

## Chapter 3: Relational Model

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

## Basic Structure

- Given sets  $A_1, A_2, \dots, A_n$  a *relation*  $r$  is a subset of  $A_1 \times A_2 \times \dots \times A_n$

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

- 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  $customer-name \times customer-street \times customer-city$

## Relation Schema

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

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

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

*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.

| <i>customer-name</i> | <i>customer-street</i> | <i>customer-city</i> |
|----------------------|------------------------|----------------------|
| Jones                | Main                   | Harrison             |
| Smith                | North                  | Rye                  |
| Curry                | North                  | Rye                  |
| Lindsay              | Park                   | Pittsfield           |

*customer*

## 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:  $\{customer\text{-}name, customer\text{-}street\}$  and  $\{customer\text{-}name\}$  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:  $\{customer\text{-}name\}$  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.** 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-many relationship sets, above super key is also the primary key.

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.

## 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 languages that people use.

# Relational Algebra

- Procedural language
- Six basic operators
  - select
  - project
  - union
  - set difference
  - Cartesian product
  - rename
- The operators take two or more relations as inputs and give a new relation as a result.



## Select Operation

- Notation:  $\sigma_P(r)$

- Defined as: 
$$\sigma_P(r) = \{t \mid t \in r \text{ and } P(t)\}$$

Where  $P$  is a formula in propositional calculus, dealing with terms of the form:

<attribute> = <attribute> or <constant>

≠

>

≥

<

≤

“connected by”:  $\wedge$  (**and**),  $\vee$  (**or**),  $\neg$  (**not**)

## Select Operation – Example

- Relation  $r$ :

| $A$      | $B$      | $C$ | $D$ |
|----------|----------|-----|-----|
| $\alpha$ | $\alpha$ | 1   | 7   |
| $\alpha$ | $\beta$  | 5   | 7   |
| $\beta$  | $\beta$  | 12  | 3   |
| $\beta$  | $\beta$  | 23  | 10  |

- $\sigma_{A=B \wedge D > 5}(r)$

| $A$      | $B$      | $C$ | $D$ |
|----------|----------|-----|-----|
| $\alpha$ | $\alpha$ | 1   | 7   |
| $\beta$  | $\beta$  | 23  | 10  |

## 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

## Project Operation – Example

- Relation  $r$ :

| $A$      | $B$ | $C$ |
|----------|-----|-----|
| $\alpha$ | 10  | 1   |
| $\alpha$ | 20  | 1   |
| $\beta$  | 30  | 1   |
| $\beta$  | 40  | 2   |

- $\Pi_{A,C}(r)$

| $A$                            | $C$          |
|--------------------------------|--------------|
| $\alpha$                       | 1            |
| <del><math>\alpha</math></del> | <del>1</del> |
| $\beta$                        | 1            |
| $\beta$                        | 2            |

=

| $A$      | $C$ |
|----------|-----|
| $\alpha$ | 1   |
| $\beta$  | 1   |
| $\beta$  | 2   |

## 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  $s$ )

## Union Operation – Example

- Relations  $r$ ,  $s$ :

| $A$      | $B$ |
|----------|-----|
| $\alpha$ | 1   |
| $\alpha$ | 2   |
| $\beta$  | 1   |

$r$

| $A$      | $B$ |
|----------|-----|
| $\alpha$ | 2   |
| $\beta$  | 3   |

$s$

- $r \cup s$

| $A$      | $B$ |
|----------|-----|
| $\alpha$ | 1   |
| $\alpha$ | 2   |
| $\beta$  | 1   |
| $\beta$  | 3   |

## 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

## Set Difference Operation – Example

- Relations  $r$ ,  $s$ :

| $A$      | $B$ |
|----------|-----|
| $\alpha$ | 1   |
| $\alpha$ | 2   |
| $\beta$  | 1   |

$r$

| $A$      | $B$ |
|----------|-----|
| $\alpha$ | 2   |
| $\beta$  | 3   |

$s$

- $r - s$

| $A$      | $B$ |
|----------|-----|
| $\alpha$ | 1   |
| $\beta$  | 1   |



## Cartesian-Product Operation

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

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

- 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.

## Cartesian-Product Operation – Example

- Relations  $r$ ,  $s$ :

| $A$      | $B$ |
|----------|-----|
| $\alpha$ | 1   |
| $\beta$  | 2   |

$r$

| $C$      | $D$ | $E$ |
|----------|-----|-----|
| $\alpha$ | 10  | +   |
| $\beta$  | 10  | +   |
| $\beta$  | 20  | -   |
| $\gamma$ | 10  | -   |

$s$

- $r \times s$

| $A$      | $B$ | $C$      | $D$ | $E$ |
|----------|-----|----------|-----|-----|
| $\alpha$ | 1   | $\alpha$ | 10  | +   |
| $\alpha$ | 1   | $\beta$  | 10  | +   |
| $\alpha$ | 1   | $\beta$  | 20  | -   |
| $\alpha$ | 1   | $\gamma$ | 10  | -   |
| $\beta$  | 2   | $\alpha$ | 10  | +   |
| $\beta$  | 2   | $\beta$  | 10  | +   |
| $\beta$  | 2   | $\beta$  | 20  | -   |
| $\beta$  | 2   | $\gamma$ | 10  | -   |

## Composition of Operations

- Can build expressions using multiple operations
- Example:  $\sigma_{A=C}(r \times s)$
- $r \times s$ 
  - Notation:  $r \bowtie s$
  - Let  $r$  and  $s$  be relations on schemas  $R$  and  $S$  respectively. The result is a relation on schema  $R \cup S$  which is obtained by considering 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  $R \cap S$ , a tuple  $t$  is added to the result, where
    - \*  $t$  has the same value as  $t_r$  on  $r$
    - \*  $t$  has the same value as  $t_s$  on  $s$

## Composition of Operations (Cont.)

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  $r$ ,  $s$ :

| $A$      | $B$ | $C$      | $D$ |
|----------|-----|----------|-----|
| $\alpha$ | 1   | $\alpha$ | a   |
| $\beta$  | 2   | $\gamma$ | a   |
| $\gamma$ | 4   | $\beta$  | b   |
| $\alpha$ | 1   | $\gamma$ | a   |
| $\delta$ | 2   | $\beta$  | b   |

$r$

| $B$ | $D$ | $E$        |
|-----|-----|------------|
| 1   | a   | $\alpha$   |
| 3   | a   | $\beta$    |
| 1   | a   | $\gamma$   |
| 2   | b   | $\delta$   |
| 3   | b   | $\epsilon$ |

$s$

- $r \bowtie s$

| $A$      | $B$ | $C$      | $D$ | $E$      |
|----------|-----|----------|-----|----------|
| $\alpha$ | 1   | $\alpha$ | a   | $\alpha$ |
| $\alpha$ | 1   | $\alpha$ | a   | $\gamma$ |
| $\alpha$ | 1   | $\gamma$ | a   | $\alpha$ |
| $\alpha$ | 1   | $\gamma$ | a   | $\gamma$ |
| $\delta$ | 2   | $\beta$  | b   | $\delta$ |

## Division Operation

$$r \div s$$

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

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

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

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

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

# Division Operation – Example

• Relations  $r, s$ :

| $A$        | $B$ |
|------------|-----|
| $\alpha$   | 1   |
| $\alpha$   | 2   |
| $\alpha$   | 3   |
| $\beta$    | 1   |
| $\gamma$   | 1   |
| $\delta$   | 1   |
| $\delta$   | 3   |
| $\delta$   | 4   |
| $\delta$   | 6   |
| $\epsilon$ | 1   |
| $\epsilon$ | 2   |

$r$

| $B$ |
|-----|
| 1   |
| 2   |

$s$

•  $r \div s$

| $A$        |
|------------|
| $\alpha$   |
| $\epsilon$ |



## Another Division Example

- Relations  $r$ ,  $s$ :

| $A$      | $B$ | $C$      | $D$ | $E$ |
|----------|-----|----------|-----|-----|
| $\alpha$ | a   | $\alpha$ | a   | 1   |
| $\alpha$ | a   | $\gamma$ | a   | 1   |
| $\alpha$ | a   | $\gamma$ | b   | 1   |
| $\beta$  | a   | $\gamma$ | a   | 1   |
| $\beta$  | a   | $\gamma$ | b   | 3   |
| $\gamma$ | a   | $\gamma$ | a   | 1   |
| $\gamma$ | a   | $\gamma$ | b   | 1   |
| $\gamma$ | a   | $\beta$  | b   | 1   |

$r$

| $D$ | $E$ |
|-----|-----|
| a   | 1   |
| b   | 1   |

$s$

- $r \div s$

| $A$      | $B$ | $C$      |
|----------|-----|----------|
| $\alpha$ | a   | $\gamma$ |
| $\gamma$ | a   | $\gamma$ |

## 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 the result of the query.
- Assignment must always be made to a temporary relation variable.
- Example: Write  $r \div s$  as

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

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

$$result = temp1 - temp2$$

- 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.

## Example Queries

- Find all customers who have an account from at least the “Downtown” and “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

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

## Example Queries

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

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

## 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 ( $\vee$ ), not ( $\neg$ )
4. Implication ( $\Rightarrow$ ):  $x \Rightarrow y$ , if  $x$  is true, then  $y$  is true

$$x \Rightarrow y \equiv \neg x \vee y$$

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-city)*

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

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

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

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

## Example Queries

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

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

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

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

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



## Example Queries

- Find the names of all customers having a loan, an account, or both at the bank

$$\{t \mid \exists s \in \text{borrower}(t[\text{customer-name}] = s[\text{customer-name}]) \\ \vee \exists u \in \text{depositor}(t[\text{customer-name}] = u[\text{customer-name}])\}$$

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

$$\{t \mid \exists s \in \text{borrower}(t[\text{customer-name}] = s[\text{customer-name}]) \\ \wedge \exists u \in \text{depositor}(t[\text{customer-name}] = u[\text{customer-name}])\}$$

## Example Queries

- Find the names of all customers having a loan at the Perryridge branch

$$\{t \mid \exists s \in \text{borrower}(t[\text{customer-name}] = s[\text{customer-name}] \\ \wedge \exists u \in \text{loan}(u[\text{branch-name}] = \text{"Perryridge"} \\ \wedge u[\text{loan-number}] = s[\text{loan-number}]))\}$$

- Find the names of all customers who have a loan at the Perryridge branch, but no account at any branch of the bank

$$\{t \mid \exists s \in \text{borrower}(t[\text{customer-name}] = s[\text{customer-name}] \\ \wedge \exists u \in \text{loan}(u[\text{branch-name}] = \text{"Perryridge"} \\ \wedge u[\text{loan-number}] = s[\text{loan-number}]) \\ \wedge \nexists v \in \text{depositor}(v[\text{customer-name}] = t[\text{customer-name}])\}$$

## Example Queries

- Find the names of all customers having a loan from the Perryridge branch and the cities they live in

$$\{t \mid \exists s \in \text{loan} (s[\text{branch-name}] = \text{"Perryridge"} \\ \wedge \exists u \in \text{borrower} (u[\text{loan-number}] = s[\text{loan-number}] \\ \wedge t[\text{customer-name}] = u[\text{customer-name}] \\ \wedge \exists v \in \text{customer} (u[\text{customer-name}] = v[\text{customer-name}] \\ \wedge t[\text{customer-city}] = v[\text{customer-city}])))\}$$

## Example Queries

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$\{t \mid \forall s \in \text{branch} (s[\text{branch-city}] = \text{"Brooklyn"} \Rightarrow \\ \exists u \in \text{account} (s[\text{branch-name}] = u[\text{branch-name}] \\ \wedge \exists s \in \text{depositor} (t[\text{customer-name}] = s[\text{customer-name}] \\ \wedge s[\text{account-number}] = u[\text{account-number}])))\}$$

## Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example,  $\{t \mid \neg t \in r\}$  results in an infinite relation if the domain of any attribute of relation  $r$  is infinite
- To guard against the problem, we restrict the set of allowable expressions to *safe* expressions.
- An expression  $\{t \mid P(t)\}$  in the tuple relational calculus is *safe* if every component of  $t$  appears in one of the relations, tuples, or constants that appear in  $P$

## Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus.
- Each query is an expression of the form:

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid P(x_1, x_2, \dots, x_n) \}$$

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

## Example Queries

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

$$\{ \langle b, l, a \rangle \mid \langle b, l, a \rangle \in \text{loan} \wedge a > 1200 \}$$

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

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

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

$$\{ \langle c, a \rangle \mid \exists l (\langle c, l \rangle \in \text{borrower} \wedge \exists b (\langle b, l, a \rangle \in \text{loan} \wedge b = \text{"Perryridge"})) \}$$

## Example Queries

- Find the names of all customers having a loan, an account, or both at the Perryridge branch:

$$\{ \langle c \rangle \mid \exists l (\langle c, l \rangle \in \text{borrower} \\ \wedge \exists b, a (\langle b, l, a \rangle \in \text{loan} \wedge b = \text{"Perryridge"})) \\ \vee \exists a (\langle c, a \rangle \in \text{depositor} \\ \wedge \exists b, n (\langle b, a, n \rangle \in \text{account} \wedge b = \text{"Perryridge"})) \}$$

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$\{ \langle c \rangle \mid \forall x, y, z (\langle x, y, z \rangle \in \text{branch} \wedge y = \text{"Brooklyn"}) \Rightarrow \\ \exists a, b (\langle x, a, b \rangle \in \text{account} \wedge \langle c, a \rangle \in \text{depositor}) \}$$



## Safety of Expressions

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid P(x_1, x_2, \dots, x_n) \}$$

is safe if all of the following hold:

1. All values that appear in tuples of the expression are values from  $dom(P)$  (that is, the values appear either in  $P$  or in a tuple of a relation mentioned in  $P$ ).
2. For every “there exists” subformula of the form  $\exists x (P_1(x))$ , the subformula is true if and only if there is a value  $x$  in  $dom(P_1)$  such that  $P_1(x)$  is true.
3. For every “for all” subformula of the form  $\forall x (P_1(x))$ , the subformula is true if and only if  $P_1(x)$  is true for all values  $x$  from  $dom(P_1)$ .

## Extended Relational-Algebra-Operations

- Generalized Projection
- Outer Join
- Aggregate Functions

## Generalized Projection

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

$$\Pi_{F_1, F_2, \dots, F_n}(E)$$

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

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

## Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that do 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 **false** by definition.

## Outer Join – Example

- Relation *loan*

| <i>branch-name</i> | <i>loan-number</i> | <i>amount</i> |
|--------------------|--------------------|---------------|
| Downtown           | L-170              | 3000          |
| Redwood            | L-230              | 4000          |
| Perryridge         | L-260              | 1700          |

- Relation *borrower*

| <i>customer-name</i> | <i>loan-number</i> |
|----------------------|--------------------|
| Jones                | L-170              |
| Smith                | L-230              |
| Hayes                | L-155              |

## Outer Join – Example

- *loan* ⋈ *Borrower*

| <i>branch-name</i> | <i>loan-number</i> | <i>amount</i> | <i>customer-name</i> |
|--------------------|--------------------|---------------|----------------------|
| Downtown           | L-170              | 3000          | Jones                |
| Redwood            | L-230              | 4000          | Smith                |

- *loan* ⋈<sub>r</sub> *borrower*

| <i>branch-name</i> | <i>loan-number</i> | <i>amount</i> | <i>customer-name</i> | <i>loan-number</i> |
|--------------------|--------------------|---------------|----------------------|--------------------|
| Downtown           | L-170              | 3000          | Jones                | L-170              |
| Redwood            | L-230              | 4000          | Smith                | L-230              |
| Perryridge         | L-260              | 1700          | <i>null</i>          | <i>null</i>        |

## Outer Join – Example

- $loan \bowtie B$  Borrower

| <i>branch-name</i> | <i>loan-number</i> | <i>amount</i> | <i>customer-name</i> |
|--------------------|--------------------|---------------|----------------------|
| Downtown           | L-170              | 3000          | Jones                |
| Redwood            | L-230              | 4000          | Smith                |
| null               | L-155              | null          | Hayes                |

- $loan \bowtie B$  borrower

| <i>branch-name</i> | <i>loan-number</i> | <i>amount</i> | <i>customer-name</i> |
|--------------------|--------------------|---------------|----------------------|
| Downtown           | L-170              | 3000          | Jones                |
| Redwood            | L-230              | 4000          | Smith                |
| Perryridge         | L-260              | 1700          | <i>null</i>          |
| null               | L-155              | null          | Hayes                |

## Aggregate Functions

- Aggregation operator  $\mathcal{G}$  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

$$G_1, G_2, \dots, G_n \mathcal{G} F_1 A_1, F_2 A_2, \dots, F_m A_m (E)$$

- $E$  is any relational-algebra expression
- $G_1, G_2, \dots, G_n$  is a list of attributes on which to group
- $F_i$  is an aggregate function
- $A_i$  is an attribute name



## Aggregate Function – Example

- Relation  $r$ :

| $A$      | $B$      | $C$ |
|----------|----------|-----|
| $\alpha$ | $\alpha$ | 7   |
| $\alpha$ | $\beta$  | 7   |
| $\beta$  | $\beta$  | 3   |
| $\beta$  | $\beta$  | 10  |

- $\text{sum}_C(r)$

|                |
|----------------|
| $\text{sum-C}$ |
| 27             |

## Aggregate Function – Example

- Relation *account* grouped by *branch-name*:

| <i>branch-name</i> | <i>account-number</i> | <i>balance</i> |
|--------------------|-----------------------|----------------|
| Perryridge         | A-102                 | 400            |
| Perryridge         | A-201                 | 900            |
| Brighton           | A-217                 | 750            |
| Brighton           | A-215                 | 750            |
| Redwood            | A-222                 | 700            |

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

| <i>branch-name</i> | <i>sum-balance</i> |
|--------------------|--------------------|
| Perryridge         | 1300               |
| Brighton           | 750                |
| Redwood            | 700                |

## 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.

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

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

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

- Delete all accounts at branches located in Needham.

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

## 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.

$$\begin{aligned} \text{account} &\leftarrow \text{account} \cup \{(\text{"Perryridge"}, \text{A-973}, 1200)\} \\ \text{depositor} &\leftarrow \text{depositor} \cup \{(\text{"Smith"}, \text{A-973})\} \end{aligned}$$

- 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.

$$\begin{aligned} r_1 &\leftarrow (\sigma_{\text{branch-name}=\text{"Perryridge"}} (\text{borrower} \bowtie \text{loan})) \\ \text{account} &\leftarrow \text{account} \cup \Pi_{\text{branch-name}, \text{loan-number}, 200} (r_1) \\ \text{depositor} &\leftarrow \text{depositor} \cup \Pi_{\text{customer-name}, \text{loan-number}} (r_1) \end{aligned}$$

## Updating

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

$$r \leftarrow \Pi_{F_1, F_2, \dots, F_n}(r)$$

- 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.

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

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

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

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

## 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

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

- Any relation that is not part 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 to be 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**

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

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

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

## Updates Through Views

- 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**

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

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

$branch-loan \leftarrow branch-loan \cup \{("Perryridge", L-37)\}$

## 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 (“Perryridge”, L-37, *null*) into the *loan* relation

## 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 and only if there is a path in the dependency graph from  $v_2$  to  $v_1$ .
- A view relation  $v$  is said to be *recursive* 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.