# Relational Calculus

# Tuple Relational Calculus

# Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form

$$\{t \mid P(t)\}$$

- It is the set of all tuples $t$ such that predicate $P$ is true for $t$
- $t$ is a *tuple variable*
- $t[A]$ denotes the value of tuple $t$ on attribute $A$
- $t \in r$ denotes that tuple $t$ is in relation $r$
- $P$ is a *formula* similar to that of the predicate calculus

# Predicate Calculus Formula

- Set of attributes and constants
- Set of comparison operators: (e.g., $<$, $\leq$, $=$, $\neq$, $>$, $\geq$)
- Set of connectives: and ($\wedge$), or (v), not ($\neg$)
- Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if true, then y is true

$$x \Rightarrow y \equiv \neg x \text{ v } y$$

- Set of quantifiers:
  - $\exists\, t \in r\,(Q\,(t\,)) \equiv$ "there exists" a tuple in $t$ in relation $r$ such that predicate $Q\,(t\,)$ is true
  - $\forall t \in r\,(Q\,(t\,)) \equiv Q$ is true "for all" tuples $t$ in relation $r$

# Example Queries

- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than $80,000

$$\{t \mid t \in instructor \wedge t\,[salary\,] > 80000\}$$

Notice that a relation on schema (*ID*, *name*, *dept_name*, *salary*) is implicitly defined by the query

- As in the previous query, but output only the *ID* attribute value

$$\{t \mid \exists\, s \in \text{instructor}\,(t\,[ID\,] = s\,[ID\,] \wedge s\,[salary\,] > 80000)\}$$

Notice that a relation on schema (*ID*) is implicitly defined by the query

# Example Queries

- Find the names of all instructors whose department is in the Watson building
- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both

$$\{t \mid \exists s \in \textit{instructor}\ (t[\textit{name}] = s[\textit{name}] \wedge \exists u \in \textit{department}\ (u[\textit{dept\_name}] = s[\textit{dept\_name}] \wedge u[\textit{building}] = \text{“Watson”}))\}$$

$$\{t \mid \exists s \in \textit{section}\ (t[\textit{course\_id}] = s[\textit{course\_id}] \wedge s[\textit{semester}] = \text{“Fall”} \wedge s[\text{year}] = 2009 \vee \exists u \in \textit{section}\ (t[\textit{course\_id}] = u[\textit{course\_id}] \wedge u[\textit{semester}] = \text{“Spring”} \wedge u[\text{year}] = 2010))\}$$

# Example Queries

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \wedge s[semester] = \text{“Fall”} \wedge s[year] = 2009 \wedge \exists u \in section\ (t[course\_id] = u[course\_id] \wedge u[semester] = \text{“Spring”} \wedge u[year] = 2010)\}$$

- Find the set of all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \wedge s[semester] = \text{“Fall”} \wedge s[year] = 2009 \wedge \neg \exists u \in section\ (t[course\_id] = u[course\_id] \wedge u[semester] = \text{“Spring”} \wedge u[year] = 2010)\}$$

# Universal Quantification

- Find all students who have taken all courses offered in the Biology department
  - $\{t \mid \exists\, r \in \textit{student}\ (t\,[\textit{ID}] = r\,[\textit{ID}]) \wedge$
    $(\forall\, u \in \textit{course}\ (u\,[\textit{dept\_name}] = \text{“Biology”} \Rightarrow$
    $\exists\, s \in \textit{takes}\ (t\,[\textit{ID}] = s\,[\textit{ID}\,] \wedge$
    $s\,[\textit{course\_id}] = u\,[\textit{course\_id}]))\}$

# Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example, { t | $\neg$ *t* $\in$ *r* } results in an infinite relation if the domain of any attribute of relation *r* is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions.
- An expression {*t* | *P* (*t* )} in the tuple relational calculus is *safe* if every component of *t* appears in one of the relations, tuples, or constants that appear in *P*
  - NOTE: this is more than just a syntax condition.
    - E.g., { *t* | *t* [*A*] = 5 $\vee$ **true** } is not safe --- it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in *P*.

# Safety of Expressions (Cont.)

- Consider again that query to find all students who have taken all courses offered in the Biology department
  - $\{t \mid \exists\, r \in \textit{student}\ (t\,[\textit{ID}] = r\,[\textit{ID}]) \wedge$
    $(\forall\, u \in \textit{course}\ (u\,[\textit{dept\_name}] = \text{“Biology”} \Rightarrow$
    $\exists\, s \in \textit{takes}\ (t\,[\textit{ID}] = s\,[\textit{ID}\,] \wedge$
    $s\,[\textit{course\_id}] = u\,[\textit{course\_id}]))\}$
- Without the existential quantification on student, the above query would be unsafe if the Biology department has not offered any courses.

# Domain Relational Calculus

# Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus
- Each query is an expression of the form:

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n) \}$$

  - $x_1, x_2, \ldots, x_n$ represent domain variables
  - $P$ represents a formula similar to that of the predicate calculus

# Example Queries

- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than $80,000
  - $\{< i, n, d, s > \mid < i, n, d, s > \in instructor \wedge s > 80000\}$
- As in the previous query, but output only the *ID* attribute value
  - $\{< i > \mid < i, n, d, s > \in instructor \wedge s > 80000\}$
- Find the names of all instructors whose department is in the Watson building

  $\{< n > \mid \exists\, i, d, s\ (< i, n, d, s > \in instructor \wedge \exists\, b, a\ (< d, b, a > \in department \wedge b =$ "Watson" $))\}$

# Example Queries

- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both

$$\{\langle c\rangle \mid \exists\, a, s, y, b, r, t\ (\langle c, a, s, y, b, r, t\rangle \in \textit{section} \wedge s = \text{“Fall”} \wedge y = 2009) \vee \exists\, a, s, y, b, r, t\ (\langle c, a, s, y, b, r, t\rangle \in \textit{section}] \wedge s = \text{“Spring”} \wedge y = 2010)\}$$

This case can also be written as

$$\{\langle c\rangle \mid \exists\, a, s, y, b, r, t\ (\langle c, a, s, y, b, r, t\rangle \in \textit{section} \wedge ((s = \text{“Fall”} \wedge y = 2009) \vee (s = \text{“Spring”} \wedge y = 2010))\}$$

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$\{\langle c\rangle \mid \exists\, a, s, y, b, r, t\ (\langle c, a, s, y, b, r, t\rangle \in \textit{section} \wedge s = \text{“Fall”} \wedge y = 2009) \wedge \exists\, a, s, y, b, r, t\ (\langle c, a, s, y, b, r, t\rangle \in \textit{section}] \wedge s = \text{“Spring”} \wedge y = 2010)\}$$

# Safety of Expressions

The expression:

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n) \}$$

is safe if all of the following hold:

1. All values that appear in tuples of the expression are values from *dom* ($P$) (that is, the values appear either in $P$ or in a tuple of a relation mentioned in $P$).
2. For every "there exists" subformula of the form $\exists x (P_1(x))$, the subformula is true if and only if there is a value of $x$ in *dom* ($P_1$) such that $P_1(x)$ is true.
3. For every "for all" subformula of the form $\forall_x (P_1(x))$, the subformula is true if and only if $P_1(x)$ is true for all values $x$ from *dom* ($P_1$).

# Universal Quantification

- Find all students who have taken all courses offered in the Biology department
  - $\{< i > \mid \exists\, n, d, tc\, (< i, n, d, tc > \in student \;\wedge$
    $(\forall\, ci, ti, dn, cr\, (< ci, ti, dn, cr > \in course \wedge dn =$ “Biology”
    $\Rightarrow \exists\, si, se, y, g\, (<i, ci, si, se, y, g> \in takes\, ))\}$
  - Note that without the existential quantification on student, the above query would be unsafe if the Biology department has not offered any courses.

# Datalog

# Datalog: Facts and Rules

Schema

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Facts: tuples in the database

Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).

Rules: queries

(notice position matters: unnamed perspective)

Q1(y) :- Movie(x,y,z), z=1940.

?

Q2(f,l) :- Actor(u,f,l), Plays(u,x),
Movie(x,y,z), z<1940.

?

Q3(f,l) :- Actor(z,f,l), Plays(z,x1), Movie(x1,y1,1910),
Plays(z,x2), Movie(x2,y2,1940).

# Datalog: Facts and Rules

Schema

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Facts: tuples in the database

Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).

Rules: queries

(notice position matters: unnamed perspective)

Q1(y) :- Movie(x,y,z), z=1940.

Find movies from 1940

Q2(f,l) :- Actor(u,f,l), Plays(u,x),
Movie(x,y,z), z<1940.

Find actors who played in a movie before 1940

Q4(f,l) :- Actor(z,f,l), Plays(z,x1), Movie(x1,y1,1910).
Q4(f,l) :- Actor(z,f,l), Plays(z,x2), Movie(x2,y2,1940).

Find actors who played in a movie from 1910 and from 1940

OR

Extensional Database (EDB) predicates: Actor, Plays, Movie

Intensional Database (IDB) predicates: Q1, Q2, Q3, Q4

# Example with Souffle

Soufflé

Schema

| |
|---|
| Actor(id, fname, lname)<br>Plays(aid, mid)<br>Movie(id, name, year) |

movie

command line if run from the same directory:

```
souffle movie.dl
```

movie.dl

```
.decl Actor(id:number, fname:symbol, lname:symbol)
.decl Plays(aid:number, mid:number)
.decl Movie(id:number, name:symbol, year:number)
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).

.decl Q2(fname:symbol, lname:symbol)
Q2(f,l) :- Actor(u,f,l), Plays(u,x), Movie(x,_,z), z<1940.
.output Q2
```

also allows to specify specific input and output directories

```
souffle -F. -D. movie.dl
```

output

tab-separated output, filename: ".csv"

Q2.csv

| |
|---|
| Douglas Fowley |

For more help on Souffle, see: https://souffle-lang.github.io/simple
Datalog example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

# Syntax of rules

- evaluates to true when relation $R_i$ contains the tuple described by $args_i$
- e.g. Actor(344759,"Douglas","Fowley") is true

$R_i(args_i)$: relational predicate with arguments (= atom / subgoal)

arithmetic predicate

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x), Movie(x,y,z), z<1940.
```

head

(or consequent)
single IDB atom

body

(or antecedent)
conjunction of atoms

{f,l}: head variables
{u,x,y,z}: existential variables

# Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<1940.

Meaning of a Datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<1940 then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\text{Movie}(x,y,z) \wedge z<1940) \Rightarrow Q(y)]$$

Ignoring the case of an empty movie table, logically equivalent to

$$\forall y\ [\exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940] \Rightarrow Q(y)\ ]$$

Thus, non-head variables are called "existential variables"

We want the smallest set Q with this property (why?)
That takes care of the empty movie table ☺: a rules only fires if the antecedent is fulfilled ...

compare with DRC

$$\{(y)\mid \exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940]\ \}$$

# Syntactic Constraints

$$Q(\mathbf{x}) :- R_1(\mathbf{x}_1,\mathbf{y}_1),\ldots,R_m(\mathbf{x}_m,\mathbf{y}_m).$$

$\mathbf{x}_i \subseteq \mathbf{x}, \mathbf{y}_i \subseteq \mathbf{y}$

(bold = vector notation)

The rule stands for the following logical formula:

$$\forall \mathbf{x}\left[Q(\mathbf{x}) \Leftarrow \exists \mathbf{y}\left[R_1(\mathbf{x}_1,\mathbf{y}_1) \wedge \cdots \wedge R_m(\mathbf{x}_m,\mathbf{y}_m)\right]\right]$$

Two restrictions:

1. Safety: every head variable should occur in the body at least once

~~R(x,z) :- S(x,y), R(y,x).~~

forbidden rule: z not in body

2. The head predicate must be an IDB (Intensional) predicate
   (Body can include both EDBs and IDBs)

~~Arc(x,y) :- Arc(x,z),Arc(z,y).~~

This is mostly of theoretic interest. Souffle calls EDBs the "facts ... sourced from tab-separated input files" but allows them also to appear in the head of a rule (https://souffle-lang.github.io/execute)

# RA to Datalog by examples: Union

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

R(A,B,C) ∪ S(D,E,F)

Datalog:

Q(x,y,z) :- R(x,y,z)
Q(x,y,z) :- S(x,y,z)

IDB EDB

# RA to Datalog by examples: Intersection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$R(A,B,C) \cap S(D,E,F)$

Datalog:

Q(x,y,z) :- R(x,y,z), S(x,y,z)

# RA to Datalog by examples: Selection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$\sigma_{B='Alice' \wedge C>10}(R)$

Datalog:

Q(x,y,z) :- R(x,y,z), y='Alice', z > 10

(also: Q(x,y,z) :- R(x,'Alice',z), z > 10 )

# RA to Datalog by examples: Selection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$\sigma_{B='Alice' \wedge C>10}(R)$

Datalog:

Q(x,y,z) :- R(x,y,z), y='Alice', z > 10

RA:

$\sigma_{B='Alice' \vee C>10}(R)$

Datalog:

Q(x,y,z) :- R(x,y,z), y='Alice'
Q(x,y,z) :- R(x,y,z), z > 10

# RA to Datalog by examples: Projection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$\pi_{A}(R)$ $\pi_{-B,C}(R)$

Datalog:

Q(x) :- R(x,y,z)

Q(x) :- R(x,_,_)

Underscore denotes an "anonymous variable".
Each occurrence of an underscore represents a different variable

# RA to Datalog by examples: Equi-join

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

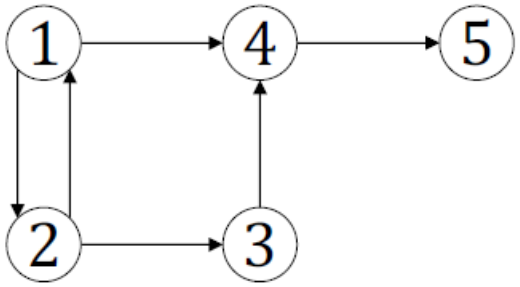
$$\pi_{-D,E}( R \bowtie_{A=D \wedge B=E} S)$$

Datalog:

?

# Recursion

## Example




A

| S | T |
|---|---|
| 1 | 2 |
| 1 | 4 |
| 2 | 1 |
| 2 | 3 |
| 3 | 4 |
| 4 | 5 |

EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```

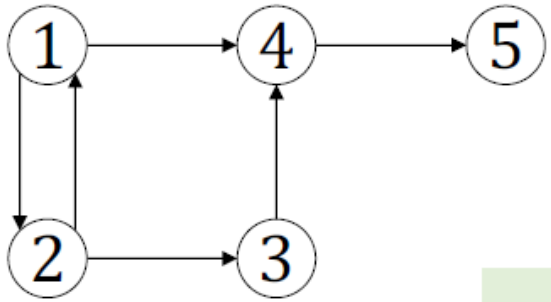
recursion due to head in rule body

A(S,T)

What does this query compute?

?

# Example

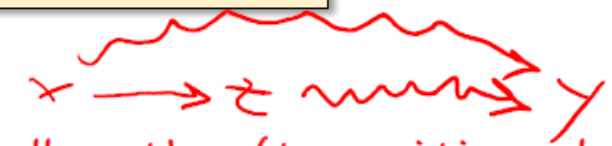



| A | S | T |
|---|---|---|
| | 1 | 2 |
| | 1 | 4 |
| | 2 | 1 |
| | 2 | 3 |
| | 3 | 4 |
| | 4 | 5 |

EDB `P(x,y) :- A(x,y).`

IDB `P(x,y) :- A(x,z), P(z,y).`

recursion due to head in rule body




Calculates all paths (transitive closure)

A(S,T)

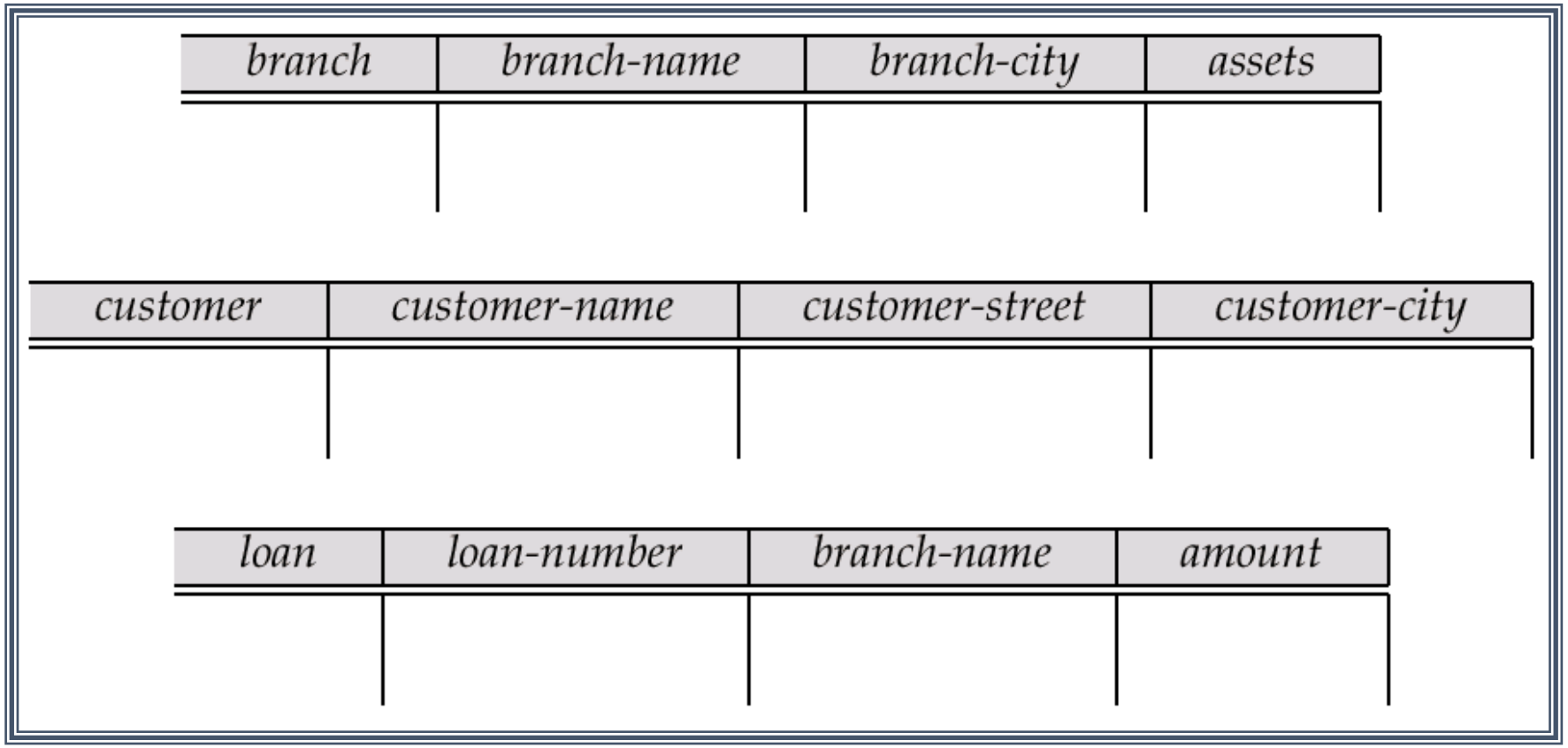
For all nodes x and y:
If there is an Arc from x to y,
then there is a Path from x to y.

For all nodes x, z, and y:
If there is an Arc from x to z, and there is a Path from z to y
then there is a Path from x to y.

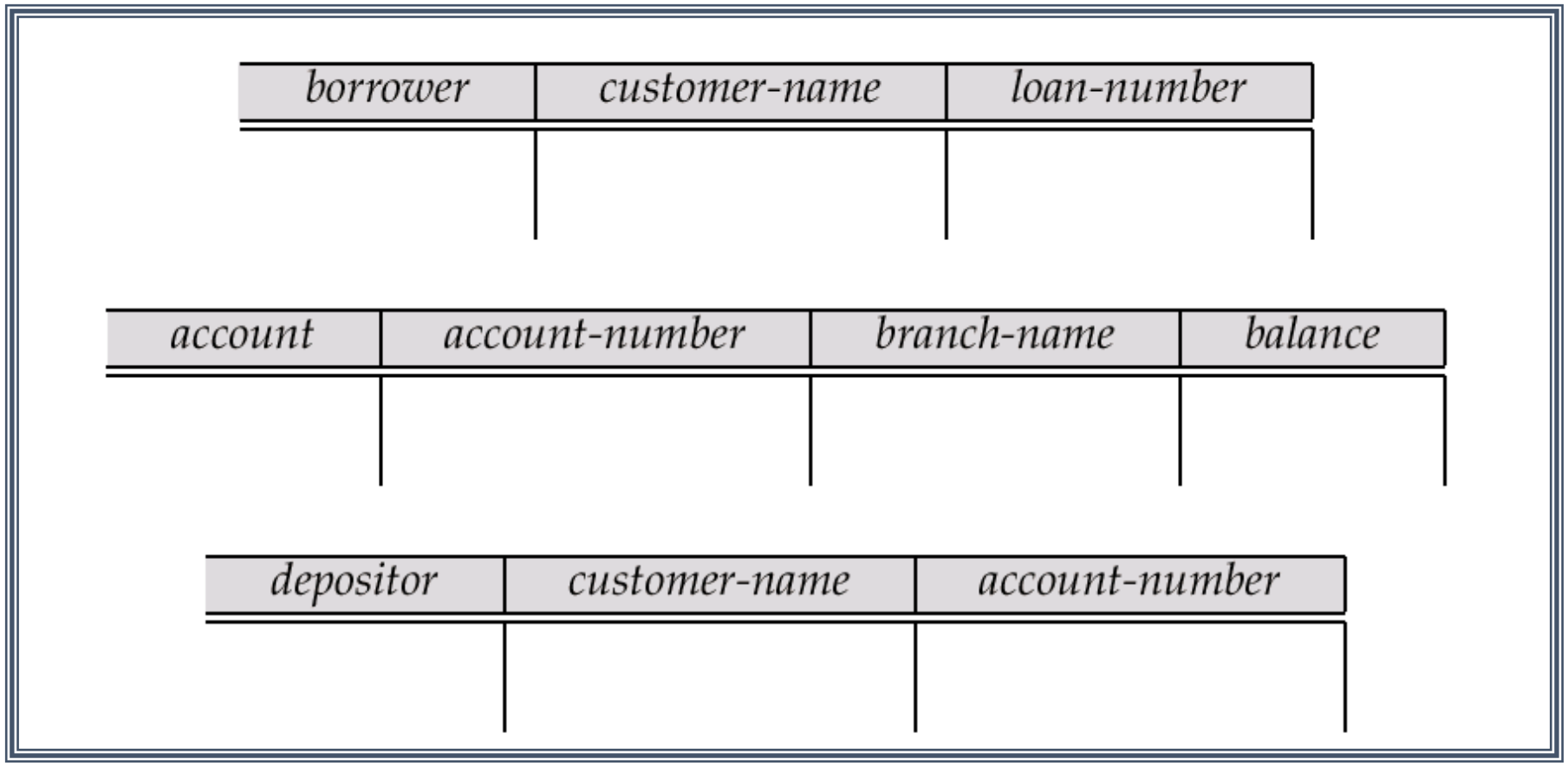
# QBE — Basic Structure

- A graphical query language which is based (roughly) on the domain relational calculus
- Two dimensional syntax – system creates templates of relations that are requested by users
- Queries are expressed “by example”
- Microsoft Access

# QBE Skeleton Tables for a Bank Example



# QBE Skeleton Tables (Cont.)



# Queries on One Relation

- Find all loan numbers at the Perryridge branch.

| *loan* | *loan-number* | *branch-name* | *amount* |
|---|---|---|---|
| | P._*x* | Perryridge | |

- _x is a variable (optional; can be omitted in above query)
- P. means print (display)
- duplicates are removed by default
- To retain duplicates use P.ALL

| *loan* | *loan-number* | *branch-name* | *amount* |
|---|---|---|---|
| | P.ALL. | Perryridge | |

# Queries on One Relation (Cont.)

- Display full details of all loans

  - Method 1:

| *loan* | *loan-number* | *branch-name* | *amount* |
|---|---|---|---|
| | P._x | P._y | P._z |

  - Method 2: Shorthand notation

| *loan* | *loan-number* | *branch-name* | *amount* |
|---|---|---|---|
| P. | | | |

# Queries on One Relation (Cont.)

- Find the loan number of all loans with a loan amount of more than $700

| *loan* | *loan-number* | *branch-name* | *amount* |
|---|---|---|---|
| | P. | | >700 |

- Find names of all branches that are not located in Brooklyn

| *branch* | *branch-name* | *branch-city* | *assets* |
|---|---|---|---|
| | P. | ¬ Brooklyn | |

# Queries on One Relation (Cont.)

- Find the loan numbers of all loans made jointly to Smith and Jones.

| *borrower* | *customer-name* | *loan-number* |
|---|---|---|
| | "Smith" | P._*x* |
| | "Jones" | _*x* |

- Find all customers who live in the same city as Jones

| *customer* | *customer-name* | *customer-street* | *customer-city* |
|---|---|---|---|
| | P._*x* | | _*y* |
| | Jones | | _*y* |

# Queries on Several Relations

- Find the names of all customers who have a loan from the Perryridge branch.

| *loan* | *loan-number* | *branch-name* | *amount* |
|---|---|---|---|
| | _x | Perryridge | |

| *borrower* | *customer-name* | *loan-number* |
|---|---|---|
| | P._y | _x |

# Queries on Several Relations (Cont.)

- Find the names of all customers who have both an account and a loan at the bank.

| *depositor* | *customer-name* | *account-number* |
|---|---|---|
| | P._*x* | |

| *borrower* | *customer-name* | *loan-number* |
|---|---|---|
| | _*x* | |

# Negation in QBE

- Find the names of all customers who have an account at the bank, but do not have a loan from the bank.

| *depositor* | *customer-name* | *account-number* |
|---|---|---|
| | P._*x* | |

| *borrower* | *customer-name* | *loan-number* |
|---|---|---|
| ¬ | _*x* | |

¬ means “there does not exist”

# Negation in QBE (Cont.)

- Find all customers who have at least two accounts.

| *depositor* | *customer-name* | *account-number* |
|---|---|---|
| | P._*x*<br>_*x* | _*y*<br>¬ _*y* |

¬ means “not equal to”

# The Condition Box

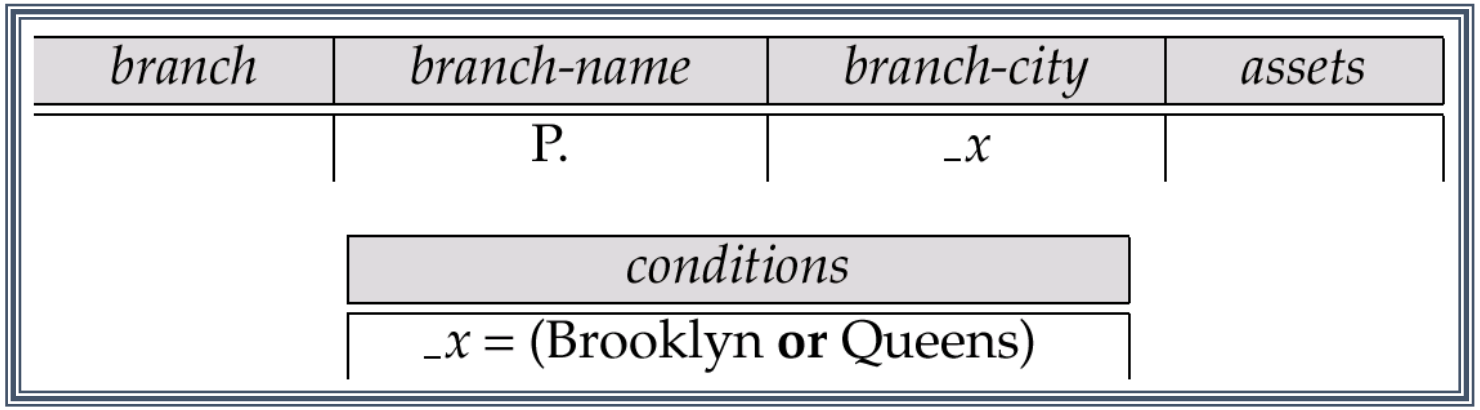
- Allows the expression of constraints on domain variables that are either inconvenient or impossible to express within the skeleton tables.
- Complex conditions can be used in condition boxes
- E.g. Find the loan numbers of all loans made to Smith, to Jones, or to both jointly

| *borrower* | *customer-name* | *loan-number* |
|---|---|---|
| | _*n* | P._*x* |

| *conditions* |
|---|
| _*n* = Smith **or** _*n* = Jones |

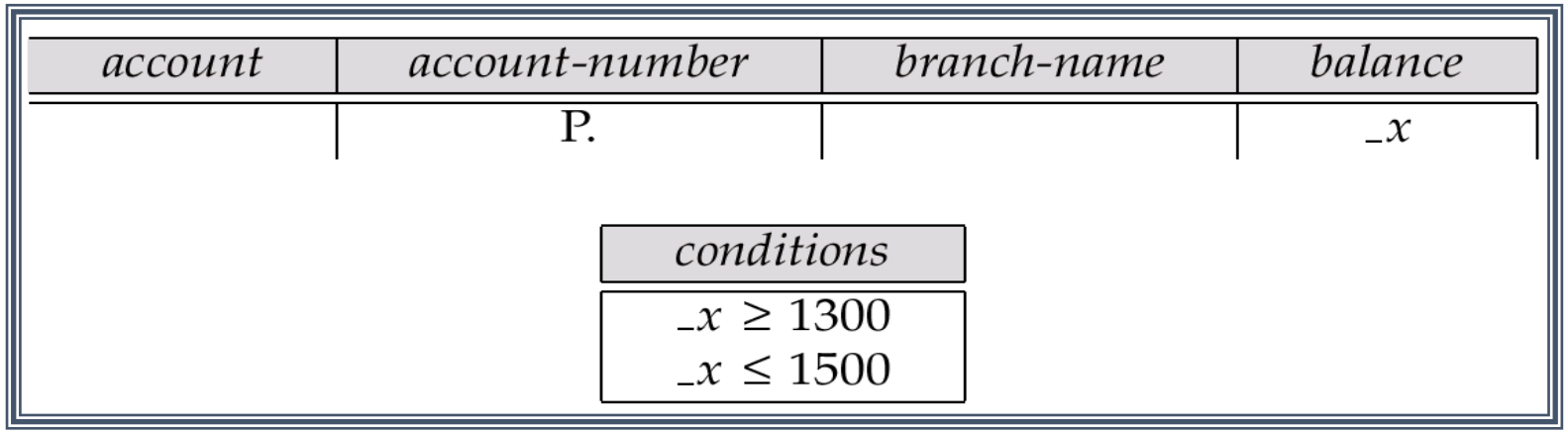
# Condition Box (Cont.)

- QBE supports an interesting syntax for expressing alternative values

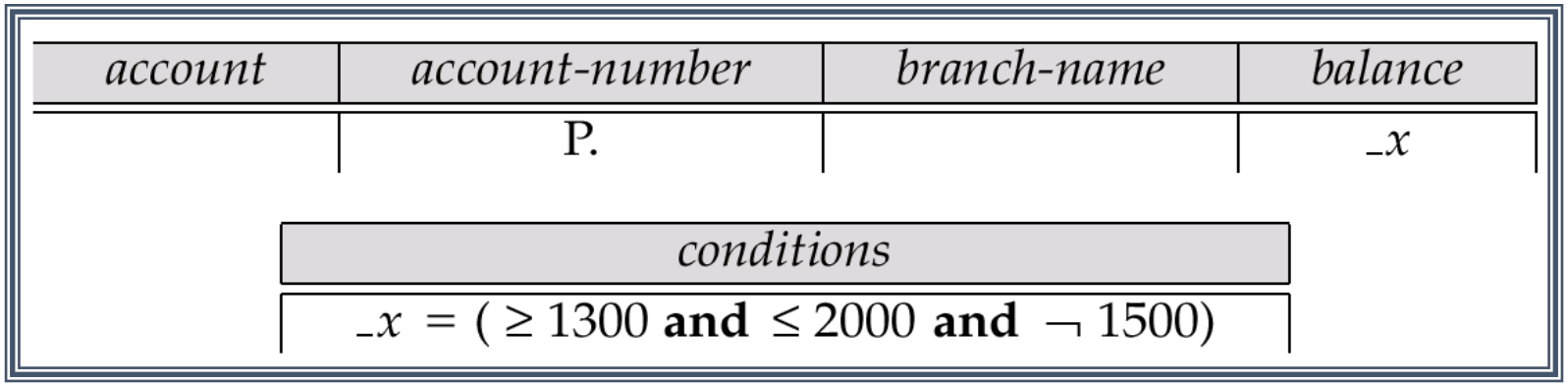


# Condition Box (Cont.)

- Find all account numbers with a balance between $1,300 and $1,500

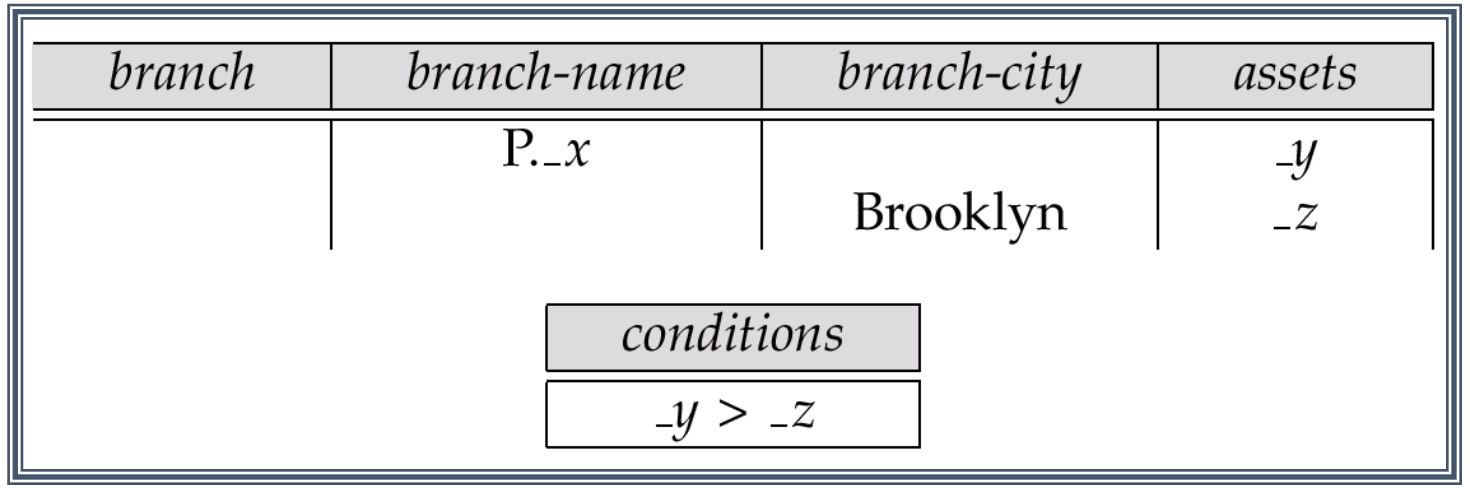



- Find all account numbers with a balance between $1,300 and $2,000 but not exactly $1,500.



# Condition Box (Cont.)

- Find all branches that have assets greater than those of at least one branch located in Brooklyn



| *branch* | *branch-name* | *branch-city* | *assets* |
|---|---|---|---|
| | P._x | | _y |
| | | Brooklyn | _z |

| *conditions* |
|---|
| _y > _z |

# The Result Relation

- Find the *customer-name*, *account-number,* and *balance* for alll customers who have an account at the Perryridge branch.
  - We need to:
    - Join *depositor* and *account.*
    - Project *customer-name, account-number* and *balance.*
  - To accomplish this we:
    - Create a skeleton table, called *result,* with attributes *customer-name, account-number,* and *balance.*
    - Write the query.

# The Result Relation (Cont.)

- The resulting query is:

| *account* | *account-number* | *branch-name* | *balance* |
|---|---|---|---|
| | _y | Perryridge | _z |

| *depositor* | *customer-name* | *account-number* |
|---|---|---|
| | _x | _y |

| *result* | *customer-name* | *account-number* | *balance* |
|---|---|---|---|
| P. | _x | _y | _z |