Relational Schema Design (end)
Relational Algebra
SQL (maybe)

April 18th, 2002

Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

A relation R is in BCNF if and only if:

Whenever there is a nontrivial dependency $A_1, A_2, \ldots A_n \rightarrow B$ for R , it is the case that $\{ A_1, A_2, \ldots A_n \}$ a super-key for R.

In English (though a bit vague):

Whenever a set of attributes of *R* is determining another attribute, should determine ***all*** the attributes of *R*.

Example

Name	SSN	Phone Number
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

What are the dependencies?

SSN → Name

What are the keys?

Is it in BCNF?

Decompose it into BCNF

SSN	Name
123-321-99	Fred
909-438-44	Joe

SSN → Name

SSN	Phone Number
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

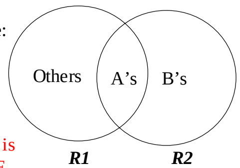
BCNF Decomposition

Find a dependency that violates the BCNF condition:

$$A_1, A_2, \ldots A_n \rightarrow B_1, B_2, \ldots B_m$$

Heuristics: choose $B_1, B_2, \ldots B_m$ "as large as possible"

Decompose:



Find a 2-attribute relation that is not in BCNF.

Continue until there are no BCNF violations left.

Correct Decompositions

A decomposition is *lossless* if we can recover:

R(A,B,C)

↓ Decompose

{ R1(A,B) , R2(A,C) }

↓ Recover

R'(A,B,C) = R(A,B,C)

R' is in general larger than R. Must ensure R' = R

Decomposition Based on BCNF is Necessarily Lossless

R(A, B, C), A → C

BCNF: R1(A,B), R2(A,C)

Some tuple (a,b,c) in R — (a,b',c') also in R
decomposes into (a,b) in R1 — (a,b') also in R1
and (a,c) in R2 — (a,c') also in R2

Recover tuples in R: (a,b,c), (a,b,c'), (a,b',c), (a,b',c') also in R ?

Can (a,b,c') be a bogus tuple? What about (a,b',c') ?

3NF: A Problem with BCNF

Unit	Company	Product

FD's: Unit → Company; Company, Product → Unit
So, there is a BCNF violation, and we decompose.

Unit	Company

Unit → Company

Unit	Product

No FDs

So What's the Problem?

Unit	Company
Galaga99	UW
Bingo	UW

Unit	Product
Galaga99	databases
Bingo	databases

No problem so far. All *local* FD's are satisfied.
Let's put all the data back into a single table again:

Unit	Company	Product
Galaga99	UW	databases
Bingo	UW	databases

Violates the dependency: company, product -> unit!

Solution: 3rd Normal Form (3NF)

A simple condition for removing anomalies from relations:

A relation R is in 3rd normal form if :

Whenever there is a nontrivial dependency $A_1, A_2, ..., A_n \rightarrow B$ for R , then $\{A_1, A_2, ..., A_n\}$ a super-key for R,
or B is part of a key.

Multi-valued Dependencies

SSN	Phone Number	Course
123-321-99	(206) 572-4312	CSE-444
123-321-99	(206) 572-4312	CSE-341
123-321-99	(206) 432-8954	CSE-444
123-321-99	(206) 432-8954	CSE-341

The multi-valued dependencies are:

SSN $\twoheadrightarrow$ Phone Number
SSN $\twoheadrightarrow$ Course

Definition of Multi-valued Dependecy

Given *R(A1,…,An,B1,…,Bm,C1,…,Cp)*

the MVD *A1,…,An* $\twoheadrightarrow$ *B1,…,Bm* holds if:

for any values of *A1,…,An*
the "set of values" of *B1,…,Bm*
is "independent" of those of *C1,…Cp*

Definition of MVDs Continued

Equivalently: the decomposition into

R1(A1,…,An,B1,…,Bm), *R2(A1,…,An,C1,…,Cp)*

is lossless

Note: an MVD *A1,…,An* $\twoheadrightarrow$ *B1,…,Bm*
Implicitly talks about "the other" attributes *C1,…Cp*

Rules for MVDs

If *A1,…An* $\rightarrow$ *B1,…,Bm*

then *A1,…,An* $\twoheadrightarrow$ *B1,…,Bm*

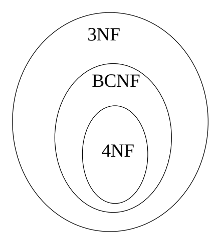
Other rules in the book

4th Normal Form (4NF)

R is in 4NF if whenever:
A1,…,An $\twoheadrightarrow$ *B1,…,Bm*
is a nontrivial MVD,
then *A1,…,An* is a superkey

Same as BCNF with FDs replaced by MVDs

Confused by Normal Forms ?



If a database doesn't violate 4NF (BCNF) then it doesn't violate BCNF (3NF) !

Querying the Database

Querying the Database

- *Find all the employees who earn more than $50,000 and pay taxes in New Jersey.*
- We *don't* want to write a program for each query.
- We design *high-level query languages:*
  - SQL (used everywhere)
  - Datalog (used by database theoreticians, their students, friends and family)
  - Relational algebra: a basic set of operations on relations that provide the *basic principles*.

Relational Algebra at a Glance

- Operators: relations as input, new relation as output
- Five basic RA operators:
  - Basic Set Operators
    - union, difference (no intersection, no complement)
  - Selection: σ
  - Projection: π
  - Cartesian Product: X
- Derived operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
- When our relations have attribute names:
  - Renaming: ρ

Set Operations

- Binary operations
- Union, difference, intersection
  - Intersection can be expressed in other ways

Set Operations: Union

- Union: all tuples in R1 or R2
- Notation: R1 U R2
- R1, R2 must have the same schema
- R1 U R2 has the same schema as R1, R2
- Example:
  - ActiveEmployees U RetiredEmployees

Set Operations: Difference

- Difference: all tuples in R1 and not in R2
- Notation: R1 – R2
- R1, R2 must have the same schema
- R1 - R2 has the same schema as R1, R2
- Example
  - AllEmployees - RetiredEmployees

Set Operations: Intersection

- Intersection: all tuples both in R1 and in R2
- Notation: $R1 \cap R2$
- R1, R2 must have the same schema
- $R1 \cap R2$ has the same schema as R1, R2
- Example
  - UnionizedEmployees $\cap$ RetiredEmployees

Selection

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- *c* is a condition: =, <, >, and, or, not
- Output schema: same as input schema
- Find all employees with salary more than $40,000:
  - $\sigma_{Salary > 40000}$(Employee)

Selection Example

Employee

SSN	Name	DepartmentID	Salary
999999999	John	1	30,000
777777777	Tony	1	32,000
888888888	Alice	2	45,000

Find all employees with salary more than $40,000.

$\sigma_{Salary > 40000}$ (Employee)

SSN	Name	DepartmentID	Salary
888888888	Alice	2	45,000

Projection

- Unary operation: returns certain columns
- Eliminates duplicate tuples !
- Notation: $\Pi_{A1,\ldots,An}(R)$
- Input schema $R(B1,\ldots,Bm)$
- Condition: $\{A1, \ldots, An\} \subseteq \{B1, \ldots, Bm\}$
- Output schema $S(A1,\ldots,An)$
- Example: project social-security number and names:
  - $\Pi_{SSN,\ Name}(Employee)$

Projection Example

Employee

SSN	Name	DepartmentID	Salary
999999999	John	1	30,000
777777777	Tony	1	32,000
888888888	Alice	2	45,000

$\Pi_{SSN,\ Name}$ (Employee)

SSN	Name
999999999	John
777777777	Tony
888888888	Alice

Cartesian Product

- Each tuple in *R1* with each tuple in *R2*
- Notation: $R1 \times R2$
- Input schemas $R1(A1,\ldots,An)$, $R2(B1,\ldots,Bm)$
- Condition: $\{A1,\ldots,An\} \cap \{B1,\ldots Bm\} = \Phi$
- Output schema is $S(A1, \ldots, An, B1, \ldots, Bm)$
- Notation: $R1 \times R2$
- Example: **Employee x Dependents**
- Very rare in practice; but joins are very common

Cartesian Product Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

EmployeeSSN	Dname
999999999	Emily
777777777	Joe

Employee x Dependents

Name	SSN	EmployeeSSN	Dname
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Renaming

- Does not change the relational instance
- Changes the relational schema only
- Notation: $\rho_{B1,\ldots,Bn}(R)$
- Input schema: $R(A1, \ldots, An)$
- Output schema: $S(B1, \ldots, Bn)$
- Example:

  $\rho_{LastName,\ SocSecNo}(Employee)$

Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

$\rho_{LastName,\ SocSecNo}$ **(Employee)**

LastName	SocSecNo
John	999999999
Tony	777777777

Derived Operations

- Intersection is derived:
  - R1 ∩ R2 = R1 – (R1 – R2) why ?
  - There is another way to express it (later)
- Most importantly: joins, in many variants

Natural Join

- Notation: *R1* ⋈ *R2*
- Input Schema: *R1(A1, …, An), R2(B1, …, Bm)*
- Output Schema: *S(C1,…,Cp)*
  - Where *{C1, …, Cp} = {A1, …, An} U {B1, …, Bm}*
- Meaning: combine all pairs of tuples in R1 and R2 that agree on the attributes:
  - *{A1,…,An}* ∩ *{B1,…, Bm}* (called the join attributes)
- Equivalent to a cross product followed by selection
- Example **Employee** ⋈ **Dependents**

Natural Join Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

SSN	Dname
999999999	Emily
777777777	Joe

Employee ⋈ **Dependents =**

$\Pi_{Name,\ SSN,\ Dname}(\sigma_{SSN=SSN2}(\text{Employee} \times \rho_{SSN2,\ Dname}(\text{Dependents}))$

Name	SSN	Dname
John	999999999	Emily
Tony	777777777	Joe

Natural Join

- R=

A	B
X	Y
X	Z
Y	Z
Z	V

S=

B	C
Z	U
V	W
Z	V

- R ⋈ S=

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

Natural Join

- Given the schemas R(A, B, C, D), S(A, C, E), what is the schema of R ⋈ S ?
- Given R(A, B, C), S(D, E), what is R ⋈ S ?
- Given R(A, B), S(A, B), what is R ⋈ S ?

Theta Join

- A join that involves a predicate
- Notation: $R1 \bowtie_{\theta} R2$ where θ is a condition
- Input schemas: $R1(A1,\ldots,An)$, $R2(B1,\ldots,Bm)$
- Output schema: $S(A1,\ldots,An,B1,\ldots,Bm)$
- It's a derived operator:

  $R1 \bowtie_{\theta} R2 = \sigma_{\theta}(R1 \times R2)$

Equi-join

- Most frequently used in practice:

  $R1 \bowtie_{A=B} R2$
- Natural join is a particular case of equi-join
- A lot of research on how to do it efficiently

Semi-join

- $R \ltimes S = \Pi_{A1,\ldots,An}(R \bowtie S)$
- Where the schemas are:
  - Input: $R(A1,\ldots An)$, $S(B1,\ldots,Bm)$
  - Output: $T(A1,\ldots,An)$

Semi-join

Applications in distributed databases:

- Product(pid, cid, pname, ...) at site 1
- Company(cid, cname, ...) at site 2
- Query: $\sigma_{price>1000}$(Product) $\bowtie_{cid=cid}$ Company
- Compute as follows:

T1 = $\sigma_{price>1000}$(Product)	site 1
T2 = P_{cid}(T1)	site 1
send T2 to site 2	(T2 smaller than T1)
T3 = T2 $\ltimes$ Company	site 2 (semijoin)
send T3 to site 1	(T3 smaller than Company)
Answer = T3 $\ltimes$ T1	site 1 (semijoin)

Relational Algebra

- Five basic operators, many derived
- Combine operators in order to construct queries: relational algebra expressions, usually shown as trees

Complex Queries

Product (pid, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, store, pid)
Company (cid, name, stock price, country)
Person(ssn, name, phone number, city)

Note:
•in Purchase: buyer-ssn, seller-ssn are **foreign keys** in Person, pid is **foreign key** in Product;
•in Product maker-cid is a **foreign key** in Company

Find phone numbers of people who bought gizmos from Fred.

Find telephony products that somebody bought

Exercises

Product (pid, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, store, pid)
Company (cid, name, stock price, country)
Person(ssn, name, phone number, city)

Ex #1: Find people who bought telephony products.
Ex #2: Find names of people who bought American products
Ex #3: Find names of people who bought American products and did not buy French products
Ex #4: Find names of people who bought American products and they live in Seattle.
Ex #5: Find people who bought stuff from Joe or bought products from a company whose stock prices is more than $50.

Operations on Bags (and why we care)

- Union: {a,b,b,c} U {a,b,b,b,e,f,f} = {a,a,b,b,b,b,b,c,e,f,f}
  - *add* the number of occurrences
- Difference: {a,b,b,b,c,c} – {b,c,c,c,d} = {a,b,b,d}
  - subtract the number of occurrences
- Intersection: {a,b,b,b,c,c} {b,b,c,c,c,c,d} = {b,b,c,c}
  - minimum of the two numbers of occurrences
- Selection: preserve the number of occurrences
- Projection: preserve the number of occurrences (no duplicate elimination)
- Cartesian product, join: no duplicate elimination

Reading assignment: 5.3

Summary of Relational Algebra

- Why bother ? Can write any RA expression directly in C++/Java, seems easy.
- Two reasons:
  - Each operator admits sophisticated implementations (think of $\bowtie$, σ_C)
  - Expressions in relational algebra can be rewritten: optimized

Glimpse Ahead: Efficient Implementations of Operators

- $\sigma_{(\textbf{age} >= 30 \text{ AND } \textbf{age} <= 35)}$**(Employees)**
  - Method 1: scan the file, test each employee
  - Method 2: use an index on **age**
  - Which one is better ? Well, depends…
- **Employees** $\bowtie$ **Relatives**
  - Iterate over Employees, then over Relatives
  - Iterate over Relatives, then over Employees
  - Sort Employees, Relatives, do "merge-join"
  - "hash-join"
  - etc

Glimpse Ahead: Optimizations

Product (pid, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, store, pid)
Person(ssn, name, phone number, city)

- Which is better:

  $\sigma_{price>100}$(Product) $\bowtie$ (Purchase $\bowtie$ $\sigma_{city=sea}$Person)

  ($\sigma_{price>100}$(Product) $\bowtie$ Purchase) $\bowtie$ $\sigma_{city=sea}$Person
- Depends ! This is the optimizer's job…

Finally: RA has Limitations !

- Cannot compute "transitive closure"

Name1	Name2	Relationship
Fred	Mary	Father
Mary	Joe	Cousin
Mary	Bill	Spouse
Nancy	Lou	Sister

- Find all direct and indirect relatives of Fred
- Cannot express in RA !!! Need to write C program

Outline

- Simple Queries in SQL (6.1)
- Queries with more than one relation (6.2)
- Subqueries (6.3)
- Duplicates (6.4)

SQL Introduction

Standard language for querying and manipulating data

Structured **Q**uery **L**anguage

Many standards out there: SQL92, SQL2, SQL3, SQL99
Vendors support various subsets of these, but all of what we'll be talking about.

SQL Introduction

Basic form: (many many more bells and whistles in addition)

```
Select   attributes
From     relations (possibly multiple, joined)
Where    conditions (selections)
```

Selections

Company(sticker, name, country, stockPrice)

Find all US companies whose stock is > 50:

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

Output schema: R(sticker, name, country, stockPrice)

Selections

What you can use in WHERE:

attribute names of the relation(s) used in the FROM.
comparison operators: =, <>, <, >, <=, >=
apply arithmetic operations: stockprice*2
operations on strings (e.g., "||" for concatenation).
Lexicographic order on strings.
Pattern matching: s LIKE p
Special stuff for comparing dates and times.

The **LIKE** operator

- s **LIKE** p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

Company(sticker, name, address, country, stockPrice)
Find all US companies whose address contains "Mountain":

```
SELECT   *
FROM     Company
WHERE    country="USA" AND
         address LIKE "%Mountain%"
```

Projections

Select only a subset of the attributes

```
SELECT   name, stockPrice
FROM     Company
WHERE    country="USA" AND stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)
Output schema: R(name, stock price)

Projections

Rename the attributes in the resulting table

```
SELECT   name AS company, stockprice AS price
FROM     Company
WHERE    country="USA" AND stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)
Output schema: R(company, price)

Ordering the Results

```
SELECT   name, stockPrice
FROM     Company
WHERE    country="USA" AND stockPrice > 50
ORDERBY  country, name
```

Ordering is ascending, unless you specify the DESC keyword.

Ties are broken by the second attribute on the ORDERBY list, etc.

Joins

Product (pname, price, category, maker)
Purchase (buyer, seller, store, product)
Company (cname, stockPrice, country)
Person(pname, phoneNumber, city)

Find names of people living in Seattle that bought gizmo products, and the names of the stores they bought from

```
SELECT   pname, store
FROM     Person, Purchase
WHERE    pname=buyer AND city="Seattle"
         AND product="gizmo"
```

Disambiguating Attributes

Find names of people buying telephony products:

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="telephony"
```

Tuple Variables

Find pairs of companies making products in the same category

```
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)

Tuple Variables

Tuple variables introduced automatically by the system:

Product (name, price, category, maker)

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

Becomes:

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

Doesn't work when Product occurs more than once:
In that case the user needs to define variables explicitely.

Meaning (Semantics) of SQL Queries

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

1. Nested loops:

```
Answer = {}
for x1 in R1 do
    for x2 in R2 do
      …..
        for xn in Rn do
            if Conditions
                then Answer = Answer U {(a1,…,ak)
return Answer
```

Meaning (Semantics) of SQL Queries

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

2. Parallel assignment

```
Answer = {}
for all assignments x1 in R1, …, xn in Rn do
    if Conditions then Answer = Answer U {(a1,…,ak)}
return Answer
```

Doesn't impose any order !
Like Datalog

Meaning (Semantics) of SQL Queries

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

3. Translation to Datalog: one rule

$Answer(a_1,\ldots,a_k) \leftarrow R_1(x_{11},\ldots,x_{1p}),\ldots,R_n(x_{n1},\ldots,x_{np})$, Conditions

Meaning (Semantics) of SQL Queries

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

4. Translation to Relational algebra:

$\Pi_{a1,\ldots,ak}(\sigma_{Conditions}(R1 \times R2 \times \ldots \times Rn))$

Select-From-Where queries are precisely Select-Project-Join

First Unintuitive SQLism

```
SELECT  R.A
FROM    R, S, T
WHERE   R.A=S.A   OR   R.A=T.A
```

Looking for $R \cap (S \cup T)$

But what happens if T is empty?

Union, Intersection, Difference

```
(SELECT  name
 FROM     Person
 WHERE    City="Seattle")

   UNION

(SELECT  name
 FROM     Person, Purchase
 WHERE    buyer=name AND store="The Bon")
```

Similarly, you can use INTERSECT and EXCEPT.
You must have the same attribute names (otherwise: rename).

Exercises

Product (pname, price, category, maker)
Purchase (buyer, seller, store, product)
Company (cname, stock price, country)
Person(per-name, phone number, city)

Ex #1: Find people who bought telephony products.
Ex #2: Find names of people who bought American products
Ex #3: Find names of people who bought American products and did not buy French products
Ex #4: Find names of people who bought American products and they live in Seattle.
Ex #5: Find people who bought stuff from Joe or bought products from a company whose stock prices is more than $50.

Subqueries

A subquery producing a single tuple:

```
SELECT Purchase.product
FROM    Purchase
WHERE  buyer =
           (SELECT  name
            FROM     Person
            WHERE    ssn = "123456789");
```

In this case, the subquery returns one value.

If it returns more, it's a run-time error.

Can say the same thing without a subquery:

```
SELECT Purchase.product
FROM    Purchase, Person
WHERE  buyer = name AND ssn = "123456789"
```

Is this query equivalent to the previous one ?

Subqueries Returning Relations

Find companies who manufacture products bought by Joe Blow.

```
SELECT  Company.name
FROM     Company, Product
WHERE   Company.name=maker
        AND  Product.name  IN
               (SELECT product
                FROM    Purchase
                WHERE  buyer = "Joe Blow");
```

Here the subquery returns a set of values

Subqueries Returning Relations

Equivalent to:

```
SELECT  Company.name
FROM     Company, Product, Purchase
WHERE   Company.name=maker
        AND  Product.name  = product
        AND  buyer = "Joe Blow"
```

Is this query equivalent to the previous one ?

Subqueries Returning Relations

You can also use: s > ALL R
s > ANY R
EXISTS R

Product (pname, price, category, maker)
Find products that are more expensive than all those produced
By "Gizmo-Works"

```
SELECT  name
FROM     Product
WHERE  price >  ALL (SELECT price
                      FROM     Purchase
                      WHERE  maker="Gizmo-Works")
```

Question for Database Fans and their Friends

- Can we express this query as a single SELECT-FROM-WHERE query, without subqueries ?
- Hint: show that all SFW queries are monotone (figure out what this means). A query with **ALL** is not monotone

Conditions on Tuples

```
SELECT  Company.name
FROM     Company, Product
WHERE  Company.name=maker
         AND  (Product.name,price)  IN
                (SELECT product, price)
                 FROM   Purchase
                 WHERE  buyer = "Joe Blow");
```

Correlated Queries

Movie (title, year, director, length)
Find movies whose title appears more than once.

```
SELECT  title
FROM   Movie AS x
WHERE  year < ANY
                (SELECT  year
                 FROM    Movie
                 WHERE  title =  x.title);
```

correlation

Note (1) scope of variables (2) this can still be expressed as single SFW

Complex Correlated Query

Product (pname, price, category, maker, year)

- Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
SELECT  pname, maker
FROM  Product AS x
WHERE  price > ALL  (SELECT  price
                     FROM    Product AS y
                     WHERE  x.maker = y.maker AND y.year < 1972);
```

Powerful, but much harder to optimize !

Removing Duplicates

```
SELECT  DISTINCT Company.name
FROM     Company, Product
WHERE  Company.name=maker
         AND  (Product.name,price)  IN
                (SELECT product, price)
                 FROM   Purchase
                 WHERE  buyer = "Joe Blow");
```

Conserving Duplicates

The UNION, INTERSECTION and EXCEPT operators operate as sets, not bags.

```
(SELECT  name
 FROM     Person
 WHERE    City="Seattle")

   UNION  ALL


(SELECT  name
 FROM     Person, Purchase
 WHERE    buyer=name AND store="The Bon")
```