CS354: Database

Relational Algebra, Tuple and Domain Relational Calculus

Query Languages

- Language in which user requests information from the database
- Categories of languages
  - Procedural:
    - Relational algebra
  - Non-procedural, or declarative:
    - Tuple relational calculus, Domain relational calculus
- These languages form underlying basis of SQL query languages

Relational Algebra

- Procedural language
- Six basic operators
  - Selection: σ
  - Projection: $\prod$
  - Union: $\cup$
  - Set difference: –
  - Cartesian product: x
  - Rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result

Selection Operation

- 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

Projection Operation

- 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

Union Operation

- 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

Set Difference Operation

- 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

Cartesian Product Operation

- 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

$r \times s$

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

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

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

  returns the result of expression E under the name x, and with the attributes renamed to A_1, A_2, …., A_n .

Banking Example

branch (branch_name, branch_city, assets)

customer (customer_name, customer_street, customer_city)

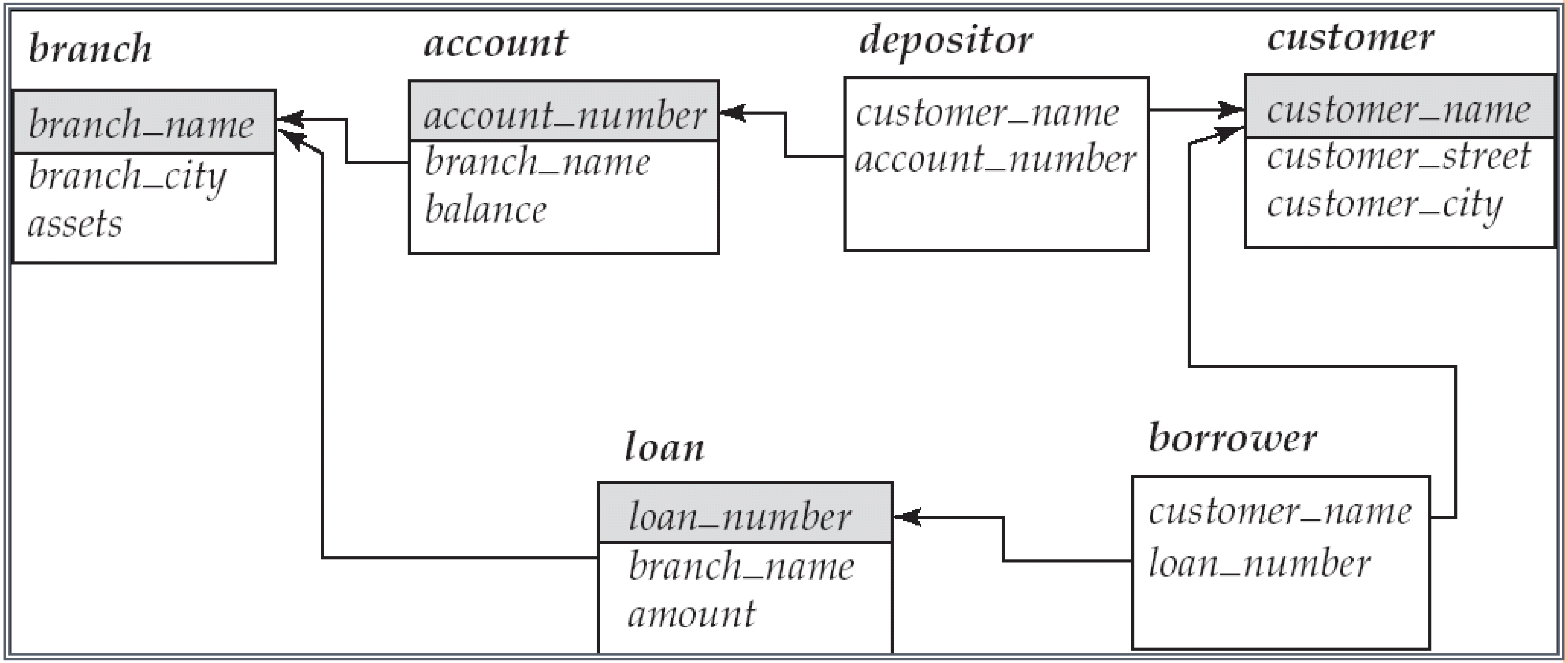
account (account_number, branch_name, balance)

loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)

Bank Example



Example Queries

- Find all loans of over Rs 2000

$$\sigma_{amount > 2000}(loan)$$

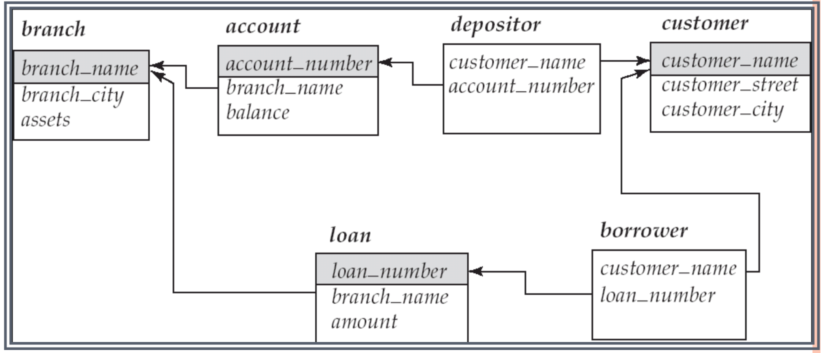
- Find the loan number for each loan of an amount greater than Rs 2000

$$\pi_{loan\_no}(\sigma_{amount > 2000}(loan))$$

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

$$\pi_{customer\_name}(depositor) \cup \pi_{customer\_name}(borrower)$$

Example Queries



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

$$\pi_{customer\_name}(\sigma_{(borrower.loan\_no=loan.loan\_no) \wedge (loan.branch=Patliputra)}(borrower \times loan))$$

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

$$\pi_{customer\_name}(\sigma_{(borrower.loan\_no=loan.loan\_no) \wedge (loan.branch=Patliputra)}(borrower \times loan))$$

$$- \pi_{customer\_name}(depositor)$$

Some other operations

- Additional Operations
  - Set intersection
  - Natural join
  - Outer Join
  - Division
- The above operations can be expressed using basic operations we have seen earlier

Set intersection operation

- Relations r,s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

A	B
α	2

$r \cap s$

Set intersection operation (contd.)

- Set intersection operation can be built using other basic operations
- How?

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

NATURAL JOIN OPERATION

- Cartesian product often requires a selection operation
- The selection operation most often requires that all attributes that are common to the relations are involved in the Cartesian product be equated
- Steps for natural join-
  1. Perform the Cartesian product of its two arguments
  2. Perform a selection forcing equality on those attributes that appear in both relational schemas
  3. Finally remove the duplicate attributes

NATURAL JOIN OPERATION

- Relations r,s

A	B	C
α	1	α
β	4	γ
γ	4	β
α	1	γ
δ	2	β

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

Natural join operation (contd.)

- A natural join operation can be rewritten as

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

- **Theta join** is a variant of natural join
- It is defined as

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

Assignment operation

- It is convenient at times to write relational algebra expression by assigning parts of it to temporary relation variables
- The assignment operation works like assignment in programming languages
- We can rewrite $r \bowtie s$ as

$$temp1 \leftarrow r \times s$$

$$temp2 \leftarrow \sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \ldots r.A_n = s.A_n}(temp1))$$

$$result \leftarrow \pi_{R \cup S}(temp2)$$

Outer join operation

- An extension of the join operation that avoids loss of information
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are **false** by definition.

Different forms of outer join

- Left outer join $⟕$
  - Includes the tuples from the left relation that did not match with any tuples in the right relation
  - Pads the tuples with null values for all other attributes from the right relation
  - Adds them to the result of the natural join
- We can rewrite $r ⟕ s$ as

$$r \bowtie s \cup (r - \pi_R(r \bowtie s)) \times \{(null, \ldots, null)\}$$

Here the constant relation {(null,…,null)} is on schema S-R

Different forms of outer join

- Similarly we can define-
- Right outer join $⟖$
- Full outer join $⟗$

Null value

- 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
- Aggregate functions simply ignore null values (as in SQL)
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)

Division Operation

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

  The result of r ÷ s is a relation on schema

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

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

Where tu means the concatenation of tuples t and u to produce a single tuple

Division operation (Contd.)

- 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

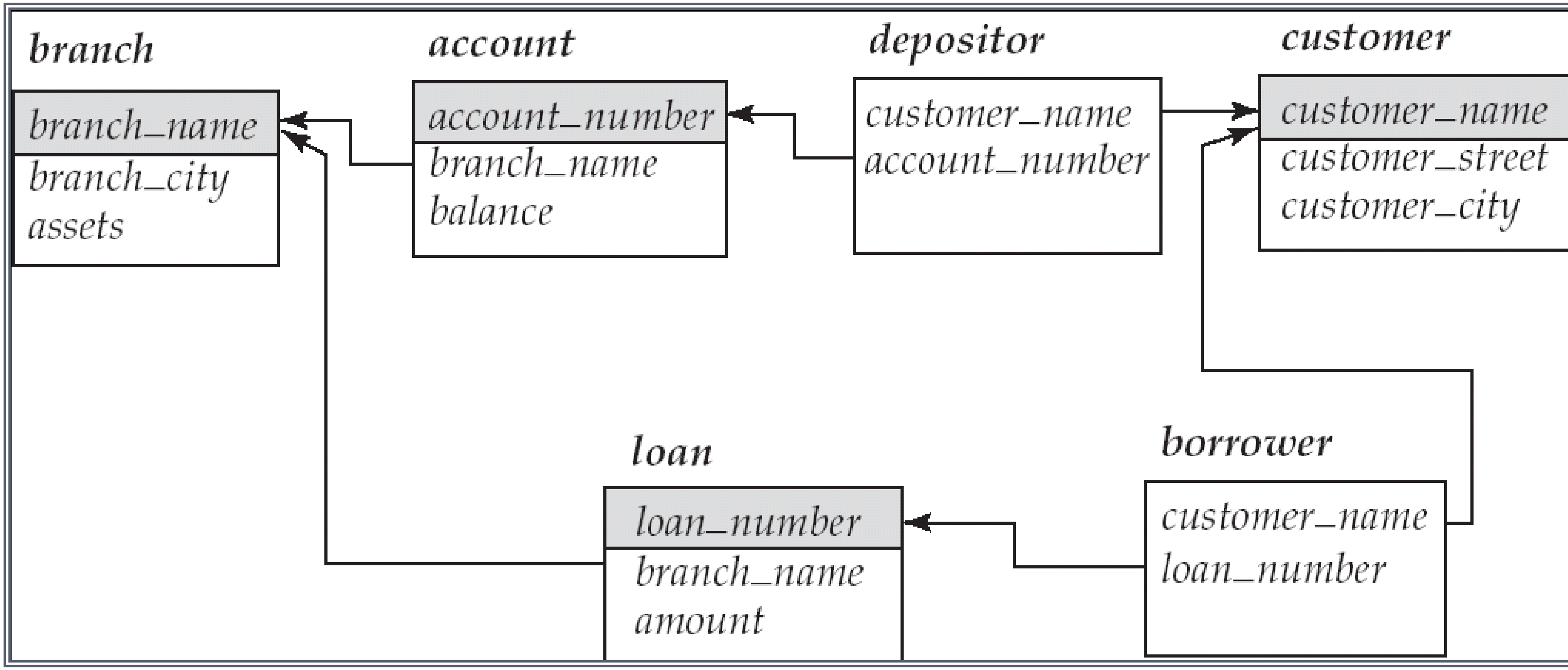
A
α
β

DIVISION OPERATION (CONTD.)

- Definition in terms of the basic algebra operation
  Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

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

Bank example



Example queries

- Find the names of all customers who have a loan and an account at bank.
  - $\Pi_{customer\_name}\ (borrower) \cap \Pi_{customer\_name}\ (depositor)$
- Find the name of all customers who have a loan at the bank and the loan amount
  - $\Pi_{customer\_name,\ loan\_number,\ amount}\ (borrower \bowtie\ loan)$

Example Queries (contd.)

- Find the largest account balance in the bank
  - $\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho_d\, account))$

EXAMPLE QUERIES (CONTD.)

- Find the name of customers who have an account at all the branches located in "Patna" city.

$$\Pi_{customer\_name, branch\_name}(depositor \bowtie account) \div \Pi_{branch\_name}(\sigma_{branch\_city=\text{"Patna"}}(branch))$$

Few more join operations

- Semi join
  - The left semi-join is similar to the natural join
  - The result of this semi-join is the set of all tuples in *r* for which there is a tuple in *s* that is equal on their common attribute names
  - $r\ semiJoin\ s = \Pi_R(r \bowtie s)$
- Anti join
  - It is similar to the natural join,
  - but the result of an anti-join is only those tuples in *r* for which there is *no* tuple in *s* that is equal on their common attribute names
  - $r\ antiJoin\ s = r - (r\ semiJoin\ s)$

Generalized projection

- An extension of projection which allows operations such as arithmetic and string functions to be used in the projection list
- $\Pi_{F1,F2,\ldots,Fn}(E)$
  - here F1,F2, …, Fn is an arithmetic expression involving constants and attributes in the schema of E
- Example: $\Pi_{ID,name,dept\_name,salary/12}(emp)$
- Example: $\Pi_{ID,(limit-balance)\ as\ credit\_available}(credit\_info)$

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.

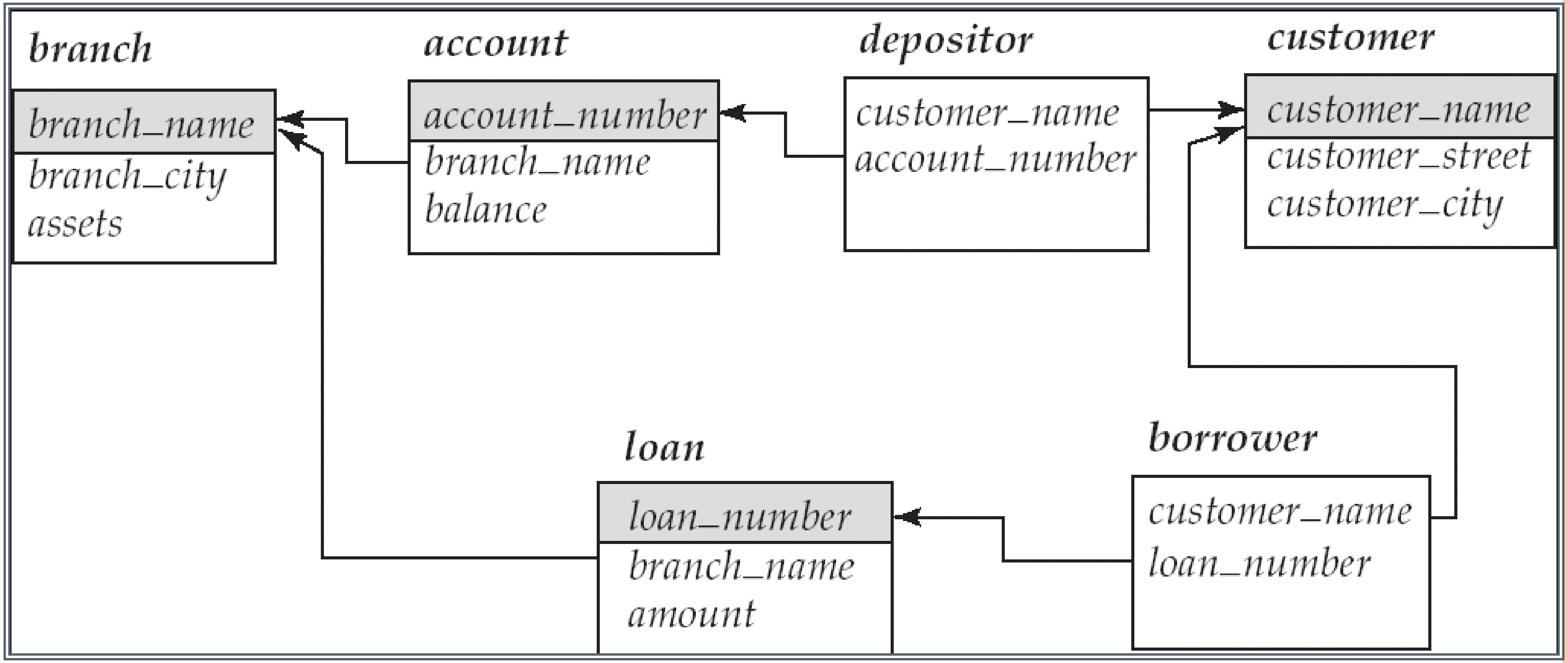
Deletion

- 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

Bank Example



Delete Example

- Delete all account records from "Patliputra" branch
  - $r1 \leftarrow \sigma_{branch\_name=\text{"}Patliputra\text{"}}$ (account $\bowtie$ depositor)
  - $depositor \leftarrow depositor - \Pi_{customer\_name,account\_number} (r1)$
  - $account \leftarrow account - \Pi_{account\_number,\ branch\_name,\ balance} (r1)$

- Delete all loan records with amount in the range of 0 to 50
  - $r2 \leftarrow \sigma_{amount>=0\ and\ amount <= 50}$ (loan $\bowtie$ borrower)
  - $loan \leftarrow loan - \Pi_{loan\_no,branch\_name,amount} (r2)$
  - $borrower \leftarrow borrower - \Pi_{customer\_name,loan\_no} (r2)$

- Delete all accounts at branches located in city 'Gaya'

$$r_1 \leftarrow \sigma_{branch\_city\,=\,\text{"}Gaya\text{"}} (account \bowtie branch)$$

$$\mathrm{r}_2 \leftarrow \prod_{account\_number,\; branch\_name,\; balance} (r_1)$$

$$r_3 \leftarrow \prod_{customer\_name,\; account\_number} (r_2 \bowtie \text{depositor})$$

$$account \leftarrow \text{account} - r_2$$

$$depositor \leftarrow \text{depositor} - r_3$$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted or
  - 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

Insert Example

- Insert information in the database specifying that Sumit has Rs 1200 in account A-973 at the Patliputra branch (assume Sumit's information is available in Customer relation)

$$account \leftarrow account \cup \{(\text{"A-973"}, \text{"Patliputra"}, 1200)\}$$

$$depositor \leftarrow depositor \cup \{(\text{"Sumit"}, \text{"A-973"})\}$$

- Provide as a gift for all loan customers in the Patliputra branch, a Rs 200 savings account. Let the loan number serve as the account number for the new savings account

$$r_1 \leftarrow (\sigma_{branch\_name = \text{"Patliputra"}} (borrower \bowtie loan))$$

$$r_2 \leftarrow (\Pi_{loan\_number,\ branch\_name,} (r_1))$$

$$account \leftarrow account \cup (r_2 \ \ x \ \{(200)\})$$

$$depositor \leftarrow depositor \cup \Pi_{customer\_name,\ loan\_number} (r_1)$$

UPDATING

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

$$r \leftarrow \Pi_{F1,F2,\ldots,Fi}(r)$$

Each F_i is either

  - the i^{th} attribute of r, if the i^{th} attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute

Update Example

- Make interest payments by increasing all account balances by 5 percent
- $account \leftarrow \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (account)$
- Pay all accounts with balances over Rs1,00,000 a six percent interest and pay all others five percent
- $account \leftarrow \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.06}\ (\sigma_{balance\ >\ 100000}\ (account\ ))$
  $\cup\ \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (\sigma_{balance \leq\ 100000}\ (account))$

Tuple Relational Calculus

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

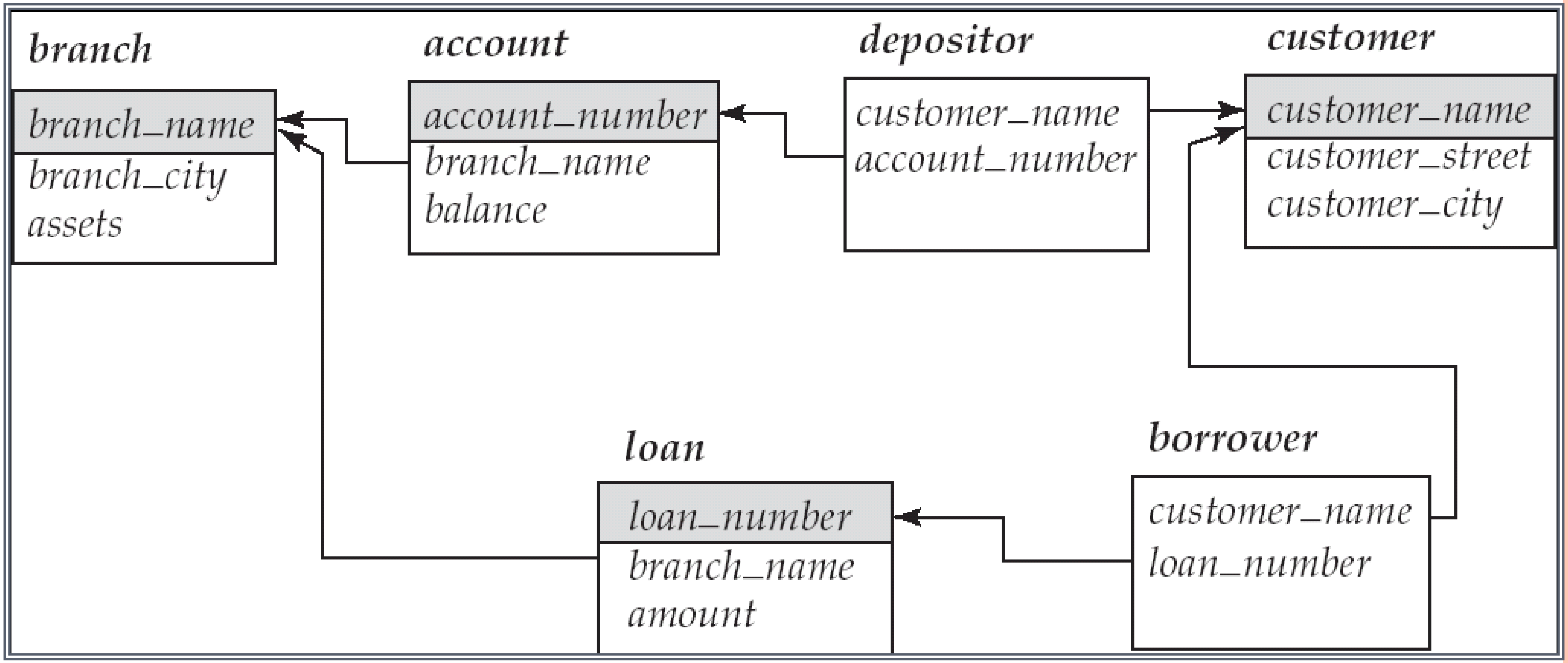
Predicate Calculus Formula

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

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

- 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
  - $\forall t \in r\, (Q\,(t\,)) \equiv Q(t)$ is true "for all" tuples t in relation r

Bank Example



Example Queries

- Find the *loan_number*, *branch_name*, and *amount* for loans of over Rs1200
  - $\{t \mid t \in loan \wedge t[amount] > 1200\}$
- Find the loan number for each loan of an amount greater than Rs1200
  - $\{t \mid \exists s \in loan\ (t\ [loan\_number\ ] = s\ [loan\_number\ ] \wedge s\ [amount\ ] > 1200)\}$
- Find the names of all customers having a loan, an account, or both at the bank
  - $\{t \mid \exists s \in borrower\ (\ t\ [customer\_name\ ] = s\ [customer\_name\ ])$
    $\vee \exists u \in depositor\ (\ t\ [customer\_name\ ] = u\ [customer\_name\ ])\}$

Example Queries (2)

- Find the names of all customers who have a loan and an account at the bank
  - $\{t \mid \exists s \in borrower\ (\ t\ [customer\_name\ ] = s\ [customer\_name\ ])$
    $\wedge \exists u \in depositor\ (\ t\ [customer\_name\ ] = u\ [customer\_name]\ )\}$
- Find the names of all customers having a loan at Patliputra branch
  - $\{t \mid \exists s \in borrower\ (t\ [customer\_name\ ] = s\ [customer\_name\ ]$
    $\wedge \exists u \in loan\ (u\ [branch\_name\ ] = \text{“Patliputra”}$
    $\wedge\ u\ [loan\_number\ ] = s\ [loan\_number\ ]))\}$

Example Queries (3)

- Find the names of all customers who have a loan at Patliputra branch, but no account at any branch of the bank
  - $\{t \mid \exists s \in borrower\ (t\ [customer\_name\ ] = s$ [customer_name]
    $\wedge \exists u \in loan\ (u\ [branch\_name\ ] =$ "Patliputra"
    $\wedge\ u\ [loan\_number\ ] = s$ [loan_$number$]))
    $\wedge$ **not** $\exists v \in depositor\ (v\ [customer\_name\ ] =$
    $t\ [customer\_name\ ])\}$

Example Queries (4)

- Find the names of all customers having a loan from Patliputra branch, and the cities in which they live
  - $\{t \mid \exists s \in loan\ (s\ [branch\_name\ ] = \text{“Patliputra”}$
    $\wedge \exists u \in borrower\ (u\ [loan\_number\ ] = s\ [loan\_number\ ]$
    $\wedge\ t\ [customer\_name\ ] = u\ [customer\_name\ ]$
    $\wedge \exists\ v \in customer\ (u\ [customer\_name\ ] = v\ [customer\_name\ ]$
    $\wedge\ t\ [customer\_city\ ] = v\ [customer\_city\ ])))\}$

Example Queries (5)

- Find the names of all customers who have an account at all branches located in branch_city = "Dhanbad":
  - $\{t \mid \forall\, u \in branch\ (u\ [branch\_city\ ] = \text{“Dhanbad”} \Rightarrow \exists\ v \in account\ (v\ [branch\_name] = u\ [branch\_name\ ] \wedge \exists\ w \in depositor\ (\ w[account\_number\ ] = v\ [account\_number\ ] \wedge (\ t\ [customer\_name\ ] = w\ [customer\_name\ ]))))\}$

Safety of Expression

- It is possible to write tuple calculus expressions that generate infinite relation.
  - For example, $\{ \text{t} \mid \neg\, t \in r \}$ results in an infinite relation
- To guard against the problem, we restrict the set of allowable expressions to safe expressions

Safe Expression

- Let's consider an expression $\{t \mid P(t)\}$ in the tuple relational calculus
  - dom(P): the set of all values referenced by P
  - They include the values mentioned in P itself, as well as values appear in a tuple of a relation mentioned in P or constants that appear in P
  - dom(t | t ∈ customer(t [*cust_city*] = 'Patna') is the set containing "Patna" as well as the values appearing in any attribute of any tuple in customer
- The expression is said to be *safe* if every component of t appears from the dom(P)
  - E.g. { t | ¬ t ∈ customer(t [*cust_city*] = 'Patna')} is not safe --- as the result is an infinite set and some of the values may not appear in any relation or tuples or constants in P.

Domain Relational Calculus

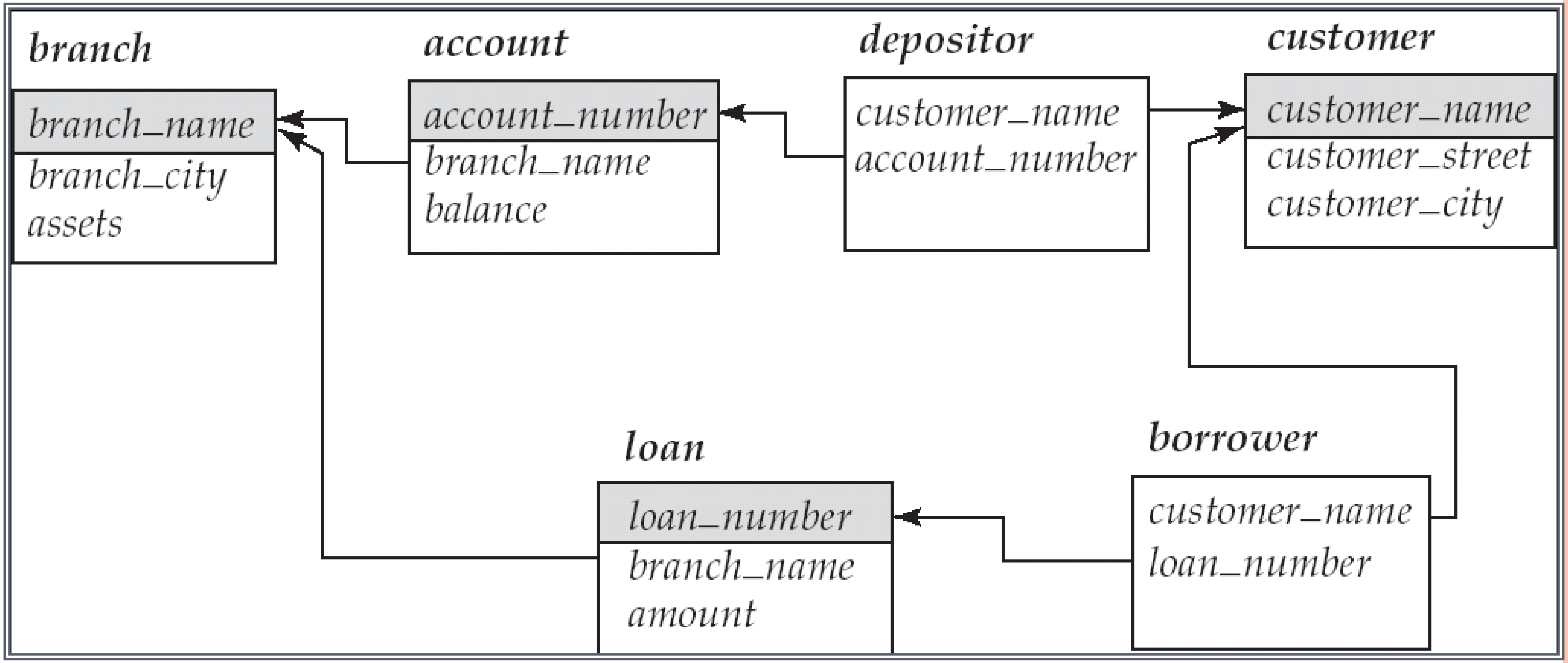
Domain Relational Calculus

- Another nonprocedural query language equivalent in power to the tuple relational calculus
- Forms the basis of widely used QBE (Query By Example) language
- Each query is an expression of the form:

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

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

Bank Example



Example Queries

- Find the *loan_number*, *branch_name*, and *amount* for loans of over Rs1200
  - $\{< l, b, a > \mid < l, b, a > \in loan \wedge a > 1200\}$
- Find the names of all customers who have a loan of over Rs1200
  - $\{< c > \mid \exists\, l, b, a\ (< c, l > \in borrower \wedge < l, b, a > \in loan \wedge a > 1200)\}$
- Find the names of all customers who have a loan from Patliputra branch and the loan amount:
  - $\{< c, a > \mid \exists\, l\ (< c, l > \in borrower \wedge \exists b\ (< l, b, a > \in loan \wedge b = \text{“Patliputra”}))\}$
  - $\{< c, a > \mid \exists\, l\ (< c, l > \in borrower \wedge < l, \text{“Patliputra”}, a > \in loan)\}$

Example Queries (2)

- Find the names of all customers having a loan, an account, or both at Patliputra branch:
  - $\{< c > \mid \exists\, l\, (< c, l > \in borrower$
    $\wedge \exists\, b,a\, (< l, b, a > \in loan \wedge b = \text{“Patliputra”}))$
    $\vee \exists\, ac\, (< c, ac > \in depositor \wedge\ \exists\, br,n,ba$
    $(< ac, br, ba > \in account \wedge br = \text{“Patliputra”}))\}$

Example Queries (3)

- Find the names of all customers who have an account at all branches located in Dhanbad:
- $\{< c > \mid \forall\ x,y,z\ (< x, y, z > \in branch \wedge y = \text{“Dhanbad”}) \Rightarrow$
  $\exists\ a,b\ (< a, x, b > \in account \wedge < c,a > \in depositor)\}$

Safety of Expression

The expression:

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

is safe if -

all values that appear in tuples of the expression are values from *dom* (P)

Expressive Power of Languages

- The following languages are equivalent
  - The basic relational algebra (without extended relational algebra operation)
  - The tuple relational calculus (restricted to safe expression)
  - The domain relational calculus (restricted to safe expression)