Unit 2
Relational Model

By :
Mrs. Suman Madan
suman.madan@jimsindia.org

Agenda

- Structure of Relational Databases
- Relational Algebra
- Tuple Relational Calculus
- Domain Relational Calculus
- Extended Relational-Algebra-Operations
- Modification of the Database
- Views

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 x D_2 x … x D_n *where* D_i *is a domain*

- 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}
  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

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

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

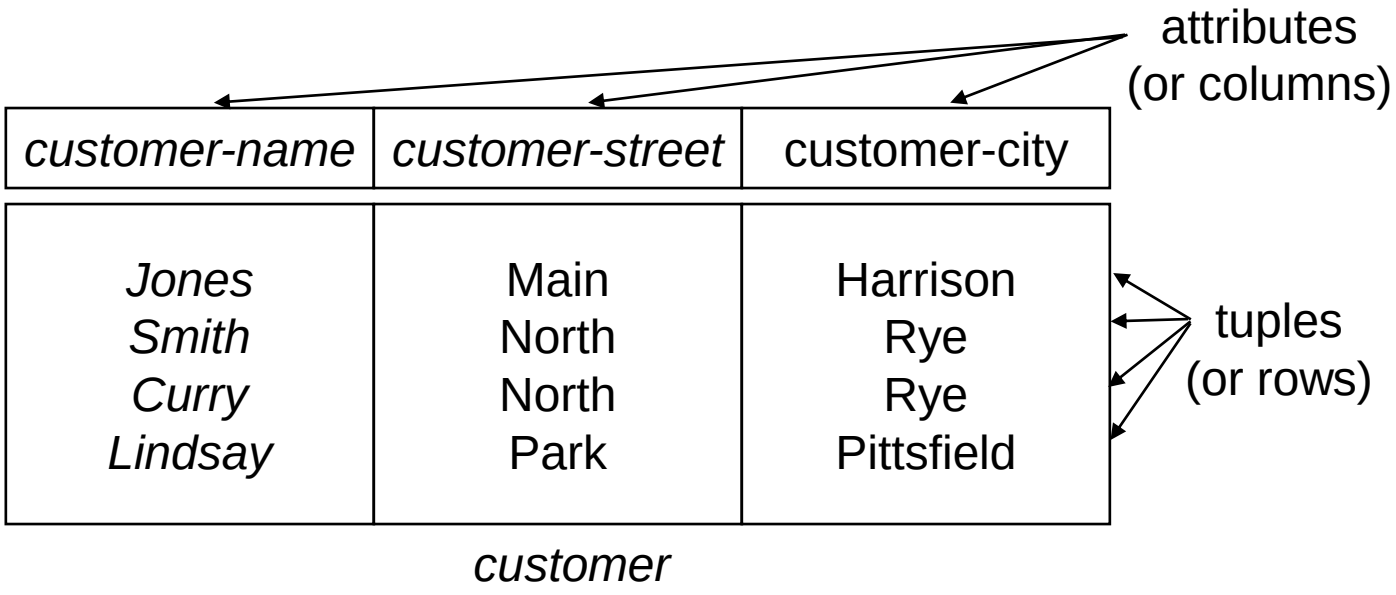
E.g. *Customer-schema* =
(*customer-name, customer-street, customer-city*)

- *r*(*R*) is a *relation* on the *relation schema R*
  E.g. *customer (Customer-schema)*

Relation Instance

- 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)
- 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

- Information about an enterprise is broken up into parts, with each relation storing one part of the information

  E.g.: *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. two customers own an account)
- the need for null values (e.g. represent a customer without an account)

- Normalization theory 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

Determining Keys from E-R Sets

- **Strong entity set**. The primary key of the entity set becomes the primary key of the relation.

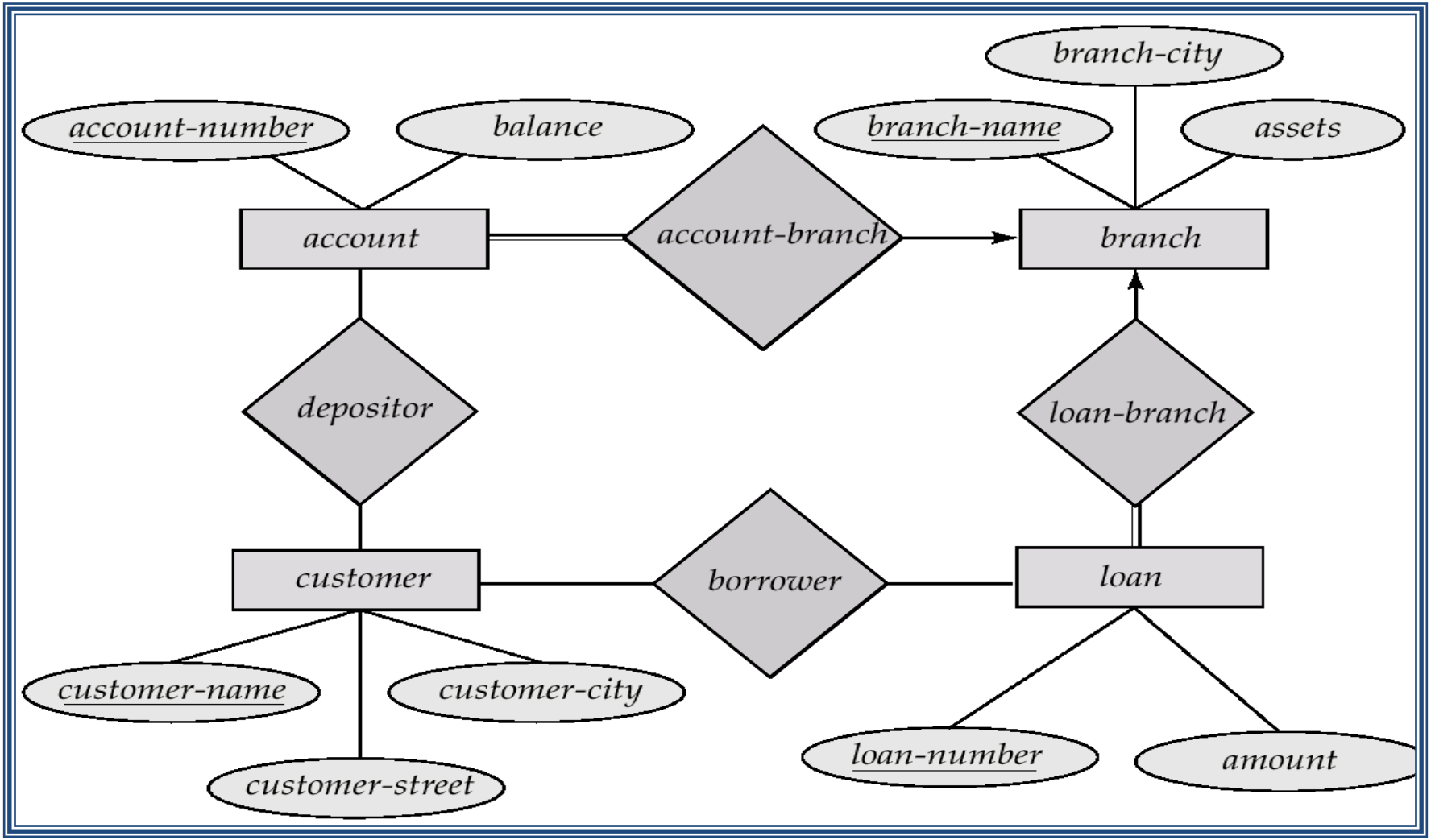
- **Weak entity set**. The primary key of the relation consists of the union of the primary key of the strong entity set and the discriminator of the weak entity set.

- **Relationship set**. The union of the primary keys of the related entity sets becomes a super key of the relation.
  - For binary many-to-one relationship sets, the primary key of the "many" entity set becomes the relation's primary key.

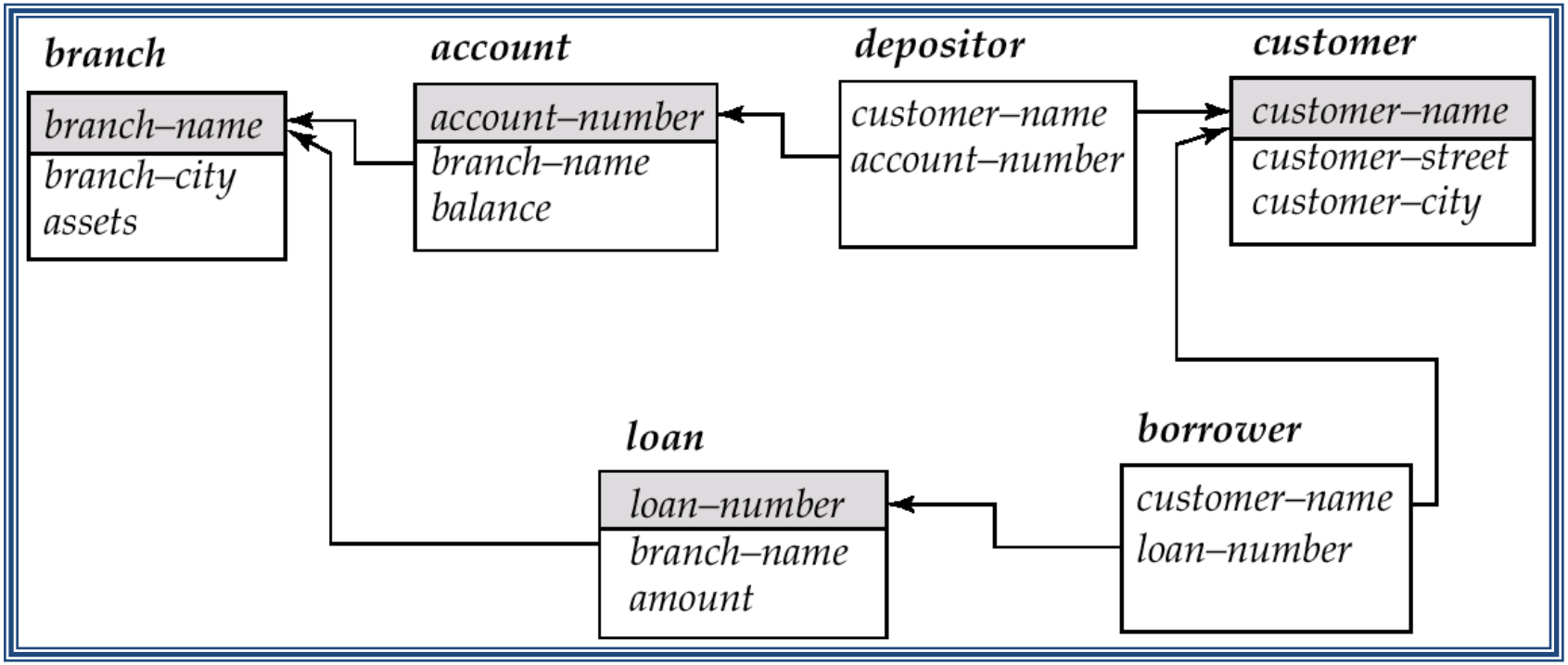
  - For one-to-one relationship sets, the relation's primary key can be that of either entity set.

  - For many-to-many relationship sets, the union of the

E-R Diagram for the Banking Enterprise



Schema Diagram for the Banking Enterprise



Query Languages

- Language in which user requests information from the database.

- Categories of languages
  - procedural
  - non-procedural

- “Pure” languages:
  - Relational Algebra
  - Tuple Relational Calculus
  - Domain Relational Calculus

- Pure languages form underlying basis of query

Relational Algebra

- Procedural language

- Seven basic operators
  - Select
  - Project
  - Union
  - Intersection
  - Set difference
  - Cartesian product
  - Rename

- The operators take two or more relations as

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

Unary Relational Operations

SELECT Operation

- SELECT operation is used to select a *subset* of the tuples from a relation that satisfy a **selection condition**. It is a filter that keeps only those tuples that satisfy a qualifying condition – those satisfying the condition are selected while others are discarded.

- **Example:** To select the EMPLOYEE tuples whose department number is four or those whose salary is greater than $30,000 the following notation is used:

  $\sigma_{DNO = 4}$ **(EMPLOYEE)**

  $\sigma_{SALARY > 30,000}$ **(EMPLOYEE)**

When both conditions are together then query is like:

$\sigma_{DNO = 4 \ V \ SALARY > 30,000}$ **(EMPLOYEE)**

- In general, the select operation is denoted by $\sigma_{<\text{selection condition}>}(R)$ where the symbol σ (sigma) is used to denote the select operator, and the selection condition is a Boolean expression specified on the attributes of relation R

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:

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

Unary Relational Operations (cont.)

PROJECT Operation

- This operation selects certain *columns* from the table and discards the other columns. The PROJECT creates a vertical partitioning – one with the needed columns (attributes) containing results of the operation and other containing the discarded Columns.

- **Example:** To list each employee's first and last name and salary, the following is used:

$$\pi_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$$

- The general form of the project operation is π<attribute list>(R) where π (pi) is the symbol used to represent the project operation and <attribute list> is the desired list of attributes from the attributes of relation R.

- The project operation *removes any duplicate tuples*, so the result of the project operation is a set of tuples and hence a valid relation.

Project Operation

- Notation:

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

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

- E.g. To eliminate the *branch-name* attribute of *account*

$$\Pi_{account\text{-}number,\ balance}\ (account)$$

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

Relational Algebra Operations From Set Theory

UNION Operation

- The result of this operation, denoted by R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

- **Example:** To retrieve the social security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5, we can use the union operation as follows:

DEP5_EMPS ← $\sigma_{DNO=5}$ (EMPLOYEE)

RESULT1 ← π_{SSN}(DEP5_EMPS)

RESULT2(SSN) ← $\pi_{SUPERSSN}$(DEP5_EMPS)

RESULT ← RESULT1 ∪ RESULT2

- The union operation produces the tuples that are in either RESULT1 or RESULT2 or both. The two operands must be “type compatible”.

Relational Algebra Operations From Set Theory

- **Type Compatibility**

  - The operand relations $R_1(A_1, A_2, ..., A_n)$ and $R_2(B_1, B_2, ..., B_n)$ must have the same number of attributes, and the domains of corresponding attributes must be compatible; that is, $dom(A_i)=dom(B_i)$ for i=1, 2, ..., n.

  - The resulting relation for $R_1 \cup R_2$, $R_1 \cap R_2$, or $R_1 - R_2$ has the same attribute names as the *first* operand relation R1 (by convention).

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*)

Relational Algebra Operations From Set Theory

- **UNION Example**

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

Relational Algebra Operations From Set Theory (cont.)

- **INTERSECTION OPERATION**

  The result of this operation, denoted by R ∩ S, is a relation that includes all tuples that are in both R and S. The two operands must be "type compatible"

  Example: The result of the intersection operation (figure below) includes only those who are both students and instructors.

FN	LN
Susan	Yao
Ramesh	Shah

(d)

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

STUDENT ∩ INSTRUCTOR

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

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

Relational Algebra Operations From Set Theory (cont.)

- **Set Difference (or MINUS) Operation**

  The result of this operation, denoted by R - S, is a relation that includes all tuples that are in R but not in S. The two operands must be "type compatible".

  Example: The figure shows the names of students who are not instructors, and the names of instructors who are not students.

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

STUDENT-INSTRUCTOR

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

INSTRUCTOR-STUDENT

Set Difference Operation

- Notation $r - s$

- Defined as:

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

- 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

- Concatenation of every row in the first relation (R) with every row in the second relation (S):

R X S

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, $R \cap S = \varnothing$).

- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.

Cartesian Product - Example

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

Courses

course#	*name*
PH	Pharmacy
CM	Computing

Students X Courses =

stud#	*Students.name*	*course*	*course#*	*Courses.name*
100	Fred	PH	PH	Pharmacy
100	Fred	PH	CM	Computing
200	Dave	CM	PH	Pharmacy
200	Dave	CM	CM	Computing
300	Bob	CM	PH	Pharmacy
300	Bob	CM	CM	Computing

Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}($)
- *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

A	B	C	D	E
α	1	α	10	a
β	2	β	20	a
β	2	β	20	b

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

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

Rename Operation

- We may want to apply several relational algebra operations one after the other. Either we can write the operations as a single **relational algebra expression** by nesting the operations, or we can apply one operation at a time and create **intermediate result relations**. In the latter case, we must give names to the relations that hold the intermediate results.

- **Example:** To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation. We can write a single relational algebra expression as follows:

$$\pi_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO=5}}(\text{EMPLOYEE}))$$

- OR We can explicitly show the sequence of operations, giving a name to each intermediate relation:

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO=5}}(\text{EMPLOYEE})$$
$$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5\_EMPS})$$

Banking Example

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

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

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 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

branch *(branch-name, branch-city, assets)* ***borrower*** *(customer-name, loan-number)*

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

loan *(loan-number, branch-name, amount)* ***depositor*** *(customer-name, account-number)*

- Find the names of all customers who have 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 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

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

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

  - Query 2

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

Theta Join

- A Cartesian product with a condition applied:

$$R \bowtie_{\text{<condition>}} S$$

Theta Join - Example

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

Courses

course#	*name*
PH	Pharmacy
CM	Computing

Students $\bowtie_{stud\# = 200}$ Courses

stud#	*Students.name*	*course*	*course#*	*Courses.name*
200	Dave	CM	PH	Pharmacy
200	Dave	CM	CM	Computing

Inner Join (Equijoin)

- A Theta join where the <condition> is the match (=) of the primary and foreign keys.

$R \bowtie_{\text{<R.primary\_key = S.foreign\_key>}} S$

Inner Join - Example

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

Courses

course#	*name*
PH	Pharmacy
CM	Computing

Students $\bowtie$ *course = course#* Courses

stud#	*Students.name*	*course*	*course#*	*Courses.name*
100	Fred	PH	PH	Pharmacy
200	Dave	CM	CM	Computing
300	Bob	CM	CM	Computing

Natural Join

- Inner join produces redundant data (in the previous example: course and course#). To get rid of this duplication:

π *< stud#, Students.name, course, Courses.name >*
(Students $\bowtie$ <course = course#> Courses)

Or

R1= Students $\bowtie$ <course = course#> Courses

R2= π *< stud#, Students.name, course, Courses.name >* R1

The result is called the natural join of Students and Courses

Natural Join - Example

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

Courses

course#	*name*
PH	Pharmacy
CM	Computing

R1= Students $\bowtie$ <course = course#> Courses

R2= π < *stud#, Students.name, course, Courses.name* > R1

stud#	*Students.name*	*course*	*Courses.name*
100	Fred	PH	Pharmacy
200	Dave	CM	Computing
300	Bob	CM	Computing

Natural-Join Operation

- Notation: r ⋈ s

- Let *r* and *s* be relations on schemas *R* and *S* respectively.
  Then, r ⋈ 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)

Division Operation

- Notation: r ÷ 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 ÷ s is a relation on schema

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

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 ÷ *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

Extended Relational-Algebra-Operations

- Outer Join
- Aggregate Functions

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

$$G1, G2, \ldots, Gn\ \mathcal{g}\ F1(A1), F2(A2), \ldots, Fn(An)\ (E)$$

  - 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

$$\mathcal{G}_{\mathbf{sum(c)}}(\mathrm{r})$$

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

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

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

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

Outer Join

- An *inner* join excludes rows from either table that don't have a matching row in the other table.
- An *outer* join provides the ability to include unmatched rows in the query results.
- The outer join combines the unmatched row in one of the tables with an artificial row for the other table. This artificial row has all columns set to *null*.
- 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.

Outer Join

- Types of Outer Join :

1. LEFT -- only unmatched rows from the left side table (*table-1*) are retained

2. RIGHT -- only unmatched rows from the right side table (*table-2*) are retained

3. FULL -- unmatched rows from both tables (*table-1* and *table-2*) are retained

Outer Joins

- Inner join + rows of one table which do not satisfy the <condition>.

- <u>Left Outer Join:</u> R  S

All rows from R are retained and unmatched rows of S are padded with NULL

- <u>Right Outer Join:</u> R  S

All rows from S are retained and unmatched rows of R are padded with NULL

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

- **Inner 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*

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

Recap of Relational Algebra Operations

TABLE 6.1 OPERATIONS OF RELATIONAL ALGEBRA

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<SELECTION CONDITION>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<ATTRIBUTE LIST>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<JOIN CONDITION>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<JOIN CONDITION>}} R_2$, OR $R_1 \bowtie (\text{<JOIN ATTRIBUTES 1>}), (\text{<JOIN ATTRIBUTES 2>})\ R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<JOIN CONDITION>}} R_2$, OR $R_1 * (\text{<JOIN ATTRIBUTES 1>}), (\text{<JOIN ATTRIBUTES 2>}) R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

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
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same.

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.

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

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

- 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.

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

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

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

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

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

- 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.

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

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

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

Banking Example

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

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

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

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

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

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

Views

- In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)

- 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

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

- 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>

  where <query expression> is any legal relational algebra query expression. The view name is represented by *v.*

- 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**

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

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

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

Updates Through View

- Database modifications expressed as views must be translated to modifications of the actual relations in the database.

- 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:

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

Updates Through Views (Cont.)

- The previous insertion must be represented by an insertion into the actual relation *loan* from which the view *branch-loan* is constructed.

- An insertion into *loan* requires a value for *amount*. The insertion can be dealt with by either.
  - rejecting the insertion and returning an error message to the user.
  - inserting a tuple (“L-37”, “Perryridge”, *null*) into the *loan* relation

- 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* 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 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* if it depends on itself.

Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form

$$\{t \mid P(t)\}$$

- It is the set of all tuples t such that predicate P is true for t

- t is a *tuple variable*, $t[A]$ denotes the value of tuple t on attribute A

- $t \in r$ denotes that tuple t is in relation r

- P is a *formula* similar to that of the predicate calculus

Predicate Calculus Formula

1. Set of attributes and constants

2. Set of comparison operators: (e.g., $<$, $\leq$, $=$, $\neq$, $>$, $\geq$)

3. Set of connectives: and ($\wedge$), or (v), not ($\neg$)

4. Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if true, then y is true

   $$x \Rightarrow y \equiv \neg x \text{ v } y$$

5. Set of quantifiers:
   - $\exists\, t \in r\ (Q(t)) \equiv$ "there exists" a tuple t in relation r such that predicate $Q(t)$ is true
   - $\forall t \in r\ (Q(t)) \equiv Q$ is true "for all" tuples t in relation r

Banking Example

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

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

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 *loan-number, branch-name,* and *amount* for loans of over $1200

$$\{t \mid t \in loan \wedge t\,[amount] > 1200\}$$

- Find the loan number for each loan of an amount greater than $1200

$$\{t \mid \exists s \in \text{loan}\ (t[loan\text{-}number] = s[loan\text{-}number] \wedge s\,[amount] > 1200)\}$$

Notice that a relation on schema [*loan-number*] is implicitly defined by the query

Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus

- Each query is an expression of the form:

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n) \}$$

  - $x_1, x_2, \ldots, x_n$ represent domain variables
  - P represents a formula similar to that of the predicate calculus

Example Queries

- Find the *loan-number, branch-name,* and *amount* for loans of over $1200

$$\{< l, b, a > \mid < l, b, a > \in loan \wedge a > 1200\}$$

- Find the names of all customers who have a loan of over $1200

$$\{< c > \mid \exists\, l, b, a\, (< c, l > \in borrower \wedge < l, b, a > \in loan \wedge a > 1200)\}$$

- Find the names of all customers who have a loan from the Perryridge branch and the loan amount:

$$\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge \exists b(< l, b, a > \in loan \wedge b = \text{“Perryridge”}))\}$$

or $\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge < l, \text{“Perryridge”}, a > \in loan)\}$

End of Unit 2

Thank You……