Database Design & Applications

Relational Query Languages

Formal relational query languages

- Two mathematical Query Languages form the basis for “real” languages (e.g., SQL), and for implementation:
  - ➢ Relational Algebra: More operational, very useful for representing execution plans.
  - ➢ Relational Calculus: Lets users describe what they want, rather than how to compute it. (Non-operational, declarative.)

Is this the Algebra you know?

Algebra -> operators and atomic operands
Expressions -> applying operators to atomic operands and/or other expressions

Algebra of arithmetic: operands are variables and constants, and operators are the usual arithmetic operators

E.g., (x+y)*2 or ((x+7)/(y-3)) + x

Relational algebra: operands are variables that stand for relations and relations (sets of tuples), and operations include *union*, *intersection*, *selection*, *projection*, *Cartesian product*, *etc*

– E.g., (π c-ownerChecking-account) ∩ (π s-ownerSavings-account)

What is a query?

A query is applied to *relation instances*, and the *result of a query is also a relation instance*. (view, query)
– *Schemas* of input and output fixed, but instances not.

• Operators refer to relation attributes by position or name:
– E.g., Account(number, owner, balance, type)

Positional ← Account.$1 = Account.number → Named field
Positional ← Account.$3 = Account.balance → Named field

– Positional notation easier for formal definitions, named-field notation more readable.
– Both used in SQL

Relational Algebra Operations

The usual set operations: union, intersection, difference

- Operations that remove parts of relations: selection, projection
- Operations that combine tuples from two relations: Cartesian product, join
- Since each operation returns a relation, operations can be *composed*!

Removing Parts of Relations

- Selection – rows
- Projection - columns

Selection: Example

σ_c R= select -- produces a new relation with the subset of the tuples in R that match the condition *C*

Sample query: $\sigma_{\text{Type = "savings"}}$ Account

Account	Number	Owner	Balance	Type
	~~101~~	~~J. Smith~~	~~1000.00~~	~~checking~~
	~~102~~	~~W. Wei~~	~~2000.00~~	~~checking~~
	103	J. Smith	5000.00	savings
	~~104~~	~~M. Jones~~	~~1000.00~~	~~checking~~
	~~105~~	~~H. Martin~~	~~10,000.00~~	~~checking~~

	Number	Owner	Balance	Type
	103	J. Smith	5000.00	savings

Another selection

$\sigma_{\text{Balance} < 4000}$ Account

Selects rows that satisfy *selection condition*

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	~~103~~	~~J. Smith~~	~~5000.00~~	~~savings~~
	104	M. Jones	1000.00	checking
	~~105~~	~~H. Martin~~	~~10,000.00~~	~~checking~~

	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	104	M. Jones	1000.00	checking

Schema of result identical to schema of input relation

Example of Projection

$\pi_{\text{AttributeList}}$ R = project -- deletes attributes that are not in *projection list*.

Sample query: $\pi_{\text{Number, Owner, Type}}$ Account

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

	Number	Owner	Type
	101	J. Smith	checking
	102	W. Wei	checking
	103	J. Smith	savings
	104	M. Jones	checking
	105	H. Martin	checking

π = project

Sample query: $\pi_{\text{Number, Owner, Type}}$ Account

Account	Number	Owner	~~Balance~~	Type
	101	J. Smith	~~1000.00~~	checking
	102	W. Wei	~~2000.00~~	checking
	103	J. Smith	~~5000.00~~	savings
	104	M. Jones	~~1000.00~~	checking
	105	H. Martin	~~10,000.00~~	checking

	Number	Owner	Type
	101	J. Smith	checking
	102	W. Wei	checking
	103	J. Smith	savings
	104	M. Jones	checking
	105	H. Martin	checking

LearnOA®

Projection removes duplicates

Projection: Another Example

π_{Owner} Account

Account	~~Number~~	Owner	~~Balance~~	~~Type~~
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

	Owner
	J. Smith
	W. Wei
	M. Jones
	H. Martin

Note: Projection operator eliminates *duplicates*
Why???

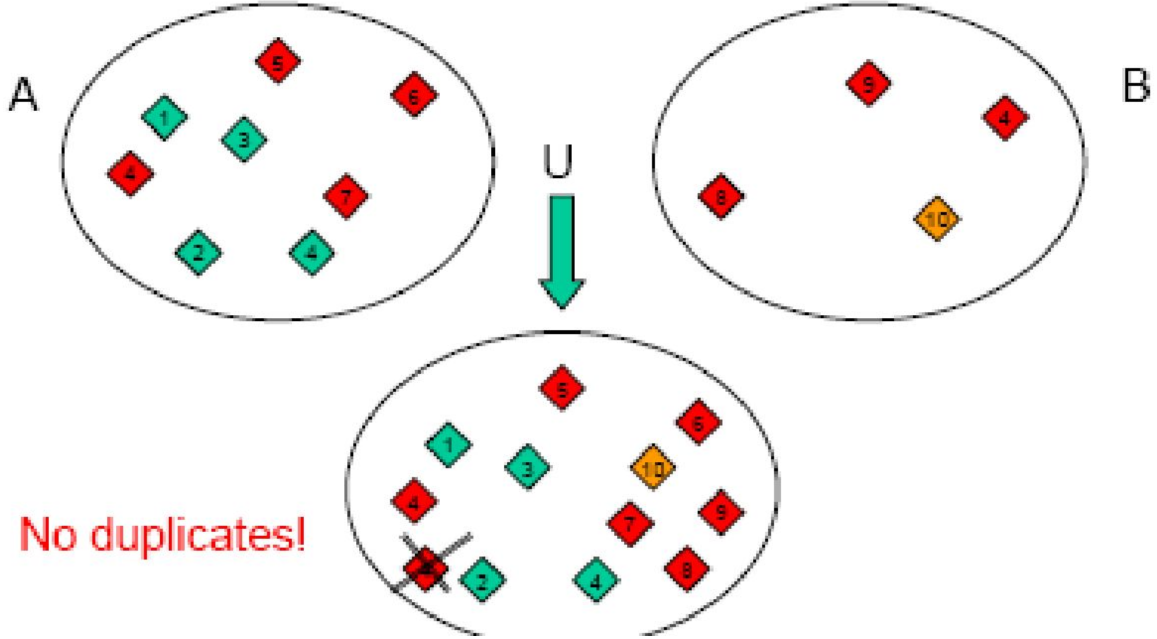
In a DBMS products, do you think duplicates should be eliminated for every query? Are they?

Set Operations

- Union
- Intersection
- Difference

What happens when sets unite?

- C = A U B



Union Operation – Example

- 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

Union Example

∪ = union Checking-account ∪ Savings-account

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00
	103	J. Smith	5000.00

Union Compatibility

- Two relations are *union-compatible* if they have the same degree (i.e., the same number of attributes) and the corresponding attributes are defined on the same domains.
- Suppose we have these tables:

  Checking-Account (c-num, c-owner, c-balance)

  Savings-Account (s-num, s-owner, s-balance)

  These are union-compatible tables.

- *Union, intersection, & difference require union-compatible tables*

Intersection

∩ = intersection

Checking-account ∩ Savings-account

What's the answer to this query?

Checking-account

c-num	c-owner	c-balance
101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-account

s-num	s-owner	s-balance
103	J. Smith	5000.00

Set Difference Operation – Example

- 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

Difference

— = difference

Checking-account — Savings-account

Checking-account

c-num	c-owner	c-balance
101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-accoun

s-num	s-owner	s-balance
103	J. Smith	5000.00

Find all the customers that own a Checking-account and do not own a Savings-account.

$(\pi_{\text{c-owner}}\text{Checking-account}) - (\pi_{\text{s-owner}}\text{Savings-account})$

- What is the *schema* of result?

Another way to show intersection?

- How could you express the intersection operation if you didn;t have an intersection operator in relational algebra? [Hint: Can you express intersection using only the difference operator?]

  A ∩ B = ???

∩

Summary so far:

- $E_1 \cup E_2$: union
- $E_1 - E_2$: difference
- $E_1 \times E_2$: cartesian product
- $\sigma_c(E_1)$: select rows, c = condition (book has p for predicate)
- $\Pi_s(E_1)$: project columns : s =selected columns
- $\rho_{x(c1,c2)}(E_1)$: rename, x is new name of E_1, c1 is new name of column

Combining Tuples of Two Relations

- Cross product (Cartesian product)
- Joins

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

Cross Product Example

X cross product

Teacher X Course

Teacher

t-num	t-name
101	Smith
105	Jones
110	Fong

Course

c-num	c-name
514	Intro to DB
513	Intro to OS

Cross product: combine information from 2 tables

- produces:

every possible combination of

a teacher and a course

t-num	t-name	c-num	c-name
101	Smith	514	Intro to DB
105	Jones	514	Intro to DB
110	Fong	514	Intro to DB
101	Smith	513	Intro to OS
105	Jones	513	Intro to OS
110	Fong	513	Intro to OS

Cross Product

- R1 X R2
- Each row of R1 is paired with each row of R2
- Result schema has one field per field of R1 and R2, with field names 'inherited' if possible.
- what about R1 X R1?

Teacher X Teacher t-num t-name t-num

Conflict!

How to resolve????

Renaming operator: ρ

Rename whole relation: Teacher X $\rho_{\text{secondteacher}}$(Teacher) □

Teacher.t-num, Teacher.t-name, secondteacher.t-num, secondteacher.t-name

OR rename attribute before combining:

Teacher X $\rho_{\text{secondteacher(t-num2, t-name2)}}$(Teacher) □

t-num, t-name, t-num2, t-name2

OR rename after combining

$\rho_{\text{c(t-num1, t-name1, t-num2, t-name2)}}$(Teacher X Teacher) □

t-num1, t-name1, t-num2, t-name2

Join: Example

$\bowtie$ = join

$\text{Account} \bowtie_{\text{Number=Account}} \text{Deposit}$

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	Transaction-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

Number	Owner	Balance	Type	Account	Transaction-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
105	H. Martin	10,000.00	checking	105	4	11/2/00	10000.00

Join : Example

⋈ join

Account $\bowtie_{Number=Account}$ Deposit

Note that when the join is based on equality, then we have two identical attributes (columns) in the answer.

Number	Owner	Balance	Type	Account	Trans-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
105	H. Martin	10,000.00	checking	105	4	11/2/00	10000.00

Condition Join

- *Condition Join*: $R \bowtie_c S = \sigma_c (R \times S)$
- *Result schema* same as that of cross-product
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

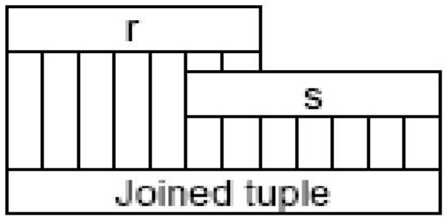
Equi and Natural Join

- *Equi-Join*: A special case of condition join where the condition *c* contains only ***equalities.***

$$\text{Student} \bowtie_{sid} \text{Takes}$$

  - *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.

- *Natural Join*: Equijoin on *all* common fields.



Why would we use Relational Algebra ?

Because:

- It is mathematically defined (where relations are sets)
- We can prove that two relational algebra expressions are equivalent. For example:

$$\sigma_{cond1}(\sigma_{cond2}\, R) \equiv \sigma_{cond2}(\sigma_{cond1}\, R) \equiv \sigma_{cond1\ and\ cond2}\, R$$

$$R1 \bowtie_{cond} R2 \equiv \sigma_{cond}\,(R1 \times R2)$$

$$R1 \div R2 \equiv \pi_x(R1) - \pi_x((\pi_x\, R1) \times R2) - R1)$$

Equivalencies help

- To help query writers - they can writes queries in several different ways
- To help query optimizers - they can choose among different ways to execute the query and in both cases we know for sure that the two queries(the original and the replacement) are identical.. that they will produce the same answer

ER vs RA

- Both ER and the Relational Model can be used to model the *structure* of a database.
- Why is it the case that there are only Relational Databases and no ER databases?

RA vs Full Programming Language

- Relational Algebra is not Turing complete. There are operations that cannot be expressed in relational algebra.
- What is the advantage of using this language to query a database?

Summary of Operators updated

- Summary so far:
- E_1 U E_2 : union
- E_1 - E_2 : difference
- E_1 x E_2 : cartesian product
- $\sigma_c(E_1)$: select rows, c = condition (book has p for predicate)
- $II_s(E_1)$: project columns : s =selected columns
- $\rho_{x(c1,c2)}(E_1)$: rename, x is new name of E_1, c1 is new name of column

- E_1 / E_2 : division
- $E_1 \bowtie_c E_2$: join, c = match condition

Extended Relational Algebra Operations

- Generalized projection
- Outer join
- Aggregate functions

Generalized projection – calculate fields

- Allows arithmetic functions to be used in the projection list.

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

- E is any relational-algebra expression
- $F_1, F_2, \ldots, F_n$ 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:

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

Can use rename to give a name to the column!

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

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

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

sum(*c*)
27

Aggregate

- Functions on more than one tuple
- Samples:
  - Sum
  - Count-distinct
  - Max
  - Min
  - Count
  - Avg
- Use “as” to rename

$$_{branchname}\, g\, _{\mathbf{sum}(balance)\ \mathbf{as}\ totalbalance}(account)$$

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

Outer Join

- Keep the outer side even if no join
- Fill in missing fields with nulls

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

  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*

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

Summary of Operators - Full

- $E_1 \cup E_2$: union
- $E_1 - E_2$: difference
- $E_1 \times E_2$: cartesian product
- $\sigma_c(E_1)$: select rows, c = condition (book has p for predicate)
- $\Pi_s(E_1)$: project columns : s =selected columns separated by commas, can have calculations included
- $\rho_{x(c1,c2)}(E_1)$: rename, x is new name of E_1, c1 is new name of column
- E_1 / E_2 : division
- $E_1 \bowtie_c E_2$: join, c = match condition
- E_1 ⟗ E_2 : outer join, c = match condition, keep the side with the arrows
- □ : assignment – give a new name to an expression to make it easy to read
- as : rename a calculated column
- ${}_{\text{attribute1}}\mathcal{G}_{\text{function (attribute2)}}(E_1)$: perform function on attribute2 whenever attribute1 changes

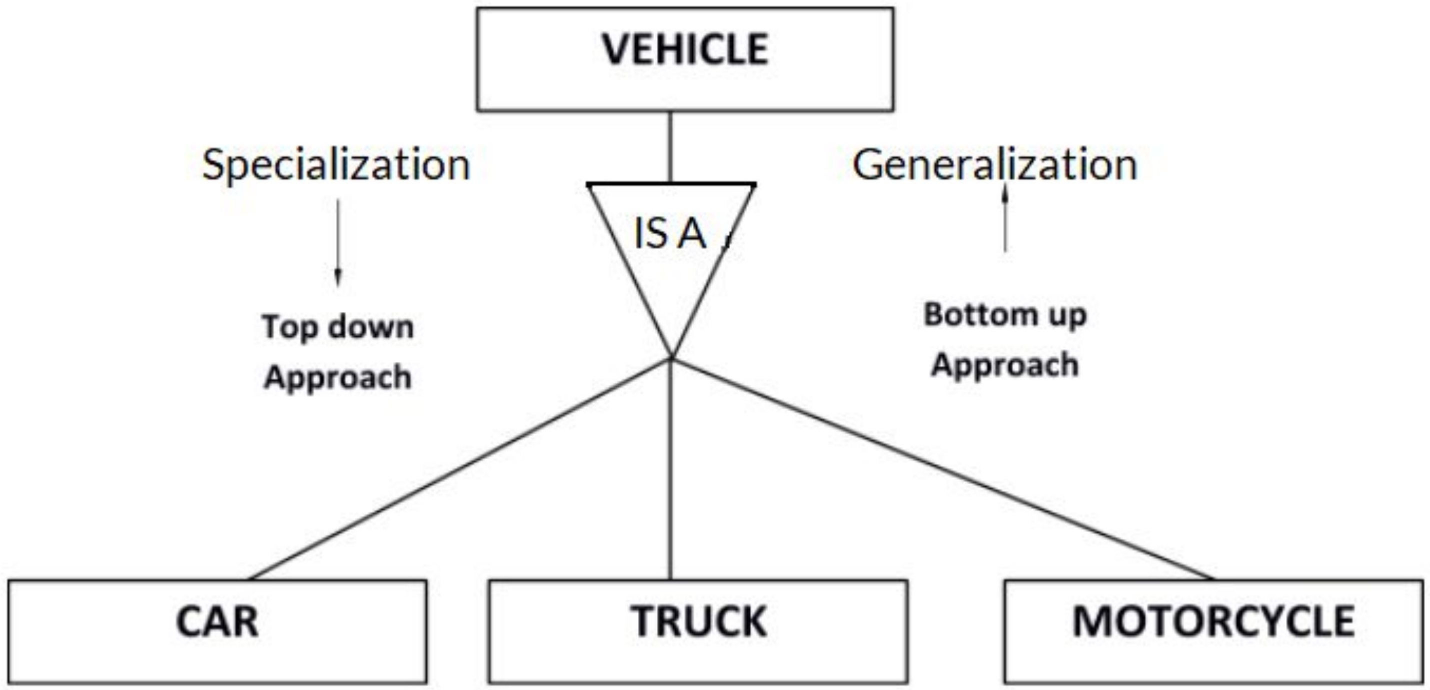
Generalization

- Generalization is a process of generalizing an entity which contains generalized attributes or properties of generalized entities. The entity that is created will contain the common features. Generalization is a Bottom up process.

- We can have three sub entities as Car, Truck, Motorcycle and these three entities can be generalized into one general super class as Vehicle.

Specialization

- Specialization is a process of identifying subsets of an entity that shares different characteristics. It breaks an entity into multiple entities from higher level (super class) to lower level (sub class). The breaking of higher level entity is based on some distinguishing characteristics of the entities in super class.
- It is a top down approach in which we first define the super class and then sub class and then their attributes and relationships.


VEHICLE
Specialization
Top down
Approach
IS A
Generalization
Bottom up
Approach
CAR
TRUCK
MOTORCYCLE

Aggregation

- Aggregation represents relationship between a whole object and its component. Using aggregation we can express relationship among relationships. Aggregation shows 'has-a' or 'is-part-of' relationship between entities where one represents the 'whole' and other 'part'.

Thank you!