Lecture 2
Relational Model and Relational Algebra

March 5, 2014

Shuigeng Zhou
School of Computer Science
Fudan University

Outline

- Relational Model for Databases
- Relational Algebra Operations

Example of a Relation

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

Basic Structure

- 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 $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$ (i=1~n)

n-tuples: n元组

- Example: if

  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 (属性类型)

- Each attribute of a relation has a name
- The domain of an attribute is the whole set of available and legal values of the attribute
- Attribute values are (normally) required to be atomic
  - E.g. multi-valued 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

Relation Schema (关系模式)

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

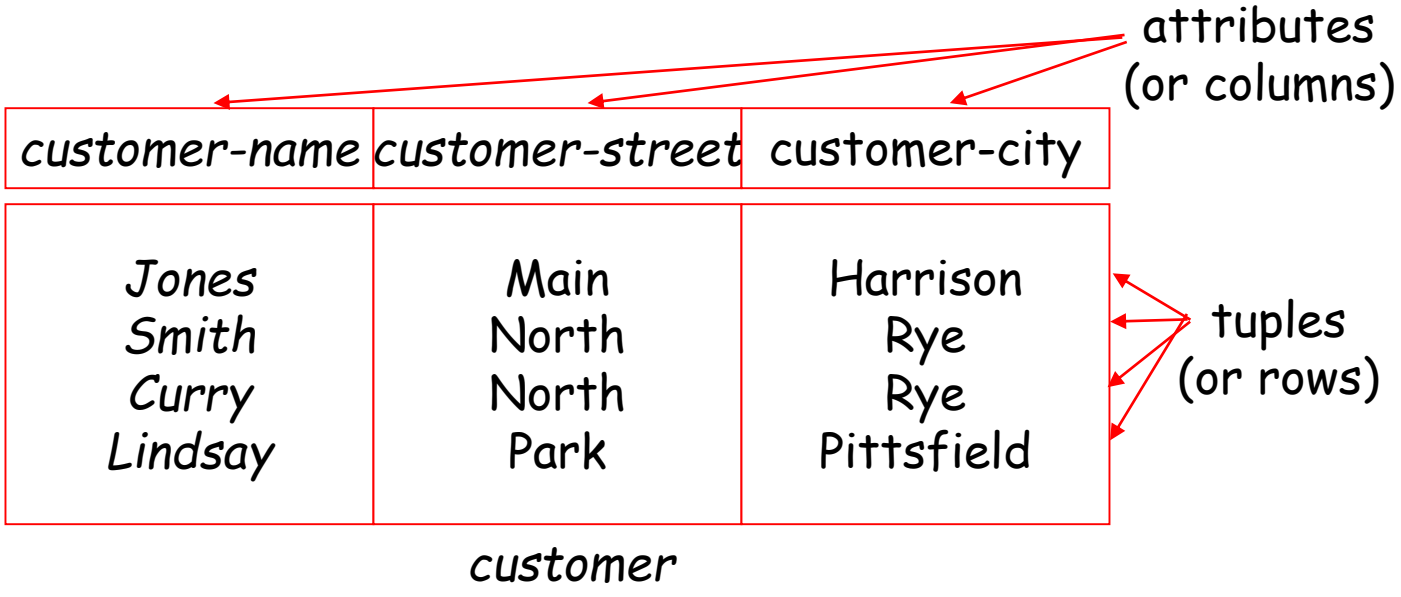
  E.g. *Customer-schema* = (*customer-name, customer-street, customer-city*)

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

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

Relation Instance (关系实例)

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



Relation vs. Variable

- Relation vs. Variable
- Relation schema vs. Variable type
- Relation instance vs. Variable value
- For example
  - int vs. *Customer-schema* =(*name*, *street*, *city*)
  - int *A* vs. customer(Customer-schema)
  - *A*=10 vs.

Jones	Main	Harrison
Smith	North	Rye
Curry	North	Rye
Lindsay	Park	Pittsfield

Relations are Unordered

- Order of tuples / attributes in a relation is irrelevant (tuples may be stored in an arbitrary order)
- E.g. *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

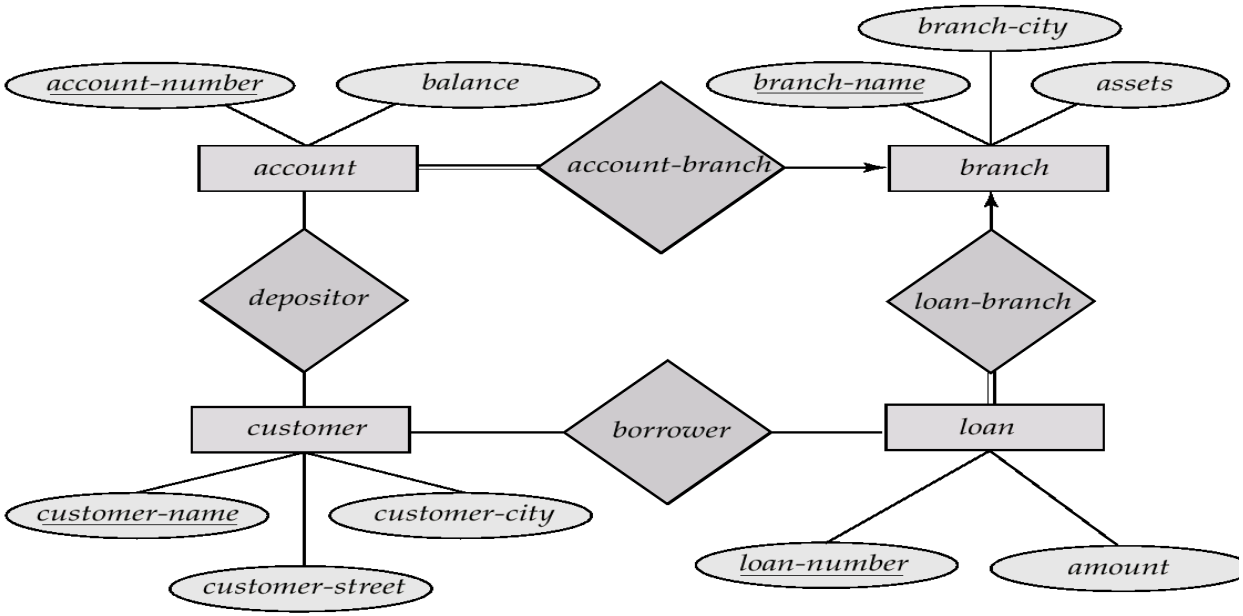
- A database consists of multiple relations
- Storing all information as a single relation results in
  - repetition of information (e.g. two customers own an account)
  - the need for null values (e.g. represent a customer without an account)
- Normalization theory (Chapter 7) deals with how to design relational schemas

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 customer 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

The depositor Relation



E-R Diagram for the Banking Enterprise

Keys (键)

- 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)*
  - Example: {*customer-name*, *customer-street*} and {*customer-name*}
    are both superkeys of *Customer*, if no two customers can possibly have the same name.
- *K* is a ***candidate key*** if *K* is minimal
  - Example: {customer-name} is a candidate key for Customer, since it is a superkey (assuming no two customers can possibly have the same name), and no subset of it is a superkey.

Determining Keys from E-R Sets

- Strong entity set
- Weak entity set
  - Discriminator plus the Key of the identifying entity set
- Relationship set
  - Union of keys of the related entity sets

Banking Database: An Example

branch (*branch-name*, *branch-city*, *assets*)

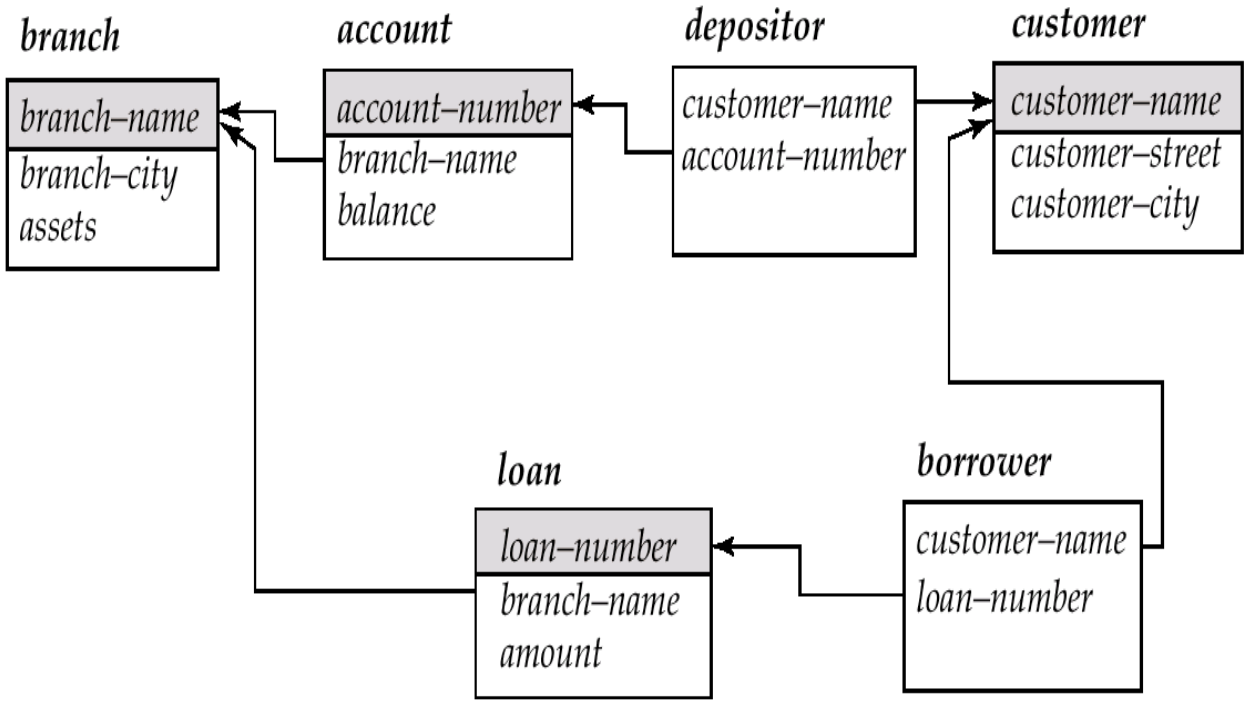
customer (*customer-name*, *customer-street*, *customer-city*)

account (*account-number*, *branch-name*, *balance*)

loan (*loan-number*, *branch-name*, *amount*)

depositor (*customer-name*, *account-number*)

borrower (*customer-name*, *loan-number*)



Schema Diagram (模式图) for the Banking Enterprise

Query Languages (查询语言)

- Language in which users request information from the database.
- Categories of languages
  - procedural
  - non-procedural
- "Pure" languages (in the sense that they contains only elements of data manipulation)
  - Procedural
    - ✓Relational Algebra（关系代数）
  - Non-procedural
    - ✓Tuple Relational Calculus（元组关系演算）
    - ✓Domain Relational Calculus（域关系演算）
- Pure languages form underlying basis of query languages that people use

Relational Algebra

- Procedural language
- Six basic operators
  - Select (选择)
  - Project (投影)
  - Union (集合并)
  - set difference (集合差)
  - Cartesian product (笛卡尔积)
  - Rename (重命名)
- The operators take two or more relations as inputs and give 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（选择谓词）
- Defined as:

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

Where p is a formula in propositional calculus （命题演算） 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:

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

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:

  $$\Pi_{A1, A2, \ldots, Ak}(r)$$

  where $A_1, A_2, \ldots, Ak$ 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
- E.g. 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* (e.g., 2nd column of r deals with the same type of values as does the 2nd column of s)
- E.g. to find all customers with either an account or a loan

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

Arity:元数/维度

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, $R \cap S = \varnothing$)
- 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: $\sigma_{A=C}(r \times s)$
- $r \times s$

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	20	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

$$\rho_{X(A1, A2, \ldots, An)}(E)$$

returns the result of expression E under the name X, and with the attributes renamed to $A1, A2, \ldots, An$.

Example Queries (1)

- 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 (2)

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

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

Example Queries (3)

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

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“}Perryridge\text{”}}$$
$$(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower\ x\ loan)))$$

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

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name\ =\ \text{“}Perryridge\text{”}}$$
$$(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(\text{borrower x loan}))\ -$$
$$\Pi_{customer\text{-}name}(\text{depositor})$$

Example Queries (4)

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

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

  - Query 2

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

Example Queries (5)

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

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

- Find the smaller account balance ?

Relational 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

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:

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

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

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

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

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

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

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 ÷ *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

$$temp_1 \leftarrow \Pi_{R-S}(r)$$

$$temp_2 \leftarrow \Pi_{R-S}((temp_1 \times s) - \Pi_{R-S,S}(r))$$

$$result = temp_1 - temp_2$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$

Example Queries (6)

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

$$\Pi_{CN}(\sigma_{BN=\text{"Downtown"}}(depositor \bowtie account)) \cap \Pi_{CN}(\sigma_{BN=\text{"Uptown"}}(depositor \bowtie account))$$

where *CN* denotes customer-name and *BN* denotes *branch-name*.

  - Query 2

$$\Pi_{customer\text{-}name,\ branch\text{-}name}(depositor \bowtie account) \div \rho_{temp(branch\text{-}name)}(\{(\text{"Downtown"}), (\text{"Uptown"})\})$$

Example Queries (7)

- 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

- Generalized Projection（广义投影）
- Outer Join（外连接）
- Aggregate Functions （聚合函数）

Generalized Projection

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

$$\Pi_{F1, F2, \ldots, Fn}(E)$$

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

$$\Pi_{\textit{customer-name, (limit - credit-balance) as credit-available}}(\textit{credit-info})$$

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

  $$G_1, G_2, \ldots, G_n \, \mathcal{G} \, F_1(A_1), F_2(A_2), \ldots, F_n(A_n) \, (E)$$

  - 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

$g_{\text{sum(c)}}(r)$:

sum-C
27

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

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

branch-name	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

$$branch\text{-}name\ \mathcal{G}\ \mathbf{sum}(balance)\ \mathbf{as}\ sum\text{-}balance\ (account)$$

Outer Join

- An extension of the join operation that avoids loss of information
- Computes the join and then adds 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.
    - Will study precise meaning of comparisons with nulls later

Outer Join – Example

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Relation *loan*

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	L-155

Relation *borrower*

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

Inner Join: *loan* $\bowtie$ *Borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Left Outer Join: *loan* $\mathbin{⟕}$ *Borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Right Outer Join: *loan* $\mathbin{⟖}$ *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

Full Outer Join: *loan* $\mathbin{⟗}$ *borrower*

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
  - Is an arbitrary decision. Could have returned null as result instead.
  - We follow the semantics of SQL in its handling of null values
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same
  - Alternative: assume each null is different from each other
  - Both are arbitrary decisions, so we simply follow SQL

Null Values

- Comparisons with null values return the special truth value *unknown*
- 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

Deletion Examples

- Delete all account records in the Perryridge branch.

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

- 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.

$$r_1 \leftarrow \sigma_{branch\text{-}city = \text{"Needham"}}(account \bowtie branch)$$
$$r_2 \leftarrow \Pi_{branch\text{-}name,\ account\text{-}number,\ balance}(r_1)$$
$$r_3 \leftarrow \Pi_{customer\text{-}name,\ account\text{-}number}(r_2 \bowtie depositor)$$
$$account \leftarrow account - r_2$$
$$depositor \leftarrow depositor - r_3$$

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.

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

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

- 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.

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

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

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

Updating

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

$$r \leftarrow \Pi_{F1, F2, \ldots, Fi,}(r)$$

- Each F_i is either
  - the *i*th attribute of *r*, if the *i*th attribute is not updated, or,
  - if the attribute 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.

$$account \leftarrow \Pi_{AN,\ BN,\ BAL * 1.05}(account)$$

where *AN*, *BN* and *BAL* stand for *account-number*, *branch-name* and *balance*, respectively.

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

$$account \leftarrow \Pi_{AN,\ BN,\ BAL * 1.06}(\sigma_{BAL > 10000}(account)) \cup \Pi_{AN,\ BN,\ BAL * 1.05}(\sigma_{BAL \leq 10000}(account))$$

Views（视图）

- In some cases, it is not desirable for all users to see the entire logical model

- Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by

  $\Pi_{customer\text{-}name,\ loan\text{-}number,\ branch\text{-}name}\ (borrower \bowtie loan)$

- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**

View Definition

- A view is defined using the **create view** statement which has the form

  create view *v* **as** < query expression >

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates

- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view

View Examples

- Consider the view (named *all-customer*) consisting of branches and their customers.

create view *all-customer* **as**

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

- We can find all customers of the Perryridge branch by writing:

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name = \text{“Perryridge”}}(all\text{-}customer))$$

Updates Through View

- Must be translated to modifications of the actual relations
- Consider the person who needs to see all loan data in the *loan* relation except *amount*. The view given to the person, *branch-loan*, is defined as:

  create view *branch-loan* **as**

  $$\Pi_{\textit{branch-name, loan-number}}(\textit{loan})$$

- Since we allow a view name to appear wherever a relation name is allowed, the person may write:

  $$\textit{branch-loan} \leftarrow \textit{branch-loan} \cup \{(\text{“Perryridge”}, \text{L-37})\}$$

Updates Through Views (Cont.)

- An insertion into *loan* requires a value for *amount*. The insertion can be dealt with by either.
  - rejecting the insertion.
  - inserting a tuple ("L-37", "Perryridge", *null*)
- Some updates through views are impossible to translate into database relation updates

  create view v as $\sigma_{branch\text{-}name = \text{"Perryridge"}}$ (*account*))

  $v \leftarrow v \cup$ (L-99, Downtown, 23)
- Others cannot be translated uniquely

  all-customer $\leftarrow$ *all-customer* $\cup$ {("Perryridge", "John")}

  Have to choose loan or account, and create a new loan/account number!

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to ***depend directly*** （直接依赖）on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to ***depend*** （依赖）*on* view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2
- A view relation v is said to be ***recursive*** (递归) if it depends on itself

View Expansion

- A way to define the meaning of views defined in terms of other views
- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations
- View expansion of an expression repeats the following replacement step:

  repeat
    Find any view relation v_i in e_1
    Replace the view relation v_i by the expression defining v_i
  until no more view relations are present in e_1

- As long as the view definitions are not recursive, this loop will terminate

Homework

- Exercises
  - 2.1, 2.3, 2.5, 2.7, and 2.9
  - 2.11

End of Lecture 2