Chapter 2: Relational Model

Chapter 2: Relational Model

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database

Basic Structure

- **Relational database:** a set of **relations**
- **Relation:** a named data table consisting of two parts:
  - **Schema:** specifies name of relation, consists of a list of attributes and type of each attribute (domains).
    - E.g., Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real).
  - **Instance:** a table of tuples (or called *records, rows*) and attributes (or called *fields, columns*).

Example of a Relation

attributes
(or columns)

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

tuples
(or rows)

Attribute Types

- Each attribute of a relation has a name
- **Domain** of the attribute: The set of allowed values for each attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
  - E.g. the value of an attribute can be an account number, but cannot be a set of account numbers
- Domain is said to be atomic if all its members are atomic
- The special value *null* :
  - Signifies that the value is unknown or does not exist
  - A member of every domain
- The null value causes complications in the definition of many operations
  - We shall ignore the effect of null values in our main presentation and consider their effect later

Relation Schema

- A_1, A_2, …, A_n are *attributes*

- $R = (A_1, A_2, \dots, An)$ is a *relation schema*

  Example:

  Customer_schema = (*customer_name, customer_street, customer_city*)

- $r(R)$ denotes a *relation* r on the *relation schema* R

  Example:

  customer (Customer_schema)

Relation Instance

- The current values (*relation instance*) of a relation are specified by a table
- An element *t* of *r* is a *tuple*, represented by a *row* in a table

attributes
(or columns)

customer_name	*customer_street*	customer_city
Jones	Main	Harrison
Smith	North	Rye
Curry	North	Rye
Lindsay	Park	Pittsfield

tuples
(or rows)

customer

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *account* relation with unordered tuples

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

Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information

  account : stores information about accounts

  depositor : stores information about which customer owns which account

  customer : stores information about customers

- Storing all information as a single relation such as
  bank(*account_number, balance, customer_name*, ..)
  results in
  - repetition of information
    - e.g.,if two customers own an account (What gets repeated?)
  - the need for null values
    - e.g., to represent a customer without an account
- Normalization theory (Chapter 7) deals with how to design relational schemas

The *customer* Relation

customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

The *depositor* Relation

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

Keys

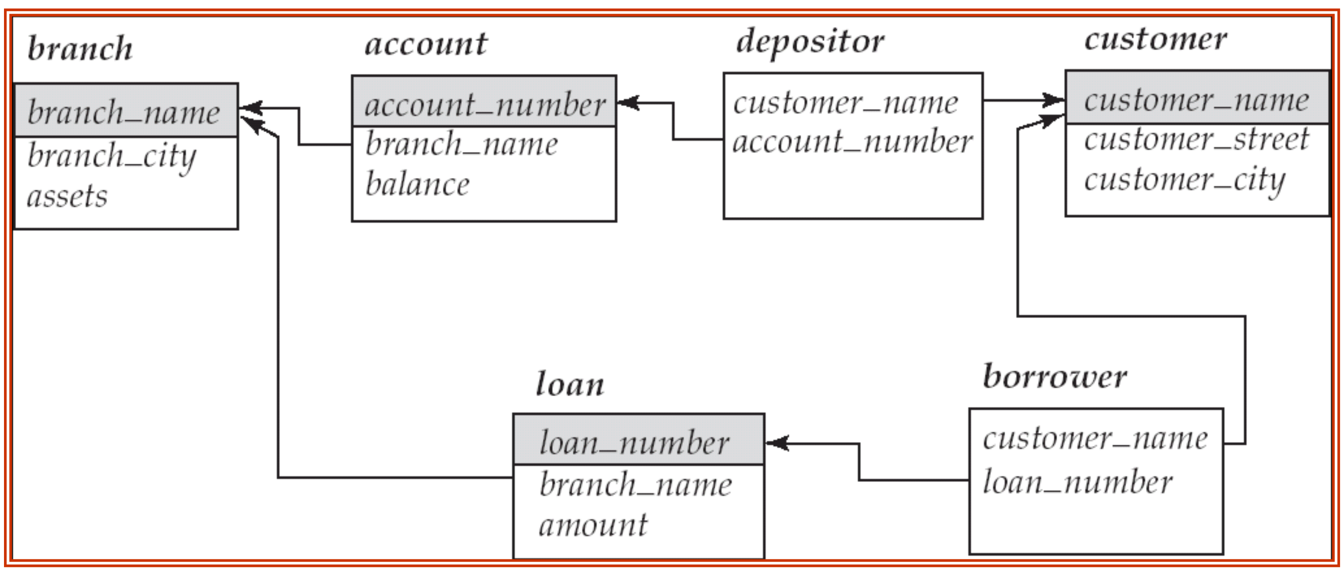
- A set of attribute *K* is a **superkey** of *R* if:
  - No two tuples can have same values in all these attributes
- Example: Customer(*customer_name, customer_street, customer_city*)
  - {*customer_name, customer_street*} and {*customer_name*} are both superkeys
  - What about *name*?
  - PS: In real life, an attribute such as *customer_id* would be used instead of *customer_name* to uniquely identify customers, but we omit it to keep our examples small, and instead assume customer names are unique.

Keys (Cont.)

- *K* is a **candidate key** if *K* is minimal
  Example: {*customer_name*} is a candidate key for *Customer*, since it is a superkey and no subset of it is a superkey.
- **Primary key:** If there's more than one candidate keys, one is chosen as the primary key
  - Should choose an attribute whose value never, or very rarely, changes.
  - E.g., email address is unique, but may change

Foreign Keys

- **Foreign key:** A relation schema may have an attribute that corresponds to the primary key of another relation.
  - E.g. *customer_name* and *account_number* attributes of *depositor* are foreign keys to *customer* and *account* respectively.
  - Can refer to itself
  - Only values occurring in the primary key attribute of the **referenced relation** may occur in the foreign key attribute of the **referencing relation.**
- **Schema diagram**



Review of last lecture

- Concepts:
  - Schema
  - Table
  - Relation
  - Attribute
  - Domain
  - Super key
  - Candidate key
  - Primary key

Class Exercise

- 1. Given a relation r defined over the schema R, which of the following can always uniquely identify the tuples in r?

  A. any non-null attributes of R

  B. super key of R

  C. the first attribute in R

  D. R itself

- 2. Given the following relation, list all candidate keys and superkeys.

A	B	C	D
A1	B1	C1	D1
A1	B2	C2	D1
A2	B1	C2	D1

Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - Procedural
  - Non-procedural, or declarative
- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- Pure languages form underlying basis of query languages that people use.

Chapter 2: Relational Model

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database

Role of Relational Algebra

- How does a relational DBMS work?
  - Queries are expressed by users in a language, e.g. SQL;
  - The DBMS translates an SQL query into relational algebra, and meanwhile looks for other algebra expressions that produce the same answers but saving the computational costs.
  - Based on the relational algebra, DBMS calculates the query results.

Relational Algebra

- Procedural language
- Six basic operators
  - select: σ
  - project: Π
  - 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.

Instance Example

ID	name	dept_name	salary
100	Kart	CS	65000
101	Crick	History	90000
102	Kim	Finance	60000
103	Wu	Physics	72000
104	John	CS	80000

instructor

Select Operation

- Notation: $\sigma_p(\boldsymbol{r})$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

<attribute> op <attribute> or <constant>

where op is one of: $=, \neq, >, \geq, <, \leq$

- Example of selection:

$$\sigma_{dept\_name=\text{“}Physics\text{”}}(instructor)$$

Select Operation

ID	name	dept_name	salary
101	Crick	History	90000

instructor

Salary greater than 80,000

$\sigma_{salary>80000}(instructor)$

Select Operation

ID	name	dept_name	salary
~~100~~	~~Kart~~	~~CS~~	~~65000~~
~~101~~	~~Crick~~	~~History~~	~~90000~~
~~102~~	~~Kim~~	~~Finance~~	~~60000~~
~~103~~	~~Wu~~	~~Physics~~	~~72000~~
104	John	CS	80000

instructor

In department CS with salary greater than 70000

$$\sigma_{dept\_name=\text{“}CS\text{”} \wedge\ salary>70000}(\text{instructor})$$

Select Operation

ID	name	dept_name	salary
100	Kart	CS	65000
101	Crick	History	90000
102	Kim	Finance	60000
~~103~~	~~Wu~~	~~Physics~~	~~72000~~
~~104~~	~~John~~	~~CS~~	~~80000~~

instructor

salary greater than 85,000 or less than 70,000

$$\sigma_{salary>85000 \lor salary<70000}(\text{instructor})$$

Project Operation

- Notation:

$$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows are removed from result, since relations are sets
- Example: To eliminate the *dept_name* attribute of *account*

$$\Pi_{ID, name, salary}(instructor)$$

Project Operation

ID	name	dept_name	salary
100	Kart	CS	65000
101	Crick	History	90000
102	Kim	Finance	60000
103	Wu	Physics	72000
104	John	CS	80000

instructor

Remove the department attribute

$\pi_{ID,\, name, salary}(\text{instructor})$

Project Operation

ID	name	salary
100	Kart	65000
101	Crick	90000
102	Kim	60000
103	Wu	72000
104	John	80000

instructor

Remove the department attribute

$\pi_{ID,\, name, salary}(\text{instructor})$

Project Operation

ID	name	dept_name	salary
100	Kart	CS	60000
101	Crick	History	90000
102	Kim	Finance	60000
103	Wu	Physics	72000
104	John	CS	80000

instructor

Only remain the dept_name attribute

$\pi_{dept\_name}(\text{instructor})$

Project Operation

ID	name	dept_name	salary
100	Kart	CS	65000
101	Crick	History	90000
102	Kim	Finance	60000
103	Wu	Physics	72000
104	John	CS	80000

instructor

Only remain the dept_name attribute

$\pi_{dept\_name}(\text{instructor})$

Project Operation

dept_name
CS
History
Finance
Physics

instructor

Only remain the dept_name attribute
(duplicate rows are removed)

$\pi_{dept\_name}(\text{instructor})$

Operator composition

ID	name	dept_name	salary
100	Kart	CS	60000
101	Crick	History	90000
102	Kim	Finance	60000
103	Wu	Physics	72000
104	John	CS	80000

instructor

List the name of instructor in CS

$$\pi_{name}(\sigma_{dept\_name=\text{“}cs\text{”}}(\text{instructor}))$$

Operator composition

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$$\pi_{sname, rating}(\sigma_{rating>8}(S2))$$

sname	rating
yuppy	9
rusty	10

Union Operation

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t | t \in r \; or \; t \in s\}$$

- For $r \cup s$ to be valid:
  - r, s must have the *same* **arity** (same number of attributes)
  - The attribute domains must be **compatible** ('corresponding' attributes have the same type)
- Example: to find all customers with either an account or a loan

$$\prod_{customer\_name}(depositor) \cup \prod_{customer\_name}(borrower)$$

Union Operation – Example

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$S_1 \cup S_2$

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

Set Difference Operation

- Notation: $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity (same number of attributes)
  - attribute domains of r and s must be compatible

Set Difference Operation – Example

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$$S1 - S2$$

sid	sname	rating	age
22	dustin	7	45.0

Cartesian-Product Operation

- Notation $r \times s$
- Output all pairs of rows from the two input relations
- If 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
  - renaming must be used.

Cartesian-Product Operation – Example

- Each row of r is paired with each row of s.
- Relations r, s:

r

A	B
α	1
β	2

s

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

- $r \times s$:

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

Cartesian-Product Operation – Example

- Each row of S1 is paired with each row of R1.

$S1 \times R1$

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1.sid	sname	rating	age	R1.sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96

Composition of Operations

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

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

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

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

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:
  - $\rho_X(E)$: the expression E under the name X
  - $\rho_{X(A_1, A_2, \ldots, A_n)}(E)$: expression E under the name X, and with the

  attributes renamed to $A_1, A_2, \ldots, A_n$.

Rename Operation

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

$\rho_{My\text{-}table(id,\ name,\ level,\ age)}$ (S1)

My-table

id	name	level	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Formal Definition: relational-algebra expressions

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
    - A constant relation is written by listing its tuples within { }
    - E.g, { (22222, Einstein, Physics, 95000), (76543, Singh, Finance, 80000) }
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_S(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_X(E_1)$, X is the new name for the result of E_1

Banking Example

- *branch (branch_name, branch_city, assets)*
- *customer (customer_name, customer_street, customer_city)*
- *account (account_number, branch_name, balance)*
- *loan (loan_number, branch_name, amount)*
- *depositor (customer_name, account_number)*
- *borrower (customer_name, loan_number)*

Example Queries

branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
account (account_number, branch_name, balance)
loan (loan_number, branch_name, amount)
depositor (customer_name, account_number)
borrower (customer_name, loan_number)

- Find all loans of over $1200
  - $\sigma_{amount>1200}(loan)$
- Find the loan number for each loan of an amount greater than $1200
  - $\prod_{loan\_number}(\sigma_{amount>1200}(loan))$

Example Queries

branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
account (account_number, branch_name, balance)
loan (loan_number, branch_name, amount)
depositor (customer_name, account_number)
borrower (customer_name, loan_number)

- Find the names of all customers who have a loan at the Perryridge branch.
  - Query 1

$$\Pi_{\text{customer\_name}} (\sigma_{\text{branch\_name = "Perryridge"}} (\sigma_{\text{borrower.loan\_number = loan.loan\_number}} (\text{borrower} \times \text{loan})))$$

Example Queries

loan

loan_number	branch_name	amount
0001	Perryridge	1000
0002	Downtown	500

borrower

customer_name	loan_number
Mark	0001
Angel	0002

loan × borrower

loan.loan_number	branch_name	amount	customer_name	borrower.loan_number
0001	Perryridge	1000	Mark	0001
0002	Downtown	500	Mark	0001
0001	Perryridge	1000	Angel	0002
0002	Downtown	500	Angel	0002

Example Queries

branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
account (account_number, branch_name, balance)
loan (loan_number, branch_name, amount)
depositor (customer_name, account_number)
borrower (customer_name, loan_number)

- Find the names of all customers who have a loan at the Perryridge branch.
  - Query 1

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

  - Query 2

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

Example Queries

loan

loan_number	branch_name	amount
0001	Perryridge	1000
0002	Downtown	500

borrower

customer_name	loan_number
Mark	0001
Angel	0002

loan × borrower

loan.loan_number	branch_name	amount	customer_name	borrower.loan_number
0001	Perryridge	1000	Mark	0001
0001	Perryridge	1000	Angel	0002

Example Queries

branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
account (account_number, branch_name, balance)
loan (loan_number, branch_name, amount)
depositor (customer_name, account_number)
borrower (customer_name, loan_number)

- Find the names of all customers who have a loan at the Perryridge branch but do not have deposit at any branch of the bank.

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

$$(\sigma_{borrower.loan\_number = loan.loan\_number}(\text{borrower} \times \text{loan}))) \; -$$

$$\Pi_{customer\_name}(\text{depositor})$$

Example Queries

branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
account (account_number, branch_name, balance)
loan (loan_number, branch_name, amount)
depositor (customer_name, account_number)
borrower (customer_name, loan_number)

- Find the <u>largest</u> account balance
  - Strategy:
    - Find those balances that are *not* the largest
      - Rename *account* relation as *d* so that we can compare each account balance with all others
    - Use set difference to find those account balances that were *not* found in the earlier step.
  - The query is:

$$\Pi_{balance}(account) - \Pi_{account.balance}$$

$$(\sigma_{account.balance < d.balance}\ (account \times \rho_d\ (account)))$$

Account

Account _number	Branch_ name	balance
1	A	50
2	B	100
3	B	70

d

Account _number	Branch_ name	balance
1	A	50
2	B	100
3	B	70

$account \times d$

Account. Account_ number	Account .Branch _name	Account .balance	d.accoun t_numer	d.Branc h_name	d.balanc e
1	A	50	1	A	50
1	A	50	2	B	100
1	A	50	3	B	70
2	B	100	1	A	50
2	B	100	2	B	100
2	B	100	3	B	70
3	B	70	1	A	50
3	B	70	2	B	100
3	B	70	3	B	70

Chapter 2: Relational Model

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database

Additional Operations

- Set intersection
- Natural join
- Division
- Assignment

- These operations can be transformed to basic operations.
- They do not add any power to relational algebra, but can simplify queries.

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- 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

- $r \cap s$

A	B
α	2

Natural-Join Operation

- $R = (A, B, C, D)$, $S = (E, B, D)$
- Equal on ***all common*** attributes
  - $r \bowtie s = \prod_{r.A,\, r.B,\, r.C,\, r.D,\, s.E}(\sigma_{r.B = s.B \wedge r.D = s.D}(r \times s))$
- Result schema = (A, B, C, D, E)

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

- Notation: $r \div s$
- Suited to queries that include the phrase "for all".
- r and s: relations on schemas R and S respectively, where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$
- The result of $r \div s$ is a relation on schema:

  $R - S = (A_1, \ldots, A_m)$

  $$r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s \ (tu \in r) \}$$

  Where *tu* means the concatenation of tuples *t* and *u* to produce a single tuple

Division Example

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

Division Example Cont.

- 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

Division Operation (Cont.)

- Property
  - Let $q = r \div s$
  - Then q is the largest relation satisfying $q \times s \subseteq r$
- Definition in terms of the basic algebra operation
  Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

To see why

- $\Pi_{R-S,S}(r)$ simply reorders attributes of r
- $\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ gives those tuples t in

  $\Pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.
- Example: Write $r \div s$ as

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

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
  - May use variable in subsequent expressions.

Bank Example Queries

branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
account (account_number, branch_name, balance)
loan (loan_number, branch_name, amount)
depositor (customer_name, account_number)
borrower (customer_name, loan_number)

- Find the names of all customers who have a loan and deposit at bank.

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

- Find the name of all customers who have a loan at the bank and the loan amount

$$\Pi_{customer\_name,\ loan\_number,\ amount}\,(borrower \bowtie loan)$$

Bank Example Queries

branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
account (account_number, branch_name, balance)
loan (loan_number, branch_name, amount)
depositor (customer_name, account_number)
borrower (customer_name, loan_number)

- Find all names of customers who have an account from at least the "Downtown" and the "Uptown" branches.
  - Query 1

$$\Pi_{customer\_name}\,(\sigma_{branch\_name = \text{“Downtown”}}\,(depositor \bowtie account\,))$$
$$\cap\ \Pi_{customer\_name}\,(\sigma_{branch\_name = \text{“Uptown”}}\,(depositor \bowtie account))$$

  - Query 2

$$\Pi_{customer\_name,\ branch\_name}\,(depositor \bowtie account\,)$$
$$\div\ \rho_{temp(branch\_name)}\,(\{(\text{“Downtown”}\,),\ (\text{“Uptown”}\,)\})$$

Note that Query 2 uses a constant relation.

Class Exercise

- BRANCH(*brh-name*, *city*)
- ACC(*acc-id*, *cust-name*, *brh-name*)
  - Assume that no two customers have the same name.

- Find all customers who have accounts at all branches in HK.
- Hint: use division

Class Exercise

- BRANCH(*brh-name*, *city*)
- ACC(*acc-id*, *cust-name*, *brh-name*)
  - Assume that no two customers have the same name.

- Find all customers who have accounts at all branches in HK.
- A wrong solution
  - $\Pi_{\textit{cust-name}} ((\text{ACC} \div \Pi_{\textit{brh-name}}(\sigma_{\textit{city} = \text{'HK'}}(\text{BRANCH})))$

- A correct solution
  - $\Pi_{\textit{cust-name, brh-name}}(\text{ACC}) \div \Pi_{\textit{brh-name}}(\sigma_{\textit{city} = \text{'HK'}}(\text{BRANCH}))$

Class Exercise

ACC

acc-id	cust-name	brh-name
0001	Mark	Kowloon
0001	Mark	Central
0002	Angel	Kowloon
0003	Angel	Central

BRANCH

brh-name	city
Kowloon	HK
Central	HK

Chapter 2: Relational Model

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database

Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions
- Outer Join

Generalized Projection

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

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

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

$$\Pi_{\textit{customer\_name, limit – credit\_balance}}\ (\textit{credit\_info})$$

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result. Duplicates are not eliminated.

  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

  - $G_1, G_2 \ldots, G_n$ 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 *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $\mathcal{G}_{\mathbf{sum(c)}}\,(\mathrm{r})$

sum(*c*)
27

- $\mathcal{G}_{\mathbf{count(c)}}\,(\mathrm{r})$

count(*c*)
4

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

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

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

Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

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

Outer Join

- An extension of the join operation that avoids loss of information.
- Compute the join and then add tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.
    - We shall study precise meaning of comparisons with nulls later

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

- Join

  loan ⋈ *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

Chapter 2: Relational Model

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values (as in SQL) except count
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)

Null Values

- Comparisons with null values return the special truth value: *unknown*
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
(*unknown* **or** *false*) = *unknown*
(*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown,*
(*false* **and** *unknown*) = *false,*
(*unknown* **and** *unknown*) = *unknown*
  - NOT*:* (**not** *unknown*) = *unknown*
  - In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	3000	*null*
L-155	*null*	*null*	Hayes

- $\mathcal{G}_{\textbf{sum}(amount)}(r)$
- $\mathcal{G}_{\textbf{count}(amount)}(r)$

Chapter 2: Relational Model

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database

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

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

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

Deletion Examples

branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
account (account_number, branch_name, balance)
loan (loan_number, branch_name, amount)
depositor (customer_name, account_number)
borrower (customer_name, loan_number)

- Delete all account records in the Perryridge branch.

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

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

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

- Delete all accounts at branches located in Needham.

  $$r_1 \leftarrow \sigma_{branch\_city = \text{“Needham”}}(account \bowtie branch)$$

  $$r_2 \leftarrow \Pi_{account\_number, branch\_name, balance}(r_1)$$

  $$r_3 \leftarrow \Pi_{customer\_name, account\_number}(r_2 \bowtie \text{depositor})$$

  $$account \leftarrow \text{account} - r_2$$

  $$depositor \leftarrow \text{depositor} - r_3$$

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:

$$r \leftarrow r \cup E$$

  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.

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

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

- Provide as a gift for all loan customers in the Perryridge branch, a \$200 savings account. Let the loan number serve as the account number for the new savings account.

$$r_1 \leftarrow (\sigma_{branch\_name\ =\ \text{“Perryridge”}}(borrower \bowtie \text{loan}))$$

$$account \leftarrow account \cup \Pi_{loan\_number,\ branch\_name,\ 200}\,(r_1)$$

$$\text{depositor} \leftarrow depositor \cup \Pi_{customer\_name,\ loan\_number}\,(r_1)$$

Updating

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

$$r \leftarrow \prod_{F_1, F_2, \ldots, F_l,} (r)$$

- 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%.

$$account \leftarrow \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (account)$$

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

$$account \leftarrow \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.06}\ (\sigma_{\,BAL\ >\ 10000}\ (account\ ))$$
$$\cup\ \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (\sigma_{BAL\ \leq\ 10000}\ (account))$$

End of Chapter 2