RELATIONAL MODEL

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

EXAMPLE OF A RELATION

account-number	*branch-name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

BASIC STRUCTURE

- Formally, given sets D_1, D_2, …. D_n a **relation** r is a subset of
  D_1 x D_2 x … x D_n
  Thus a relation is a set of n-tuples (a_1, a_2, …, a_n) where each $a_i \in D_i$
- Example: if

  customer-name = {Jones, Smith, Curry, Lindsay}
  customer-street = {Main, North, Park}
  customer-city = {Harrison, Rye, Pittsfield}

  Then r = { (Jones, Main, Harrison),
  (Smith, North, Rye),
  (Curry, North, Rye),
  (Lindsay, Park, Pittsfield)}
  is a relation over *customer-name* x *customer-street* x *customer-city*

ATTRIBUTE TYPES

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**, that is, indivisible
  - E.g. multivalued attribute values are not atomic
  - E.g. composite attribute values are not atomic
- The special value *null* is a member of every domain
- The null value causes complications in the definition of many operations
  - we shall ignore the effect of null values in our main presentation and consider their effect later

RELATION SCHEMA

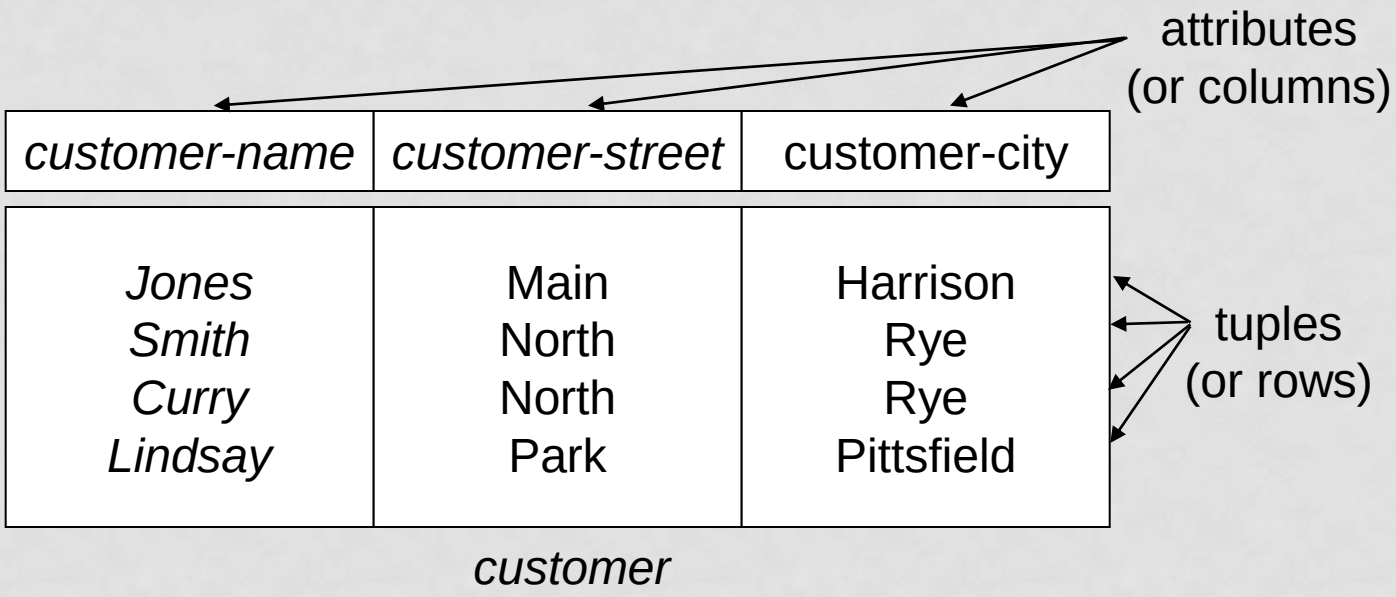
- $A_1, A_2, \ldots, A_n$ are *attributes*
- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

  E.g. *Customer-schema = (customer-name, customer-street, customer-city)*
- *r(R)* is a *relation* on the *relation schema R*

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

RELATION INSTANCE

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



RELATIONS ARE UNORDERED

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- E.g. *account* relation with unordered tuples

account-number	*branch-name*	*balance*
A-101	Downtown	500
A-215	Mianus	700
A-102	Perryridge	400
A-305	Round Hill	350
A-201	Brighton	900
A-222	Redwood	700
A-217	Brighton	750

DATABASE

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

  E.g.: *account* : stores information about accounts
  depositor : stores information about which customer
  owns which account
  customer : stores information about customers
- Storing all information as a single relation such as *bank(account-number, balance, customer-name, ..)* results in
  - repetition of information (e.g. two customers own an account)
  - the need for null values (e.g. represent a customer without an account)
- Normalization theory deals with how to design relational schemas

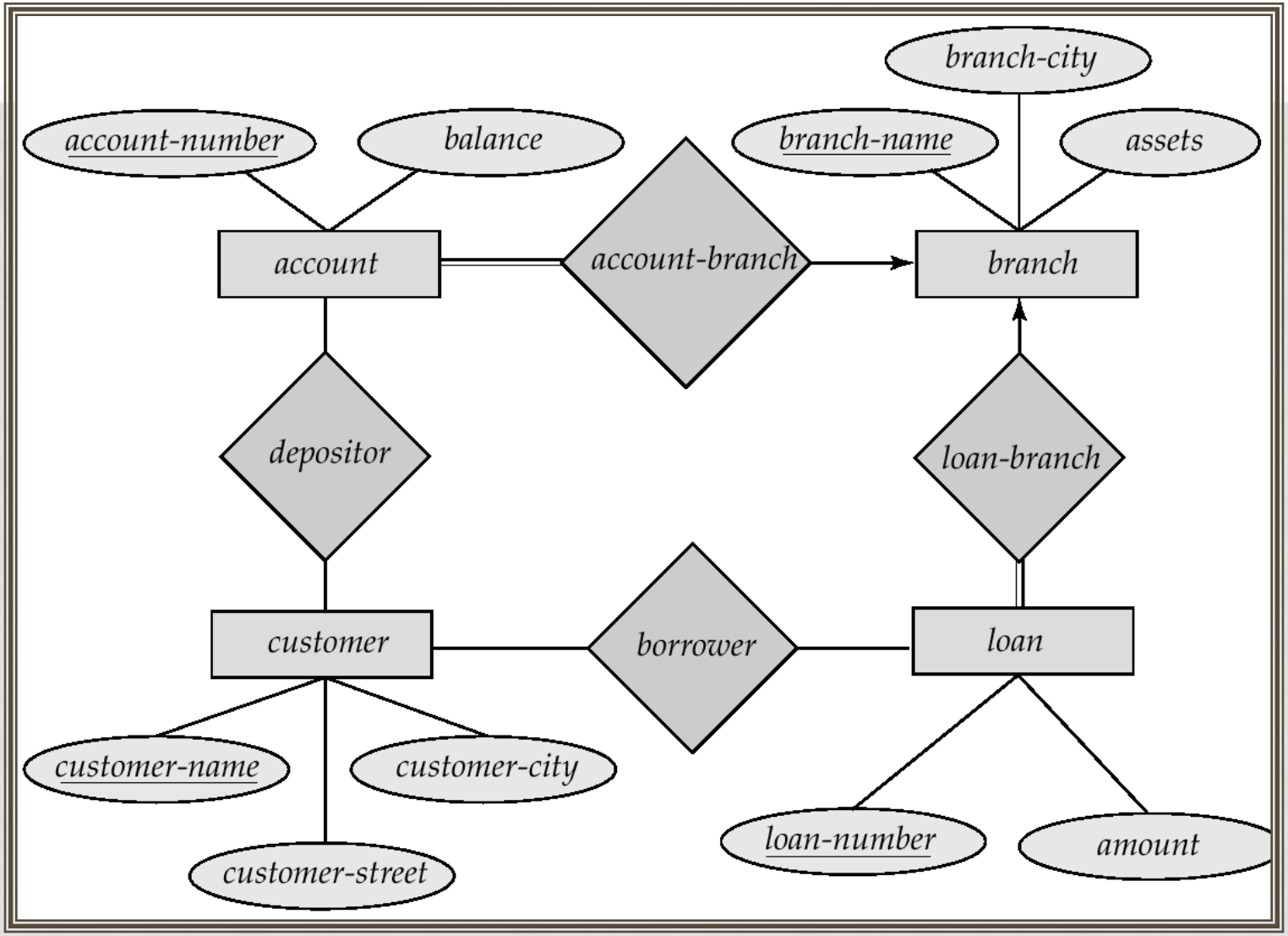
THE *CUSTOMER* RELATION

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

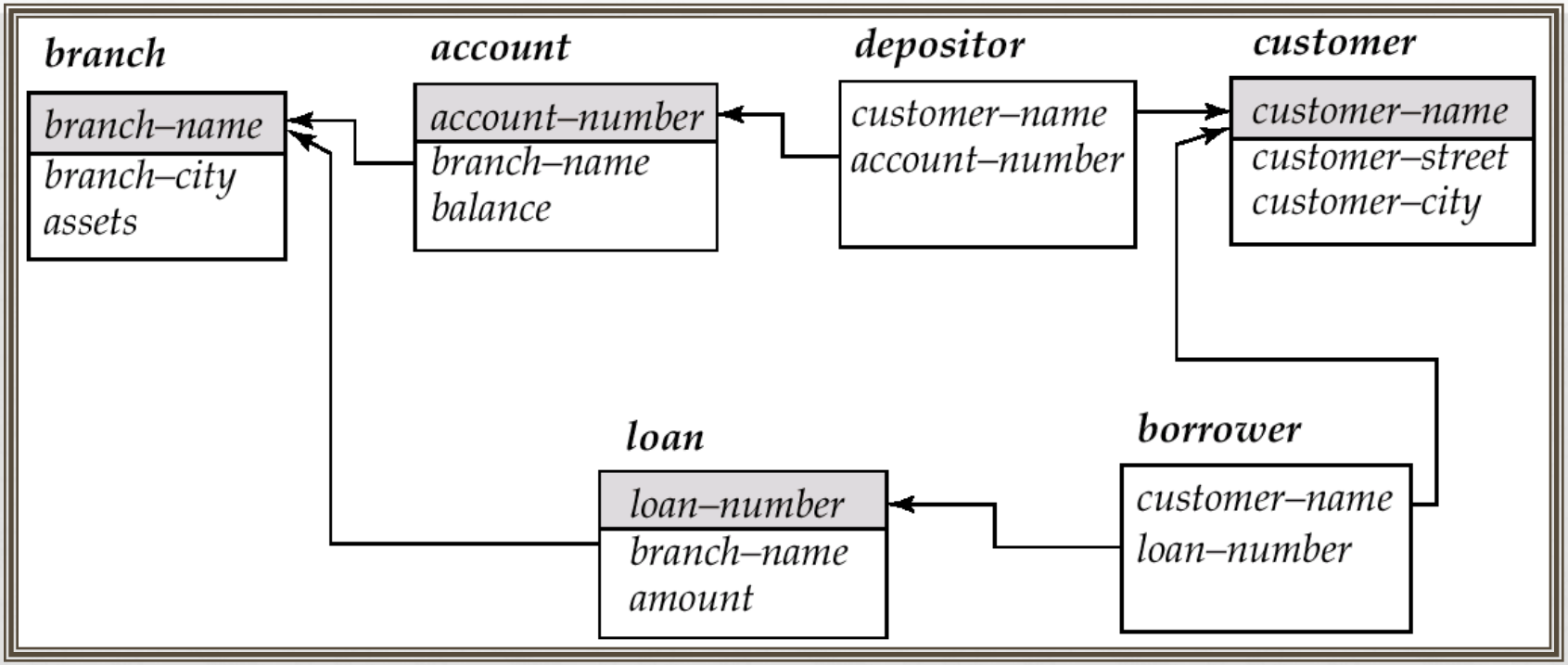
THE *DEPOSITOR* RELATION

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

E-R DIAGRAM FOR THE BANKING ENTERPRISE



SCHEMA DIAGRAM FOR THE BANKING ENTERPRISE



QUERY LANGUAGES

- Language in which user requests information from the database.
- Categories of languages
  - procedural
  - non-procedural
- "Pure" languages:
  - Relational Algebra
  - Tuple Relational Calculus
  - Domain Relational Calculus
- Pure languages form underlying basis of query languages that people use.

RELATIONAL ALGEBRA

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

SELECT OPERATION – EXAMPLE

- Relation *r*

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

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

A	B	C	D
α	α	1	7
β	β	23	10

SELECT OPERATION

- Notation: $\sigma_p(r)$
- p is called the selection predicate
- 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:

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

PROJECT OPERATION – EXAMPLE

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

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

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

PROJECT OPERATION

- Notation:

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

where A_1, A_2 are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- E.g. To eliminate the *branch-name* attribute of *account*

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

UNION OPERATION – EXAMPLE

- Relations *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r $\cup$ s:

A	B
α	1
α	2
β	1
β	3

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*)

SET DIFFERENCE OPERATION – EXAMPLE

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r – *s*:

A	B
α	1
β	1

SET DIFFERENCE OPERATION

- Notation $r - s$
- Defined as:

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

- 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

CARTESIAN-PRODUCT OPERATION-EXAMPLE

Relations *r, s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

r x *s*:

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

CARTESIAN-PRODUCT OPERATION

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

$$r \times s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \varnothing$).
- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.

COMPOSITION OF OPERATIONS

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

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

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

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

RENAME OPERATION

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

Example:

$$\rho_X (E)$$

returns the expression *E* under the name *X*

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

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

returns the result of expression *E* under the name *X*, and with the

attributes renamed to *A1*, *A2*, …., *An*.

BANKING EXAMPLE

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

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

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

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

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

EXAMPLE QUERIES

- Find all loans of over $1200

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

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

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

EXAMPLE QUERIES

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

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

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

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

EXAMPLE QUERIES

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

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

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

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

EXAMPLE QUERIES

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

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

  - Query 2

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

EXAMPLE QUERIES

Find the largest account balance

- Rename *account* relation as *d*
- The query is:

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

FORMAL DEFINITION

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\Pi_s(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

ADDITIONAL OPERATIONS

We define additional operations that do not add any power to the

relational algebra, but that simplify common queries.

- Set intersection
- Natural join
- Division
- Assignment

SET-INTERSECTION OPERATION

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - *r, s* have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

SET-INTERSECTION OPERATION - EXAMPLE

- Relation r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

NATURAL-JOIN OPERATION

- Notation: r ⋈ s

- Let *r* and *s* be relations on schemas *R* and *S* respectively.
  Then, r ⋈ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*
- Example:
  R = (*A*, *B*, *C*, *D*)
  S = (*E*, *B*, *D*)
  - Result schema = (*A*, *B*, *C*, *D*, *E*)
  - *r* ⋈ *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
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

$r \bowtie s$

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

DIVISION OPERATION

$$r \div s$$

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

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

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

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

DIVISION OPERATION – EXAMPLE

Relations *r, s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
$\in$	6
$\in$	1
β	2

r

B
1
2

s

r ÷ *s*:

A
α
β

ANOTHER DIVISION EXAMPLE

Relations *r, s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	*1*
α	a	γ	a	*1*
α	a	γ	b	*1*
β	a	γ	a	*1*
β	a	γ	b	*3*
γ	a	γ	a	*1*
γ	a	γ	b	*1*
γ	a	β	b	*1*

r

D	*E*
a	*1*
b	*1*

s

r ÷ *s*:

A	*B*	*C*
α	a	γ
γ	a	γ

DIVISION OPERATION (CONT.)

- Property
  - Let $q - r \div s$
  - Then q is the largest relation satisfying $q \times s \subseteq r$
- Definition in terms of the basic algebra operation
  Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

To see why

- $\Pi_{R-S,S}(r)$ simply reorders attributes of r
- $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ gives those tuples t in $\Pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

ASSIGNMENT OPERATION

- The assignment operation (←) 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 ← is assigned to the relation variable on the left of the ←.
  - May use variable in subsequent expressions.

EXAMPLE QUERIES

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

Query 1

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

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

Query 2

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

EXAMPLE QUERIES

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

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

VIEWS

- In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)
- Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by $\Pi_{customer\text{-}name,\ loan\text{-}number}\ (borrower \bowtie loan)$
- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.

VIEW DEFINITION

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

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

  where <query expression> is any legal relational algebra query expression. The view name is represented by *v*.
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

VIEW EXAMPLES

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

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

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

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

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

UPDATES THROUGH VIEW

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

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

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

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

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

TUPLE RELATIONAL CALCULUS

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

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

- It is the set of all tuples *t* such that predicate *P* is true for *t*
- *t* is a *tuple variable*, *t*[*A*] denotes the value of tuple *t* on attribute *A*
- $t \in r$ denotes that tuple *t* is in relation *r*
- *P* is a *formula* similar to that of the predicate calculus

PREDICATE CALCULUS FORMULA

1. Set of attributes and constants
2. Set of comparison operators: (e.g., $<$, $\leq$, $=$, $\neq$, $>$, $\geq$)
3. Set of connectives: and ($\wedge$), or (v), not ($\neg$)
4. Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if true, then y is true

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

5. Set of quantifiers:
   - $\exists\, t \in r\ (Q(t)) \equiv$ "there exists" a tuple in *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 (account-number, branch-name, balance)*
- *loan (loan-number, branch-name, amount)*
- *depositor (customer-name, account-number)*
- *borrower (customer-name, loan-number)*

EXAMPLE QUERIES

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

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

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

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

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