Chapter 2: Relational Model

Database System Concepts, 5th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use

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 of a Relation

Account Relation		
branch_name	**account_no**	**balance**
College road	A-111	10000
C.G. Road	A-211	20000
M.G. Road	A-235	32500
Ashram Road	A-425	2500
City Station	A-421	15420
Central City	A-752	25634
Ring road	A-524	242516

Depositor Relation	
Customer_name	**account_no**
Amit	111
Amir	211
Leena	235
Himanshu	425
Azahar	421
Sachin	752
Priyanka	524

Branch Relation		
branch_name	**branch_city**	**assets**
College road	Nadiad	9000000
C.G. Road	Ahmedabad	2100000
M.G. Road	Surat	1700000
Ashram Road	Vadodara	400000
City Station	Vapi	8000000
Central City	Gandhinagar	300000
Maninagar	Jamnagar	3700000
Ring road	Ahmedabad	7100000
Mansarovar	Ahmedabad	2500000

©Silberschatz, Korth and Sudarshan

Example of a Relation

Customer Relation		
customer_name	**customer_street**	**customer_city**
Amit	Main	Nadiad
Suresh	North	Ahmedabad
Leena	Park	Surat
Amita	Putnam	Vadodara
Azahar	Nassau	Vapi
Sachin	Senator	Gandhinagar
Yuvraj	Sand Hill	Jamnagar
Amir	North	Ahmedabad
Priyanka	North	Ahmedabad
Sulekha	Senator	Gandhinagar
Himanshu	Putnam	Vadodara
Anjum	Main	Nadiad

Borrower Relation	
customer_name	**loan_no**
Amit	L-11
Amir	L-23
Leena	L-15
Himanshu	L-14
Azahar	L-93
Sachin	L-11
Priyanka	L-16

Loan Relation		
branch_name	**loan_no**	**balance**
College road	L-11	10000
C.G. Road	L-23	20000
M.G. Road	L-15	32500
Ashram Road	L-14	2500
City Station	L-93	15420
Central City	L-11	25634
Ring road	L-16	242516

Basic Structure

- Table , Attributes , Domain (permitted values) D.
- Formally, given sets D_1, D_2, …. D_n a **relation** *r* is a subset of

  $D_1 \times D_2 \times \ldots \times D_n$

  Thus, a relation is a set of *n*-tuples (a_1, a_2, …, a_n) where each $a_i \in D_i$
- Example: If
  - *customer_name* = {Jones, Smith, Curry, Lindsay, …}

    /* Set of all customer names */
  - *customer_street* = {Main, North, Park, …} /* set of all street names*/
  - *customer_city* = {Harrison, Rye, Pittsfield, …} /* set of all city names */

  Then *r* = { (Jones, Main, Harrison),

  (Smith, North, Rye),

  (Curry, North, Rye),

  (Lindsay, Park, Pittsfield) }

  is a relation over

  customer_name x *customer_street* x *customer_city*

©Silberschatz, Korth and Sudarshan

Attribute Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the 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* is 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 (Database) Schema

- $A_1, A_2, \ldots, A_n$ are *attributes*

- $R = (A_1, A_2, \ldots, A_n)$ 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)

©Silberschatz, Korth and Sudarshan

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

©Silberschatz, Korth and Sudarshan

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

©Silberschatz, Korth and Sudarshan

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

©Silberschatz, Korth and Sudarshan

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

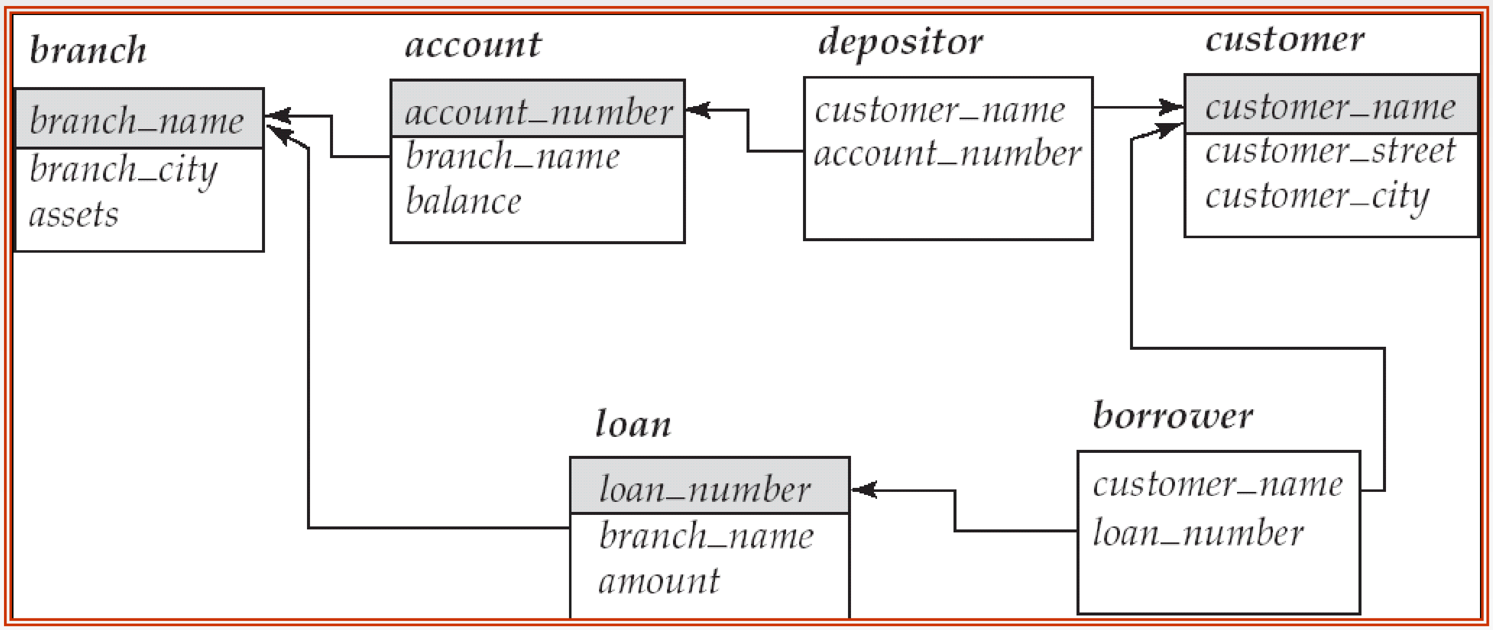
Keys

- Let $K \subseteq R$
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
- *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:** a candidate key chosen as the principal means of identifying tuples within a relation
  - Should choose an attribute whose value never, or very rarely, changes.
  - E.g. email address is unique, but may change

Foreign Keys

- A relation schema may have an attribute that corresponds to the primary key of another relation. The attribute is called a **foreign key**.
  - E.g. *customer_name* and *account_number* attributes of *depositor* are foreign keys to *customer* and *account* respectively.
  - 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**



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.

Relational Algebra

- It is a Procedural Query Language
- Six basic operators
  - **select: σ** **(unary operator)**
  - **project: $\prod$** **(unary operator)**
  - **union: $\cup$** **(binary operator)**
  - **set difference: –** **(binary operator)**
  - **Cartesian product: x** **(binary operator)**
  - **rename: ρ** **(unary operator)**
- Other Operations: **set intersection, natural join, division and assignment**
- The operators take one or two relations as inputs and produce a new relation as a result.

Select 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

©Silberschatz, Korth and Sudarshan

Select Operation

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

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

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:**

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

Select Operation – Example

The same table **E** (for **EMPLOYEE**) as above.

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td>select *
from E
where salary < 200</td>
<td><table><tr><th>nr</th><th>name</th><th>salary</th></tr><tr><td>1</td><td>John</td><td>100</td></tr><tr><td>7</td><td>Tom</td><td>100</td></tr></table></td>
<td>$\textbf{SELECT}_{\text{salary} < 200}(\text{E})$</td>
</tr>
<tr>
<td>select *
from E
where salary < 200
and nr >= 7</td>
<td><table><tr><th>nr</th><th>name</th><th>salary</th></tr><tr><td>7</td><td>Tom</td><td>100</td></tr></table></td>
<td>$\textbf{SELECT}_{\text{salary} < 200 \text{ and nr} >= 7}(\text{E})$</td>
</tr>
</table>

Project 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

©Silberschatz, Korth and Sudarshan

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 removed from result, since relations are sets
- **Example: To eliminate the *branch_name* attribute of *account***

$$\Pi_{\textit{account\_number, balance}}\ (\textit{account})$$

Project Operation – Example

Example: The table **E** (for **EMPLOYEE**)

nr	**name**	**salary**
1	John	100
5	Sarah	300
7	Tom	100

SQL	**Result**	**Relational algebra**
`select salary from E`	**salary** 100 300	**PROJECT**$_{salary}$(E)
`select nr, salary from E`	**nr** | **salary** 1 | 100 5 | 300 7 | 100	**PROJECT**$_{nr,\ salary}$(E)

Union Operation – Example

- Relations *r, s:*

r

A	*B*
α	*1*
α	*2*
β	*1*

s

A	*B*
α	*2*
β	*3*

- r $\cup$ s:

A	*B*
α	*1*
α	*2*
β	*1*
β	*3*

©Silberschatz, Korth and Sudarshan

Union Operation

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

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.

1. *r, s* must have the *same* **arity** (same number of attributes)
2. The attribute domains must be **compatible** (example: 2^{nd} column of *r* deals with the same type of values as does the 2^{nd} column of *s*)

- **Example: to find all customers with either an account or a loan**

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

©Silberschatz, Korth and Sudarshan

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

©Silberschatz, Korth and Sudarshan

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**
  - attribute domains of *r* and *s* must **be compatible**

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

Cartesian-Product Operation

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

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

- 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

Example: The table **E** (for **EMPLOYEE**)

enr	ename	dept
1	Bill	A
2	Sarah	C
3	John	A

Example: The table **D** (for **DEPARTMENT**)

dnr	dname
A	Marketing
B	Sales
C	Legal

SQL	Result	Relational algebra
`select * from E, D`	see table below	E **X** D

enr	ename	dept	dnr	dname
1	Bill	A	A	Marketing
1	Bill	A	B	Sales
1	Bill	A	C	Legal
2	Sarah	C	A	Marketing
2	Sarah	C	B	Sales
2	Sarah	C	C	Legal
3	John	A	A	Marketing
3	John	A	B	Sales
3	John	A	C	Legal

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:

$$\rho_X(E)$$

returns the expression E under the name X

- If a relational-algebra expression E has arity n, then

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression E under the name X, and with the attributes renamed to $A_1, A_2, \ldots., A_n$.

©Silberschatz, Korth and Sudarshan

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.

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

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

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

Example Queries

- 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

- Find the largest 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:

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

©Silberschatz, Korth and Sudarshan

Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- 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
  - $\Pi_S(E_1)$, 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

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

©Silberschatz, Korth and Sudarshan

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

- Notation: r ⋈ s
- Natural join is a binary operator that is written as (*R* *S*) where *R* and *S* are relations. The result of the natural join is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names.
- Example:

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

  S = (*E, B, D*)

  - Result schema = (*A, B, C, D, E*)
  - *r* ⋈ *s* is defined as:

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

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

©Silberschatz, Korth and Sudarshan

Natural Join Operation – Example

For an example consider the tables *Employee* and *Dept* and their natural join:

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Employee $\bowtie$ *Dept*

Name	EmpId	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

©Silberschatz, Korth and Sudarshan

THETA JOIN (θ-Join)

- General form

  $$R \bowtie_{\theta} S$$

  where
  - R, S are relations,
  - F is a Boolean expression, called a join condition.
- A derivative of Cartesian product
  - $R \bowtie_{\theta} S = \sigma_{\theta} (R \times S)$
- $R(A_1, A_2, \ldots, A_m, B_1, B_2, \ldots, B_n)$ is the resulting schema of a θ-Join over R_1 and R_2:

  $$R_1(A_1, A_2, \ldots, A_m) \bowtie_{\theta} R_2 (B_1, B_2, \ldots, B_n)$$

Theta Join Operation – Example

Consider tables *Car* and *Boat* which list models of cars and boats and their respective prices. Suppose a customer wants to buy a car and a boat, but she doesn't want to spend more money for the boat than for the car. The θ-join on the relation *CarPrice* ≥ *BoatPrice* produces a table with all the possible options.

Car

CarModel	CarPrice
CarA	20'000
CarB	30'000
CarC	50'000

Boat

BoatModel	BoatPrice
Boat1	10'000
Boat2	40'000
Boat3	60'000

$$Car \underset{CarPrice \geq BoatPrice}{\bowtie} Boat$$

CarModel	CarPrice	BoatModel	BoatPrice
CarA	20'000	Boat1	10'000
CarB	30'000	Boat1	10'000
CarC	50'000	Boat1	10'000
CarC	50'000	Boat2	40'000

Equi Join Operation

In case the operator θ is the equality operator (=) then this join is also called an equijoin. <u>Example</u>: Given the two sample relational instances S1 and R1

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

Figure 4.1 Instance $S1$ of Sailors

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

Figure 4.3 Instance $R1$ of Reserves

The operator S1 $\bowtie_{\text{R.sid=Ssid}}$ R1 yields

sid	*sname*	*rating*	*age*	*bid*	*day*
22	Dustin	7	45.0	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

Figure 4.13 $S1 \bowtie_{R.sid=S.sid} R1$

©Silberschatz, Korth and Sudarshan

Division Operation

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

  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

©Silberschatz, Korth and Sudarshan

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

©Silberschatz, Korth and Sudarshan

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

©Silberschatz, Korth and Sudarshan

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 $u \in s$, $tu \notin r$.

 ©Silberschatz, Korth and Sudarshan

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

- Find the names of all customers who have a loan and an account 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

- Find all 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.

Bank Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.

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

©Silberschatz, Korth and Sudarshan

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, …, 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:

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

 ©Silberschatz, Korth and Sudarshan

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
  - $G_1, G_2 \ldots, G_n$ 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

- $\mathcal{g}_{\text{sum(c)}}(\text{r})$

sum(*c*)
27

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 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.
- Computes the join and then adds tuples form 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*

©Silberschatz, Korth and Sudarshan

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

©Silberschatz, Korth and Sudarshan

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)
- 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*

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.

 ©Silberschatz, Korth and Sudarshan

Deletion Examples

- 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

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

- 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.

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

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

- 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.

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

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

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

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 percent.

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

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

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

©Silberschatz, Korth and Sudarshan