Relational Model

Chapter -3

Chapter 3: Relational Model

- Structure of Relational Databases
- Relational Algebra
- Modification of the Database
- Views

Cartesian-Product Operation

- It allows to combine information from any two relation.
- Notation *r* x *s*
- Defined as:

$$r \text{ x } s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).
- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.

Cartesian-Product Operation-Example

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 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

Composition of Operations

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

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

$r \, x s$

- $\sigma_{A=C}(r \times s)$

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

– If the relation schema for r = borrower X loan then

– It conatinas all attributes of borrower and loan

– [borrwer.customer_name , borrower.loan_no, loan.loan_no, loan.branch_name, loan.amount]

E.G Find the name of all customer who have a loan at the perryridge branch

σ

custome
Ada
Curr
Hay
Jack
Jone
Smit
Smit
Willi

loan-number	*branch-name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

- Assume that we have n1 tuples in borrower and n2 tuples in loan. Then, there are
- n1 ∗ n2 ways of choosing a pair of tuples—one tuple from each relation; so there are n1 ∗ n2 tuples in r
- In general, if we have relations r1(R1) and r2(R2), then r1 × r2 is a relation whose
- schema is the concatenation of R1 and R2.

– relation schema for r = borrower × loan, we construct a tuple of r out of each possible pair of tuples: one from the borrower relation and one from the loan relation

customer-name	borrower. loan-number	loan. loan-number	branch-name	amount
Adams	L-16	L-11	Round Hill	900
Adams	L-16	L-14	Downtown	1500
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Adams	L-16	L-17	Downtown	1000
Adams	L-16	L-23	Redwood	2000
Adams	L-16	L-93	Mianus	500
Curry	L-93	L-11	Round Hill	900
Curry	L-93	L-14	Downtown	1500
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Curry	L-93	L-17	Downtown	1000
Curry	L-93	L-23	Redwood	2000
Curry	L-93	L-93	Mianus	500
Hayes	L-15	L-11		900
Hayes	L-15	L-14		1500
Hayes	L-15	L-15		1500
Hayes	L-15	L-16		1300
Hayes	L-15	L-17		1000
Hayes	L-15	L-23		2000
Hayes	L-15	L-93		500
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
Smith	L-23	L-11	Round Hill	900
Smith	L-23	L-14	Downtown	1500
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Smith	L-23	L-17	Downtown	1000
Smith	L-23	L-23	Redwood	2000
Smith	L-23	L-93	Mianus	500
Williams	L-17	L-11	Round Hill	900
Williams	L-17	L-14	Downtown	1500
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300
Williams	L-17	L-17	Downtown	1000
Williams	L-17	L-23	Redwood	2000
Williams	L-17	L-93	Mianus	500

customer-name	*borrower. loan-number*	*loan. loan-number*	*branch-name*	*amount*
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Hayes	L-15	L-15	Perryridge	1500
Hayes	L-15	L-16	Perryridge	1300
Jackson	L-14	L-15	Perryridge	1500
Jackson	L-14	L-16	Perryridge	1300
Jones	L-17	L-15	Perryridge	1500
Jones	L-17	L-16	Perryridge	1300
Smith	L-11	L-15	Perryridge	1500
Smith	L-11	L-16	Perryridge	1300
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300

$\sigma_{branch\text{-}name = \text{"Perryridge"}}(borrower \times loan)$

Cartesian product with condition

– $(\sigma_{borrower.loan\_number=loan.loan\_number}(\sigma_{branch\text{-}name=\text{“Perryridge”}}(borrowerXloan))$ → will give all the attributes . of cartesian product

– $\Pi_{customer\text{-}name}(\sigma_{borrower.loan\_number=loan.loan\_number}(\sigma_{branch\text{-}name=\text{“Perryridge”}}(borrowerXloan))$

customer-name
Adams
Hayes

E.G 2 id is primary key

In student table

Activity is primary key

In activity table

Students Table

Student	ID*
John Smith	084
Jane Bloggs	100
John Smith	182
Mark Antony	219

Participants Table

ID*	Activity*
084	Tennis
084	Swimming
100	Squash
100	Swimming
182	Tennis
219	Golf
219	Swimming
219	Squash

Activities Table

Activity*	Cost
Golf	$47
Sailing	$50
Squash	$40
Swimming	$15
Tennis	$36

- Find the name students who has participate in activity swimming
  - (student X participant) show all the combination of records and fields it contains is
  - (student.student,student.ID,participant.activity,participant.id)

- (studentXparticipant)

Student.student	Student.ID	Participant.id	Participant.activity
John Smith	084	084	tennis
John Smith	084	084	swimming
John Smith	084	100	sqaush
John Smith	084	100	swimming
John Smith	084	182	tennis
John Smith	084	219	golf
John Smith	084	219	Swimming
John Smith	084	219	sqaush

- $\sigma_{participant.activity=\text{"swimming"}}(student \times participant)$

student	studentID	Participant.id	Participant.activity
John smith	084	084	swimming
John smith	084	100	swimming
John smith	084	219	Swimming
Jane Blogge	100	084	swimming
Jane Blogge	100	100	swimming
Jane Blogge	100	219	Swimming
John smith	182	084	swimming
John smith	182	100	swimming
John smith	182	219	Swimming
Mark antony	219	084	swimming
Mark antony	219	100	swimming
Mark antony	219	219	Swimming

- $\sigma_{student.ID=participant.ID}(\sigma_{participant.activity=”swimming”}(student\ x\ participant))$

student	studentID	Participant.id	Participant.activity
John smith	084	084	swimming
Jane Blogge	100	100	swimming
Mark antony	219	219	Swimming

- $\Pi_{student.student}(\sigma_{student.ID=participant.ID}(\sigma_{participant.activity=”swimming”}(student\ x\ participant)))$

Rename Operation

– Allows us to give a name to the results of relational-algebra expressions.

– Allows us to refer to a relation by more than one name.

Notation: $\rho_{X}(E)$

– returns the expression *E* under the name *X*

– If a relational-algebra expression *E* has arity *n*, then $\rho_{X\ (A1,\ A2,\ \ldots,\ An)}(E)$

– returns the result of expression *E* under the name *X*, and with the

– attributes renamed to $A_1, A2, \ldots., An$.

- ρ_{stud_part} (studentxpraticipant)
- Stud_part is then name student x participant
- $\rho_{stud\_part(stud\_name,stud\_id,part\_id,part\_activity)}$ (stude
- ntxpraticipant)

Additional Operations

We define additional operations that do not add any power to the

relational algebra, but that simplify common queries.

– Set intersection

– Natural join

– Division

– Assignment

Set-Intersection Operation

- It is used to find the common from both the relation.
- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r$ **and** $t \in s \}$
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

Set-Intersection Operation - Example

- Relation 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

r- (r-s) =

A	B
α	2

- $r \cap s$

A	B
α	2

Example Queries

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

$$\Pi_{customer\text{-}name}\ (borrower) \cup \Pi_{customer\text{-}name}\ (depositor)$$

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

$$\Pi_{customer\text{-}name}\ (borrower) \cap \Pi_{customer\text{-}name}\ (depositor)$$

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:

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

Natural Join Operation – Example

- 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

Division Operation

$$r \div s$$

- Suited to queries that include the phrase "**for all**".
- Let *r* and *s* be relations on schemas R and S respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  Then devision is $r \div s$

Division Operation – Example

Relations *r, s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
$\in$	6
$\in$	1
β	2

r

B
1
2

s

$r \div s$:

A
α
β

Another Division Example

Relations *r, s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	*1*
α	a	γ	a	*1*
α	a	γ	b	*1*
β	a	γ	a	*1*
β	a	γ	b	*3*
γ	a	γ	a	*1*
γ	a	γ	b	*1*
γ	a	β	b	*1*

r

D	*E*
a	*1*
b	*1*

s

$r \div s$:

A	*B*	*C*
α	a	γ
γ	a	γ

find all customers who have an account at all branches located in broklyn

Result of $\Pi_{branch\text{-}name}(\sigma_{branch\text{-}city\ =\ \text{"Brooklyn"}}(branch))$

branch-name
Brighton
Downtown

account

account-number	*branch-name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

depositor

customer-name	*account-number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Result of $\Pi_{customer\text{-}name,\ branch\text{-}name}(depositor \bowtie account)$

Return all the customer name and branch name who are having an account

customer-name	*branch-name*
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Lindsay	Redwood
Smith	Mianus
Turner	Round Hill

branch-name
Brighton
Downtown

- $\Pi_{\text{customer-name, branch-name}}(\text{depositor} \bowtie \text{account}) \div \Pi_{\text{branch-name}}(\sigma_{\text{branch-city} = \text{“Brooklyn”}}(\text{branch}))$

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
- We can assign part of relational algebra expression to temporary relation variable
  - Assignment must always be made to a temporary relation variable.
- Example: Write $r \div s$ as

$$temp1 \leftarrow \Pi_{R-S}\,(r)$$

$$temp2 \leftarrow \Pi_{R-S}\,((temp1 \times s) - \Pi_{R-S,S}\,(r))$$

$$result = temp1 - temp2$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
  - Queries can be written as program having series of assignments

Extended Relational-Algebra-Operations

- – Generalized Projection
- – Outer Join
- – Aggregate Functions

Generalized Projection

– Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$\prod$ *customer-name, limit – credit-balance (credit-info)*

$$\prod_{F1,\ F2,\ \ldots,\ Fn}(E)$$

– *E* is any relational-algebra expression

– Each of F_1, F_2, …, F_n are are arithmetic expressions involving constants and attributes in the schema of *E*.

– Given relation *credit-info(customer-name, limit, credit-balance),* find how much more each person can spend:

– Result of aggregation does not have a name

- Can use rename operation to give it a name

– $\square_{\text{customer-name, (salary+500) as credit-available}}(\text{employee})$

$\Pi_{\textit{customer-name, (limit – credit-balance) as credit-available}}(\textit{credit-info})$

customer-name	*credit-available*
Curry	250
Jones	5300
Smith	1600
Hayes	0

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

There can be multiple occurences of values on which aggregate function works

- **Aggregate operation** in relational algebra

$$_{G1,\ G2,\ \ldots,\ Gn}\ g\ _{F1(\ A1),\ F2(\ A2),\ldots,\ Fn(\ An)}\ (E)$$

  - E is any relational-algebra expression
  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

Aggregate Operation – Example

- Relation *account*

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

Find total sum of balance amount of all accounts

$$\mathcal{G}_{sum(balance)}(account)$$

Sum(balance)
3500

– We may want to eliminate duplicate values

– Function name is hyphenated with “distinct” word

$$\mathcal{G}_{\,sum\text{-}distinct(balance)}\,(account)$$

2750

e.g.Find number of branches in account relation

$$\mathcal{G}_{\,count\text{-}distinct(branch\text{-}name)\ as\ no\text{-}of\text{-}branches}\,(account)$$

no-of-branches
3

Aggregate Operation – Example

We may want to group tuples on some attribute and then apply aggregate function

e.g. find sum of balance for each branch seperately

– 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

$$_{branch\text{-}name}\,\mathcal{G}_{\,sum(balance)}\,(account)$$

branch-name	*balance*
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Operation – Example

e.g. find sum of balance and maximum of balance for each branch seperately

- 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

$$_{branch\text{-}name}\, g\, _{sum(balance), max(balance)}\, (account)$$

branch-name	*sum(balance)*	*max(balance)*
Perryridge	1300	900
Brighton	1500	750
Redwood	700	700

Outer Join

– An extension of the join operation that avoids loss of information.
– Computes the join and then adds tuples form one relation that do not match tuples in the other relation to the result of the join.
– Uses *null* values

Outer Join – Example

- Relation *loan*

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relation *borrower*

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	L-155

Outer Join – Example

- Inner Join ; tuples common in both relation

loan $\bowtie$ *Borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Left Outer Join

loan ⟕ *Borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	null	null	hayes

Outer Join – Example

- **Right Outer Join**

  loan ⟖ *borrower* =

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

- **Full Outer Join**

  loan ⟗ *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Modification of the Database

– The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
– All these operations are expressed using the assignment operator.

Deletion

- It is similar to selection, except that selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra query.

Deletion Examples

Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch\text{-}name\, =\, \text{“Perryridge”}}\,(account)$$

Delete all loan records with amount in the range of 0 to 50

$$loan \leftarrow loan - \sigma_{amount\, \geq\, 0\ \wedge\ amount\, \leq\, 50}\,(loan)$$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- in relational algebra, an insertion is expressed by: r U { }

$$r \leftarrow r \cup E$$

where *r* is a relation and *E* is a relational algebra expression.

- The insertion of a single tuple is expressed by letting *E* be a constant relation containing one tuple.

Insertion Examples

– Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

account $\leftarrow$ *account* $\cup$ {(A-973, “Perryridge”, 1200)}

depositor $\leftarrow$ *depositor* $\cup$ {(“Smith”, A-973)}

Updating

- A mechanism to change a value in a tuple without changing *all* values in the tuple.
- Use the generalized projection operator to do this task.

$$r \leftarrow \Pi_{F1,\ F2,\ \ldots,\ Fl,}\ (r)$$

- Each F_i is either
- the *i*th attribute of r, if the *i*th attribute is not updated or if the attribute is to be updated.
- F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute.

Update Examples

– Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \Pi_{AN,\ BN,\ BAL\ *\ 1.05}\ (account)$$

Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent

$$account \leftarrow \quad \Pi_{AN,\ BN,\ BAL\ *\ 1.06}\ (\sigma_{BAL\ >\ 10000}\ (account)) \\ \cup \ \Pi_{AN,\ BN,\ BAL\ *\ 1.05}\ (\sigma_{BAL\ \leq\ 10000}\ (account))$$

End of Chapter 3