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

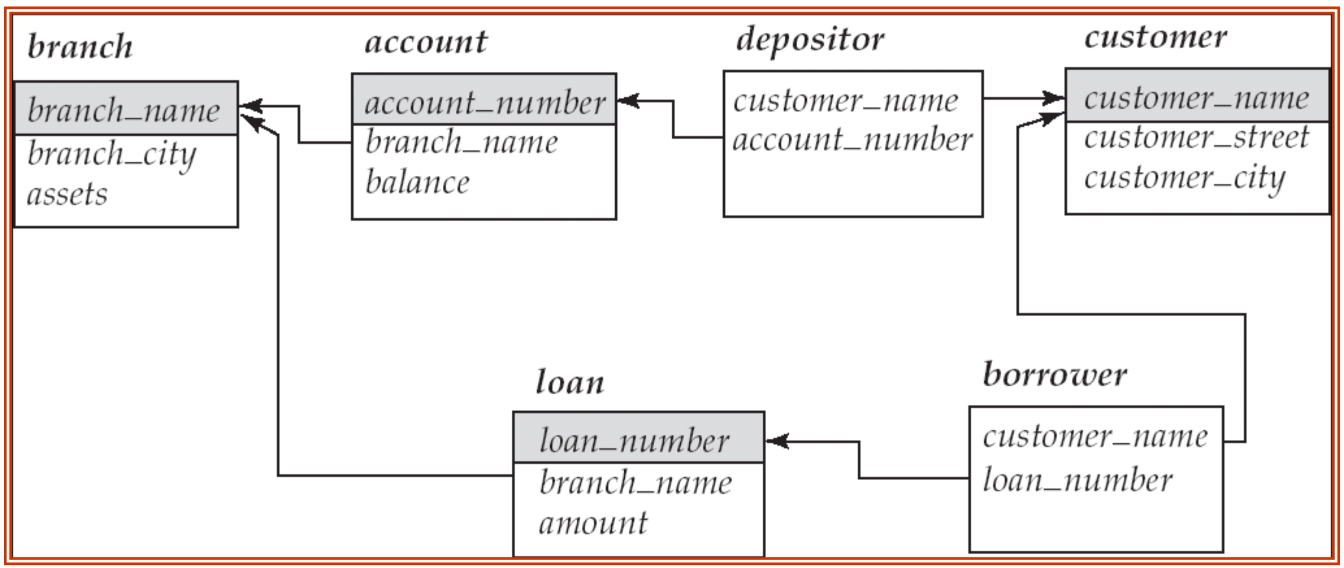
- A set of attribute *K* is a **superkey** of *R* if:
  - No two tuples can have same values in all these attributes
- Example: Customer(*customer_name, customer_street, customer_city*)
  - {*customer_name, customer_street*} and {*customer_name*} are both superkeys
  - What about *name*?
  - PS: In real 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

ClassExercise

- 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: $\sigma_p(r)$
- 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

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

Project Operation

- Notation:

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

  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 *dept_name* attribute of *account*

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

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)

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

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

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

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 $r \times s$
- Defined as:

$$r \times s = \{(t, q) |\ t \in r \textbf{ and } q \in s\}$$

- 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

- 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

- $r \times s$:

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 $A_1, A_2, \ldots, A_n$.

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
  - $\prod_{loan\_number}(\sigma_{amount>1200}(loan))$
- Find the names of all customers who have a loan, an account, or both, from the bank
  - $\prod_{customer\_name}(borrower)$

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

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

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 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

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 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.

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
- 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