

DBST课程群



该二维码7天内(9月18日前)有效, 重新进入将更新

Relational Model

Roadmap of This Lecture

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

Example of a Relation

<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
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

Basic Structure

- Formally, given sets D_1, D_2, \dots, D_n , a **relation** r is a subset of

$$D_1 \times D_2 \times \dots \times D_n$$

Thus, a relation is a **set** of n -tuples (a_1, a_2, \dots, a_n) where each $a_i \in D_i$

- Example: If

- $customer_name = \{\text{Jones, Smith, Curry, Lindsay, ...}\}$
/* Set of all customer names */
- $customer_street = \{\text{Main, North, Park, ...}\}$ /* set of all street names*/
- $customer_city = \{\text{Harrison, Rye, Pittsfield, ...}\}$ /* set of all city names */

Then $r = \{$
 (Jones, Main, Harrison),
 (Smith, North, Rye),
 (Curry, North, Rye),
 (Lindsay, Park, Pittsfield) $\}$

is a relation over

$customer_name \times customer_street \times customer_city$

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. 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 Schema

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

Example:

$Customer_schema = (customer_name, customer_street, customer_city)$

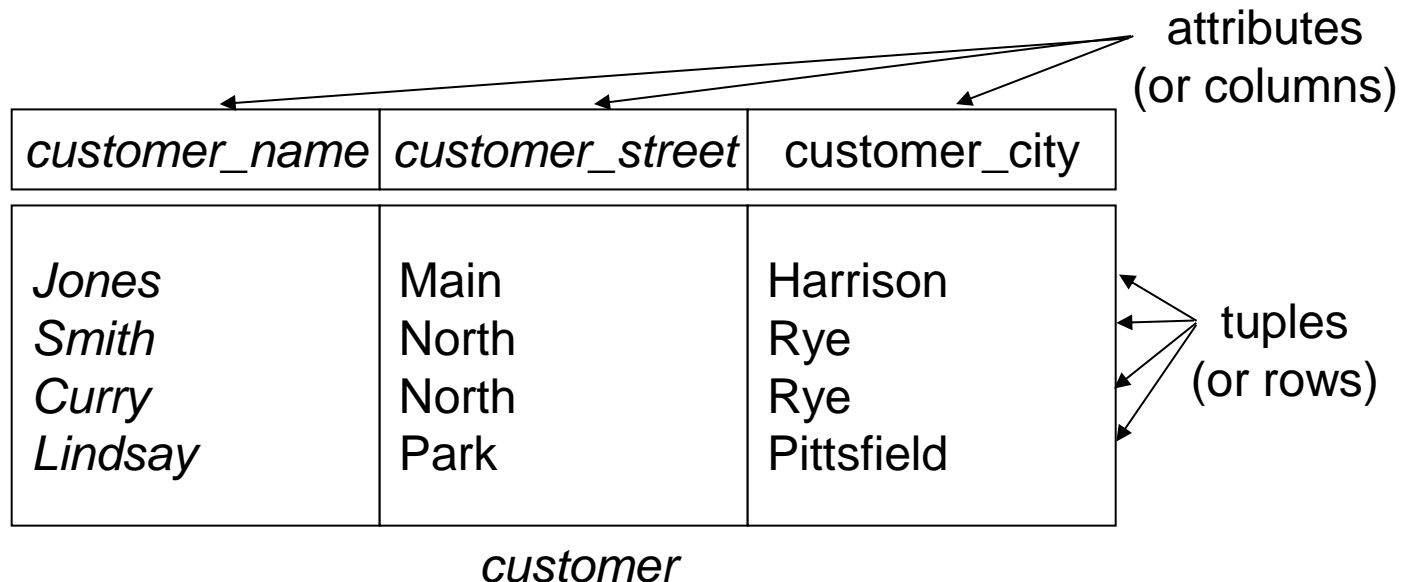
- $r(R)$ denotes a *relation* r on the *relation schema* R

Example:

$customer (Customer_schema)$

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



Relations are Unordered

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

<i>account_number</i>	<i>branch_name</i>	<i>balance</i>
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 (later) deals with how to design relational schemas

The *customer* Relation

<i>customer_name</i>	<i>customer_street</i>	<i>customer_city</i>
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

<i>customer_name</i>	<i>account_number</i>
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 $r(R)$
 - by “possible r ” we mean a relation r that could exist in the enterprise we are modeling.
 - Example: $\{customer_name, customer_street\}$ and $\{customer_name\}$
are both superkeys of *Customer*, if no two customers can possibly have the same name
 - ▶ 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

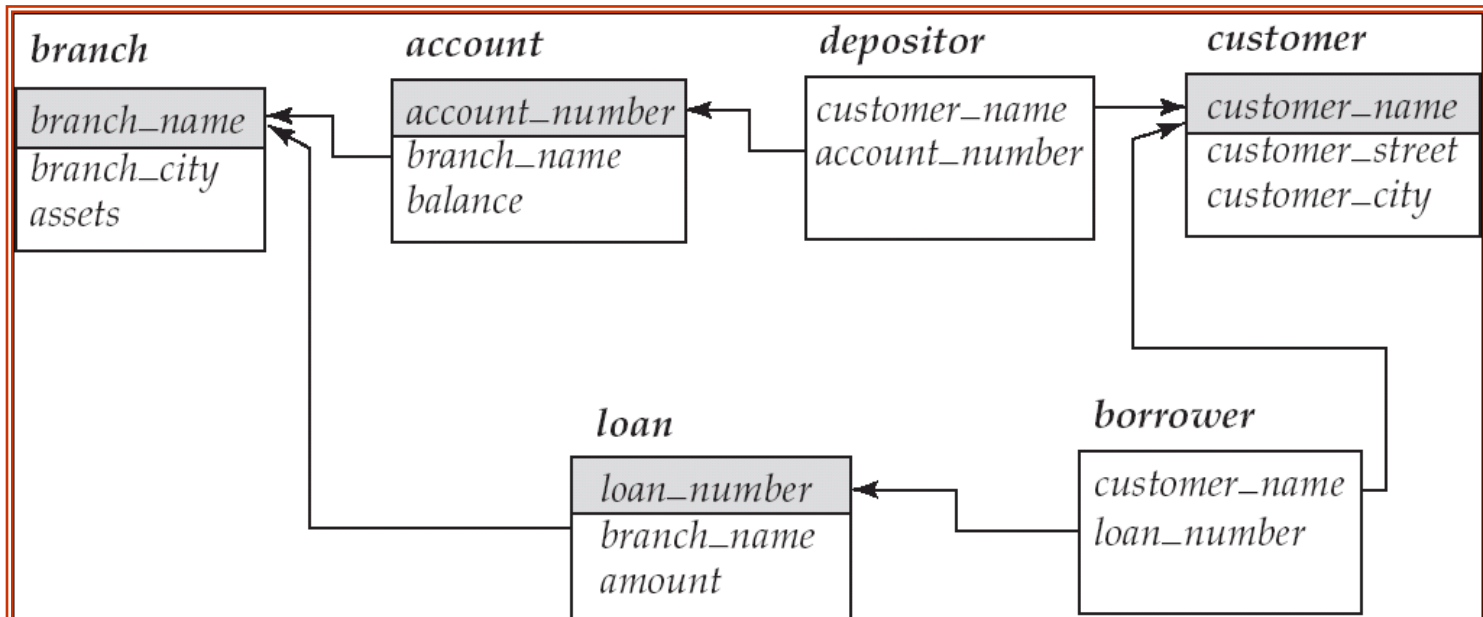
Example: $\{customer_name\}$ is a candidate key for *Customer*, since it is a superkey and no subset of it is a superkey.

- **Primary key:** 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 a set of attributes that corresponds to the primary key of another relation. These attributes are 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.
 - Pure languages use immutable variables only!
 - They are *functional*.

Relational Algebra

- Procedural language
- Six basic operators
 - select: σ
 - project: Π
 - union: \cup
 - set difference: $-$
 - Cartesian product: \times
 - rename: ρ
- 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

Select Operation

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

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

Where p is a formula in propositional calculus consisting of **terms** connected by logical connectives: \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:

$$\sigma_{branch_name="Perryridge"}(account)$$

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

Project Operation

- Notation:

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

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*

$$\Pi_{\text{account_number, balance}}(\text{account})$$

Union Operation – Example

- Relations r, s :

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cup s$:

A	B
α	1
α	2
β	1
β	3

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: 2nd column of r deals with the same type of values as does the 2nd column of s)
- Example: to find all customers with either an account or a loan

$$\Pi_{customer_name}(depositor) \cup \Pi_{customer_name}(borrower)$$

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

Set Difference Operation

- Notation $r - s$
- Defined as:

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

- 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

■ $r \times s$:

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

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

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

- 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.

Composition of Operations

- Can build expressions using multiple operations

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

- $r \times s$

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
α	1	α	10	<i>a</i>
α	1	β	10	<i>a</i>
α	1	β	20	<i>b</i>
α	1	γ	10	<i>b</i>
β	2	α	10	<i>a</i>
β	2	β	10	<i>a</i>
β	2	β	20	<i>b</i>
β	2	γ	10	<i>b</i>

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

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
α	1	α	10	<i>a</i>
β	2	β	10	<i>a</i>
β	2	β	20	<i>b</i>

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

$$\rho_{X(A_1, A_2, \dots, A_n)}(E)$$

returns the result of expression E under the name X , and with the attributes renamed to A_1, A_2, \dots, 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*)

Highlighted attributes are primary keys.

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

$$\Pi_{customer_name} (borrower) \cup \Pi_{customer_name} (depositor)$$

<p><i>branch</i> (<i>branch_name</i>, <i>branch_city</i>, <i>assets</i>) <i>customer</i> (<i>customer_name</i>, <i>customer_street</i>, <i>customer_city</i>) <i>account</i> (<i>account_number</i>, <i>branch_name</i>, <i>balance</i>) <i>loan</i> (<i>loan_number</i>, <i>branch_name</i>, <i>amount</i>) <i>depositor</i> (<i>customer_name</i>, <i>account_number</i>) <i>borrower</i> (<i>customer_name</i>, <i>loan_number</i>)</p>
--

Example Queries

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

$$\Pi_{customer_name} (\sigma_{branch_name = \text{“Perryridge”}} (\sigma_{borrower.loan_number = loan.loan_number} (borrower \times loan)))$$

- 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.

$$\Pi_{customer_name} (\sigma_{branch_name = \text{“Perryridge”}} (\sigma_{borrower.loan_number = loan.loan_number} (borrower \times loan))) - \Pi_{customer_name} (depositor)$$

<p><i>branch</i> (<i>branch_name</i>, <i>branch_city</i>, <i>assets</i>) <i>customer</i> (<i>customer_name</i>, <i>customer_street</i>, <i>customer_city</i>) <i>account</i> (<i>account_number</i>, <i>branch_name</i>, <i>balance</i>) <i>loan</i> (<i>loan_number</i>, <i>branch_name</i>, <i>amount</i>) <i>depositor</i> (<i>customer_name</i>, <i>account_number</i>) <i>borrower</i> (<i>customer_name</i>, <i>loan_number</i>)</p>
--

Example Queries

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

$$\Pi_{\text{customer_name}} (\sigma_{\text{branch_name} = \text{"Perryridge"}} (\sigma_{\text{borrower.loan_number} = \text{loan.loan_number}} (\text{borrower} \times \text{loan})))$$

What's the benefit of the second query?

- Query 2

$$\Pi_{\text{customer_name}} (\sigma_{\text{loan.loan_number} = \text{borrower.loan_number}} (\sigma_{\text{branch_name} = \text{"Perryridge"}} (\text{loan}) \times \text{borrower}))$$

```
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

- 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* selected in the earlier step.
 - ▶ That missing balance is the MAX among all account balances
 - 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 (shown later)
- 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

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$
- 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: $r \bowtie s$
- Let r and s be relations on schemas R and S respectively. Then, $r \bowtie s$ is a relation on schema $R \cup S$ obtained as follows:
 - Consider each pair of tuples t_r from r and t_s from s .
 - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where
 - ▶ t has the same value as t_r on r
 - ▶ t has the same value as t_s on s

- Example:

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

$S = (E, B, D)$

- Result schema = (A, B, C, D, E)
- $r \bowtie s$ 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	ϵ

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”.
- Let r and s be relations on schemas R and S respectively where
 - $R = (A_1, \dots, A_m, B_1, \dots, B_n)$
 - $S = (B_1, \dots, B_n)$

The result of $r \div s$ is a relation on schema

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

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

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

Division Operation – Example

■ Relations r, s :

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ϵ	6
ϵ	1
β	2

r

B
1
2

s

■ $r \div s$:

A
α
β

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

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

$$temp1 \leftarrow \Pi_{R-S}(r)$$

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

$$temp1 - temp2$$

- 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, the loan number and the loan amount

$$\Pi_{customer_name, loan_number, amount} (borrower \bowtie loan)$$

<p><i>branch</i> (<i>branch_name</i>, branch_city, assets) <i>customer</i> (<i>customer_name</i>, customer_street, customer_city) <i>account</i> (<i>account_number</i>, branch_name, balance) <i>loan</i> (<i>loan_number</i>, branch_name, amount) <i>depositor</i> (customer_name, account_number) <i>borrower</i> (customer_name, loan_number)</p>
--

Bank Example Queries

- Find all customers who have an account from both 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.

<p><i>branch</i> (<i>branch_name</i>, <i>branch_city</i>, <i>assets</i>) <i>customer</i> (<i>customer_name</i>, <i>customer_street</i>, <i>customer_city</i>) <i>account</i> (<i>account_number</i>, <i>branch_name</i>, <i>balance</i>) <i>loan</i> (<i>loan_number</i>, <i>branch_name</i>, <i>amount</i>) <i>depositor</i> (<i>customer_name</i>, <i>account_number</i>) <i>borrower</i> (<i>customer_name</i>, <i>loan_number</i>)</p>
--

Bank Example Queries

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

$$\Pi_{customer_name, branch_name} (depositor \bowtie account) \\ \div \Pi_{branch_name} (\sigma_{branch_city = \text{“Brooklyn”}} (branch))$$

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