Introduction to SQL

Select-From-Where Statements

Multirelation Queries

Subqueries

Why SQL?

- SQL is a very-high-level language.
  - Say "what to do" rather than "how to do it."
  - Avoid a lot of data-manipulation details needed in procedural languages like C++ or Java.
- Database management system figures out "best" way to execute query.
  - Called "query optimization."

SQL历史

- SQL
  - Structured Query Language，结构化查询语言
    - 1974年由Boyce和Chamberlin提出
    - 1975~1979年IBM公司实现
  - 目前国际标准：ANSI-99 标准
  - 2003年添加了XML的操作...

SQL特点

- SQL是一种介于关系代数和关系演算之间的结构化查询语言。
- SQL语言集数据查询（Data Query）、数据操纵（Data Manipulation）、数据定义（Data Definition）、数据控制（Data Control）于一体。
  - 关系模型中实体和实体间的联系都用关系来表示，使得操作符单一，每种操作只使用一个操作符。

SQL的特点

- 面向集合的操作方式
  - SQL语言采用集合操作方式，查询、插入、删除、修改操作的对象都是集合。
- 以同一种语法结构提供两种使用方式
  - SQL语言既是自含式语言，又是嵌入式语言，而且语法结构基本一致

SQL的特点

- 语言简洁，易学易用
  - 核心功能一共9个动词

SQL功能	动词
数据查询	SELECT
数据定义	CREATE，DROP，ALTER
数据操纵	INSERT，UPDATE，DELETE
数据控制	GRANT，REVOKE

Select-From-Where Statements

SELECT desired attributes

FROM one or more tables

WHERE condition about tuples of the tables

Our Running Example

- All our SQL queries will be based on the following database schema.
  - Underline indicates key attributes.

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

Example

- Using Beers(name, manf), what beers are made by Anheuser-Busch?

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

Result of Query

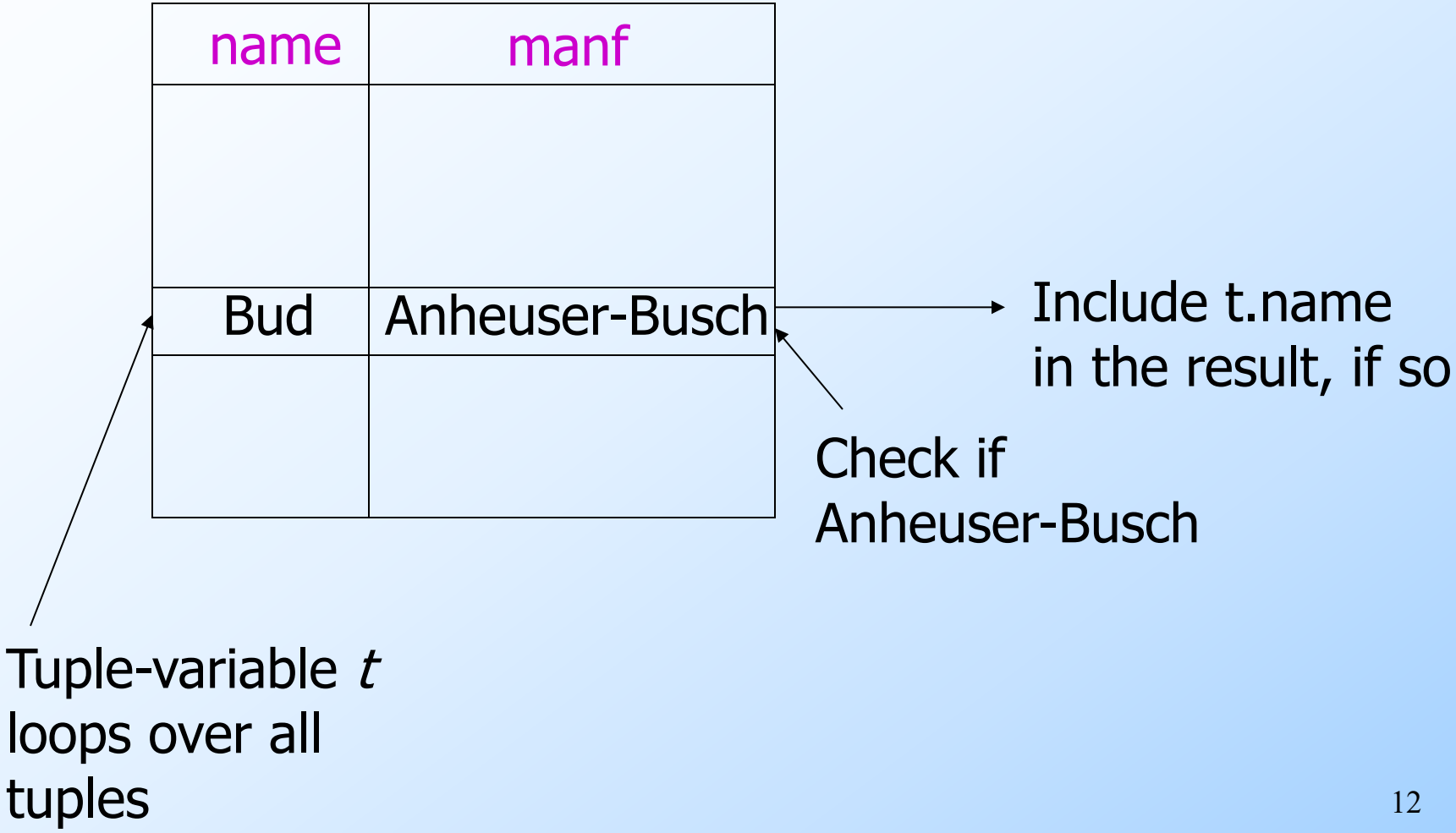
name
Bud
Bud Lite
Michelob
. . .

The answer is a relation with a single attribute, name, and tuples with the name of each beer by Anheuser-Busch, such as Bud.

Meaning of Single-Relation Query

- Begin with the relation in the FROM clause.
- Apply the selection indicated by the WHERE clause.
- Apply the extended projection indicated by the SELECT clause.

Operational Semantics



Operational Semantics --- General

- Think of a *tuple variable* visiting each tuple of the relation mentioned in FROM.
- Check if the "current" tuple satisfies the WHERE clause.
- If so, compute the attributes or expressions of the SELECT clause using the components of this tuple.

* In SELECT clauses

- When there is one relation in the FROM clause, * in the SELECT clause stands for "all attributes of this relation."
- Example: Using Beers(name, manf):

```
SELECT *
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

Result of Query:

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
. . .	. . .

Now, the result has each of the attributes of Beers.

Renaming Attributes

- If you want the result to have different attribute names, use "AS <new name>" to rename an attribute.
- Example: Using Beers(name, manf):

```
SELECT name AS beer, manf
FROM Beers
WHERE manf = 'Anheuser-Busch'
```

Result of Query:

beer	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
. . .	. . .

Expressions in SELECT Clauses

- Any expression that makes sense can appear as an element of a SELECT clause.
- Example: Using Sells(bar, beer, price):

```
SELECT bar, beer,
       price*114 AS priceInYen
FROM Sells;
```

Result of Query

bar	beer	priceInYen
Joe's	Bud	285
Sue's	Miller	342
...	...	...

Example: Constants as Expressions

- Using Likes(drinker, beer):

```
SELECT drinker,
       'likes Bud' AS whoLikesBud
FROM Likes
WHERE beer = 'Bud';
```

Result of Query

drinker	whoLikesBud
Sally	likes Bud
Fred	likes Bud
...	...

Example: Information Integration

- We often build "data warehouses" from the data at many "sources."
- Suppose each bar has its own relation Menu(beer, price) .
- To contribute to Sells(bar, beer, price) we need to query each bar and insert the name of the bar.

Information Integration --- (2)

- For instance, at Joe's Bar we can issue the query:

```
SELECT 'Joe''s Bar', beer, price
FROM Menu;
```

Complex Conditions in WHERE Clause

- Boolean operators AND, OR, NOT.
- Comparisons =, <>, <, >, <=, >=.
  - And many other operators that produce boolean-valued results.

Example: Complex Condition

- Using Sells(bar, beer, price), find the price Joe's Bar charges for Bud:

```
SELECT price
FROM Sells
WHERE bar = 'Joe''s Bar' AND
      beer = 'Bud';
```

predicate

- 使用WHERE子句

查询条件	谓词
比　　较	=,>,<,>=,<=,!=,<>,!>,!<,NOT
确定范围	BETWEEN…AND…
确定集合	IN
字符匹配	LIKE
空　　值	IS NULL
多重条件	AND,OR,NOT

Patterns

- A condition can compare a string to a pattern by:
  - <Attribute> LIKE <pattern> or <Attribute> NOT LIKE <pattern>
- *Pattern* is a quoted string with % = "any string"; _ = "any character."

Patterns

%	0~任意多个字符
_	任意单个字符
[]	集合范围内的任意单个字符
[^]	不在集合范围内的任意单个字符
[…-…]	前一字符至后一字符中的任一字符
ESCAPE	取消后面通配字符的通配作用

Example: LIKE

- Using Drinkers(name, addr, phone) find the drinkers with exchange 555:

```
SELECT name
FROM Drinkers
WHERE phone LIKE '%555-_ _ _ _';
```

LIKE

like ‘baby%’ like ‘%baby’

like ‘%baby%’ like ‘_en’

like ‘ba_by’ like ‘baby_’

like ‘[abc]%’ like ‘[a-c]ing’

like ‘[^p]en%’ like p[^q]en%’

like ‘DB_Design’ESCAPE ‘ \’

BETWEEN AND

```
SELECT beer beername, price
FROM Sells
WHERE price BETWEEN 10 AND 20
```

beertname	*price*
Chai	18
Chang	19
Aniseed Syrup	10
Genen Shouyu	15.5
Pavlova	17.45
Sir Rodney's Scones	10
…	…

IN

```
SELECT name, manf
FROM beers
WHERE manf IN ('Yanjing', 'Qingdao')
```

name	*manf*
draft	Yanjing
draught	Yanjing
draft	Qingdao
draught	Qingdao

NULL Values

- Tuples in SQL relations can have NULL as a value for one or more components.
- Meaning depends on context. Two common cases:
  - *Missing value* : e.g., we know Joe's Bar has some address, but we don't know what it is.
  - *Inapplicable* : e.g., the value of attribute spouse for an unmarried person.

Comparing NULL's to Values

- The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN.
- Comparing any value (including NULL itself) with NULL yields UNKNOWN.
- A tuple is in a query answer iff the WHERE clause is TRUE (not FALSE or UNKNOWN).

NULL

- 查询满足条件的元组

```
SELECT name, fax
FROM beers
WHERE manf IS NULL
```

用来表示无形值、[illegible]省值、不可用值，以及取最低值或取最[illegible]值。

companyname	manf
Exotic Liquids	NULL
New Orleans Cajun Delights	NULL
Tokyo Traders	NULL
Cooperativa de Quesos ‘Las Cabras’	NULL
…	…

- [illegible]、/[illegible]则该[illegible]有[illegible]’。

Three-Valued Logic

- To understand how AND, OR, and NOT work in 3-valued logic, think of TRUE = 1, FALSE = 0, and UNKNOWN = ½.
- AND = MIN; OR = MAX, NOT(*x*) = 1-*x*.
- Example:

TRUE AND (FALSE OR NOT(UNKNOWN)) =
MIN(1, MAX(0, (1 - ½))) =
MIN(1, MAX(0, ½)) = MIN(1, ½) = ½.

Surprising Example

- From the following Sells relation:

bar	beer	price
Joe's Bar	Bud	NULL

```
SELECT bar
FROM Sells
WHERE price < 2.00 OR price >= 2.00;
```

UNKNOWN UNKNOWN

UNKNOWN

Reason: 2-Valued Laws != 3-Valued Laws

- Some common laws, like commutativity of AND, hold in 3-valued logic.
- But not others, e.g., the *law of the excluded middle* : *p* OR NOT *p* = TRUE.
  - When *p* = UNKNOWN, the left side is MAX(½, (1 – ½)) = ½ != 1.

Multirelation Queries

- Interesting queries often combine data from more than one relation.
- We can address several relations in one query by listing them all in the FROM clause.
- Distinguish attributes of the same name by "<relation>.<attribute>" .

Example: Joining Two Relations

- Using relations Likes(drinker, beer) and Frequents(drinker, bar), find the beers liked by at least one person who frequents Joe's Bar.

```
SELECT beer
FROM Likes, Frequents
WHERE bar = 'Joe''s Bar' AND
   Frequents.drinker =
       Likes.drinker;
```

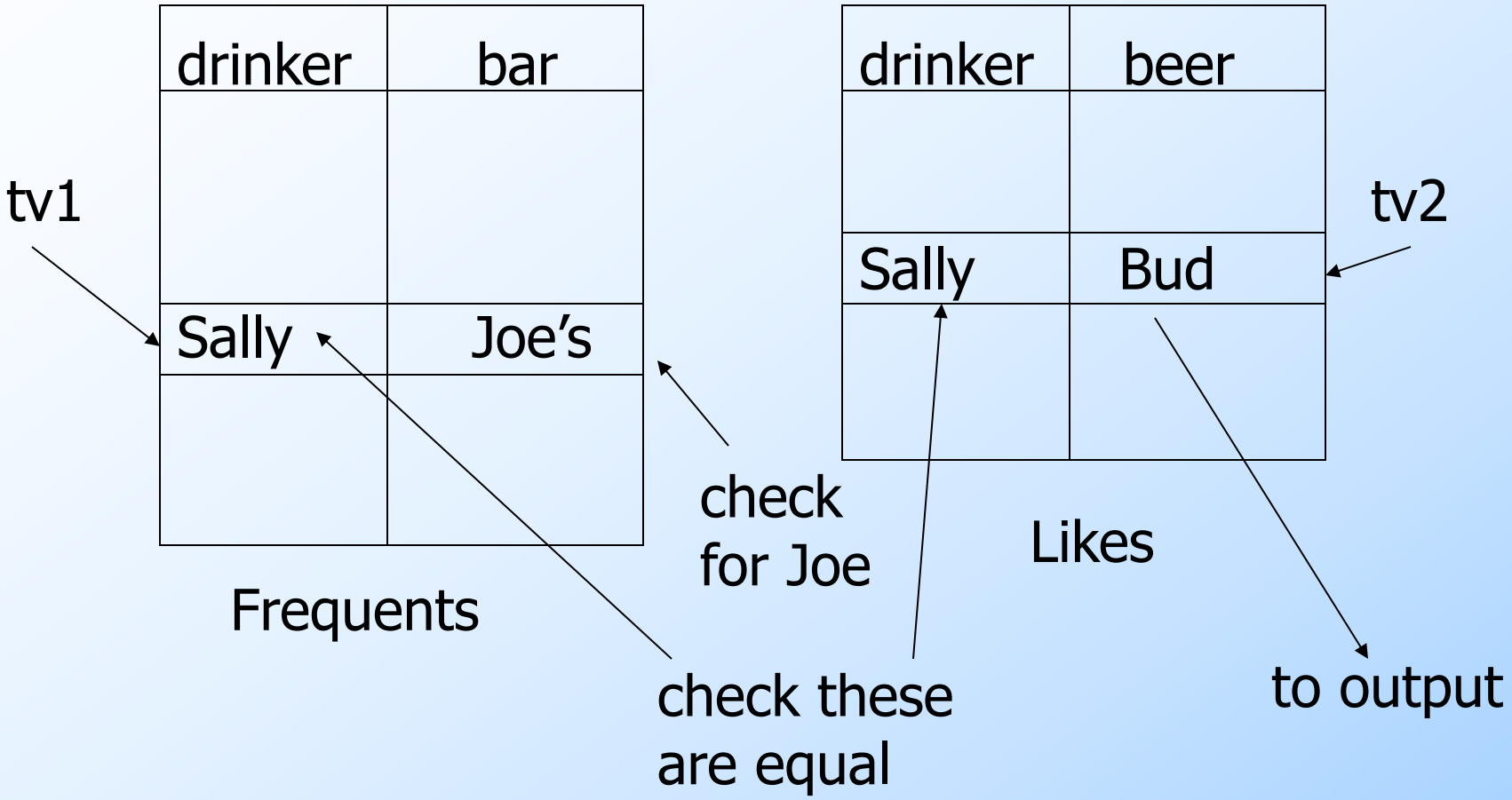
Formal Semantics

- Almost the same as for single-relation queries:
  1. Start with the product of all the relations in the FROM clause.
  2. Apply the selection condition from the WHERE clause.
  3. Project onto the list of attributes and expressions in the SELECT clause.

Operational Semantics

- Imagine one tuple-variable for each relation in the FROM clause.
  - These tuple-variables visit each combination of tuples, one from each relation.
- If the tuple-variables are pointing to tuples that satisfy the WHERE clause, send these tuples to the SELECT clause.

Example


drinker
bar
Sally
Joe's
Frequents
drinker
beer
Sally
Bud
Likes
tv1
tv2
check
for Joe
check these
are equal
to output

Explicit Tuple-Variables

- Sometimes, a query needs to use two copies of the same relation.
- Distinguish copies by following the relation name by the name of a tuple-variable, in the FROM clause.
- It's always an option to rename relations this way, even when not essential.

Example: Self-Join

- From Beers(name, manf), find all pairs of beers by the same manufacturer.
  - Do not produce pairs like (Bud, Bud).
  - Produce pairs in alphabetic order, e.g. (Bud, Miller), not (Miller, Bud).

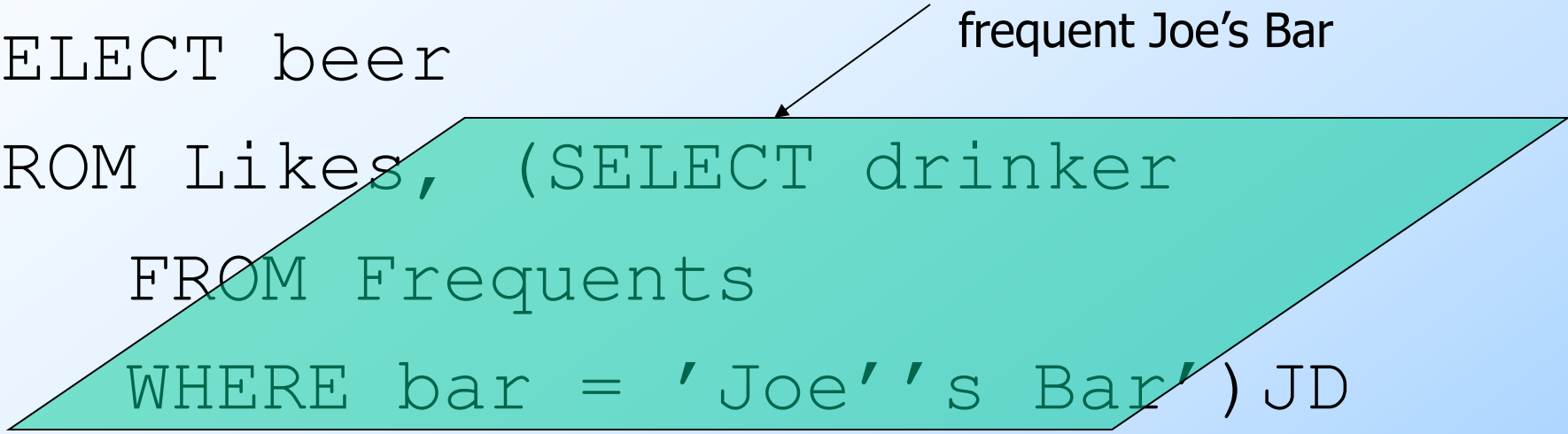
```
SELECT b1.name, b2.name
FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND
   b1.name < b2.name;
```

Subqueries

- A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a number of places, including FROM and WHERE clauses.
- Example: in place of a relation in the FROM clause, we can use a subquery and then query its result.
  - Must use a tuple-variable to name tuples of the result.

Example: Subquery in FROM

- Find the beers liked by at least one person who frequents Joe's Bar.




```
SELECT beer
FROM Likes, (SELECT drinker
     FROM Frequents
     WHERE bar = 'Joe''s Bar')JD
WHERE Likes.drinker = JD.drinker;
```

Subqueries That Return One Tuple

- If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value.
  - Usually, the tuple has one component.
  - A run-time error occurs if there is no tuple or more than one tuple.

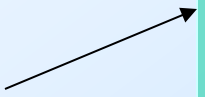
Example: Single-Tuple Subquery

- Using Sells(bar, beer, price), find the bars that serve Miller for the same price Joe charges for Bud.
- Two queries would surely work:
  1. Find the price Joe charges for Bud.
  2. Find the bars that serve Miller at that price.

Query + Subquery Solution

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
    price = (SELECT price
             FROM Sells
             WHERE bar = 'Joe''s Bar'
               AND beer = 'Bud');
```

The price at which Joe sells Bud



The IN Operator

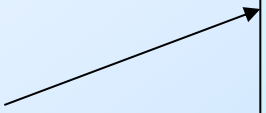
- <tuple> IN (<subquery>) is true if and only if the tuple is a member of the relation produced by the subquery.
  - Opposite: <tuple> NOT IN (<subquery>).
- IN-expressions can appear in WHERE clauses.

Example: IN

- Using Beers(name, manf) and Likes(drinker, beer), find the name and manufacturer of each beer that Fred likes.

```
SELECT *
FROM Beers
WHERE name IN (SELECT beer
               FROM Likes
               WHERE drinker = 'Fred');
```

The set of beers Fred likes



Remember These From Lecture #1?

```
SELECT a
FROM R, S
WHERE R.b = S.b;

SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

IN is a Predicate About R's Tuples

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

Two 2's

One loop, over the tuples of R

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) satisfies the condition; 1 is output once.

This Query Pairs Tuples from R, S

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

Double loop, over the tuples of R and S

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) with (2,5) and (1,2) with (2,6) both satisfy the condition;
1 is output twice.

ORDR BY

```
SELECT productid, productname,
     categoryid, unitprice
 FROM products
 ORDER BY categoryid, unitprice DESC
```

- ASC：升序（默认）　DESC：降序

productid	*productname*	*categoryid*	*unitprice*
38	Cote de Blaye	1	263.5000
43	Ipoh Coffee	1	46.0000
2	Chang	1	19.0000
…	…	…	…
63	Vegie-spread	2	43.9000
8	Northwoods Cranberry Sauc	2	40.0000
61	Sirop d'érable	2	28.5000
…	…	…	…

The Exists Operator

- EXISTS(<subquery>) is true if and only if the subquery result is not empty.
- Example: From Beers(name, manf) , find those beers that are the unique beer by their manufacturer.

EXISTS

- 带有EXISTS谓词的子查询
  - 全称量词∀
    - SQL语言中没有全称量词∀，但可以进行谓词等价转换，使用存在量词表示，并进行适当的查询。 $(\forall x)P \equiv \neg (\exists x (\neg P))$
  - 蕴涵逻辑运算
    - SQL语言中没有蕴涵逻辑运算，但可以进行谓词等价转换，并进行适当的查询。

$$p \rightarrow q \equiv \neg p \vee q$$

EXISTS

- 带有EXISTS谓词的子查询（续）
  - EXISTS代表存在量词∃。
  - Use with Correlated Subqueries
  - Determine Whether Data Exists in a List of Values

Example: EXISTS

```
SELECT name
FROM Beers b1
WHERE NOT EXISTS (
    SELECT *
    FROM Beers
    WHERE manf = b1.manf AND
          name <> b1.name);
```

Notice scope rule: manf refers to closest nested FROM with a relation having that attribute.

Set of beers with the same manf as b1, but not the same beer

Notice the SQL "not equals" operator

The Operator ANY

- *x* = ANY(<subquery>) is a boolean condition that is true iff *x* equals at least one tuple in the subquery result.
  - = could be any comparison operator.
- Example: *x* >= ANY(<subquery>) means *x* is not the uniquely smallest tuple produced by the subquery.
  - Note tuples must have one component only.

ANY

- 带有ANY（SOME）谓词的子查询

谓词	含义
> ANY	大于子查询结果的某个值
>=ANY	大于等于子查询结果的某个值
< ANY	小于子查询结果的某个值
<=ANY	小于等于子查询结果的某个值
= ANY	等于子查询结果的某个值
<>ANY	不等于子查询结果的某个值

ANY

- 带有ANY（SOME）谓词的子查询

Example 1

```
SELECT * FROM xs
   WHERE csrq < ANY
      (SELECT csrq FROM xs)
      AND xb='男'
```

Example 2

```
SELECT * FROM xs
   WHERE csrq <
      (SELECT MAX(csrq) FROM xs)
      AND xb='男'
```

The Operator ALL

- *x* <> ALL(<subquery>) is true iff for every tuple *t* in the relation, *x* is not equal to *t*.
  - That is, *x* is not in the subquery result.
- <> can be any comparison operator.
- Example: *x* >= ALL(<subquery>) means there is no tuple larger than *x* in the subquery result.

ALL

- 带有ALL谓词的子查询

谓词	含义
> ALL	大于子查询结果的所有值
>=ALL	大于等于子查询结果的所有值
< ALL	小于子查询结果的所有值
<=ALL	小于等于子查询结果的所有值
= ALL	等于子查询结果的所有值（无意义）
<>ALL	不等于子查询结果的所有值

Example: ALL

- From Sells(bar, beer, price), find the beer(s) sold for the highest price.

```
SELECT beer
FROM Sells
WHERE price >= ALL(
    SELECT price
    FROM Sells);
```

price from the outer Sells must not be less than any price.

ALL

Example 1

```
SELECT * FROM xs
    WHERE csrq >= ALL
        (SELECT csrq FROM xs)
        AND xb='女'
```

Example 2

```
SELECT * FROM xs
   WHERE csrq >=
       (SELECT MAX(csrq) FROM xs)
       AND xb='女'
```

Union, Intersection, and Difference

- Union, intersection, and difference of relations are expressed by the following forms, each involving subqueries:
  - (<subquery>) UNION (<subquery>)
  - (<subquery>) INTERSECT (<subquery>)
  - (<subquery>) EXCEPT (<subquery>)

Example: Intersection

- Using Likes(drinker, beer), Sells(bar, beer, price), and Frequents(drinker, bar), find the drinkers and beers such that:
  1. The drinker likes the beer, and
  2. The drinker frequents at least one bar that sells the beer.

Notice trick: subquery is really a stored table.

Solution

The drinker frequents a bar that sells the beer.

```
(SELECT * FROM Likes)
    INTERSECT
(SELECT drinker, beer
 FROM Sells, Frequents
 WHERE Frequents.bar = Sells.bar
);
```

Bag Semantics

- Although the SELECT-FROM-WHERE statement uses bag semantics, the default for union, intersection, and difference is set semantics.
  - That is, duplicates are eliminated as the operation is applied.

Motivation: Efficiency

- When doing projection, it is easier to avoid eliminating duplicates.
  - Just work tuple-at-a-time.
- For intersection or difference, it is most efficient to sort the relations first.
  - At that point you may as well eliminate the duplicates anyway.

Controlling Duplicate Elimination

- Force the result to be a set by SELECT DISTINCT . . .
- Force the result to be a bag (i.e., don't eliminate duplicates) by ALL, as in . . . UNION ALL . . .

Example: DISTINCT

- From Sells(bar, beer, price), find all the different prices charged for beers:

```
SELECT DISTINCT price
FROM Sells;
```

- Notice that without DISTINCT, each price would be listed as many times as there were bar/beer pairs at that price.

Example: ALL

- Using relations Frequents(drinker, bar) and Likes(drinker, beer):

```
(SELECT drinker FROM Frequents)
    EXCEPT ALL
(SELECT drinker FROM Likes);
```

- Lists drinkers who frequent more bars than they like beers, and does so as many times as the difference of those counts.

SQL SERVER的补充命令

- TOP n
  - 选取关系中的前n个元组
  - 一般和ORDER BY组合使用
  - 元组本质是无序的，因此该命令不是标准的SQL命令。
  - 基本格式：
    - SELECT [TOP n] [PERCENT] [WITH TIES] FROM…

SQL SERVER的补充命令

- TOP n

```
SELECT TOP 5 orderid, productid, quantity
 FROM [order details]
 ORDER BY quantity DESC

SELECT TOP 5 WITH TIES orderid, productid,
  quantity
 FROM [order details]
 ORDER BY quantity DESC
```


Join Expressions

- SQL provides several versions of (bag) joins.
- These expressions can be stand-alone queries or used in place of relations in a FROM clause.

Products and Natural Joins

- Natural join:

  R NATURAL JOIN S;

- Product:

  R CROSS JOIN S;

- Example:

```
Likes NATURAL JOIN Sells;
```

- Relations can be parenthesized subqueries, as well.

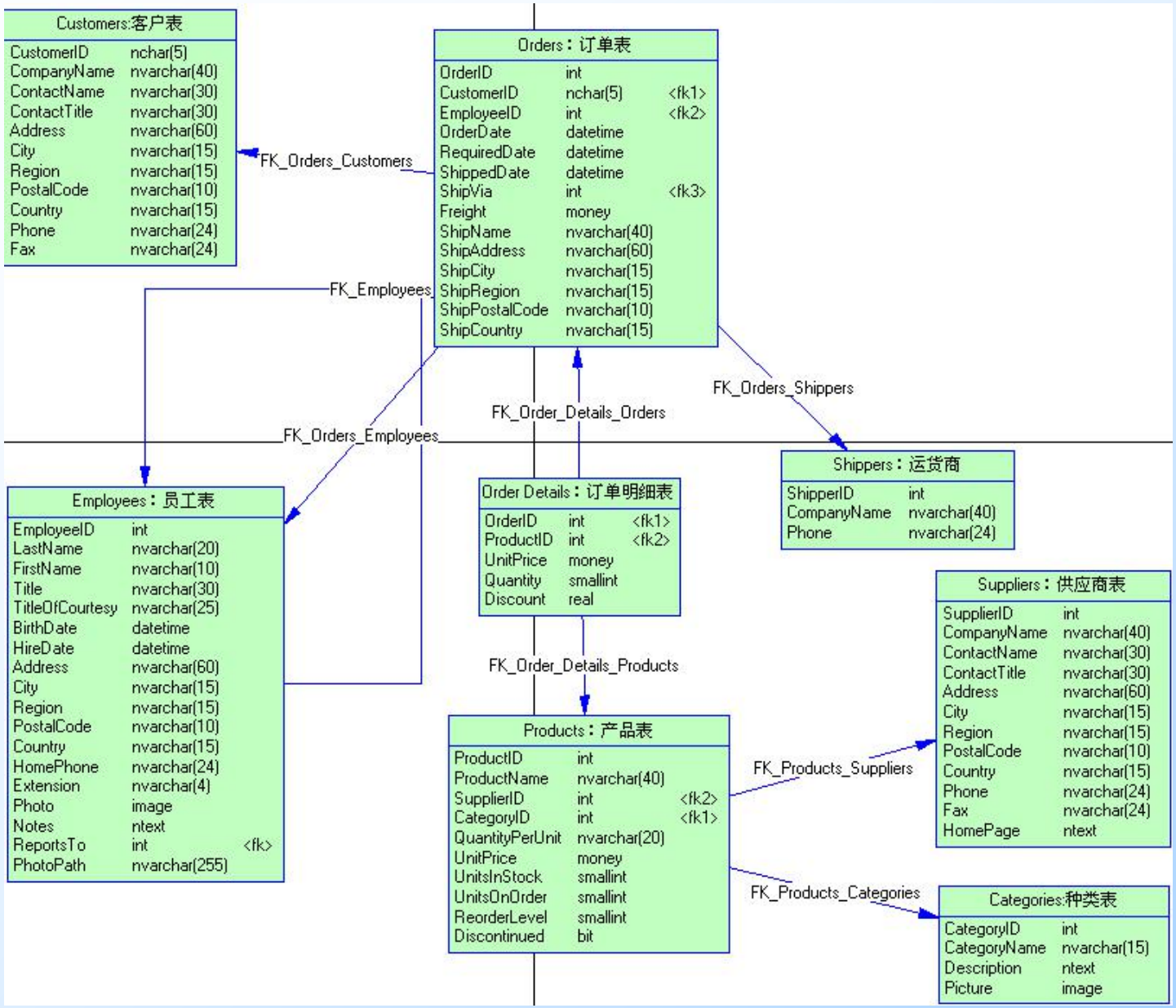
Theta Join

- R JOIN S ON <condition>
- Example: using Drinkers(name, addr) and Frequents(drinker, bar):

```
Drinkers JOIN Frequents ON
    name = drinker;
```

gives us all (*d, a, d, b*) quadruples such that drinker *d* lives at address *a* and frequents bar *b*.

Example Database



Using Inner Joins

- 连接方法
  - 目标列中的属性可能需要加表前缀。

```
SELECT buyer_name, sales.buyer_id, qty
FROM buyers  INNER JOIN sales
ON buyers.buyer_id = sales.buyer_id
```

Example 1

buyers

buyer_name	buyer_id
Adam Barr	1
Sean Chai	2
Eva Corets	3
Erin O'Melia	4

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer_name	buyer_id	qty
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003

Using Inner Joins

```
SELECT buyer_name, sales.buyer_id, qty
FROM buyers , sales
WHERE buyers.buyer_id = sales.buyer_id
```

Example 1

buyers

buyer_name	buyer_id
Adam Barr	1
Sean Chai	2
Eva Corets	3
Erin O'Melia	4

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer_name	buyer_id	qty
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003

Using Outer Joins

```
SELECT buyer_name, sales.buyer_id, qty
 FROM buyers  LEFT OUTER JOIN sales
  ON buyers.buyer_id = sales.buyer_id
```

Example 1

buyers

buyer_name	buyer_id
Adam Barr	1
Sean Chai	2
Eva Corets	3
Erin O'Melia	4

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer_name	buyer_id	qty
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003
Sean Chai	NULL	NULL

Using Outer Joins

```
SELECT buyer_name, sales.buyer_id, qty
 FROM sales  RIGHT OUTER JOIN buyers
  ON buyers.buyer_id = sales.buyer_id
```

Example 1

buyers

buyer_name	buyer_id
Adam Barr	1
Sean Chai	2
Eva Corets	3
Erin O'Melia	4

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer_name	buyer_id	qty
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003
Sean Chai	NULL	NULL

Using Cross Joins

```
SELECT buyer_name, qty
 FROM buyers
 CROSS JOIN sales
```

Example 1

buyers

buyer_id	buyer_name
1	Adam Barr
2	Sean Chai
3	Eva Corets
4	Erin O'Melia

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer_name	qty
Adam Barr	15
Adam Barr	5
Adam Barr	37
Adam Barr	11
Adam Barr	1003
Sean Chai	15
Sean Chai	5
Sean Chai	37
Sean Chai	11
Sean Chai	1003
Eva Corets	15
...	...

Jo...es

```
SELECT buyer_name, prod_name, qty
 FROM buyers
 INNER JOIN sales
  ON buyers.buyer_id = sales.buyer_id
 INNER JOIN produce
  ON sales.prod_id = produce.prod_id
```

Example 1

buyers

buyer_i	buyer_name
1	Adam Barr
2	Sean Chai
3	Eva Corets
4	Erin O'Melia

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
3	1	37
4	5	11
2	2	1003

produce

prod_id	prod_name
1	Apples
2	Pears
3	Oranges
4	Bananas
5	Peaches

Result

buyer_name	prod_name	qty
Erin O'Melia	Apples	37
Adam Barr	Pears	15
Erin O'Melia	Pears	1003
Adam Barr	Oranges	5
Eva Corets	Peaches	11

```
SELECT a.buyer_id AS buyer1, a.prod_id
      ,b.buyer_id AS buyer2
 FROM sales AS a
 JOIN sales AS b
  ON a.prod_id = b.prod_id
WHERE  a.buyer_id > b.buyer_id
```

Example 3

sales a

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

sales b

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer1	prod_id	buyer2
4	2	1

Evaluating a Correlated Subquery

1. Outer query passes column values to the inner query
2. Inner query uses that value to satisfy the inner query

```
SELECT orderid, customerid
 FROM orders AS or1
 WHERE 20 < (SELECT quantity
             FROM [order details] AS od
             WHERE or1.orderid = od.orderid
              AND  od.productid = 23)
```

Example 1

3. Inner query returns a value back to the outer query
4. The process is repeated for the next row of the outer query

Back to Step 1

Mimicking a JOIN Clause

- Correlated Subqueries Can Produce the Same Result as a JOIN Clause
- Joins Let the Query Optimizer Determine How to Correlate Data Most Efficiently

Example 1

```
SELECT DISTINCT t1.type
 FROM titles AS t1
 WHERE t1.type IN
   (SELECT t2.type
     FROM titles AS t2
     WHERE t1.pub_id <> t2.pub_id)
```

Mimicking a HAVING Clause

- Subquery with the Same Result As a HAVING Clause

Example 1

```
SELECT t1.type, t1.title, t1.price
 FROM titles AS t1
 WHERE t1.price > ( SELECT AVG(t2.price)  FROM titles AS t2
                      WHERE t1.type = t2.type )
```

- Using a HAVING Clause Without a Subquery

Example 2

```
SELECT t1.type, t1.title, t1.price
 FROM titles AS t1
 INNER JOIN titles AS t2  ON t1.type = t2.type
 GROUP BY t1.type, t1.title, t1.price
  HAVING t1.price > AVG(t2.price)
```

Using a Correlated Subquery in a HAVING Clause

- Use a Correlated Subquery in a HAVING Clause of an Outer Query

```
SELECT t1.type
FROM titles t1
GROUP BY t1.type
HAVING MAX(t1.advance) >= ALL
    (SELECT 2 * AVG(t2.advance)
    FROM titles t2
    WHERE t1.type = t2.type)
```

Using the EXISTS and NOT EXISTS Clauses

- Use with Correlated Subqueries
- Determine Whether Data Exists in a List of Values
- SQL Server Process
  - Outer query tests for the existence of rows
  - Inner query returns TRUE or FALSE
  - No data is produced

Example 1

```
SELECT lastname, employeeid
 FROM employees AS e
 WHERE EXISTS (SELECT * FROM orders AS o
                WHERE e.employeeid = o.employeeid
                 AND  o.orderdate = '9/5/2020')
```