Chapter 3
Relational Model

3. Relational Model

3.1 Structure of a relational database

3.2 The relational algebra

3.2.1 Select, Project, Product, Union, Difference, Rename, Intersection, Division, Assignment, Natural Join, Outer Join, Aggregate functions, Generalized projection.

3.2.2 Database manipulation: Insertion, Update, and Deletion

Structure of Relational Databases

- A relational database consists of collection of tables, each of which is assigned a unique name.
- A row in a table represents a relationship among a set of values.
- A table is a collection of relationships.
- The concept of table closely corresponds to mathematical concept of relations.
- Thus, the relational model represents the database of collection of relations.

Basic Structure (1)

- For each attributes, there is a set of permitted values, called the domain of that attribute.
- E.g.: for branch-name attribute, the domain is the set of all branch names.
- If account has D1 that is set of all account numbers, D2 is set of all branch-name, and D3 the set of all balance,

  then any row of account must consist of a 3-tuple(v_1,v_2,v_3),
  - Where, v_1 is an account number that is in domain D1
  - v_2 is a branch-name that is in domain D2
  - v_3 is a balance that is in domain D3
- Therefore: account is a subset of all possible rows. Thus, account is a subset of

  D1 x D2 x D3
- In general a table of n attributes must be a subset of

  $D_1 \times D_2 \times D_3 \times \ldots . \times D_{n-1} \times D_n$

Basic Structure (2)

- Formally, given sets $D_1, D_2, \ldots D_n$ domains, a relation r is a subset of $D_1 \times D_2 \times \ldots\ldots D_n$
- *In otherwords, Mathematically, a relation is a subset of cartesian product of a list of domains.*
- *Thus a relation is a set of n-tuples $(a_1, a_2, .., a_n)$, $a_i \in D_i$*

Example: if
customer-name = {Jones, Smith, Curry, Lindsay}
customer-street = {Main, North, Park}
customer-city = {Munich, Stuttgart, Berlin}
Then r = {(Jones, Main, Munich),
(Smith, North, Stuttgart),
(Curry, North, Stuttgart),
(Lindsay, Park, Berlin)}
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 that attribute.
- Attribute values are (normally) required to be atomic, that is, indivisible.
  - E.g. Multivalued attributes values are not atomic
  - E.g. Composite attributes values are not atomic
  - The special value null is a member of every domain.

Relational Schema

- $A_1, A_2, .., A_n$ are attributes
- $R=(A_1, A_2, .., A_n)$ is a relation schema
  - E.g.: Customer-schema=(customer-name, customer street, customer city)
- r(R) is a relation on relation schema R
  - E.g.: customer(Customer-Schema)
  - means *customer* is a relation on Customer-schema by *customer(Customer-schema)*

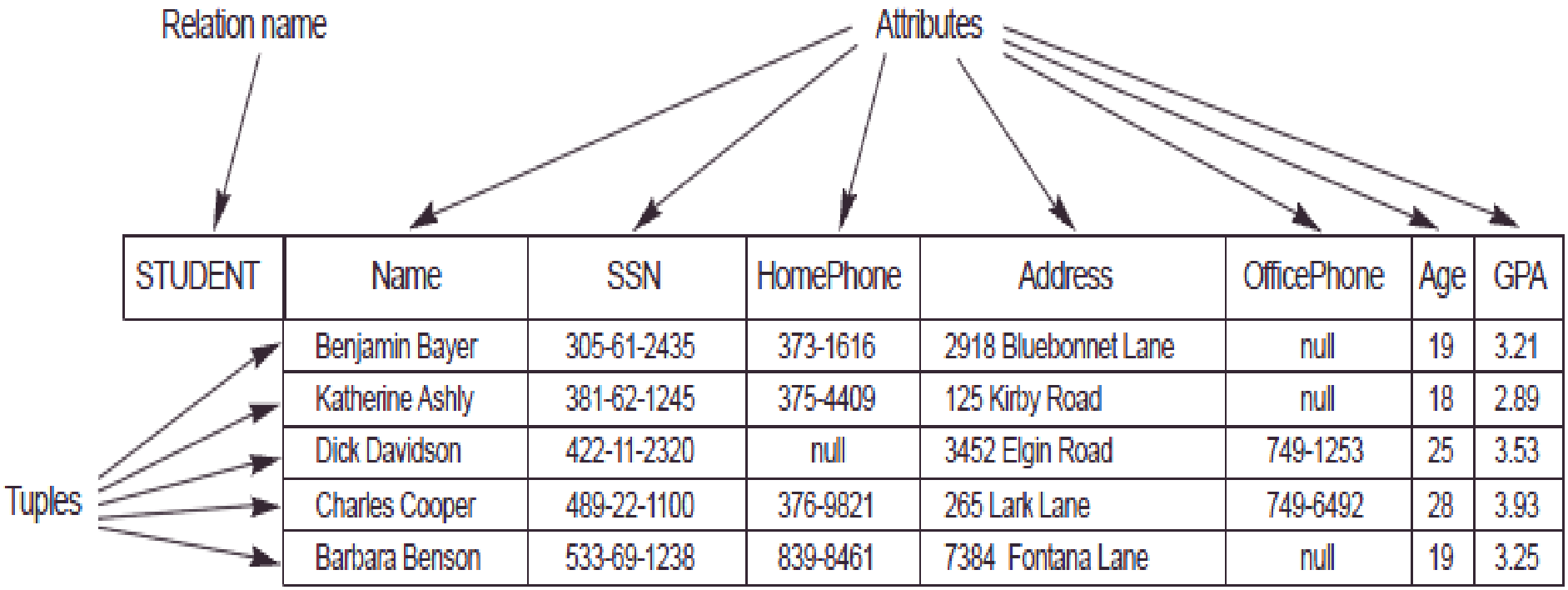
Naming convention

- Relation schema names begin with uppercase
- Relation names are in lowercase

Relation Instance

- The current values of a relation are specified by a table. Thus, a relation state at a given time is called a relation instance.
- An element t of r is a tuple, represent by row in a table.

Attributes and Tuples of a relation Student



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Relation student is

student(Name, SSN, HomePhone, Address, OfficePhone, Age, GPA)

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)

The *customer* relation

customer-name	*customer-street*	*customer-city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

The *depositor* relation

customer-name	*account-number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Keys (1)

- Let R be a relation schema.
- *If a subset K of R is a superkey for R, then in relation r(R), no two distinct tuples have the same values on all attributes in K. That is, if t_1 and t_2 are in r and $t_1 \neq t_2$, then $t_1[K] \neq t_2[K]$*
- *Example: {branch-name, branch-city}*

  {branch-name}

  Are both superkeys of branch, if no two branches can possibly have the same name.
- K is a candidate key if K is minimal.
- Example: *branch-name* is a candidate key for branch, since it is a superkey (assuming no two branch 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 union of the primary key of the strong entity set and the discriminator of the weak entity set.
- Relationship set: The union of 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 "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 primary keys of entity sets becomes the relation's primary key.

Determining Keys from E-R sets

- Multivalued attributes:
  - The primary key of entity set or relationship set, together with the attribute C becomes the relation‘s primary key. Where C is a column which holds an individual value of multivalued attribute M.
- Combined attributes:
  - Binary many-to-one relationship set from entity set A to entity set B can be represented by a table consisting of the attribute of A and attributes of the relationship set. The primary key of the “many” entity set A becomes the primary key of the relation.
  - Binary one-to-one relationship set from entity set A to entity set B can be constructed like in many-to-one. But either entity set‘s primary key is chosen as the primary key of the relation because both keys are candidate keys.

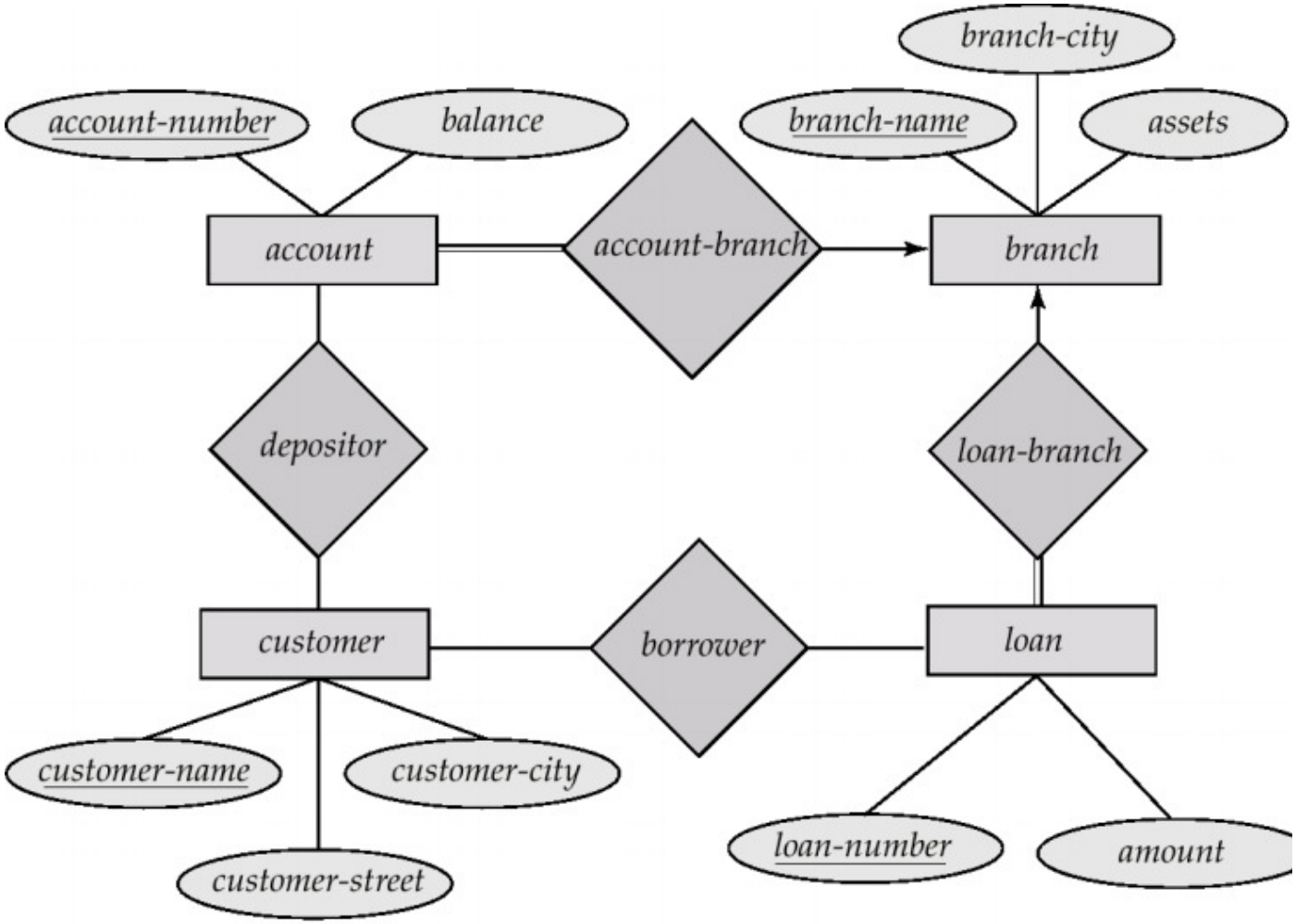
Foreign Key

- A relation schema *r1* may include among its attributes, the primary key of another relation schema *r2*. This attribute is called a foreign key from *r1*, referencing *r2*.
- The relation *r1* is also called the referencing relation of the foreign key.
- The relation *r2* is also called the referenced relation of the foreign key.
- Eg. Branch-name in Account-schema is a foreign key from Account-schema referencing Branch-schema where branch-name is the primary key of Branch-schema.
- Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

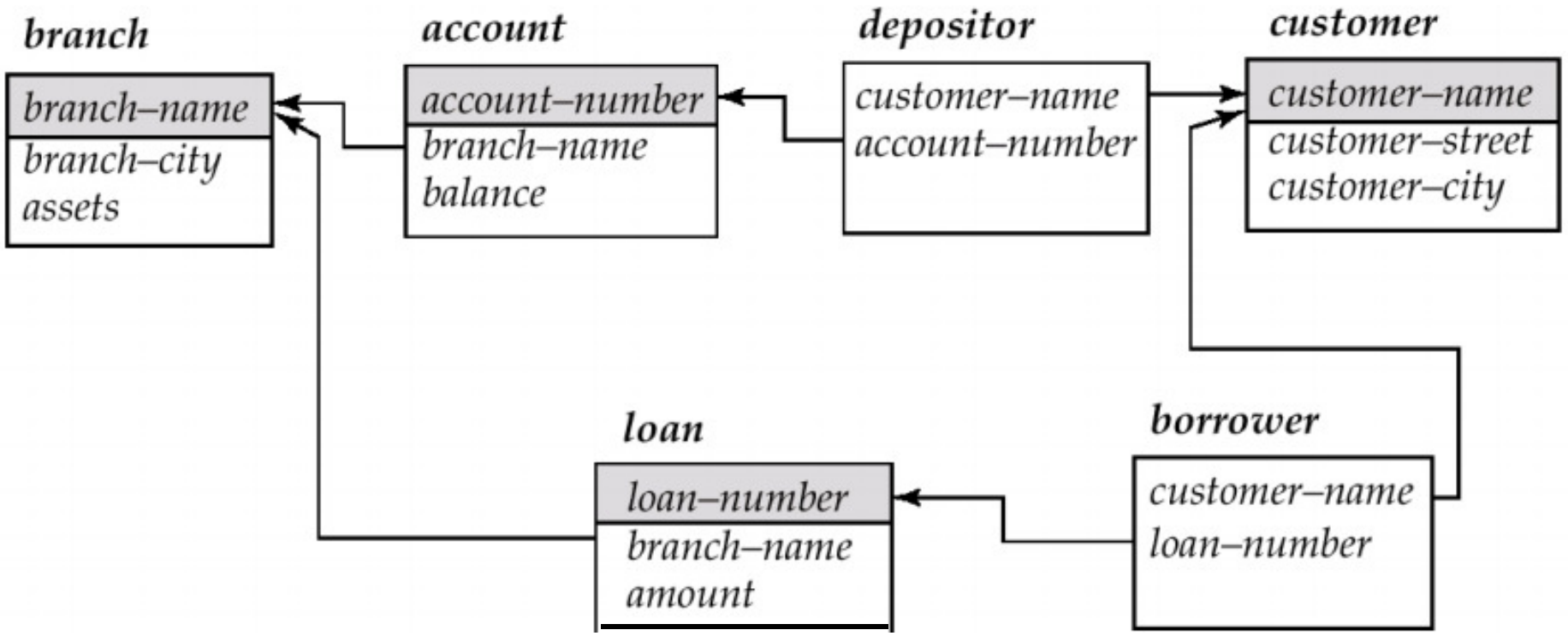
Schema diagram

- A database schema along with primary key and foreign key, can be depicted pictorially by schema diagram.
- Boxes represents *relations*
- Above boxes, are *relation names*
- Inside boxes, are *list of attributes*
- Horizontal line crosses the box for *primary key*
- Arrows for the *foreign key dependencies* from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

E-R Diagram for the Banking Enterprise



Schema Diagram for the banking enterprise



Query Languages

- Query Language is a language in which user requests information from the database.
- Categories of languages
  - Procedural: user instruct the system to *perform a sequence of operations* on the database to compute desired result.
  - non-procedural : user describes the desired information *without giving a specific procedure* for obtaining information.
- Procedural languages:
  - Relational Algebra (procedural language)
- Non-Procedural languages:
  - Tuple Relational Calculus
  - Domain Relational Calculus
- Pure languages form underlying basis of query languages(e.g. SQL) that people use.

Relational Algebra

- Relational Algebra is a procedural query language.
- Consists of set of operations
  - that takes one or more relations as input and
  - produces a new realtion as their results
- Fundamental operations

  Select σ
  Project Π
  Union $\cup$
  Set difference $-$
  Cartesian product x
  Rename ρ

- Other operations

  Set intersection $\cap$
  Natural join $\bowtie$
  Division $\div$
  Assignment $\leftarrow$

Banking example

Consider the relation database of Bank

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)

Give an expression in the relational algebra to express queries.

The *Select* Operation (σ)

- 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

The *Select* Operation (σ) contd..

- The select operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- p is called the selection predicate (selection condition).
- Predicate may consists of following clauses
  - <attribute name> <comparison operator> <attribute name>
  - <attribute name> < comparison operator > <constant value>

  Comparision operators in predicate

  $=, \neq, <, >, \leq, \geq$

  Connectives in predicate

  $\wedge$ (And), $\vee$ (or), $\neg$ (Not)

The *Select* Operation (σ) contd..

E.g. Find all loans that belongs to branch name "Putalisadak".

E.g. Find all loans that with loan amount greater than 1200.

E.g. Find all loans that belongs to branch name "Putalisadak" and loan amount greater than 1200.

Loan-number	Branch-name	amount
L-11	Biratnagar	900
L-14	Pokhara	1500
L-15	Putalisadak	1500
L-16	Putalisadak	1300
L-17	Pokhara	1000
L-23	Pulchowk	2000
L-93	Birgunj	500

E.g. Find all customers who have the same name as their loan officer. *(Consider a relation loan-officier which has columns customer-name, banker-name and loan-number)*

The *Select* Operation (σ) contd..

- The selection operation is commutative; that is

  $\sigma_{<cond1>}(\sigma_{<cond2>}(r)) = \sigma_{<cond2>}(\sigma_{<cond1>}(r)$

- The cascaded SELECT operation can be combined to a single SELECT operation with a conjuctive AND($\wedge$).

  $\sigma_{<cond1>}(\sigma_{<cond2>}(\ldots. \sigma_{<condn>}(r)) = \sigma_{<cond1> \wedge <cond2> \wedge \ldots \wedge <condn>}(r)$

The *Project* Operation (Π)

- 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

The *Project* Operation (∏) contd..

- Notation $\Pi_{A1, A2, \ldots Ak}(r)$
  where A1, A2, ..Ak are attribute names and *r* is a relation.
- Selects certain columns from the table and discards other columns.
- 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 relation is a set.

Example:List all loan numbers and the amount of the loan

$$\Pi_{loan\text{-}number,\ amount}(loan)$$

- Commutativity does not hold on PROJECTION

$$\Pi_{<attribute\ list1>}(\Pi_{<attribute\ list2>}(r)) = \Pi_{<attribute\ list1>}(r)$$

Combination of relational operators

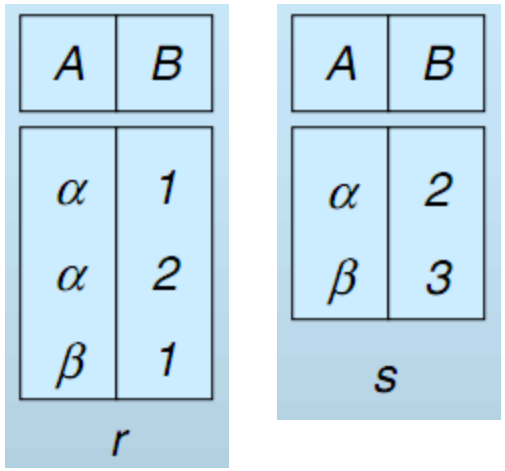
- The result of a relational algebra operation is itself a relation.
- Can build a relational-algebra expression using multiple operations
- E.g. Find all the customer who lives in “Harrison”.

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

$$\Pi_{\textit{customer-name}} (\sigma_{\textit{customer-city=“Harrison”}} (r))$$

The *Union* Operation (∪)

- Relation *r*, *s*



A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- *r* ∪ *s*

A	B
α	1
α	2
β	1
β	3

The *Union* Operation (∪) contd..

- Notation: r ∪ s
- The result of this operation is a relation that includes all tuples that are either in *r* or in *s* or in both.
- Duplicate tuples are eliminated.
- Union must be taken between compatible relations,

  that means, for r ∪ s to be valid, the two conditions must hold.
  1. The *relations r and s* must have the *same arity*. That is, they must have *same number of attributes*.
  2. The domains of the i^{th} attribute of r and the i^{th} attribute of s must be the same, for all *i*.
- Set union is commutative, that is

  $r \cup s = s \cup r$
- E.g. to find all customers with either an account or a loan

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

Set Difference (—)

- 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 (—) contd..

- Notation: $r - s$
- The result of this operation is a relation that includes all tuples that are in *r* but not in *s*.
- Set differences must be taken between compatible relations.
  1. The *relations r and s* must have the *same arity*. That is, they must have *same number of attributes*.
  2. The domains of the i^{th} attribute of r and the i^{th} attribute of s must be the same, for all *i*.
- Set difference is not commutative, that is

$$r - s \neq s - r$$

Example:

E.g. to find all customers who have an account but not a loan

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

Cartesian Product (x)

- 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 (x) contd..

- Notation *r1 x r2*
- Also called *cross product*
- This operation is used to combine tuples from any two relations.
- In general, if *relations r1(R1) and r2(R2), then r1 x r2 is a relation*
  1. whose schema is the concatenation of R1 and R2.
  2. which contains all the tuples t for which there is a tuple t1 in r1 and a tuple t2 in r2 for which t[R1]=t1[R1] and t[R2]=t2[R2]

- *If we have n1 tuples r1 and n2 tuples in r2, then there are n1 * n2 ways of choosing pair of tuples, so there is n1 * n2 tuples in r, where r=r1 x r2.*

Cartesian Product (x) contd..

- If attributes of relation schemas R1 and R2 are not disjoint, then,
  - such attributes are used by attaching to an attribute the name of the relation from which the attribute belongs to.

  OR
  - the renaming must be done.
- *Example:*
  borrower (customer-name, loan-number)
  loan (loan-number, branch-name, amount)

  The relation schema for r= borrower x loan is

  (borrower.customer-name, borrower.loan-number, loan.loan-number, loan.branch-name, loan.amount)

  This can also be written without any ambiguity
  (customer-name, borrower.loan-number, loan.loan-number, branch-name, amount)

Cartesian Product (x) Example

Find names of all customers who have a loan at Putalisadak branch.

- Step 1: we need information from both customer and loan tables

  Borrower x loan

- Step 2: Find tuples which belongs to branch "Putalisadak"

  $\sigma_{branch.name="Putalisadak"}$(Borrower x loan)

- Step 3: Find tuples that pertain to customers who have loan at Putalisadak branch

$\sigma_{borrower.loan\text{-}number=loan.loan\text{-}number}$(
$\sigma_{branch.name="Putalisadak"}$(Borrower x loan))

Step 4: We only want to show customer-name

$\Pi_{customer\text{-}name}$(
$\sigma_{borrower.loan\text{-}number=loan.loan\text{-}number}$(
$\sigma_{branch.name="Putalisadak"}$(Borrower x loan)))

Example ....

Find names of all customers who have a loan at Putalisadak branch.

customer-name	loan-number
Shankar	L-15
John	L-16
Ashok	L-93
Anil	L-14

Table *borrower*

loan-numberc	**branch-name**	**amount**
L-14	Pokhara	1500
L-15	Putalisadak	1500
L-16	Putalisadak	1300
L-93	Birgunj	500

Table *Loan*

Borrower x loan

customer-name	borrower.loan -number	loan.loan -number	branch-name	amount
Shankar	L-15	L-14	Pokhara	1500
Shankar	L-15	L-15	Putalisadak	1500
Shankar	L-15	L-16	Putalisadak	1300
Shankar	L-15	L-93	Birgunj	500
John	L-16	L-14	Pokhara	1500
John	L-16	L-15	Putalisadak	1500
John	L-16	L-16	Putalisadak	1300
John	L-16	L-93	Birgunj	500
Ashok	L-93	L-14	Pokhara	1500
Ashok	L-93	L-15	Putalisadak	1500
Ashok	L-93	L-16	Putalisadak	1300
Ashok	L-93	L-93	Birgunj	500
Anil	L-14	L-14	Pokhara	1500
Anil	L-14	L-15	Putalisadak	1500
Anil	L-14	L-16	Putalisadak	1300
Anil	L-14	L-93	Birgunj	500

$\sigma_{branch.name="Putalisadak"}$(Borrower x loan)

customer-name	borrower.loan -number	loan.loan -number	branch-name	amount
Shankar	L-15	L-14	Pokhara	1500
Shankar	L-15	L-15	Putalisadak	1500
Shankar	L-15	L-16	Putalisadak	1300
Shankar	L-15	L-93	Birgunj	500
John	L-16	L-14	Pokhara	1500
John	L-16	L-15	Putalisadak	1500
John	L-16	L-16	Putalisadak	1300
John	L-16	L-93	Birgunj	500
Ashok	L-93	L-14	Pokhara	1500
Ashok	L-93	L-15	Putalisadak	1500
Ashok	L-93	L-16	Putalisadak	1300
Ashok	L-93	L-93	Birgunj	500
Anil	L-14	L-14	Pokhara	1500
Anil	L-14	L-15	Putalisadak	1500
Anil	L-14	L-16	Putalisadak	1300
Anil	L-14	L-93	Birgunj	500

Step 3

customer-name	borrower.loan -number	loan.loan -number	branch-name	amount
Shankar	L-15	L-15	Putalisadak	1500
Shankar	L-15	L-16	Putalisadak	1300
John	L-16	L-15	Putalisadak	1500
John	L-16	L-16	Putalisadak	1300
Ashok	L-93	L-15	Putalisadak	1500
Ashok	L-93	L-16	Putalisadak	1300
Anil	L-14	L-15	Putalisadak	1500
Anil	L-14	L-16	Putalisadak	1300

$\sigma_{borrower.loan\text{-}number=loan.loan\text{-}number}($
$\sigma_{branch.name=\text{“}Putalisadak\text{”}}$(Borrower x loan))

Step 4

customer-name	borrower.loan -number	loan.loan -number	branch-name	amount
Shankar	L-15	L-15	Putalisadak	1500
John	L-16	L-16	Putalisadak	1300

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

Rename (ρ)

- The symbol ρ (rho) denotes the rename operator.
- Allows us to name the results of relational algebra expression E

  $\rho_x(E)$ returns the result of expression under name x.
- Allows rename either the relation name, attribute names or both.
- Thus, if a relation algebra expression E has arity n (i.e. n columns), then

  $\rho_{x(A1, A2,\ldots,An)}(E)$ returns the result of expression under name x and attributes renamed to A1, A2, .., An

  $\rho_{x(A1, A2,\ldots,An)}(r)$ renames both relation and its attributes.

  $\rho_x(r)$ renames relation only.

  $\rho_{(A1, A2,\ldots,An)}(r)$ renames attributes only.

If the attributes of r are C1, C2, ..., Cn in that order, then each C_i is renamed as A_i .

Rename (ρ) contd..

- Find a largest account balance in the bank

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

Rename (ρ) contd..

Step 1. gives those balances except the largest balance

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

Step 2. applying set difference gives the largest balance in a bank

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

Rename (ρ) contd..

Find the names of all customers who live on the same street and city as smith.

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

Rename (ρ) contd..

Step 1. get Smith‘s street-name and city by an expression

$$\Pi_{customer\text{-}street,\ customer\text{-}city}(\sigma_{customer\text{-}name=\text{“}Smith\text{”}}(customer))$$

Step 2. find other customers who lives in Smith‘s street and city

$$\Pi_{account.balance}(\sigma_{customer.\ customer\text{-}street\ =smith\text{-}addr.street}$$

$$\wedge\ customer.customer\text{-}city=smith\text{-}addr.city$$

$$(\text{customer} \times \rho_{\text{smith-addr(street, city)}}$$

$$(\Pi_{customer\text{-}street,\ customer\text{-}city}(\sigma_{customer\text{-}name=\text{“}Smith\text{”}}$$

$$(customer)))))$$

Formal Definition of the Relational Algebra

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- A constant relation is written by listing its tuples within { }.
  e.g. account ={ (A-101, Biratnagar, 500) (A-215, Pokhara, 700)}

A general expression in relational algebra composed of smaller expressions.

Let E1 and E2 are relational algebra expressions. Then, relation algebra expressions are

a. E1 ∪ E2

b. E1 – E2

c. E1 x E2

d. $\sigma_p(E)$, where *P* is a predicate on attributes in *E1*

e. $\Pi_S(E1)$, where *S is a list consisting of some attributes*

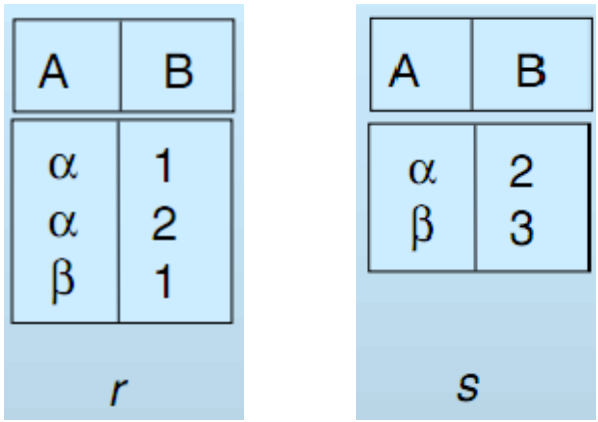
f. $\rho_x(E1)$, where x is the new name for the result of *E1*

Additional Operations

- The fundamental operations of relational algebra are sufficient to express any query.
- Certain common queries are lenghty to express only by using fundamental operations
- Additional operations that do not add any power to the relational algebra, but that simplify common queries.
- Additional Algebra Operations
  - Set intersection $\cap$
  - Natural join $\bowtie$
  - Division $\div$
  - Assignment $\leftarrow$

Set-Intersection Operation ∩

- *Relation r, s:*



A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- *r* ∩ *s*

A	B
α	2

Set-Intersection Operation ∩

- Notation: $r \cap s$
- The result of this operation, denoted by r ∩ s, is a relation that includes all tuples that are in both r and s.
- For intersection, *r* and *s* must be compatible relations.

E.g. Find all customers who have both a loan and an account.

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

Note: The equivalent relational algebra expressions

$$r \cap s = r - (r - s)$$

Natural-Join Operation ⋈

- Notation: r⋈s
- Consider two relations *r(R)* and *s(S)* where R and S are schemas.
  - *The result is a relation on schema R ∪ 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 ∩ 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.*

Example:

R=(A, B, C, D)

S=(E, B, D)

Result schema=(A, B, C, D, E)

r ⋈ s is defined as

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

Natural-Join Operation ⋈

- Relation r, s

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ϵ

s

- $r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Natural-Join Operation

- *Consider two relations r(R) and s(S)*
- *The natural join of r and s, denoted by* $r \bowtie s$*, is a relation on schema* $R \cup S$ formally defined as follows

$$r \bowtie s = \Pi_{R \cup S} (\sigma_{r.A_1=s.A_1 \wedge r.A_2=s.A_2 \wedge .. \wedge r.A_n=s.A_n} (r \times s))$$

Where $R \cap S = \{A_1, A_2, .. , A_n\}$

Table *account*

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

Table *depositor*

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

Table *customer*

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

1. Find the names of all branches with customers who have an account in the bank and who live in harrison.
2. Find all the customer who have both a loan and an account at the bank.

theta-Join Operation

The theta join is an extension to the natural join operation which allows to combine a selection and a cartesian product.

Consider relations *r(R)* and *s(S)*, θ be the predicate on attributes in the schema R ∪ S. Then theta join operation $r \bowtie_\theta s$ is defined as

$$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

E.g.

$$\text{Department} \bowtie_{\text{MGRSSN=SSN}} \text{Employee}$$

Division Operation ÷

- Division operation denoted by ÷ is suited to queries that includes the phrase "for all".
- Formally, let two relations r(R) and s(S) and let $S \subseteq R$. That is, every attributes of schema S is also in R.

  The relation r ÷ s is a relation on schema R – S.

  A tuple t is in r ÷ s if and only if both of two conditions holds:

  1. *t is in* $\Pi_{R-S}(r)$
  2. *For every tuple* t_s *in s, there is a tuple* t_r *in r satisfying both of following:*

     a. $t_r[S] = t_s[S]$

     b. $t_r[R-S] = t$

Division Operation ÷

Example 1

- Relation r, s

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

r

B
1
2

s

- *r ÷ s*

A
α
β

1. *t is in Π_{R-S} (r)*
2. *For every tuple t_s in s, there is a tuple t_r in r satisfying both of following:*
   a. t_r [S] = t_s [S]
   b. t_r [R-S] = t

Division Operation ÷

Example 2

- Relation 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

Example

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

Branch-name	Branch-city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
..		

Table *branch*

Assignment Operators ←

- The assignment operation (←) 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 r ÷ s can be written as

  temp1 ← Π_{R-S} *(r)*

  temp2 ← Π_{R-S} *(temp1 x s)* − $\Pi_{R-S\ ,S}$ *(r))*

  Result ← temp1 − temp2
- The result to the right of the ← is assigned to the relation variable on the left of the ←.

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.

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

- E is any relational-algebra expression.
- Each of $F_1, F_2, \ldots, F_n$ is an arithmetic expression involving constants and attributes in the schema of E.
- An arithmatic expression may be simply an attributes or a constant.

Generalized Projection contd..

Table *employee*

ssn	name	salary	tax
111221	John	10000	0.05
511249	Smith	20000	0.1

Write a query to
find the employee net earning.

Table *result of the expression*

ssn	name	net-salary
111221	John	
511249	Smith	

Aggregate functions

- Aggregation function takes a collection of values and returns single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- General form of the aggregation operation

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

- *E* is any relational-algebra expression
- $G_1, G_2, \ldots, G_n$ is 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 functions- Example

- Relation r

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

sum-C
27

Aggregate functions- Example

Table *account*

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

Write an algebra expression
to find sum of all balance that belongs to each branch.

branch-name	*balance*
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate functions contd..

- 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

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

Aggregate functions contd..

A	B	value
a1	*a1*	*11*
a1	*a1*	*100*
b1	*a1*	*22*
b1	*a1*	*30*
a1	*a1*	*6*
b1	*a1*	*50*

table *r*

Write an algerbra expression to get following result relation from table *r*.

Result a.

B	mininum	maximum
a1	11	100
a1	22	50

Result b.

A	B	mininum	maximum
a1	a1	11	100
b1	a1	6	50

Outer Join

- 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 comparison between nulls are false.

Inner Join

Table *loan*

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

Table *customer*

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

Inner join:

loan ⋈ customer

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

Left OuterJoin

The left outer join ⟕ of any two relations is
- the result of natural join plus
- all tuples in the left relation that did not match with any tuples in the right relations, pads the tuples with null values for all other attributes

Table *loan*

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

Table *customer*

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

Left outer join:

loan ⟕ customer

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

Right Outer Join

The right outer join ⟖ on any two relations is

- the result of natural join plus
- all tuples in the right relation that did not match with any tuples in the left relations, pads the tuples with null values for all other attributes

Table *loan*

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

Table *customer*

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

Right outer join:

loan ⟖ customer

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

Table *loan*

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

Table *customer*

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

Right outer join:

loan ⟖ customer

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

Modification of database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- Using assignment operation ← database modifications are done

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 attribute.
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

where r is a relational and E is a relation algebra expression.

Table *account*

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

Table *depositor*

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

- Delete all of Smith's account records from depositor
- Delete all loan records with amount in the range of 0 to 50
- Delete all accounts at branches located in Brooklyn.

branch-name	*branch-city*	*assets*
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000

Deletion contd..

- Delete all of Smith's account records from depositor

  depositor← depositor- $\sigma_{cusotmer\text{-}name=\text{“}smith\text{”}}(depositor)$

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

  loan ← loan – $\sigma_{amount\geq 0 \text{ and } amount\leq 50}(loan)$

- Delete all accounts at branches located in Brooklyn.

  r1← $\sigma_{branch\text{-}city=\text{“}Brooklyn\text{”}}(account \bowtie branch)$

  r2←$\Pi_{\text{branch-name, account-number, balance}}$(r1)

  account ← account-r1

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:

  $$r \leftarrow r \cup E$$

  r is a relational and E is a relation algebra expression.
- The insertion of a single tuple is expressed by letting *E* be a constant relation containing one tuple.

Insertion contd..

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

- 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.

Table *account*

account-number	*branch-name*	*balance*
A-101	Downtown	500
A-215	Mianus	700
A-102	Perryridge	400

Table *depositor*

customer-name	*account-number*
Hayes	A-102
Johnson	A-101
Johnson	A-201

Insertion contd..

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

  account ← account ∪ {(A-973, “Perryridge”, 1200)}

  depositor← depositor ∪ {(“smith”, 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.

  temp1← $\sigma_{branch\text{-}name=\text{“Perryridge”}}$ (*borrower* ⋈ *loan*)

  temp2← $\Pi_{loan\text{-}number,\ branch\text{-}name}$(*temp1*)

  account ← account ∪ (temp2 x {200})

  depositor← depositor ∪ $\Pi_{customer\text{-}name,\ loan\text{-}number}$(temp1)

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

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

Each *F, 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 F_i, which gives the new value for the attribute*

Updating contd..

E.g. 1. Interest payments are being changed and that all balances are to be increased by 5 percent.

E.g. 2. accounts with balances over $10,000 receive 6 percent interest, whereas all other receive 5 percent.

Updating contd..

E.g. 1. Interest payments are being changed and that all balances are to be increased by 5 percent.

$$account \leftarrow \pi_{account\text{-}number,\ branch\text{-}name,\ balance\ *\ 1.05}\ (account)$$

E.g. 2. accounts with balances over $10,000 receive 6 percent interest, whereas all other receive 5 percent.

$$account \leftarrow \pi_{account\text{-}number,\ branch\text{-}name,\ balance\ *\ 1.06}\ (\sigma_{balance>10000}\ (account))$$

$$\cup$$

$$\pi_{account\text{-}number,\ branch\text{-}name,\ balance\ *\ 1.05}\ (\sigma_{balance\leq 10000}\ (account))$$

End of chapter 3