальными значениями, для несовпадающих строк поля заполняются в соответствии с правилами левого и правого соединений.

Bce

