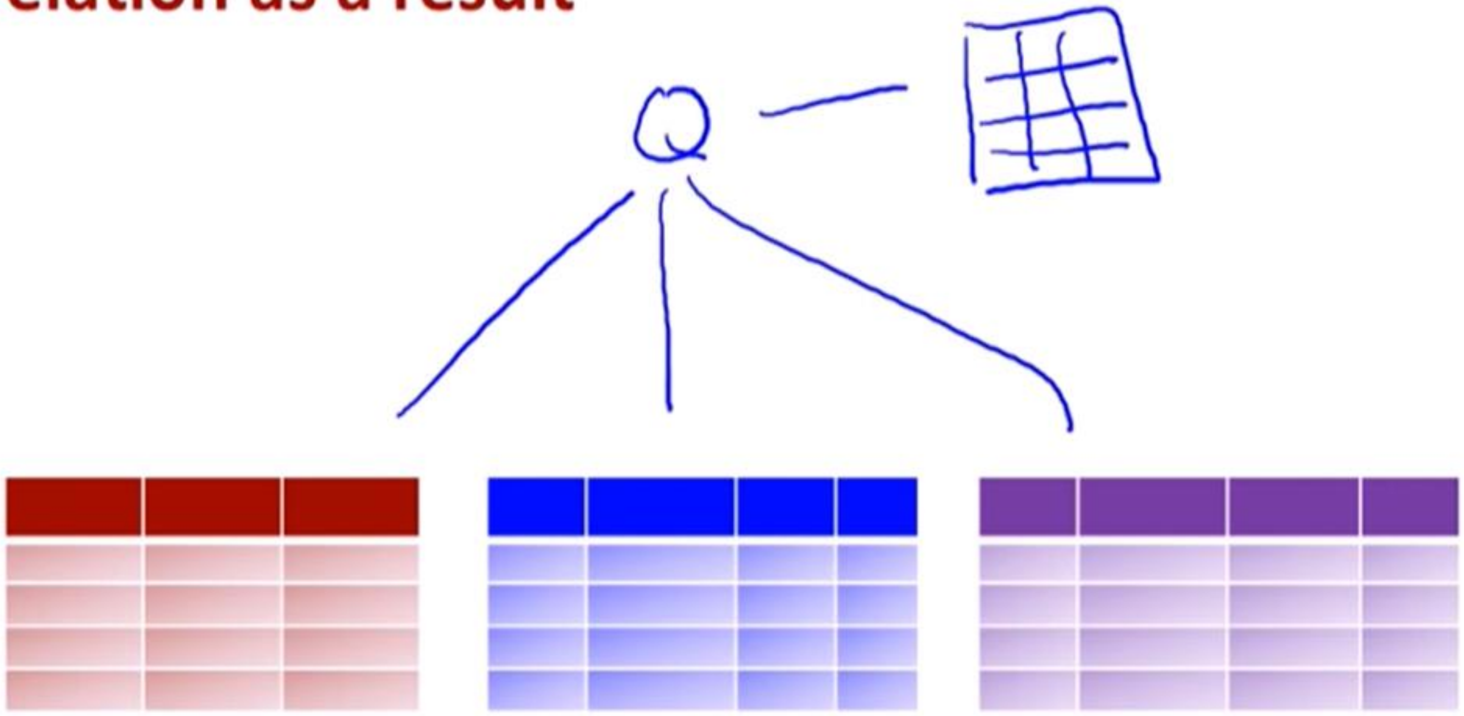
CENG 3005
Database Management Systems
Week 5

Relational Algebra

Summary of our course so far

- Introduction to databases
- ER diagram
- Relational Algebra
- Basic SQL commands (select, insert,delete,inner join, set operations)
- Advanced SQL commands (stored procedures, views, aggregate functions with group by)
- How to design a database (0 NF, 1NF, 2NF,3NF)
- Secondary Storage file structures, B+ Trees
- Indexing
- Transactions

Query (expression) on set of relations produces relation as a result



What is an "Algebra"

- Mathematical system consisting of:
  - *Operands* --- variables or values from which new values can be constructed.
  - *Operators* --- symbols denoting procedures that construct new values from given values.

What is Relational Algebra?

- An algebra whose operands are relations/tables
- Operators are designed to do the most common things that we need to do with relations in a database.
  - The result is an algebra that can be used as a *query language* for tables.

Core Relational Algebra

- Union, intersection, and difference.
  - Usual set operations, but require both operands have the same relation schema (<u>same column names!</u>)
- Select: picking certain rows.
- Project: picking certain columns.
- Cartesian products and joins: combining relations.
- Renaming of relations and attributes.

Core Relational Algebra

- Six basic operators **to filter / slice / combine relations**
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

- The operators take one or two relations as inputs and produce a new relation as a result.

Selection operator σ

- R1 := SELECT_C (R2)
  - *C* is a condition (as in "if" statements) that refers to attributes of R2.
  - R1 is all those tuples of R2 that satisfy *C*.

Select (σ) Operation – Example

- Relation **r**

A	*B*	*C*	*D*
α	α	*1*	*7*
α	β	*5*	*7*
β	β	*12*	*3*
β	β	*23*	*10*

- $\sigma_{A=B \wedge D > 5}(r)$

A	*B*	*C*	*D*
α	α	*1*	*7*
β	β	*23*	*10*

Select Example

Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

JoeMenu := $\text{SELECT}_{bar=\text{“Joe's”}}$(Sells):

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Projection Π

- R1 <- PROJ$_L$ (R2)
  - L is a list of attributes from the schema of R2.
    - NOT A CONDITION!!
  - R1 is constructed by looking at each tuple of R2, extracting the attributes on list L, in the order specified, and creating from those components a tuple for R1.
  - Eliminate duplicate tuples, if any.

Project (Π) Operation – Example

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Another Project Π Example

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Prices := $\mathrm{PROJ}_{\mathrm{beer,price}}$(Sells):

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

*Cartesian Product **

- R3 := R1 * R2
  - Pair each tuple t1 of R1 with each tuple t2 of R2.
  - Concatenation t1t2 is a tuple of R3.
  - Schema of R3 is the attributes of R1 and then R2, in order.
  - But beware attribute *A* of the same name in R1 and R2: use R1.*A* and R2.*A*.

*Cartesian Product R1 * R2*

R1(

A,	B)
1	2
3	4

R2(

B,	C)
5	6
7	8
9	10

R3(

A,	R1.B,	R2.B,	C)
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

Cartesian-Product Operation *

- Relations *r*, *s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* * *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Union Operation – ∪

- Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

Set Difference Operation –

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r - s$:

A	B
α	1
β	1

Rename Operation ρ

A	B
α	1
β	2

r

$$\rho_{A=D}(r)$$

D	B
α	1
β	2

r

$$\rho_t(r)$$

D	B
α	1
β	2

t

$$\rho_{t(A=D)}(r)$$

Composition of Operations

- You can build expressions using multiple operations
- Example: $\sigma_{A=C}(r * s)$

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

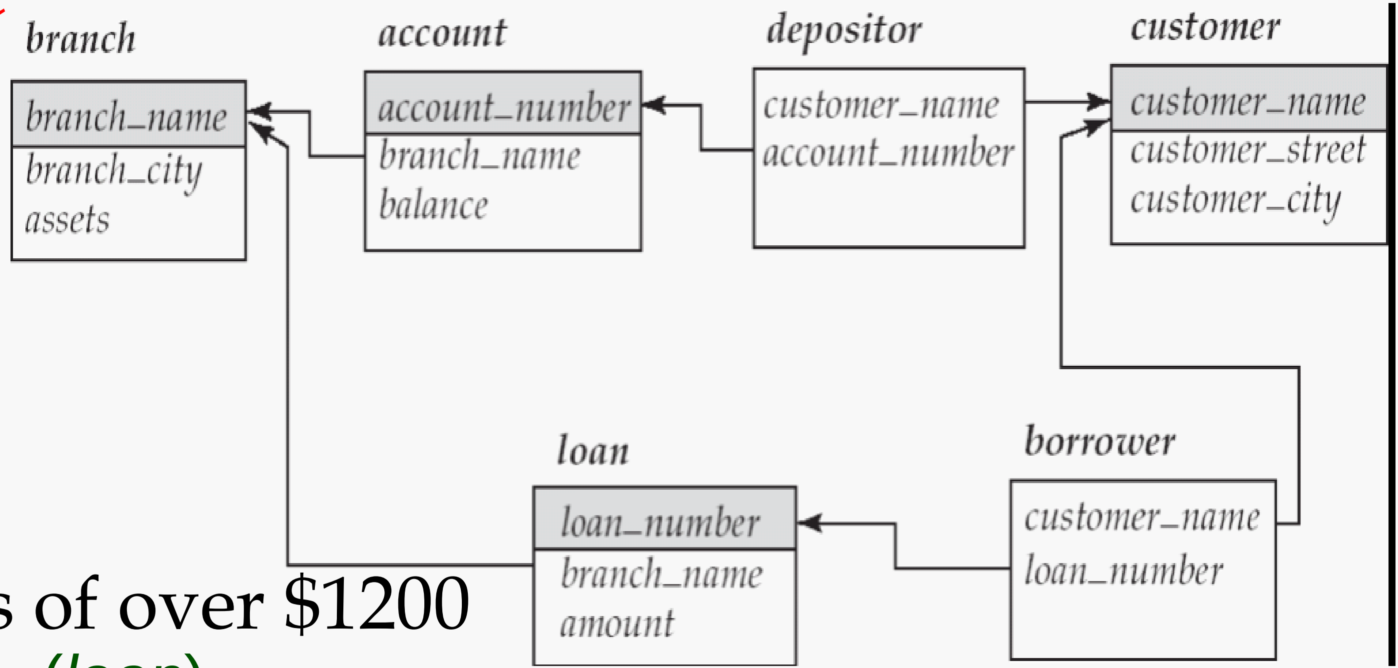
s

- $r \ x \ s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

3 Example Queries



- Find all loans of over $1200

$$\sigma_{amount\,>\,1200}\,(loan)$$

- Find the loan number for each loan of an amount greater than $1200

$$\Pi_{loan\_number}\,(\sigma_{amount\,>\,1200}\,(loan))$$

- Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer\_name}\,(borrower) \cup \Pi_{customer\_name}\,(depositor)$$

Example Queries

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

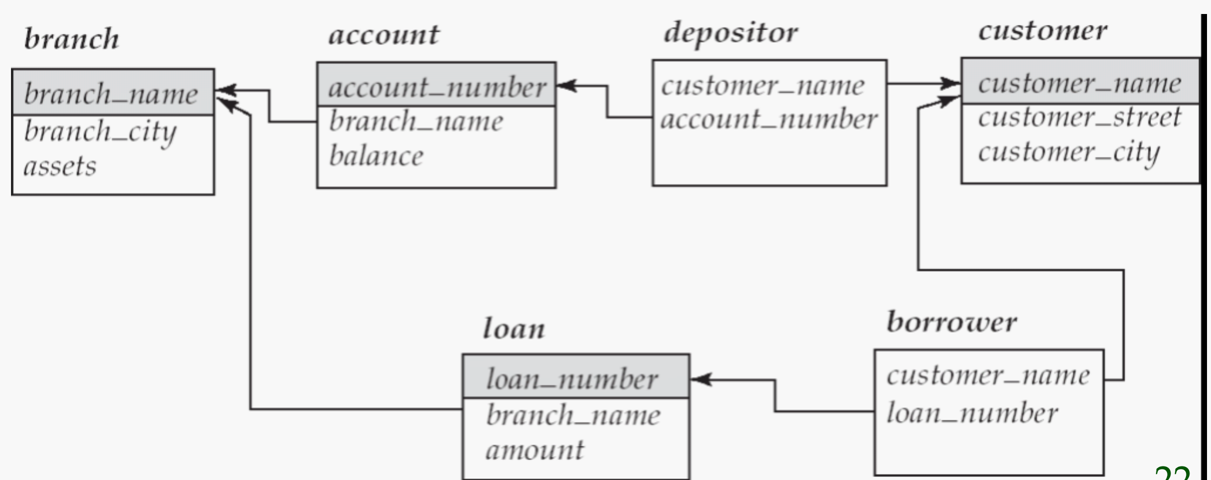
$\Pi_{customer\_name}$ ($\sigma_{branch\_name=\text{“Perryridge}}$

($\sigma_{borrower.loan\_number\ =\ loan.loan\_number}$(borrower * loan)))

- Find the names of all customers who have a loan at the NY branch but do not have an account at any branch of the bank

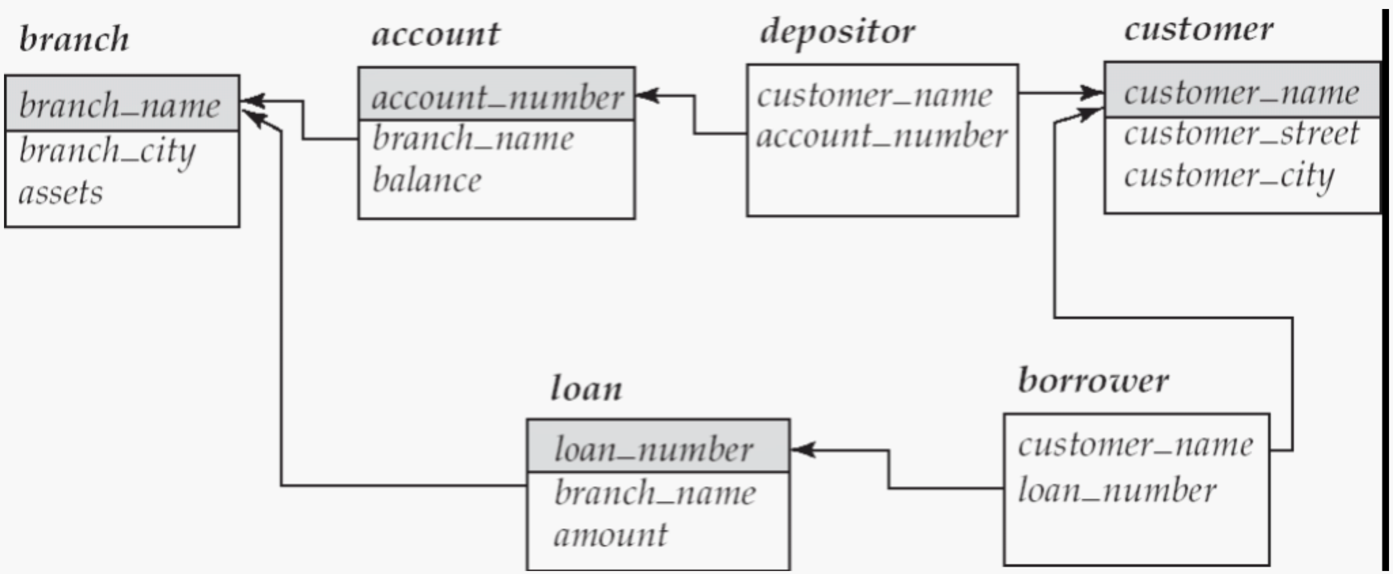
$\Pi_{customer\_name}$ ($\sigma_{branch\_name\ =\ \text{“NY”}}$ ($\sigma_{borrower.loan\_number\ =\ loan.loan\_number}$(borrower * loan)))

– $\Pi_{customer\_name}$(depositor)



Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.
  - $\Pi_{customer\_name}\ (\sigma_{branch\_name\ =\ \text{“Perryridge”}}\ (\sigma_{borrower.loan\_number\ =\ loan.loan\_number}\ (borrower * loan)))$
  - $\Pi_{customer\_name}(\sigma_{loan.loan\_number\ =\ borrower.loan\_number}\ ((\sigma_{branch\_name\ =\ \text{“Perryridge”}}\ (loan)) * borrower))$



Set-Intersection Operation ∩

- Relation *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Natural-Join Operation

□ Notation: r ⋈ s

- Let *r* and *s* be **relations** on schemas *R* and *S* respectively.
  Then, r ⋈ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$

  - Result schema = (A, B, C, D, E)
  - $r \bowtie s$ is defined as:

$$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B\ =\ s.B\ \wedge\ r.D\ =\ s.D}\ (r\ *\ s))$$

Natural Join Operation

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

- $r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Difference between Select (σ) and Project (∏)

– σ is a commutative operation

- Order of two σ is NOT important

– ∏ is an associative operation (ORDER is important)

$\prod_{list1}(\prod_{list2}(R)) = \prod_{list1}(R)$

list1 must be a subset of list 2!

->that's why not commutative

– Usually we use select and project together

- First we **σ** based on a ***condition*** and select the **rows**
- Then we ∏ the necessary **columns** for the "answer"

General Strategy for Intersections and Unions

For intersections and unions, it's useful to

- reduce everything to a **common column set** (like customer name) and
- take $\cap$ $\cup$ or - , and then
- use $\bowtie$ to fill in additional fields and then
- project with $\prod$ for the answer

Outline

- **Relational Algebra examples**
- **Relational Algebra (aggregate functions)**

$\mathcal{G}_{\mathbf{sum}(salary)}(pt\text{-}works)$

employee-name	*branch-name*	*salary*
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Rao	Austin	1500
Sato	Austin	1600

Figure 3.27 The *pt-works* relation

employee-name	*branch-name*	*salary*
Rao	Austin	1500
Sato	Austin	1600
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300

Figure 3.28 The *pt-works* relation after grouping.

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G_1, G_2, \ldots, G_n}\vartheta_{F_1(A_1), F_2(A_2, \ldots, F_n(A_n)}(E)$$

  E is any relational-algebra expression
  - $G_1, G_2 \ldots, G_n$ are the list of attributes to **group**
  - F_i are aggregate functions
  - A_i are attribute names

Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$_{branch\_name}\,\mathcal{G}\,_{\mathbf{sum}(balance)}\,(account)$

branch_name	**sum**(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $g_{\textbf{sum(c)}}$ (r)

sum(*c*)
27

$$\mathcal{G}_{\mathbf{sum}(salary)}(pt\text{-}works)$$

employee-name	*branch-name*	*salary*
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Rao	Austin	1500
Sato	Austin	1600

Figure 3.27 The *pt-works* relation

employee-name	*branch-name*	*salary*
Rao	Austin	1500
Sato	Austin	1600
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300

Figure 3.28 The *pt-works* relation after grouping.