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

- Formally,
  given sets $D_1, D_2,$ ¡. D_n a *relation* r is a subset of $D_1 \times D_2 \times$ ¡ $\times D_n$

  $R \subseteq D_1, \times$ ¡.. $\times D_n$ (n: degree of R)

  or

  $R = \{ <d_1, ..., d_n> \mid d_1 \in D_1, ..., d_n \in D_n \}$ (set of tuples)

- Example: *customer-name* = {Jones, Smith, Curry, Lindsay}

  customer-street = {Main, North, Park}

  customer-city = {Harrison, Rye, Pittsfield}

  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*

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. multivalued attribute values are not atomic
  - E.g. composite attribute values are not 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 Schema

- A_1, A_2, ¡, A_n are *attributes*
- $R = (A_1, A_2,$ ¡, $A_n)$ is a *relation schema*

  E.g.   *Customer-schema* = (*customer-name, customer-street, customer-city*)
- $r(R)$ is a *relation* (*variable*) on the *relation schema R*

  E.g.   *customer(Customer-schema)*

Relational Algebra

- Algebra : operators and operands
  - Relational algebra
    - operands : relations
    - operators : basic operators (+ additional operations)
      - take two or more relations as inputs and give a new relation as a result.
- Procedural language
- 6 Fundamental Operators
  - select
  - project
  - union
  - set difference
  - Cartesian product
  - rename

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

$$\Pi_{loan\text{-}number}\ (\sigma_{amount > 1200}\ (loan))$$

Example Queries

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

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

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

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

Example Queries

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

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=¡\ Perryridge¡}$$
$$(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower \times loan)))$$

- 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.

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name\ =\ ¡\ Perryridge¡}$$
$$(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower \times loan)))$$

$$?\quad \Pi_{customer\text{-}name}(depositor)$$

Example Queries

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

Query 1

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name\ =\ \text{¡ Perryridge¡}}$$
$$(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower \times loan)))$$

Query 2

$$\Pi_{customer\text{-}name}($$
$$(\sigma_{loan.loan\text{-}number\ =\ borrower.loan\text{-}number}($$
$$(\ \sigma_{branch\text{-}name\ =\ \text{¡ Perryridge¡}}\ (loan)\ )\ \times\ borrower$$
$$)$$
$$)$$

Example Queries

- Find the largest account balance
  - Rename *account* relation as *d*
  - The query is:

$$\Pi_{balance}(account)\ ?$$

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

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
  - $\prod_s(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_N(E_1)$, N 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

Division Operation

$$r \div s$$

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

  The result of r ÷ s is a relation on schema

  $R ? S = (A_1, ¡, \quad A_m)$

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

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

Division Operation ? 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

Division Operation

- 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 $S \subseteq R$

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

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

Example Queries

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

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{¡ Downtown¡}}(depositor \bowtie account)) \cap$$

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{¡ Uptown¡}}(depositor \bowtie account))$$

  - Query 2

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

$$\div\ \rho_{temp(branch\text{-}name)}(\{(\text{¡Downtown¡}), (\text{¡Uptown¡})\})$$

Example Queries

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

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

Extended Relational-Algebra Operations

adds power and convenience to the relational algebra

- Generalized Projection
- Aggregate Functions
- Outer Join

Generalized Projection

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

$$\Pi_{F1,\ F2,\ ¡\ ,\ Fn}(E)$$

  - E is any relational-algebra expression
  - Each of F_1, F_2, ¡ , F_n are are arithmetic expressions involving constants and attributes in the schema of E.

- Find how much more each person can spend

credit-info(customer-name, limit, credit-balance)

$$\Pi_{customer\text{-}name,\ limit\ ?\ credit\text{-}balance}\ (credit\text{-}info)$$

Aggregate Functions and Operations

- Aggregation function
  - takes a collection of values and returns a single value as a result.
  - avg, min, max, sum, count
- Aggregate operation in relational algebra

$$_{G1,\, G2,\, \text{¡}\,,\, Gn}\; g_{\,F1(A1),\, F2(A2),\, \text{¡}\,,\, Fm(Am)}\;(E)$$

  - E is any relational-algebra expression
  - G_1, G_2 ¡ , G_n is a list of attributes on which to group (can be empty)
  - F_i is an aggregate function, and
  - A_i is an attribute name, for i=1, ¡ , m

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 Operation ? Example

- Relation *account*

branch-name	*account-number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

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

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

- Result of aggregation does not have a name
  - Can use rename operation to give it a name (as in SQL)

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

Outer Join

- An extension of the join operation that avoids loss of information
  - Compute the join
  - add tuples that do not have matching tuples in the other relation
    - fill in undefined attribute values with *null*
- Three types
  - left outer join
  - right outer join
  - full outer join
- Conventional join is called inner join

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

- Inner join: *loan* $\bowtie$ *Borrower*

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

Outer Join ? Example

- 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*

- 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

Null Values

- *null*: an unknown or non-existing value
- The result of any arithmetic expression involving *null* is *null*
- Aggregate functions
  - simply ignore null values (SQL semantics)
  - Is an arbitrary decision. Could have returned null as result instead.
- For duplicate elimination and grouping
  - null is treated like any other value, and two nulls are assumed to be the same (SQL semantics)
  - Alternative: assume each null is different from each other
  - Both are arbitrary decisions

Null Values

- Comparisons with null values return the special truth value *unknown*
  - Why do we need 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.
- Deletion operation in relational algebra :

$$r \leftarrow r \, ? \, E$$

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

- You can only delete whole tuples;
  - cannot delete values on only particular attributes

    => should use update operation

Deletion Examples

- Delete all account records in the Perryridge branch.

  $account \leftarrow account \ ? \ \sigma_{branch\text{-}name\ =\ ¡Perryridge¡} \ (account)$

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

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

- Delete all accounts at branches located in Needham.

  $r_1 \leftarrow \sigma_{branch\text{-}city\ =\ ¡Needham¡} \ (account \quad branch) \ \bowtie$

  $r_2 \leftarrow \Pi_{branch\text{-}name,\ account\text{-}number,\ balance} \ (r_1)$

  $r_3 \leftarrow \Pi_{customer\text{-}name,\ account\text{-}number} \ (r_2 \quad \text{depositor}) \bowtie$

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

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

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
- Insertion operation in relational algebra:

$$r \leftarrow r \cup E$$

  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.

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

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

- 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.

$$r_1 \leftarrow (\sigma_{branch\text{-}name\ =\ ¡\ Perryridge¡}\ (borrower \bowtie loan))$$

$$account \leftarrow account \cup \Pi_{branch\text{-}name,\ account\text{-}number, 200}\ (r_1)$$

$$depositor \leftarrow depositor \cup \Pi_{customer\text{-}name,\ loan\text{-}number}\ (r_1)$$

Updating

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

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

Each F_i, is either

- the *i*th attribute of *r:* the attribute is not updated, or,
- an expression, involving only constants and the attributes of *r*, which gives the new value for the attribute: the attribute is updated

Update Examples

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

$$account \leftarrow \Pi_{\,account\#,\ branch\text{-}name,\ balance*1.05}\ (account)$$

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

$$account \leftarrow$$

$$\Pi_{\,account\#,\ branch\text{-}name,\ balance*1.06}\ (\sigma_{\,balance\ >\ 10000}\ (account)\ )\ \cup$$
$$\Pi_{account\#,\ branch\text{-}name,\ balance*1.05}\ (\sigma_{balance\ \leq\ 10000}\ (account)\ )$$

END OF CHAPTER 2