CS 2550 / Spring 2006
Principles of Database Systems

02 – Relational Model

Alexandros Labrinidis
University of Pittsburgh

Relational Model

- It is the most popular implementation model
  - Simplest, most uniform data structures
  - Most formal (algebra to describe operations)
- Introduced in 1970 (by E. F. Codd)
- Everything from real world is represented by **relations** (i.e. tables)
- Each table has multiple rows and columns
  - Row in a table "binds" values together (row = tuple)

Relations

Attributes (=columns)

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

The **account** relation

Domain: set of permitted values

tuple **t**

t[account-number] = A-215

Relations are sets

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

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

Tuple order is not important
The two relations are exactly the same

The Mathematical Concept of Relation

- Let $D_1, D_2, \ldots, D_n$ be domains (not necessarily distinct)
- the Cartesian product of these *n* sets
  $$D_1 \times D_2 \times \ldots \times D_n$$
  is the set of all possible ordered n-tuples
  $(v_1, v_2, \ldots, v_n)$ such that $v_1 \in D_1, v_2 \in D_2, \ldots, v_n \in D_n$
- **Example**: let $D_1 = \{$Nick, Susan$\}$ and $D_2 = \{$BS, MS, PhD$\}$
- $D_1 \times D_2$ = {(Nick, BS), (Nick, MS), (Nick, PhD), (Susan,BS), (Susan, MS), (Susan, PhD)}
- A relation is any subset of the Cartesian product
  - R_1= {(Nick,BS),(Nick, MS), (Susan, BS), (Susan, PhD)}
  - R_2= {}

Relation Schema

- A relation schema specifies:
  - Name of relation
  - Names of attributes of the relation
  - The domain for each attribute
- Database schema = set of relation schemas, constraints (i.e. the logical design)
- Database instance = snapshot of the data in database

Relation Schema Examples

- Account-schema = (account-number, branch-name, balance)
- Branch-schema = (branch-name, branch-city, assets)
- Customer-schema=(customer-name, customer-street, customer-city)
  - For simplicity assume *customer-name* unique
- Depositor-schema = (customer-name, account-number)
- Loan-schema = (loan-number, branch-name, amount)
- Borrower-schema = (customer-name, loan-number)

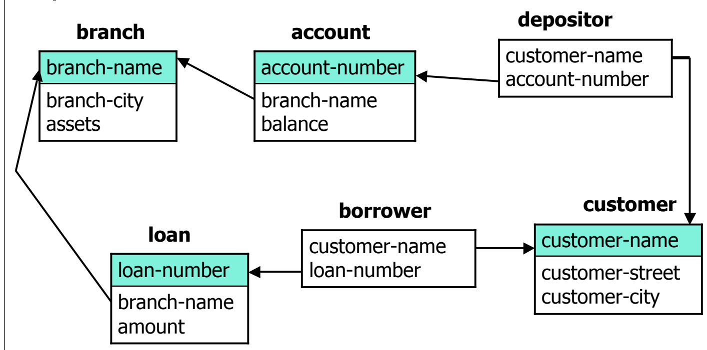
Keys

- Same definitions from Entity-Relationship model
- Superkey
  - Set of one or more attributes that, taken collectively, uniquely identify a tuple within the relation
  - E.g., {customer-name}, {customer-name, customer-city}
- Candidate key
  - Superkey for which no proper subset is superkey (i.e. minimal)
  - E.g., {customer-name}
- Primary key
  - Candidate key chosen by database designer as principal means of identifying tuples within relation

Foreign Keys

- A relation r_1 may include among its attributes the primary key of another relation, r_2
- This attribute is called foreign key from r_1 referencing r_2
- r_1 is called the referencing relation
- r_2 is called the referenced relation
- Example
  - loan-schema includes "branch-name" which is a primary key for branch-schema
  - therefore: branch-name is foreign key

Schema Diagram



Roadmap

- Relational Model
- Relational Schema
- Keys
- Schema Diagrams
- Relational Algrebra
  - Fundamental operators

Relational Algebra

- Procedural Query Language
- Fundamental Operators
- Unary
  - Select
  - Project
  - Rename
- Binary
  - Union
  - Set Difference
  - Cartesian-Product

Select Operator

- select operator selects tuples that satisfy given predicate

  $\sigma_{predicate}$ (relation)

- selection predicate:
  - comparisons: =, !=, <, <=, >, =>
  - combinations: and $\wedge$ or $\vee$ not $\neg$
- Example:
  - $\sigma_{amount>1200}$ (loan)

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

Project Operator

- project operator returns relation with attributes left out

  $\Pi_{attribute\text{-}list}$ (relation)

  - attribute-list $\subseteq$ relation
- Example:
  - $\Pi_{customer\text{-}name}$ (depositor)

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

Composition of Relational Operators

- Result of relational operator is a relation!
- Can arbitrary combine operators
- Relations are sets ➔ eliminate duplicate values
- Example:
  - $\Pi_{branch\text{-}name}$ ($\sigma_{amount>1200}$ (loan))

loan-number	branch-name	amount
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Redwood	1300
L-23	Redwood	2000

Union Operator

- Set operation: r U s
- Produces union of two sets

- For union operation to be valid between r and s:
  - 1. both relations must be of same arity
  - 2. domains of corresponding attributes must match
- Commutative operation
  - r U s = s U r

Set Difference Operator

- Set operation: r – s
- Allows us to find tuples that are in relation r, but not in s

- Relations must have same arity and matching attribute domains, as with the Union operator
- Non-commutative operation
  - r – s != s – r

Cartesian-Product Operator

- Cartesian product operator (x) allows us to combine information from any two relations
- Combine attribute-lists:
  - Relation r, schema R =(A, B, C)
  - Relation s, schema S =(C, D, E)
  - r x s, schema =(r.A, r.B, r.C, s.C, s.D, s.E) =(A, B, r.C, s.C, D, E)
- Cardinality of relation r = number of tuples in r
  - notation: cardinality(r) = |r|
- Cardinality of Cartesian product:
  - |r x s| = |r| * |s|

Rename Operator

- rename operator gives name to results

  $\rho_{new\text{-}name}$ (expression)

- If we also want to rename attributes to $A_1, A_2, \ldots, A_n$

  $\rho_{(A1,\ A2,\ \ldots,\ An)}$ (expression)

Relational Algebra (Formal Definition)

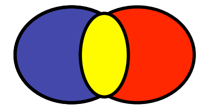
- A relational algebra expression is:
  - A relation in the database
  - A constant relation, e.g., {(A-101, Pgh, $20), (A-203, Oak, $5)}
  - E1 U E2
  - E1 – E2
  - E1 x E2
  - $\sigma_{predicate}$(E1)
  - $\Pi_{attr\text{-}list}$(E1)
  - $\rho_{new\text{-}name}$(E1)
    - where E1 and E2 are also relational-algebra expressions

Relational Operators

- Fundamental Operators
  - Select
  - Project
  - Union
  - Set Difference
  - Cartesian Product
  - Rename Operator
- **Additional Operators**
  - Set Intersection
  - Natural Join
  - Division
  - Assignment
- Extended Relational Operators
  - Generalized Projection
  - Aggregation
  - Aggregation with Grouping
  - Outer-Join

Set Intersection Operator

- Set operation: $r \cap s$
- Allows us to find tuples that are in both relations r and s



- Relations must have same arity and matching attribute domains, as with the Union operator
- $r \cap s = r - (r - s)$
- Question: is it commutative?

Natural-Join Operator $\bowtie$

- Forms a cartesian product of its two argument relations
- Performs selection, forcing equality on attributes that appear on both relations
- Removes duplicate attributes
- If r, s have no common attributes then $r \times s = r \bowtie s$

Relational Database Example

- **E**mployee (SSN, name, street, city)
- **W**orks (SSN, comp-name, salary)
- **C**ompany (comp-name, comp-city, state)
- **M**anages (SSN, manager-SSN)
- Note: we will sometimes use the initials of the relations instead of their full names

Example Queries /1

- Find all employees that live in "Pittsburgh"
  $\sigma_{\text{city = "Pittsburgh"}}$ (Employee)
- Find the names of all employees that live in Pittsburgh
  Π_{name} ($\sigma_{\text{city = "Pittsburgh"}}$ (Works))
- Find the names of all the employees that work for "Blockbuster"
  Π_{name} ($\sigma_{\text{comp-name = "Blockbuster"}}$ (Employee $\bowtie$ Works))
- OR, equivalently:
  Π_{name} (Employee $\bowtie$ $\sigma_{\text{comp-name = "Blockbuster"}}$ (Works))

Example Queries /2

- Find the names and cities of all the employees that work for "Blockbuster"
  $\Pi_{\text{name,city}}$ ($\sigma_{\text{comp-name = "Blockbuster"}}$ (Employee $\bowtie$ Works))
- Find the names, salaries and cities of all employees that work for "Blockbuster" and make over \$15,000
  $\Pi_{\text{name, salary, city}}$ ($\sigma_{\text{comp-name = "Blockbuster"} \wedge \text{salary > 15000}}$ (E$\bowtie$ W))
- Find the names of all employees who live in the same city as the company they work for
  Π_{name} ($\sigma_{\text{city = comp-city}}$ (Employee $\bowtie$ Works $\bowtie$ Company))
  - We would not need the selection, if comp-city was named city.

Θ-Join (condition-join)

- $\Theta = \{=, <, \leq, >, \geq, \neq\}$
- Θ-join of r(R) and s(S) on attributes r.A_i and s.A_j
  $r \infty_{r.Ai\ \theta\ s.Aj}\ s$
  $= \sigma_{r.Ai\ \theta\ s.Aj}(r \times s)$
- ≥-join of r(R) and s(S):
  $r \infty_{r.B \geq s.D}\ s$ = ?

relation **r**

A	B	C	D
1	2	3	4
2	3	3	5
2	4	6	8
1	2	6	8
2	6	6	8
8	2	3	4
2	4	3	4

relation **s**

C	D
3	4
6	8

Equi-Join

- Θ in join is = (equi-join)
- $r \infty_{r.Ai = s.Aj}\ s$
- equi-join of r(R) and s(S):
  $r \infty_{r.B = s.D}\ s$ = ?

relation **r**

A	B	C	D
1	2	3	4
2	3	3	5
2	4	6	8
1	2	6	8
2	6	6	8
8	2	3	4
2	4	3	4

relation **s**

C	D
3	4
6	8

Natural-Join

- Equi-join without duplicate columns
  r $*_P$ s
- P=list of attributes: P=R ∩ S
- r∗s =π $_{R\cup S}$ (r ∞ $_{r.P\ =\ s.P}$ s)
- r∗s = ?

relation **r**

A	B	C	D
1	2	3	4
2	3	3	5
2	4	6	8
1	2	6	8
2	6	6	8
8	2	3	4
2	4	3	4

relation **s**

C	D
3	4
6	8

Division

- Let r(R) and s(S) be relations such as S⊂R
- The division of r by s, denoted by r÷s,
  - is relation whose schema is Q=R-S and
  - includes all t such as $t_r[Q]$ = t and $t_r[S]$ = t

relation **r**

A	B	C	D
1	2	3	4
2	3	3	5
2	4	6	8
1	2	6	8
8	2	3	4
2	4	3	4

relation **s**

C	D
3	4
6	8

r÷s

A	B
1	2
2	4

Division Usage

- Query: "Retrieve the names of students who took *all* the classes that John took."

relation **r**

FN	LN	DP	No
S	A	CS	4
M	K	CS	5
K	L	EE	8
S	A	EE	8
F	S	CS	4
K	L	CS	4

relation **s**

DP	No
CS	4
EE	8

r÷s

FN	LN
S	A
K	L

Note:
Division can be expressed using π, x and – operations:
r÷s= π_Q(r) – π_Q((π_Q(r) x s) – r)

Division

- r ÷ s
- suitable for queries that include phrase "for all"
- Definition:
  - relation r, schema R = (A_1, A_2, …, A_m, B_1, B_2, …, B_n)
  - relation s, schema S = (B_1, B_2, …, B_n)
  - relation r ÷ s, schema Q = R – S = (A_1, A_2, …, A_m)
  - Tuple t in r ÷ s if
    - t in Π_Q(r), and
    - for every tuple u in s, tu is in r
- Alternative definition
  - r ÷ s = Π_Q(r) – Π_Q**(** (Π_Q(r) x s) – r **)**

Assignment Operator

- Assign parts of relational expression to temporary variables
- Assignment operation ←
- Works like assignment in programming languages
- Example:
  - tmp1 ← $\Pi_Q(r) \times s$
  - tmp2 ← $\Pi_Q(\text{tmp1} - r)$

Relational Operators

- Fundamental Operators
  - Select
  - Project
  - Union
  - Set Difference
  - Cartesian Product
  - Rename Operator
- Additional Operators
  - Set Intersection
  - Natural Join
  - Division
  - Assignment
- **Extended Operators**
  - Generalized Projection
  - Aggregation
  - Aggregation with Grouping
  - Outer-Join

Generalized Projection

- Extend projection operator by allowing arithmetic functions in the projection list
- General form:

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

- Example:
  - $\Pi_{\text{customer\_name, credit\_limit – balance}}$ (credit-info)
  - $\Pi_{\text{customer\_name, (credit\_limit – balance) } \textbf{as} \text{ available\_credit}}$ (credit-info)

Aggregate Functions

- Take collection of values and return single result
- Examples:
  - sum(), min(), max(), count(), avg()
- Notation:

$$\mathcal{G}_{\textbf{sum}(\text{salary})}\ (\text{employees})$$

- sets: no duplicates
- multisets: duplicates allowed

Aggregate Function Example

employee-name	branch-name	salary
Adams	Perryridge	40K
Brown	Perryridge	50K
Gopal	Perryridge	25K
Johnson	Downtown	35K
Loreena	Downtown	50K
Peterson	Downtown	40K
Rao	Austin	30K
Sato	Austin	20K

$\mathcal{G}_{\textbf{avg}(\text{salary})}$ (employees)

Sum of salary
36.25k

Aggregate Function Example – II

employee-name	branch-name	salary
Adams	Perryridge	40K
Brown	Perryridge	50K
Gopal	Perryridge	25K
Johnson	Downtown	35K
Loreena	Downtown	50K
Peterson	Downtown	40K
Rao	Austin	30K
Sato	Austin	20K

$\mathcal{G}_{\textbf{count-distinct}(\text{branch-name})}$ (employees)

Count of branch-name
3

Aggregation with Grouping

- Example:
  - total employee salaries per Branch
    - partition relation employees into groups
    - apply aggregate per group

employee-name	branch-name	salary
Adams	Perryridge	40K
Brown	Perryridge	50K
Gopal	Perryridge	25K
Johnson	Downtown	35K
Loreena	Downtown	50K
Peterson	Downtown	40K
Rao	Austin	30K
Sato	Austin	20K

$_{\text{branch-name}}\,\mathcal{G}_{\textbf{sum}(\text{salary})}$ (employees)

branch-name	salary
Perryridge	115K
Downtown	125K
Austin	50K

Outer Join

- Join selects only tuples satisfying the join condition
- *Outer Join*
  - **Left outer join** (r]∞ s) also keeps every tuple in first or left relation
  - **Right outer join** (r ∞[s) also keeps every tuple in second or right relation
  - **Full outer join** (r]∞[s) also keeps every tuple
- Attributes of tuples with no matching tuples are set to NULL

Outer Join

relation **r**

A	B	C
a	b	c
d	a	f
c	b	d

relation **s**

A	B	D
b	g	a
d	a	f

r]∞s

A	B	C	D
a	b	c	Null
d	a	f	f
c	b	d	Null

r ∞[s

A	B	C	D
d	a	f	f
b	g	Null	a

r]∞[s

A	B	C	D
a	b	c	Null
d	a	f	f
c	b	d	Null
b	g	Null	a

Relational Database Example

- **E**mployee (SSN, name, street, city)
- **W**orks (SSN, comp-name, salary)
- **C**ompany (comp-name, comp-city, state)
- **M**anages (SSN, manager-SSN)
- Note: we will sometimes use the initials of the relations instead of their full names

Example Queries /1

- Find all employees that live in "Pittsburgh"
  $\sigma_{\text{city = "Pittsburgh"}}$ (Employee)
- Find the names of all employees that live in Pittsburgh
  Π_{name} ($\sigma_{\text{city = "Pittsburgh"}}$ (Works))
- Find the names of all the employees that work for "Blockbuster"
  Π_{name} ($\sigma_{\text{comp-name = "Blockbuster"}}$ (Employee $\bowtie$ Works))
- OR, equivalently:
  Π_{name} (Employee $\bowtie$ $\sigma_{\text{comp-name = "Blockbuster"}}$ (Works))

Example Queries /2

- Find the names and cities of all the employees that work for "Blockbuster"
  $\Pi_{\text{name,city}}$ ($\sigma_{\text{comp-name = "Blockbuster"}}$ (Employee $\bowtie$ Works))
- Find the names, salaries and cities of all employees that work for "Blockbuster" and make over \$15,000
  $\Pi_{\text{name, salary, city}}$ ($\sigma_{\text{comp-name = "Blockbuster"} \wedge \text{ salary > 15000}}$ (E$\bowtie$ W))
- Find the names of all employees who live in the same city as the company they work for
  Π_{name} ($\sigma_{\text{city = comp-city}}$ (Employee$\bowtie$ Works$\bowtie$ Company))
  - We would not need the selection, if comp-city was named city.

Example Queries /3

- Find the names of all employees who live on the same street and city as their managers
  - Note: we will need to join employee (E), manager (M), and employee again to get the street, city info for the manager. For this we rename the second instance of employee as emp2

$$\Pi_{E.name}(\sigma_{manager\text{-}SSN = emp2.SSN \wedge E.city = emp2.city \wedge E.street = emp2.street}(E \bowtie M \bowtie \rho_{emp2}(E))$$

Example Queries /4

- Find the names of all people who do not work for "First Bank"

$$\Pi_{name}(\sigma_{comp\text{-}name \neq \text{"First Bank"}}(Employee \bowtie Works))$$

  - **Note**: the above assumes that all people are currently employed, i.e. have an entry in the Works relation. If we are not to assume this, the query should be written as follows:

$$\Pi_{name}(E) - \Pi_{name}(\sigma_{comp\text{-}name = \text{"First Bank"}}(E \bowtie W))$$

Example Queries /5

- Get the average salary of all employees

$$G_{avg(salary)}(Works)$$

- Get the average salary of all employees at "Blockbuster"

$$G_{avg(salary)}(\sigma_{comp\text{-}name = \text{"Blockbuster"}}(Works))$$

- Get the average salary of all employees per company:

$$_{comp\text{-}name}G_{avg(salary)}(Works)$$

Example Queries /6

- Find all the names of employees who live in Pittsburgh and make between $30K and $50K

$$\Pi_{name}(\sigma_{salary >= 30000 \wedge salary <= 50000 \wedge city = \text{"PGH"}}(E \bowtie W))$$

- Find the names of the managers of all employees who live in Pittsburgh and make between $30K and $50K

$$\mathbf{t} \leftarrow \Pi_{mgr\text{-}SSN}(\sigma_{salary>=30K \wedge salary<=50K \wedge city=\text{"PGH"}}((E \bowtie W)) \bowtie M$$

$$\Pi_{name}(\sigma_{SSN = manager\text{-}SSN}(\mathbf{t} \bowtie E))$$