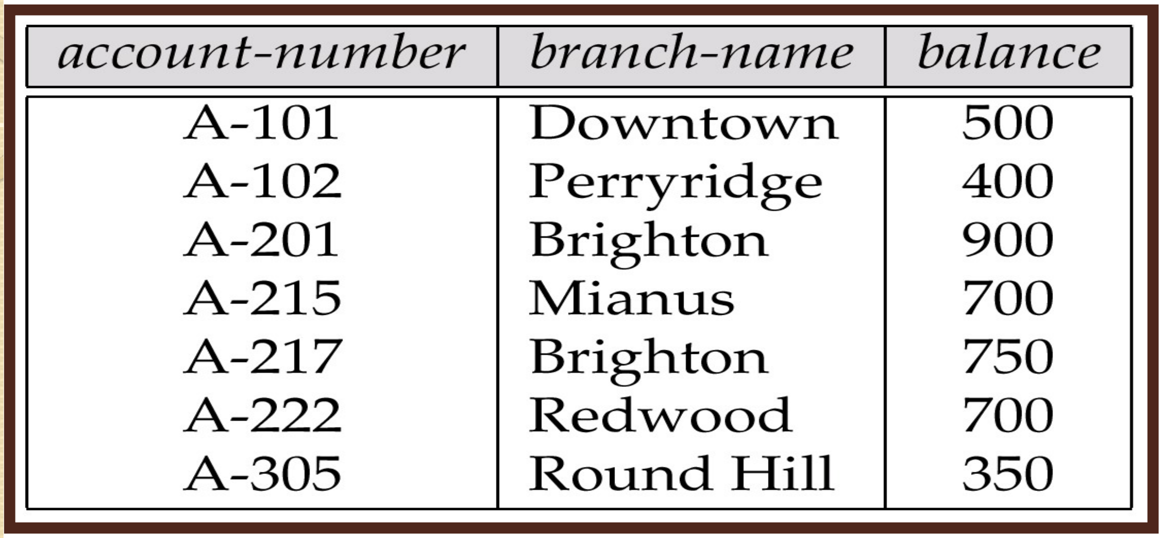
Chapter 3: Relational Model

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

Structure of Relational Databases

- A relational database consists of a collection of tables, each of which is assigned a unique name.
- A row in a table represents a relationship among a set of values.
- There is close correspondence between table and mathematical concept of relation.
    - Hence, known as relational data model.

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

- Column header : attributes
- For each attribute, there is a set of permitted values, called the domain of that attribute.

Basic Structure

- Formally, given sets $D_1, D_2, \ldots. D_n$, a **relation** r is a subset of $D_1 \times D_2 \times \ldots \times D_n$
  Thus a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$
- Use the mathematical terms relation and tuple for table and row respectively.
- 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

- Let the tuple variable t refer to the first tuple of the relation.
  - E.g. account relation

    t[account-number] – "A-101"

    t[2] – "Downtown"
  - Since, a relation is a set of tuples, t ϵ r; tuple t is in relation r.
- The order in which tuples appear in a relation is irrelevant; since a relation is a set of tuples.

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

Attribute Types

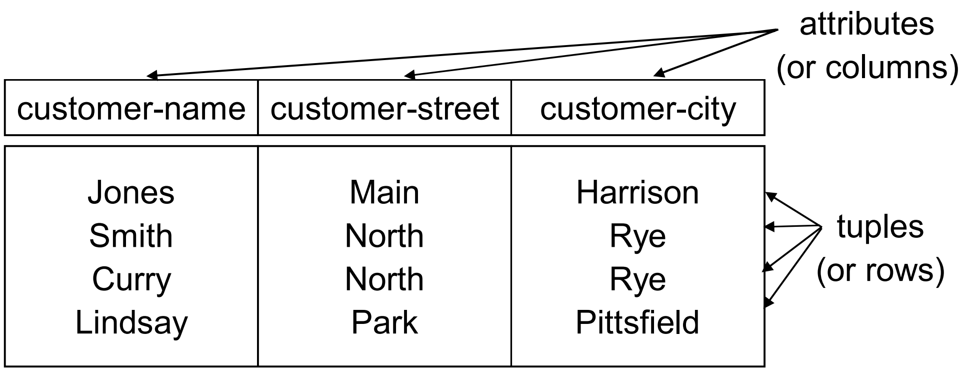
- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**, that is, indivisible
  - E.g. multivalued attribute values are not atomic
  - E.g. composite attribute values are not atomic
- The special value null is a member of every domain
- It is possible for several attributes to have the same domain.
  - E.g. customer-name and employee-name : the set of all person names, which at physical level is set of all character strings.
  - Balance and branch-name: distinct
  - Customer-names and branch-names should have same domain
  - At the physical level, both are character strings; but at the logical level, we may want both to have distinct domains.

Database Schema

- Database schema : the logical design of the database
- Database instance : snapshot of the data in the database at a given instant in time.
- Relation corresponds to the programming-language notion of a variable.
- $A_1, A_2, \ldots, A_n$ 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



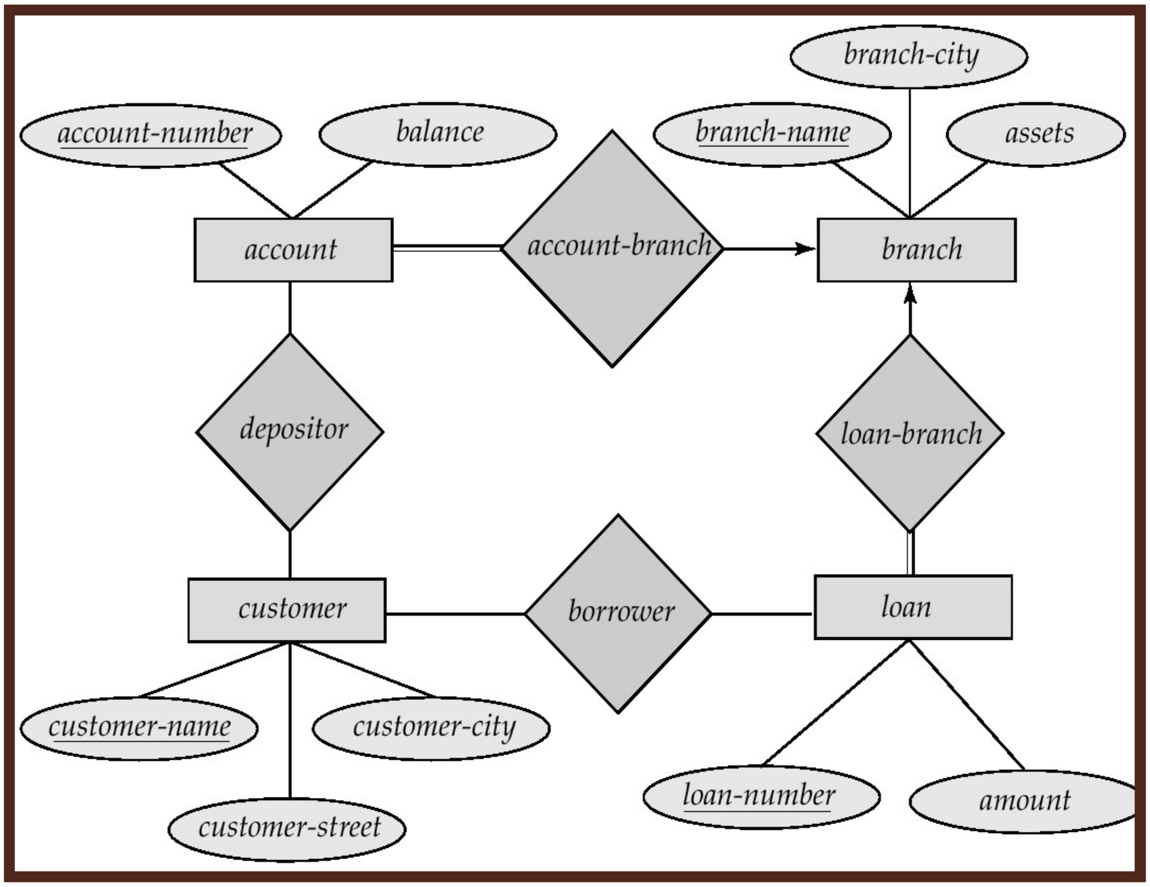
customer-name	customer-street	customer-city
Jones	Main	Harrison
Smith	North	Rye
Curry	North	Rye
Lindsay	Park	Pittsfield

customer

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)
  - To represent incomplete tuples, we must use null values i.e. unknown or does not exist.

E-R Diagram for the Banking Enterprise



The account 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

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 loan Relation

loan-number	*branch-name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The branch Relation

branch-name	*branch-city*	*assets*
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

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

The borrower Relation

customer-name	*loan-number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

Keys

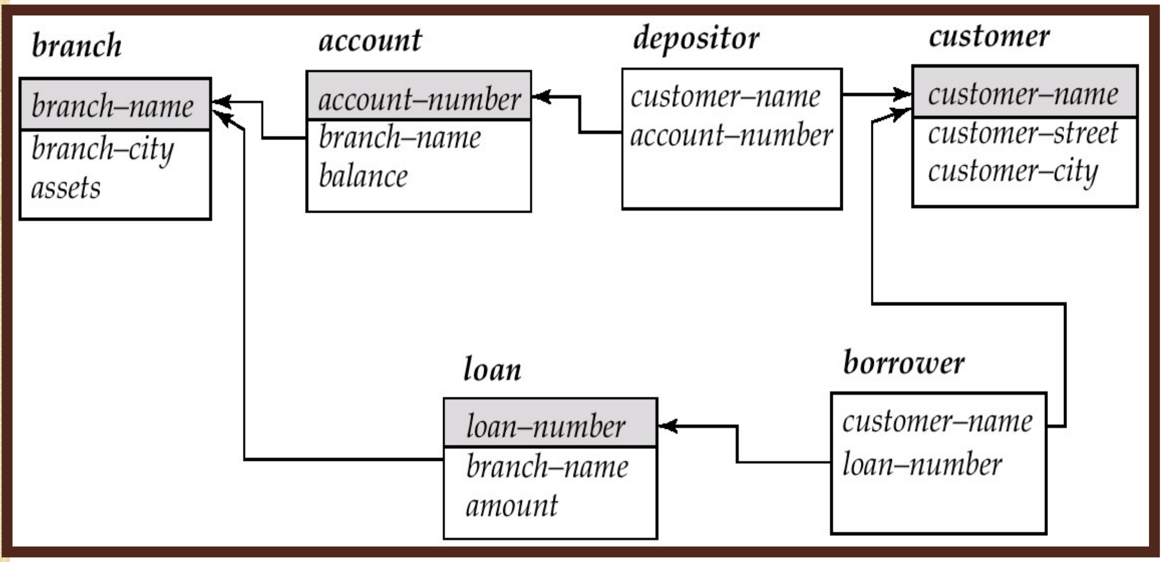
- Let $K \subseteq R$
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation r(R)
- i.e. $K \subseteq R$ is superkey for R, no two distinct tuples have the same values on all attributes in K.
  - If t1 and t2 are in r and $t1 \neq t2$, then $t1[K] \neq t2[K]$.
  - by "possible r" we mean a relation r that could exist in the enterprise we are modeling.
  - Example: {customer-name, customer-street} and {customer-name} are both superkeys of Customer, if no two customers can possibly have the same name.
- K is a **candidate key** if K is minimal Example: {customer-name} is a candidate key for Customer, since it is a superkey (assuming no two customers can possibly have the same name), and no subset of it is a superkey.

Determining Keys from E-R Sets

- Strong entity set : 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 primary keys becomes the relation's primary key

- Multivalued attributes : the primary key of the entity set or relationship set of which M is an attribute + Column C holding an individual value of M
- Foreign key : A relation schema say r1, derived from an E-R schema 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 called the referencing relation of the foreign key dependency, and r2 is called the referenced relation of the foreign key.
  - E.g. branch-name in Account-schema is a foreign key from Account-schema referencing Branch-schema; since branch-name is primary key of Branch-schema.
  - In any database instance, given any tuple, say ta from the account relation, there must be some tuple, say tb, in the branch relation such that value of branch-name attribute of ta is the same as the value of the primary key, branch-name, of tb.

Schema Diagram for the Banking Enterprise



- A database schema, along with primary key and foreign key dependencies, can be depicted pictorially by schema diagrams.
- E-R diagrams do now show foreign key attributes explicitly, whereas schema diagrams show them explicitly.

Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - Procedural – user instructs the system to perform sequence of operations on the database to compute the desired result
  - non-procedural – user describes the desired information without giving a specific procedure for obtaining that information.
- "Pure" languages:
  - Relational Algebra – procedural
  - Tuple Relational Calculus - nonprocedural
  - Domain Relational Calculus - nonprocedural
- Pure languages form underlying basis of query languages that people use.
  - They are terse, formal and lack "syntactic sugar' of commercial languages; but illustrate fundamental techniques for extracting data from the database.

Relational Algebra

- Procedural language
- Six basic operators
  - select
  - Project — unary operator
  - rename
  - set difference
  - Cartesian product — binary operator
  - union
- Other operations are intersection, natural join, division and assignment.
- The operators take one or two relations as inputs and give a new relation as a result.

Select Operation – Example

- Relation r

A	B	C	D
□	□	1	7
□	□	5	7
□	□	12	3
□	□	23	10

- □$_{A=B \text{ ^ } D > 5}$ (r)

A	B	C	D
□	□	1	7
□	□	23	10

Select Operation

- Notation: $\sigma_{p}(r)$
- p is called the selection predicate
- Defined as:

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

  Where p is a formula in propositional calculus consisting of terms connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
  Each term is one of:

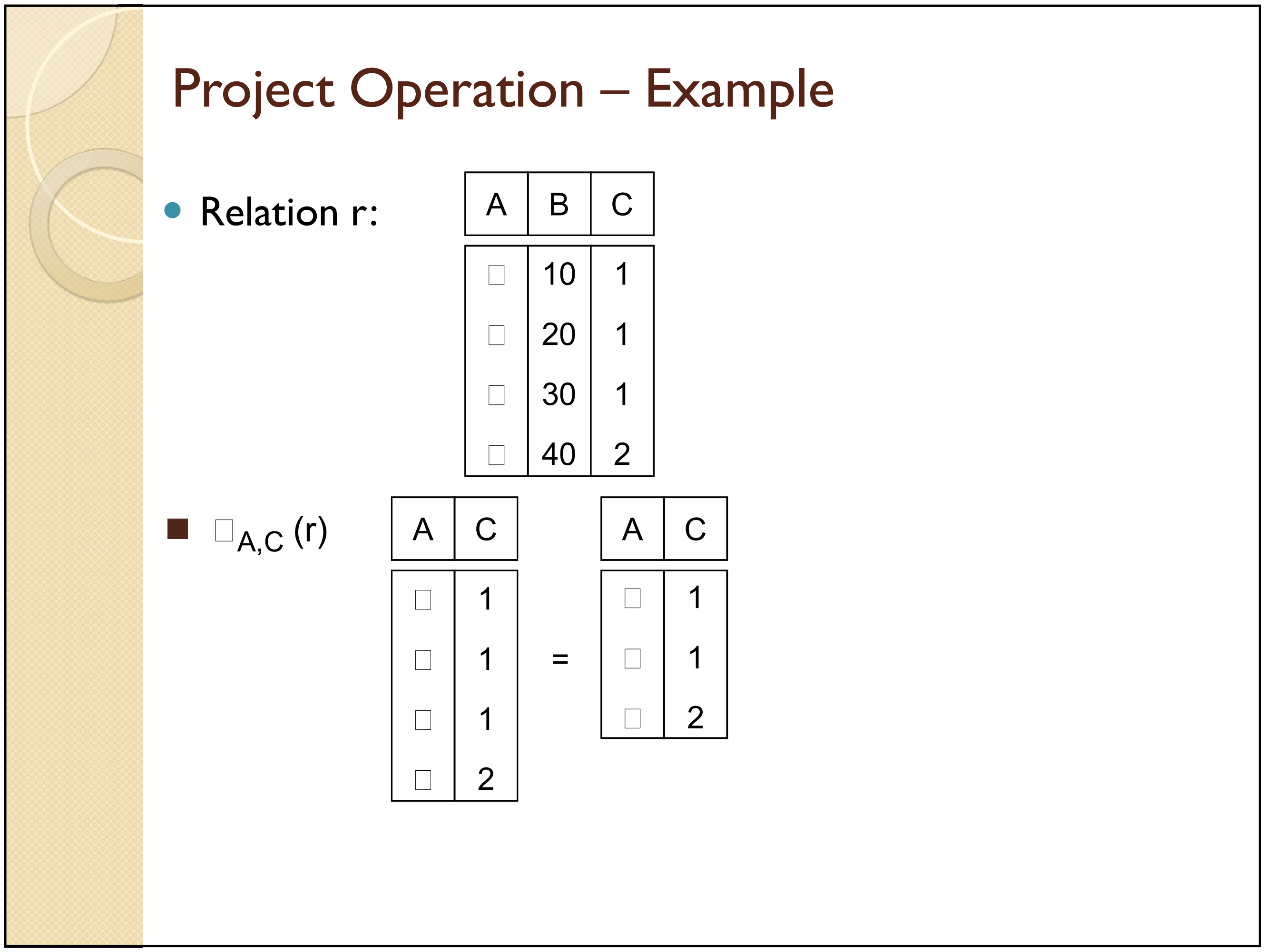
  <attribute> op <attribute> or <constant>

  where op is one of: $=, \neq, >, \geq. <. \leq$
- Example of selection:

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

loan-number	*branch-name*	*amount*
L-15	Perryridge	1500
L-16	Perryridge	1300

$\sigma_{balance>1200}(loan)$



Project Operation – Example

- Relation r:

A	B	C
□	10	1
□	20	1
□	30	1
□	40	2

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

A	C
□	1
□	1
□	1
□	2

=

A	C
□	1
□	1
□	2

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

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

loan-number	*amount*
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

Composition of Relational Operations

- Find those customers who live in Harrison”

  $\Pi_{\text{customer-name}}\ (\sigma_{\text{customer-city=“Harrison”}}(\text{customer}))$
- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

Union Operation – Example

- Relations r, s:

A	B
□	1
□	2
□	1

r

A	B
□	2
□	3

s

r □ s:

A	B
□	1
□	2
□	1
□	3

Union Operation

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

  $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$

- For $r \cup s$ to be valid.

  1. r, s must have the same arity (same number of attributes)
  2. The attribute domains must be compatible (e.g., 2nd column of r deals with the same type of values as does the 2nd column of s)

- E.g. to find all customers with either an account or a loan

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

Names of All Customers Who Have Either a Loan or an Account

customer-name
Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

Set Difference Operation – Example

- Relations r, s:

r

A	B
□	1
□	2
□	1

s

A	B
□	2
□	3

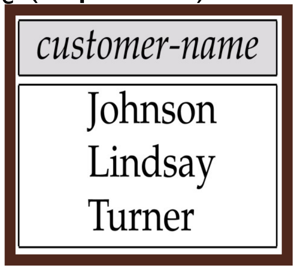
r – s:

A	B
□	1
□	1

Set Difference Operation

- Notation r – s
- Defined as:

  $r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$
- 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
- Find all customers of the bank who have an account but not a loan.
  - $\Pi_{\text{customer-name}}$ (depositor) - $\Pi_{\text{customer-name}}$ (borrower)



customer-name
Johnson
Lindsay
Turner

Cartesian-Product Operation-Example

Relations r, s:

A	B
□	1
□	2

r

C	D	E
□	10	a
□	10	a
□	20	b
□	10	b

s

r x s:

A	B	C	D	E
□	1	□	10	a
□	1	□	10	a
□	1	□	20	b
□	1	□	10	b
□	2	□	10	a
□	2	□	10	a
□	2	□	20	b
□	2	□	10	b

Cartesian-Product Operation

- Notation r x s
- Defined as:

  r x s = {t q | t ∈ r **and** q ∈ s}
- Assume that attributes of r(R) and s(S) are disjoint. (That is, R ∩ S = ∅).
- If attributes of r(R) and s(S) are not disjoint, then renaming must be used.
- E.g. r = borrower x loan is

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

This simplification does not lead to any ambiguity.

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

- Naming convention requires relations should have distinct names, causes problems when
  - Cartesian product of a relation with itself is desired
  - If we use the result of a relational-algebra expression in a Cartesian product.
  - Rename operation is used to avoid these problems.

Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$
- r x s

A	B	C	D	E
□	1	□	10	a
□	1	□	10	a
□	1	□	20	b
□	1	□	10	b
□	2	□	10	a
□	2	□	10	a
□	2	□	20	b
□	2	□	10	b

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

A	B	C	D	E
□	1	□	10	a
□	2	□	20	a
□	2	□	20	b

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.

Example:

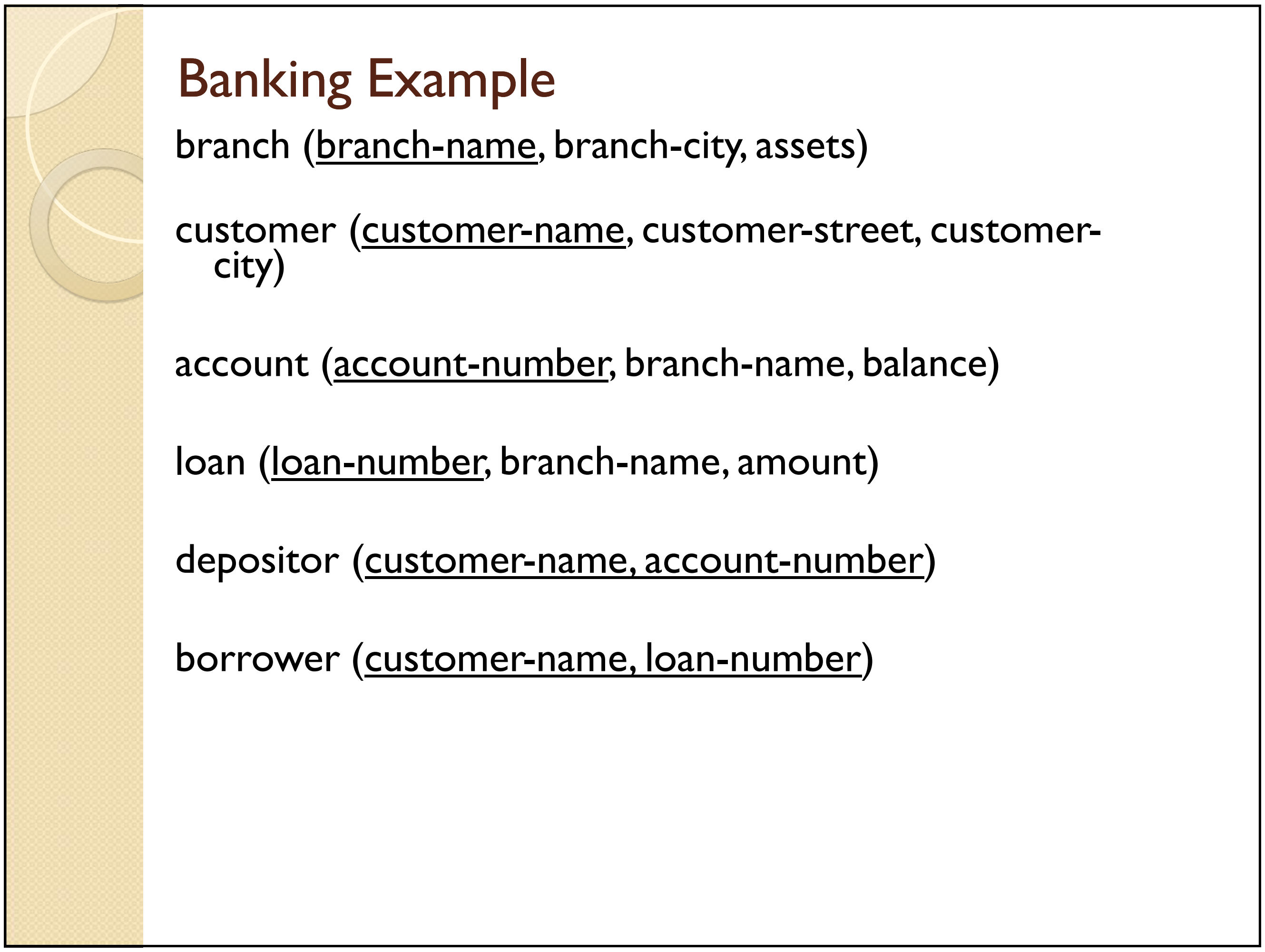
$$\rho_{x}\ (E)$$

returns the expression E under the name X

If a relational-algebra expression E has arity n, then

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

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



Banking Example

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

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

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

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

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

Example Queries

- Find all loans of over $1200

  $\square_{\text{amount} > 1200}$ (loan)

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

  $\square_{\text{loan-number}}$ ($\square_{\text{amount} > 1200}$ (loan))

- Find the names of all customers who have a loan, an account, or both, from the bank

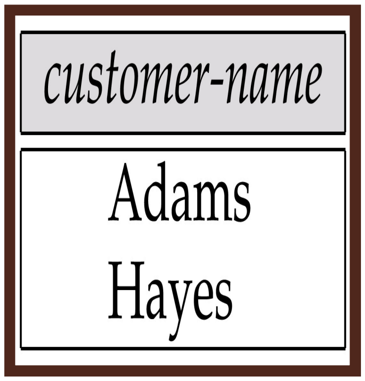
  $\square_{\text{customer-name}}$ (borrower) □ $\square_{\text{customer-name}}$ (depositor)

Example Queries

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

$\square_{\text{customer-name}}\ (\square_{\text{branch-name="Perryridge"}}$

$(\square_{\text{borrower.loan-number = loan.loan-number}}(\text{borrower x loan})))$



customer-name
Adams
Hayes

Result of borrower × loan

customer-name	*borrower. loan-number*	*loan. loan-number*	*branch-name*	*amount*
Adams	L-16	L-11	Round Hill	900
Adams	L-16	L-14	Downtown	1500
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Adams	L-16	L-17	Downtown	1000
Adams	L-16	L-23	Redwood	2000
Adams	L-16	L-93	Mianus	500
Curry	L-93	L-11	Round Hill	900
Curry	L-93	L-14	Downtown	1500
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Curry	L-93	L-17	Downtown	1000
Curry	L-93	L-23	Redwood	2000
Curry	L-93	L-93	Mianus	500
Hayes	L-15	L-11		900
Hayes	L-15	L-14		1500
Hayes	L-15	L-15		1500
Hayes	L-15	L-16		1300
Hayes	L-15	L-17		1000
Hayes	L-15	L-23		2000
Hayes	L-15	L-93		500
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
Smith	L-23	L-11	Round Hill	900
Smith	L-23	L-14	Downtown	1500
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Smith	L-23	L-17	Downtown	1000
Smith	L-23	L-23	Redwood	2000
Smith	L-23	L-93	Mianus	500
Williams	L-17	L-11	Round Hill	900
Williams	L-17	L-14	Downtown	1500
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300
Williams	L-17	L-17	Downtown	1000
Williams	L-17	L-23	Redwood	2000
Williams	L-17	L-93	Mianus	500

Result of $\sigma_{\text{branch-name} = \text{“Perryridge”}}$ (borrower $\times$ loan)

customer-name	*borrower. loan-number*	*loan. loan-number*	*branch-name*	*amount*
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Hayes	L-15	L-15	Perryridge	1500
Hayes	L-15	L-16	Perryridge	1300
Jackson	L-14	L-15	Perryridge	1500
Jackson	L-14	L-16	Perryridge	1300
Jones	L-17	L-15	Perryridge	1500
Jones	L-17	L-16	Perryridge	1300
Smith	L-11	L-15	Perryridge	1500
Smith	L-11	L-16	Perryridge	1300
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300

- 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.

$\Box_{\text{customer-name}}$ ($\Box_{\text{branch-name = “Perryridge”}}$

($\Box_{\text{borrower.loan-number = loan.loan-number}}$(borrower x loan))) –
$\Box_{\text{customer-name}}$(depositor)

Example Queries

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

– Query 1

$$\square_{customer\text{-}name}(\square_{branch\text{-}name = \text{“Perryridge”}}(\square_{borrower.loan\text{-}number = loan.loan\text{-}number}(borrower \times loan)))$$

□ Query 2

$$\square_{customer\text{-}name}(\square_{loan.loan\text{-}number = borrower.loan\text{-}number}((\square_{branch\text{-}name = \text{“Perryridge”}}(loan)) \times borrower))$$

Example Queries

Find the largest account balance

- Rename account relation as d
- The query is:

$$\square_{\text{balance}}(\text{account}) - \square_{\text{account.balance}}$$
$$(\square_{\text{account.balance} < \text{d.balance}} (\text{account} \times \rho_{d} (\text{account})))$$

balance
500
400
700
750
350

$\square_{\text{account.balance}} (\square_{\text{account.balance} < \text{d.balance}} (\text{account} \times \rho_{d} (\text{account})))$

balance
900

Largest account balance

- Find the names of all customers who live on the same street and in the same city as Smith.

$\square_{\text{customer.customer-name}}$

$(\square_{\text{ customer.customer-steet = smith-addr.street }} \square_{\text{ customer.customer-city = smith-add}}$

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

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

customer-name
Curry
Smith

Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
    - E.g. {(A-101, Downtown, 500)(A-215, mianus, 700)}
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_s(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

Additional Operations

- We define additional operations that do not add any power to the relational algebra, but that simplify common queries.
- For each new operation, we give an equivalent expression that uses only fundamental operations.
- Set intersection
- Natural join
- Division
- Assignment

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r$ **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
□	1
□	2
□	1

r

A	B
□	2
□	3

s

- r ∩ s

A	B
□	2

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

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

customer-name
Hayes
Jones
Smith

Natural-Join Operation

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

Example:

R = (A, B, C, D)

S = (E, B, D)

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

$r \bowtie s$ is defined as:

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

Natural Join Operation – Example

- Relations r, s:

A	B	C	D
□	1	□	a
□	2	□	a
□	4	□	b
□	1	□	a
□	2	□	b

r

B	D	E
1	a	□
3	a	□
1	a	□
2	b	□
3	b	□

s

r ⋈ s

A	B	C	D	E
□	1	□	a	□
□	1	□	a	□
□	1	□	a	□
□	1	□	a	□
□	2	□	b	□

Natural Join Operation – Example

- Find the names of all customers who have a loan at the bank, along with the loan number and the loan amount.

Query-1 (Using cartesian product)

$$\Pi_{\text{customer-name, loan.loan-number, amount}}$$
$$(\sigma_{\text{borrower.loan-number = loan.loan-number}}(\text{borrower x loan}))$$

Query- 2 (Using natural join)

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

customer-name	*loan-number*	*amount*
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-23	2000
Smith	L-11	900
Williams	L-17	1000

Natural Join Operation – Example

- Find the names of all branches with customers who have an account in the bank and who live in Harrison.

$\square_{\text{branch-name}}$

$(\square_{\text{customer-city = “Harrison”}}(\text{customer} \quad \text{account} \quad \text{depositor}))$

$\bowtie \qquad \bowtie$

- Natural join is associative

i.e. $(\text{customer} \bowtie \text{account}) \bowtie \text{depositor}$

$\text{customer} \bowtie (\text{account} \bowtie \text{depositor})$

branch-name
Brighton
Perryridge

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

  $$\square_{\text{customer-name}}(\text{borrower} \bowtie \text{depositor}) \qquad \text{OR}$$

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

- Let r(R) and s(S) be relations without any attributes in common; (That is, $R \cap S = \emptyset$)

  then, $r \bowtie s = r \times s$.

- The theta join operation is an extension to the natural-join operation
  - combines a selection and a Cartesian product into single operation.

  $$r \bowtie_{\theta} s \;=\; \overset{\square}{\ }_{\theta}(r \times s)$$

Division Operation

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

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

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

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

Division Operation – Example

Relations r, s:

A	B
□	1
□	2
□	3
□	1
□	1
□	1
□	3
□	4
□	6
□	1
□	2

r

B
1
2

s

r □ s:

A
□
□

Division Operation (Cont.)

- Property
    - Let q – r ÷ 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 $u \in s$, $tu \notin r$.

Example Queries

- Find all customers who have an account at all branches located in Brooklyn city. (r2 ÷ r1)

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

$\div$ $\Pi_{\text{branch-name}}$ ($\sigma_{\text{branch-city = "Brooklyn"}}$ (branch))

customer-name	*branch-name*
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Lindsay	Redwood
Smith	Mianus
Turner	Round Hill

r2

branch-name
Brighton
Downtown

r1

Customer-name : Johnson

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.
- Example: Write $r \div s$ as

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

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
  - Assignments to permanent relation constitute a database modification.

Example Queries

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

Query 1

$$\square_{CN}(\square_{BN=\text{“Downtown”}}(\text{depositor} \bowtie \text{account})) \ \square$$

$$\square_{CN}(\square_{BN=\text{“Uptown”}}(\text{depositor} \bowtie \text{account}))$$

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

Query 2

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

Extended Relational-Algebra-Operations

- Generalized Projection
- Outer Join
- Aggregate Functions

Generalized Projection

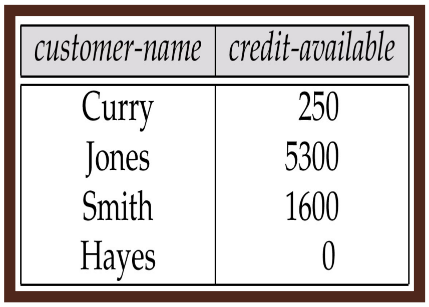
- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

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

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

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

customer-name	limit	credit-balance
Curry	2000	1750
Hayes	1500	1500
Jones	6000	700
Smith	2000	400



customer-name	*credit-available*
Curry	250
Jones	5300
Smith	1600
Hayes	0

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

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

  - 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)
    - All tuples in a group have the same value for $G_1, G_2 \ldots, G_n$
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

Aggregate Operation – Example

- Relation r:

A	B	C
□	□	7
□	□	7
□	□	3
□	□	10

$g_{\textbf{sum(c)}}(r)$

sum-C
27

Aggregate Operation – Example

- Relation 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

Find total sum of balance amount of all accounts

$g_{\text{sum(balance)}}$ (account)

Sum(balance)
3500

- We may want to eliminate duplicate values
- Function name is hyphenated with "distinct" word
- Result of aggregation does not have a name
- Can use rename operation to give it a name

e.g. Find number of branches in account relation

$\mathcal{g}_{\text{count-}\textbf{distinct}\text{(branch-name)}\ \textbf{as no-of-branches}}\ (\text{account})$

no-of-branches
3

Aggregate Operation – Example

We may want to group tuples on some attribute and then apply aggregate function

e.g. Find sum of balance for each branch separately

- Relation account grouped by branch-name:

branch-name	account-number	balance
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

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

branch-name	Sum of balance
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Operation – Example

E.g. Find sum of balance and maximum of balance for each branch separately

- Relation account grouped by branch-name:

branch-name	account-number	balance
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$_{\text{branch-name}}\mathcal{g}_{\text{sum(balance)},\text{max(balance)}}(\text{account})$$

branch-name	sum(balance)	max(balance)
Perryridge	1300	900
Brighton	1500	750
Redwood	700	700

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses null values:
  - null signifies that the value is unknown or does not exist

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 $\bowtie$ 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

Null Values

- It is possible for tuples to have a null value, denoted by null, for some of their attributes
- null signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving null is null.
- Aggregate functions simply ignore null values
  - Is an arbitrary decision. Could have returned null as result instead.
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same
  - Alternative: assume each null is different from each other
  - Both are arbitrary decisions, so we simply follow SQL

Null Values

- Comparisons with null values return the special truth value unknown
- Three-valued logic using the truth value unknown:
  - OR: (unknown **or** true) = true,
    (unknown **or** false) = unknown
    (unknown **or** unknown) = unknown
  - AND: (true **and** unknown) = unknown,
    (false **and** unknown) = false,
    (unknown **and** unknown) = unknown
  - NOT: (**not** unknown) = unknown

Null value in various operation

- Select operation
  - It evaluates predicate for each tuple.
  - If value is true, tuple is added to result.
  - If unknown or false then not added to result.
- Join operation
  - If both relations have null value for common attribute, tuples do not match.
- Projection and generalized projection operation
  - If two tuples in result are exactly same and both have null values for same field they are treated as duplicate and hence eliminated.
- Union, intersection, difference
  - Tuples that have same values on all fields are treated as duplicates even if some of fields have null values in both tuples.

    Behavior is arbitrary.

- Aggregate function
  - When null occurs in grouping attributes → If two tuples are same on all grouping attributes, placed in same group even if some attributes has null values
  - When null occurs in aggregated attributes →operation deletes null values before applying aggregation. If result is empty, aggregate result is null.
- Outer join operation
  - Just like join except that if tuple do not occur in join, added to result by adding null values.

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 □ account – □□$_{\text{branch-name = “Perryridge”}}$ (account)

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

  loan □ loan – □□$_{\text{amount □□0□and amount □ 50}}$ (loan)

- Delete all accounts at branches located in Needham.

  r_1 □ □$_{\text{□branch-city = “Needham”}}$ (account ⋈ branch)

  r_2 □ □$_{\text{branch-name, account-number, balance}}$ (r_1)

  r_3 □ □$_{\text{customer-name, account-number}}$ (r_2 ⋈depositor)

  account □ account – r_2

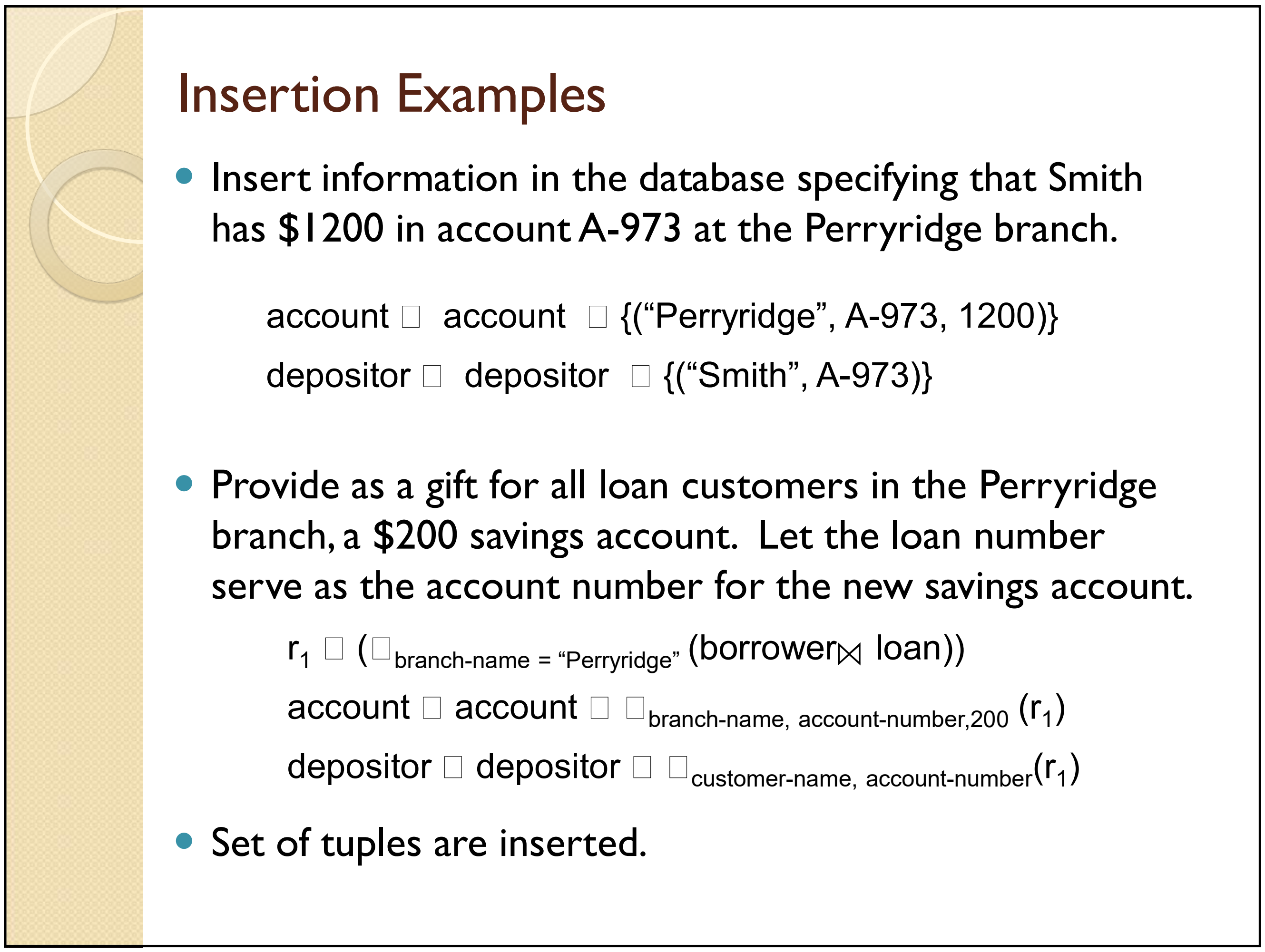
  depositor □ depositor – r_3

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 □ account □ {("Perryridge", A-973, 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.

  r_1 □ (□$_{\text{branch-name = “Perryridge”}}$ (borrower ⋈ loan))

  account □ account □ □$_{\text{branch-name, account-number,200}}$ (r_1)

  depositor □ depositor □ □$_{\text{customer-name, account-number}}$($r_1$)

- Set of tuples are inserted.

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

  $r \leftarrow \prod_{F1, F2, \ldots, Fl,} (r)$
- Each F_i is either
  - the ith attribute of r, if the ith 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 □ □$_{\text{AN, BN, BAL * 1.05}}$(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 □ □$_{\text{AN, BN, BAL * 1.06}}$(□$_{\text{BAL □ 10000}}$(account))
  □ □$_{\text{AN, BN, BAL * 1.05}}$(□$_{\text{BAL □ 10000}}$(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.)
- Security aspect may require that certain data be hidden from users.
- 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

  $\Pi_{\text{customer-name, loan-number}}$ (borrower $\bowtie$ loan)
- Any relation that is not part of the logical 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**

$\Pi_{\text{branch-name, customer-name}}$ (depositor ⋈ account)

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

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

$\Pi_{\text{customer-name}}$

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

- View names may appear in any place where a relation name may appear.

- View definition differs from the relational-algebra assignment operation.

  r1<- □$_{\text{branch-name, customer-name}}$ (depositor ⋈ account)

  □ □$_{\text{branch-name, customer-name}}$ (borrower ⋈ loan)

  - Assignment operation is evaluated once.
  - r1 does not change when we update the relations depositor, account, borrower or loan.
  - Any modifications we make to these relations changes the set of tuples in the view all-customer as well.
- If a view relation is computed and stored, it may become out of date if the relations used to define it are modified.
  - Hence, database system stores the definition of view itself; rather than the result of evaluation of expression that defines the view.
  - Wherever a view relation appears in a query, it is replaced by the stored query expression.
  - Thus, whenever we evaluate the query, the view relation gets

- Certain db systems allow view relations to be stored, but they make sure that, if the actual relations used in the view definition change, the view is kept up to date.
- Such views are called materialized views.
- The process of keeping the view up to date is called view maintenance.
- Applications that use a view frequently get benefit from the use of materialized views.
  - Fast response
  - Weighed against the storage cost and added overhead for updates.

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**

  $\Pi_{\text{branch-name, loan-number}}$ (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 (“L-37”, “Perryridge”, null) into the loan relation
- Some updates through views are impossible to translate into database relation updates
  - create view v as $\sigma_{\text{branch-name = “Perryridge”}}$ (account)

    v $\leftarrow$ v $\cup$ (L-99, Downtown, 23)
- Others cannot be translated uniquely
  - all-customer $\leftarrow$ all-customer $\cup$ {(“Perryridge”, “John”)}

- Hence, modifications are generally not permitted on view relations except in limited cases.
- Different database systems specify different conditions under which they permit updates on view relation.

Views Defined Using Other Views

- One view may be used in the expression defining another view.

  create view perryridge-customer as

  $\square_{\text{customer-name}}(\square_{\text{branch-name = "Perryridge"}}(\text{all-customer}))$

- 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.

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

View expansion

$\square_{\text{customer-name = “John”}}(\text{perryridge-customer})$

$\downarrow$

$\square_{\text{customer-name = “John”}}(\square\ _{\text{customer-name}}(\square_{\text{branch-name = “Perryridge”}}(\text{all-customer})))$

$\downarrow$

$\square_{\text{customer-name = “John”}}(\square\ _{\text{customer-name}}(\square_{\text{branch-name = “Perryridge”}}(\square_{\text{branch-name, customer-name}}(\text{depositor} \bowtie \text{account}) \; \square\ \square_{\text{branch-name, customer-name}}(\text{borrower} \bowtie \text{loan}))))$