Relational Model

崇志宏

东南大学数据与智能实验室（D &. Intel Lab）

https://cse.seu.edu.cn/2019/0105/c23024a257490/page.htm

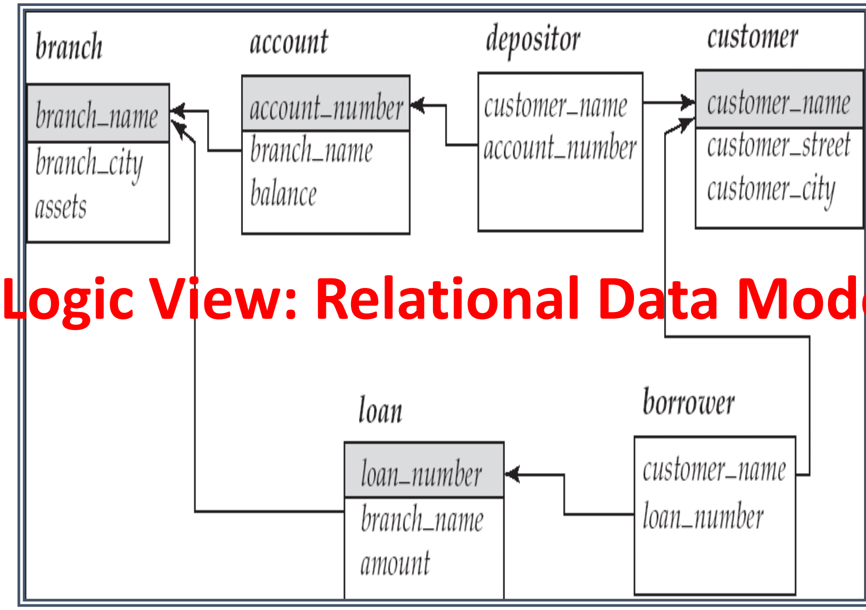
Chapter 2: Relational Model

- **Data Model and Relational Model**
  - Data Structure, Operation, Constraint
- **Relational Data Model and Relational Databases**
  - Fundamental Relational-Algebra-Operations
  - Additional Relational-Algebra-Operations
  - Extended Relational-Algebra-Operations
- **Null Values**
- **Modification of the Database**

关系数据库的成就（1）

逻辑模式

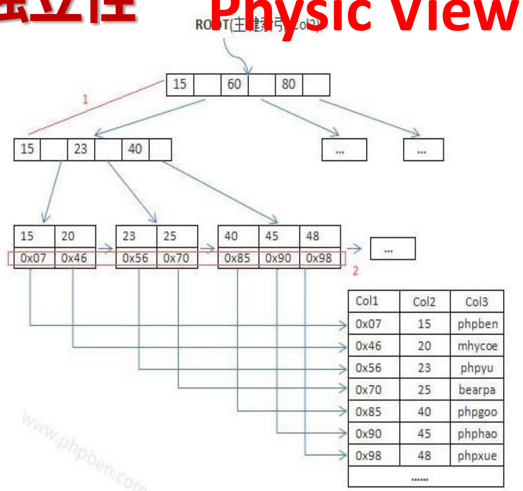
Logic View: Relational Data Model

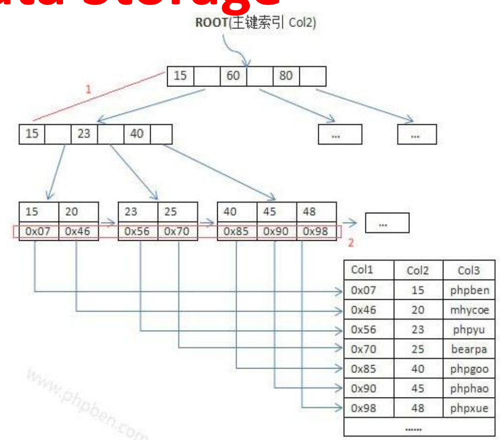


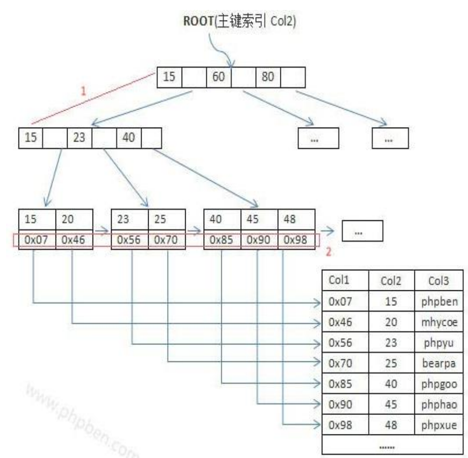
数据独立性

Physic View: Data Storage

物理模式



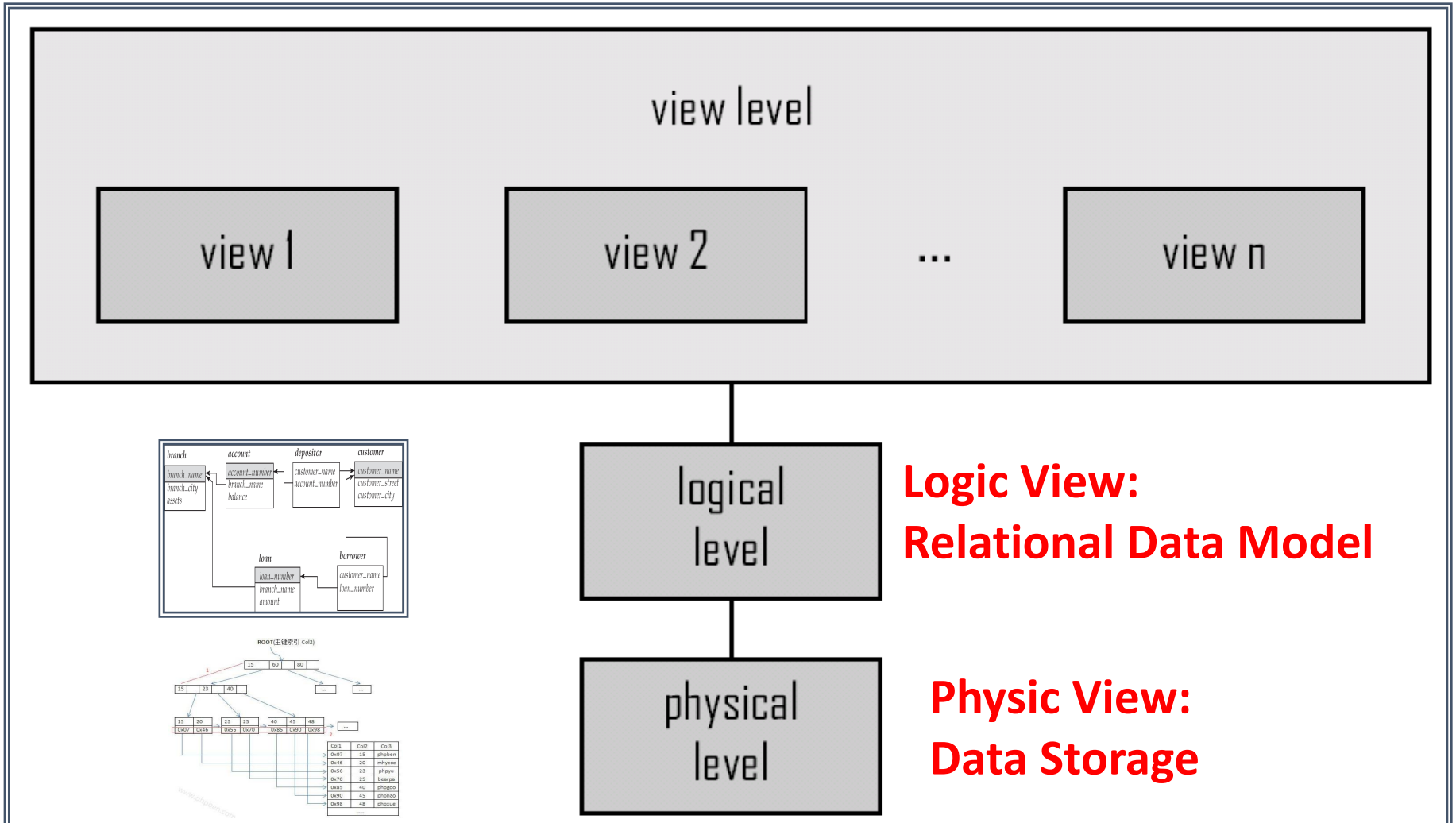




工业级的强一致性协议：全球同步和保证高吞吐的一致性

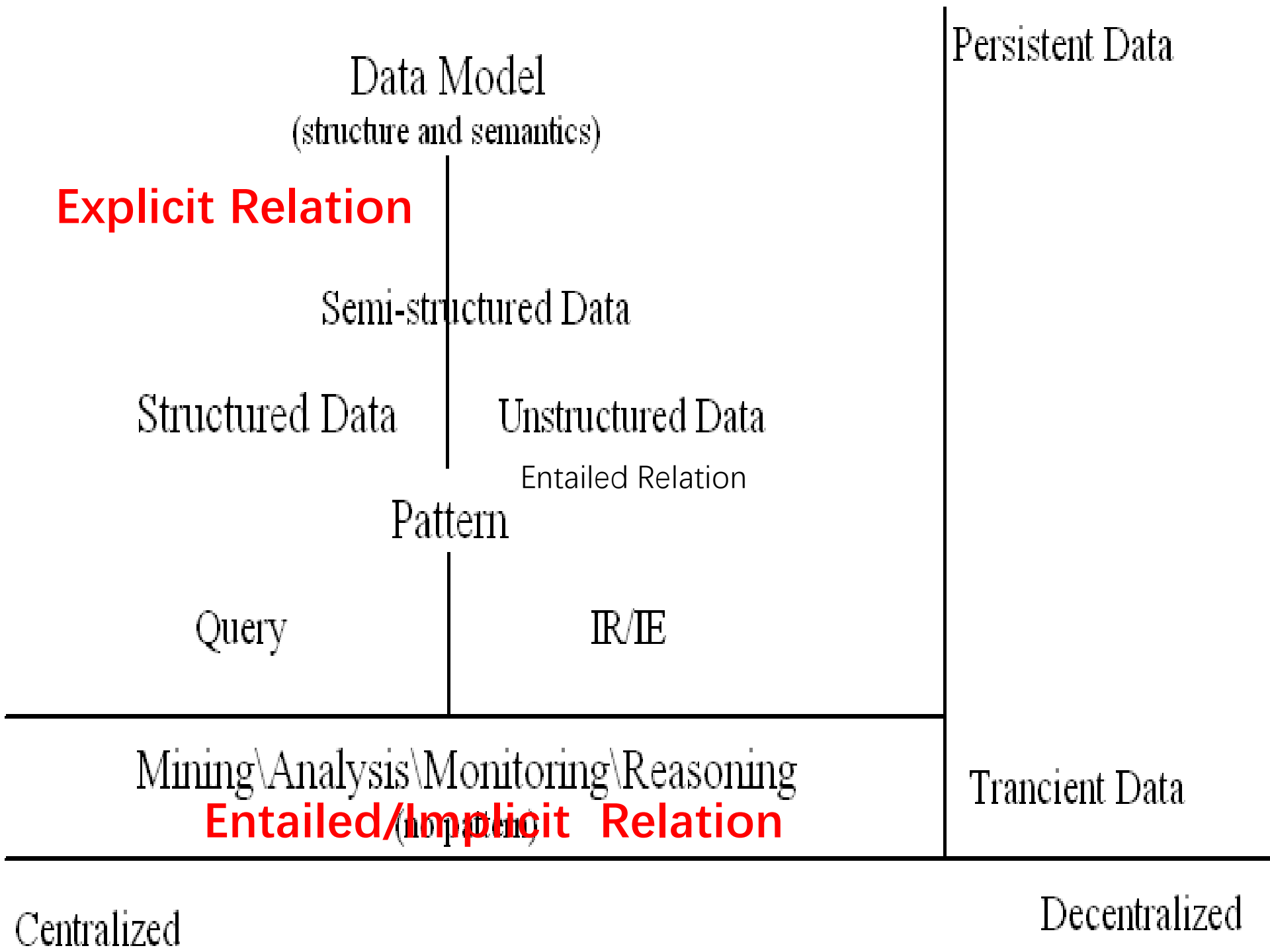
View of Database

An architecture for a database system



Relation

- **Explicit Relation**
  - Relation: Table
  - Relation of Relation: Relation of Tables
  - Entailed Relation： Logic entailed
- **Implicit Relation: Probabilistic Inference**


Persistent Data
Data Model
(structure and semantics)
Explicit Relation
Semi-structured Data
Structured Data
Unstructured Data
Entailed Relation
Pattern
Query
IR/IE
Mining\Analysis\Monitoring\Reasoning
(no pattern)
Entailed/Implicit Relation
Trancient Data
Centralized
Decentralized

Data Model

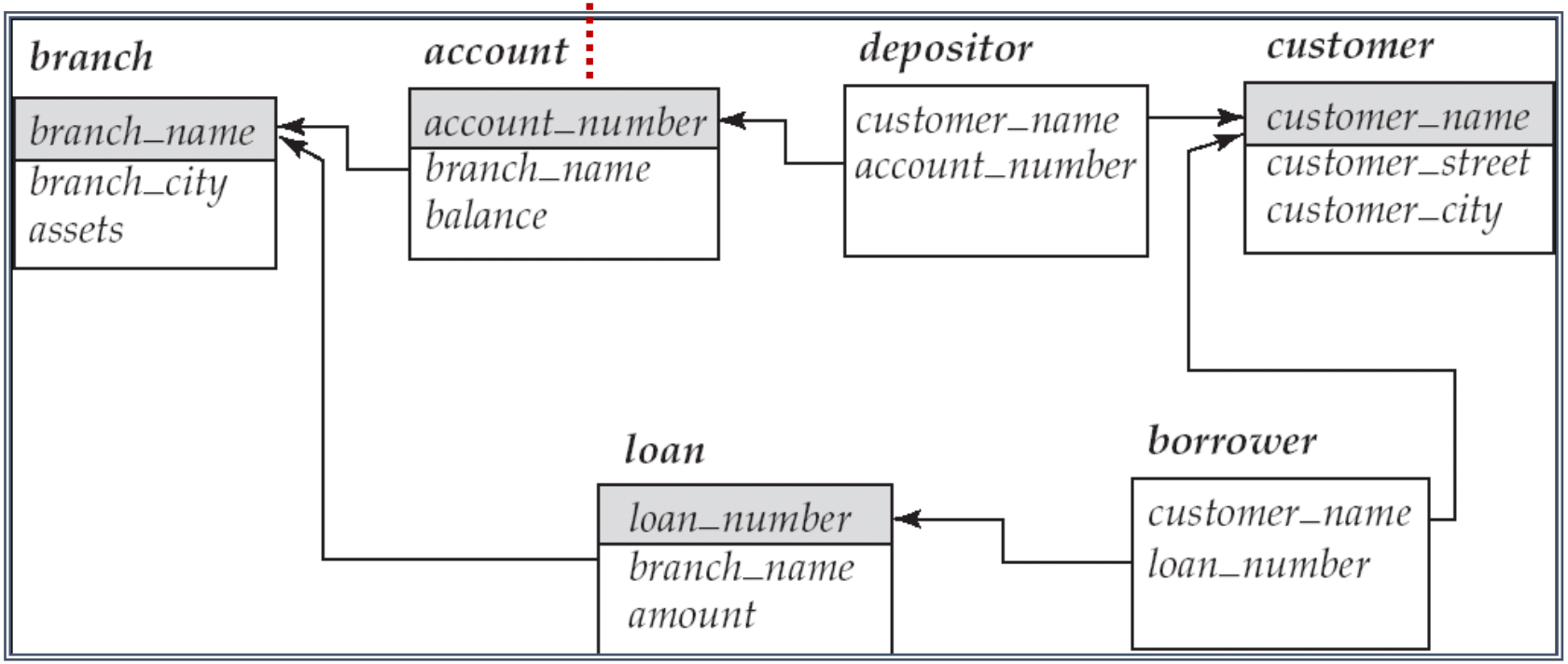
- **Conceptions and Tools for Data Description**
  - Data Structure
  - Data Operation
  - Data Constraint
- **Relational Data Model**
  - Relation as data Structure: Table
  - Algebra as Data Operation
  - Primary/ Foreign Key Pairs as Data Constraint
    - Entity Completion Constraint
    - Reference Completion Constraint
    - Assertion Constraint

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

Example of Relations

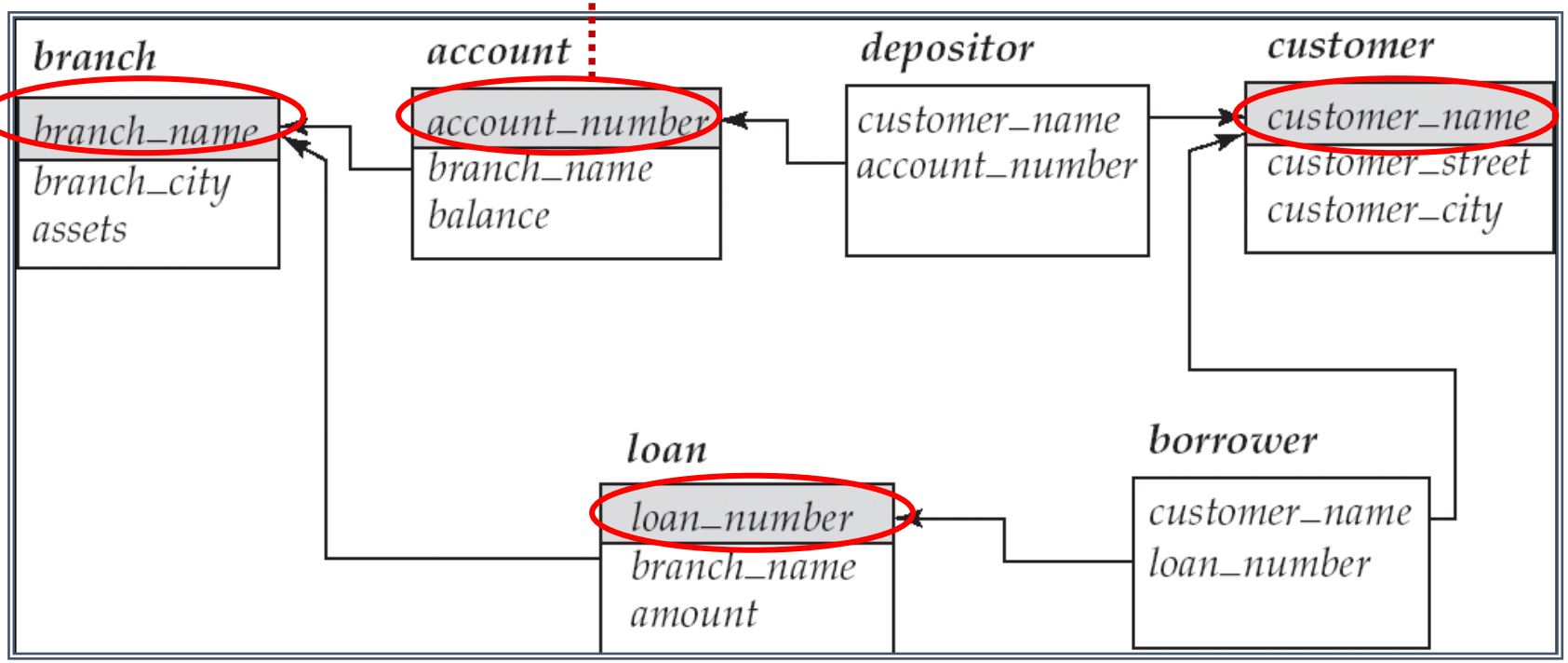
account_number	branch_name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350



Example of Relations

Data Structure as Tables(Relations)

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

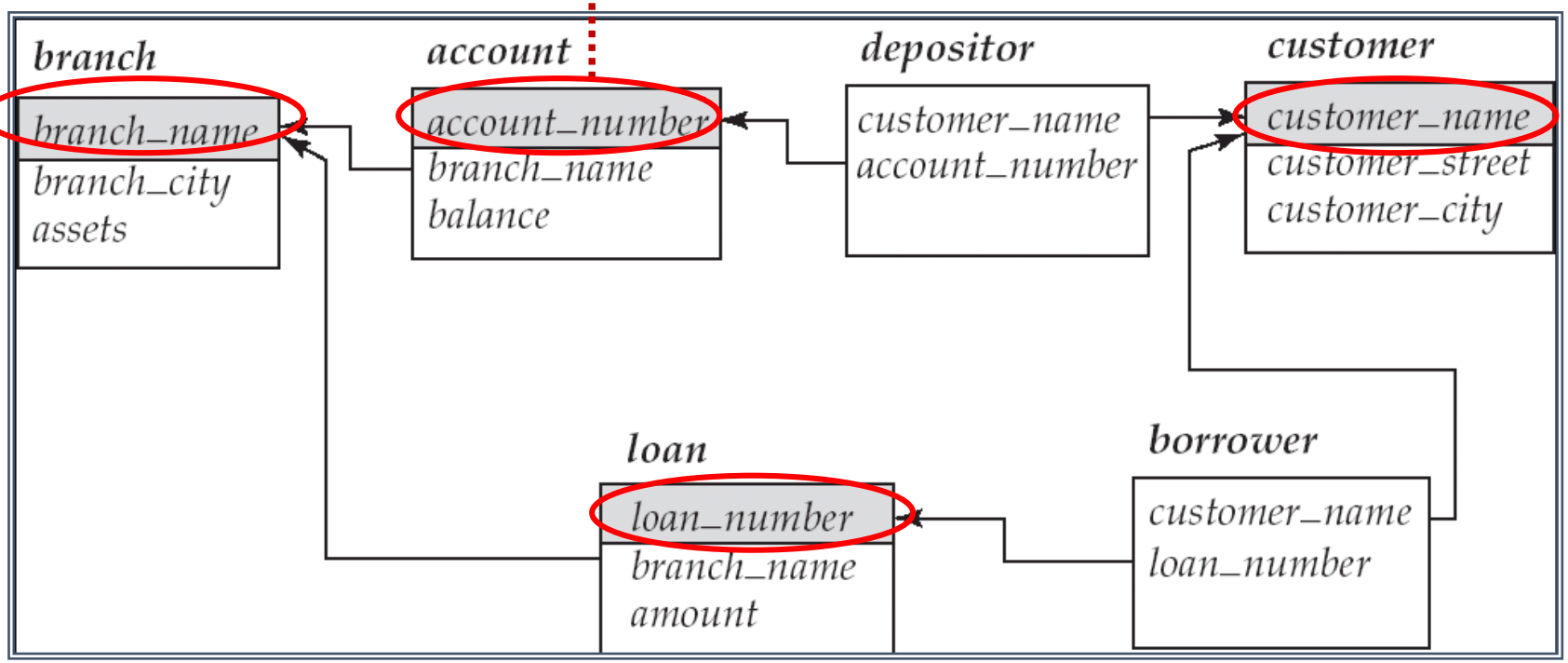


Example of Relations

Data Structure as Tables(Relations)

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

1. **How to find depositors/borrowers?**
2. **How to find those who are both depositors and borrowers?**



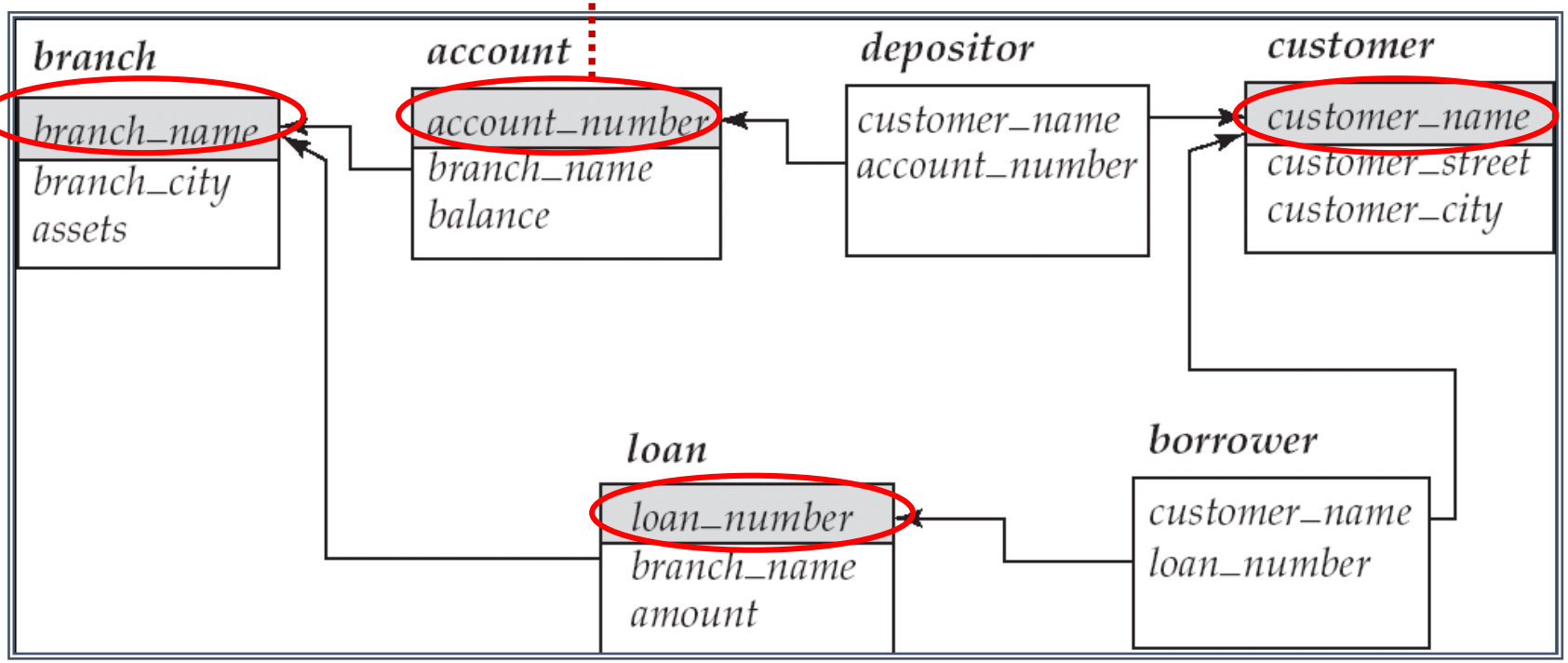
Example of Relations

Data Structure as Tables(Relations)

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

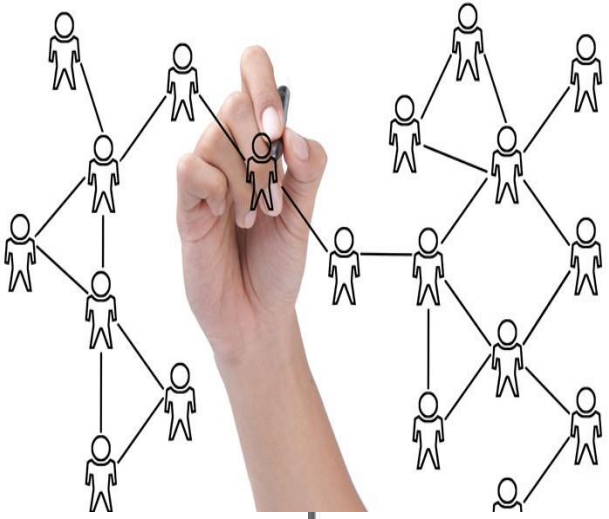
1. **How to find depositors/borrowers?**
2. **How to find those who are both depositors and borrowers?**

Need some data operations to compute answers!

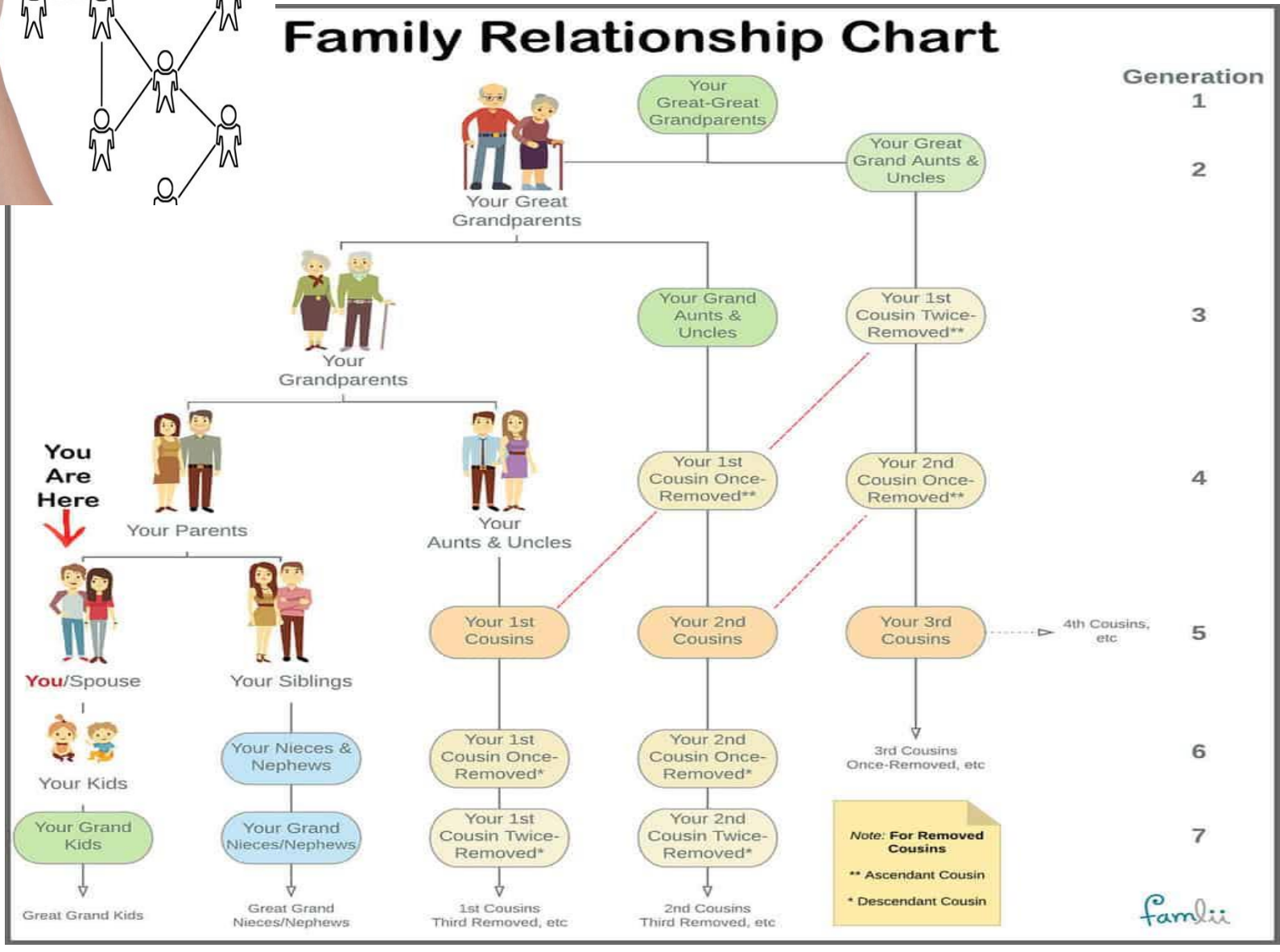


Data Model

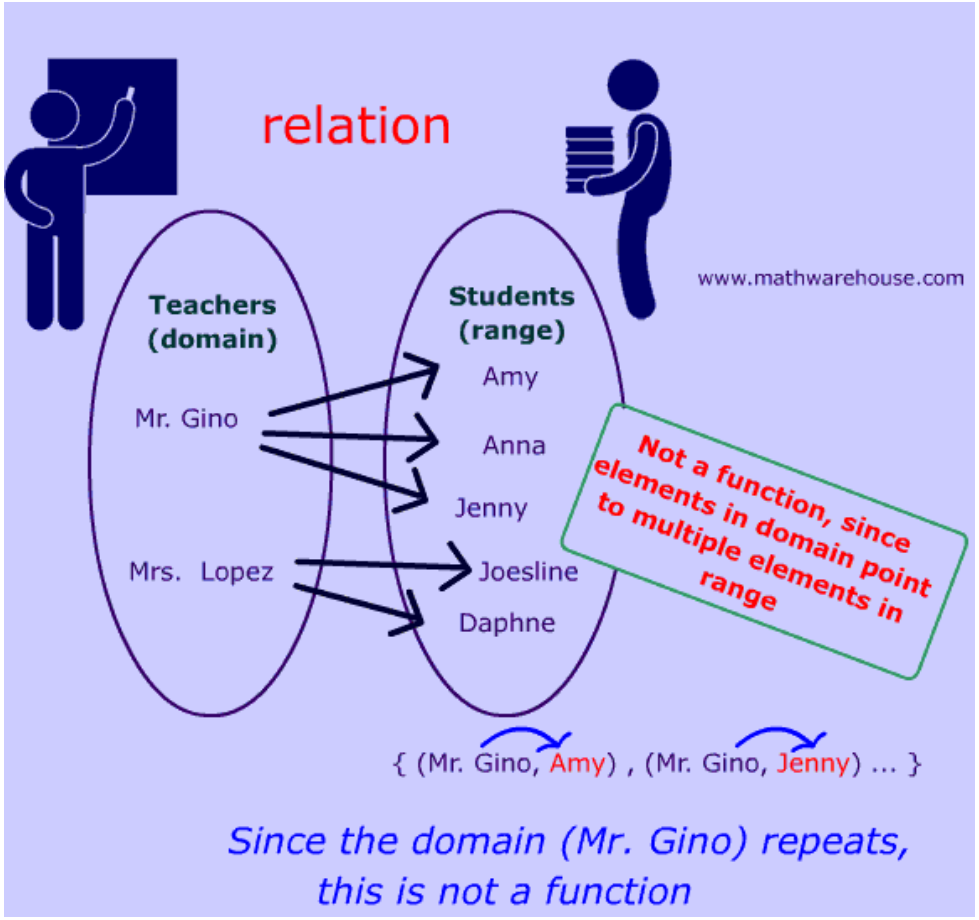
- **Conceptions and Tools for Data Description**
  - **Data Structure**
  - **Data Operation**
  - Data Constraint
- **Relational Data Model**
  - **Relation as data Structure: Table**
  - **Algebra as Data Operation**
  - Primary/ Foreign Key Pairs as Data Constraint
    - Entity Completion Constraint
    - Reference Completion Constraint
    - Assertion Constraint

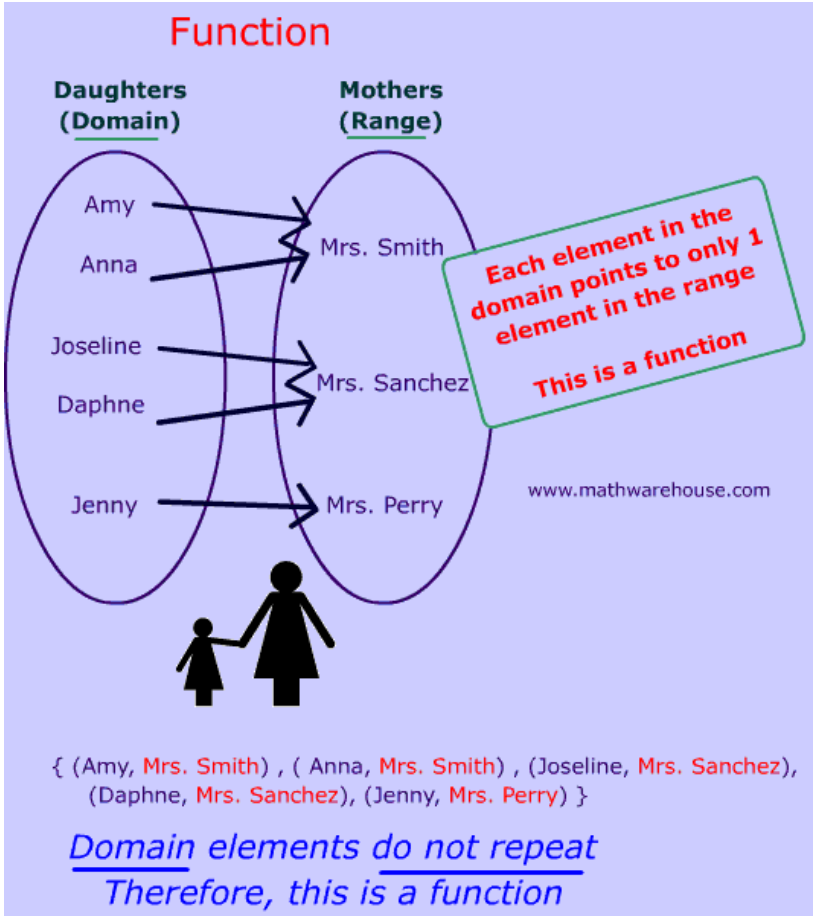


What is Relation?

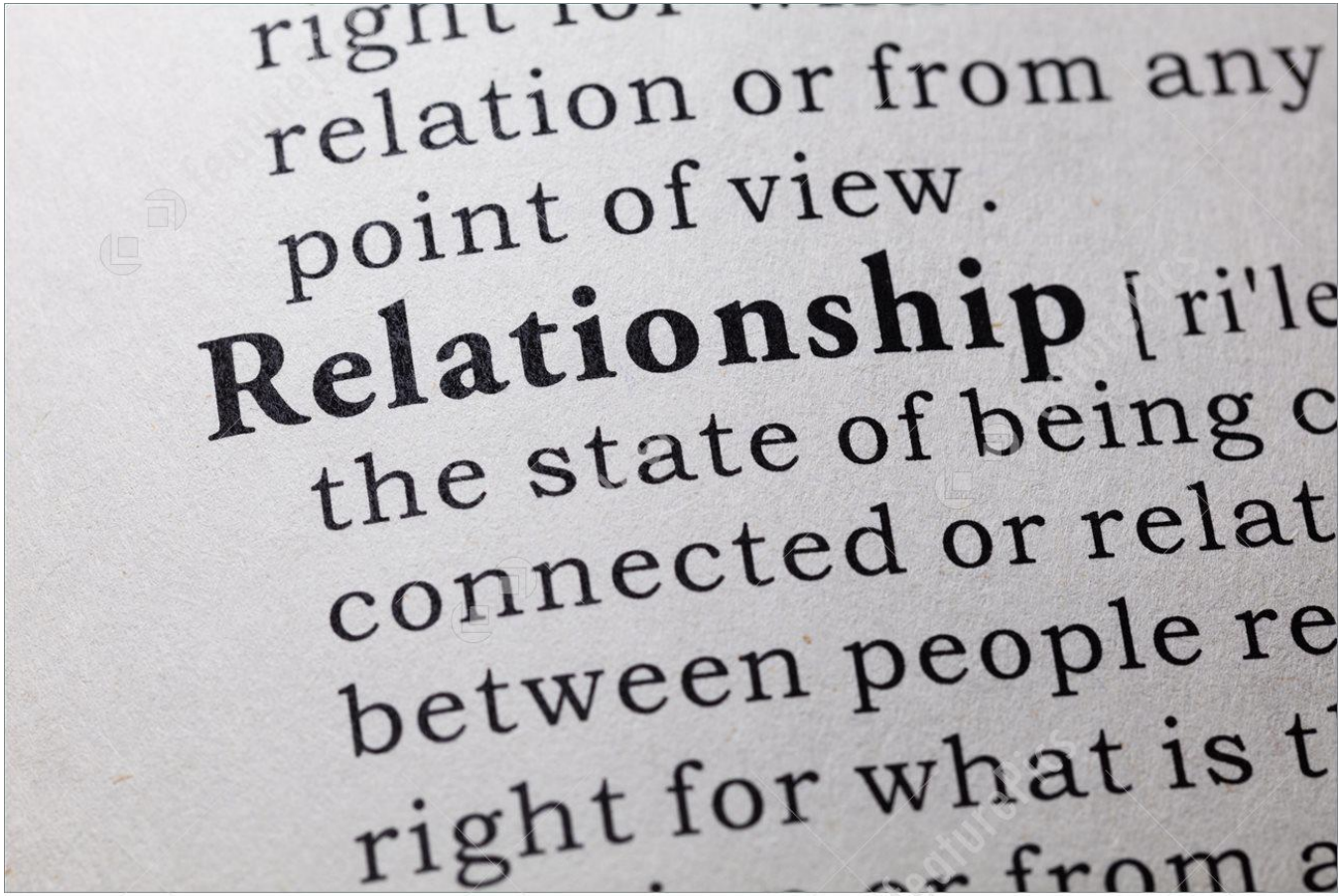


What is Relation?





What is Relation?

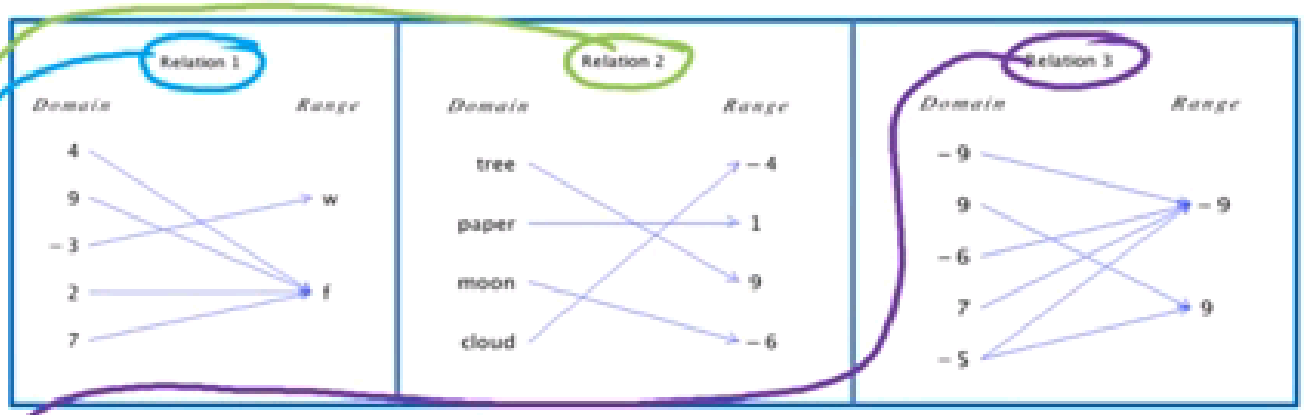


What is Relation?

What is Relation?

Relations versus Functions

Ex: For each relation below, decide whether or not it is a function.



- The set of first components = domain
- The set of second components = range

Relation 1 = {(4, f), (9, f), (-3, w), (2, f), (7, f)}

Relation 2 = {(tree, 9), (paper, 1), (moon, -6), (cloud, -4)}

Relation 3 = {(-9, -9), (9, 9), (-6, -9), (7, -9), (-5, -9), (-5, 9)}

What is Relation?

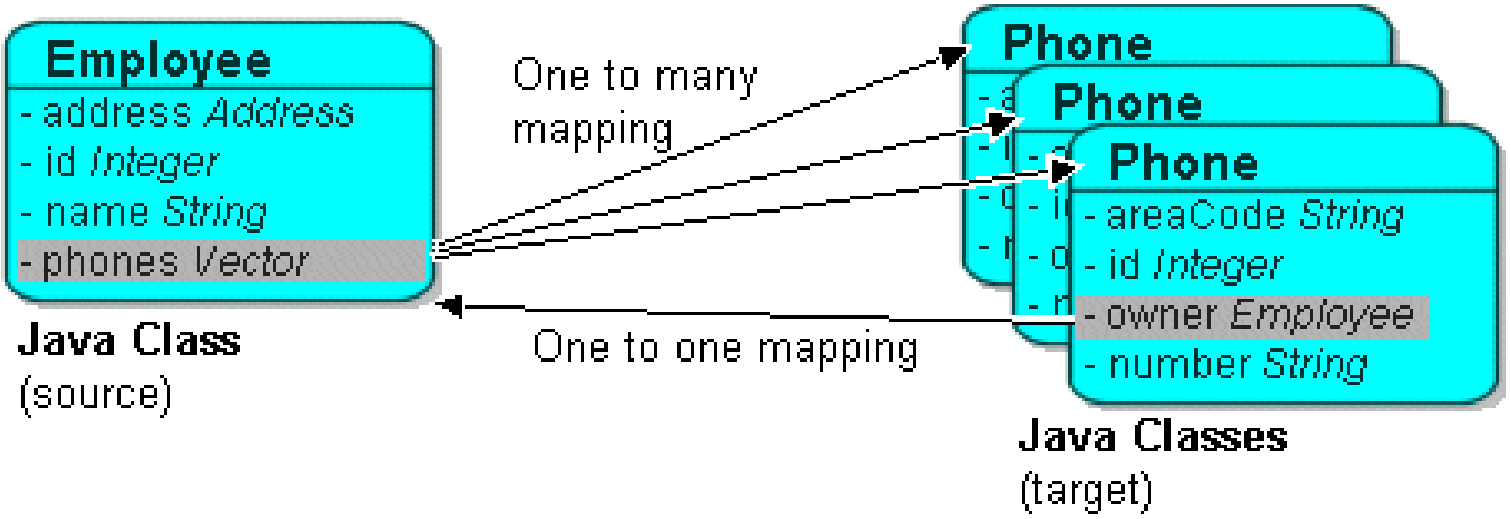
Definition 3 A relation R in a set A is called

(i) reflexive, if $(a, a) \in R$, for every $a \in A$,

(ii) symmetric, if $(a_1, a_2) \in R$ implies that $(a_2, a_1) \in R$, for all $a_1, a_2 \in A$.

(iii) transitive, if $(a_1, a_2) \in R$ and $(a_2, a_3) \in R$ implies that $(a_1, a_3) \in R$, for all $a_1, a_2, a_3 \in A$.

What is Relation?



EMPLOYEE table

EMP_ID	NAME	ADDR_ID
103	John Doe	305
104	Jane Smith	226
105	Tom Jones	274

Relational Database

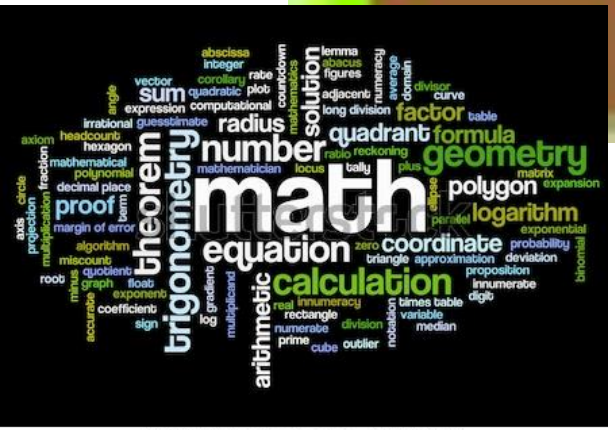
PHONE table (target)

Java Classes

PH_ID	AREA_CODE	EMP_ID	NUMBER
25	613	105	555-7635
26	603	96	555-8251
27	407	105	555-5649

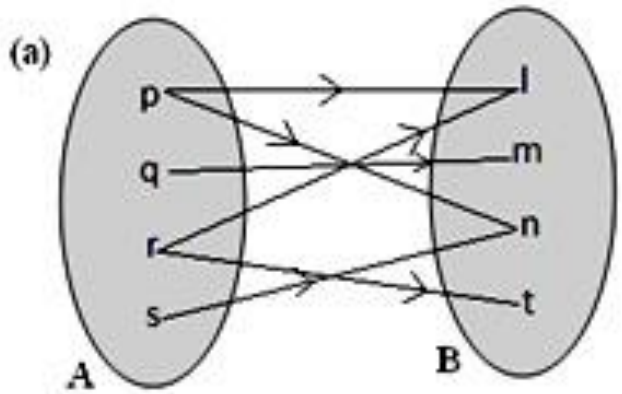
Foreign key reference

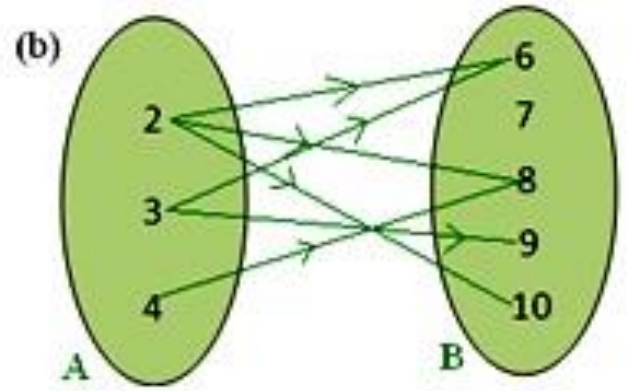
What is Relation?

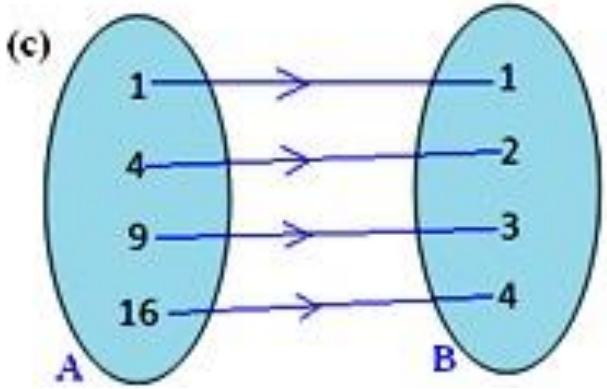


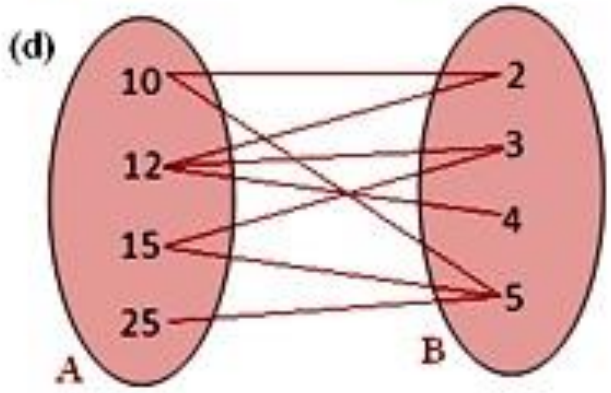
What is Relation?

$R \subseteq A \times B$

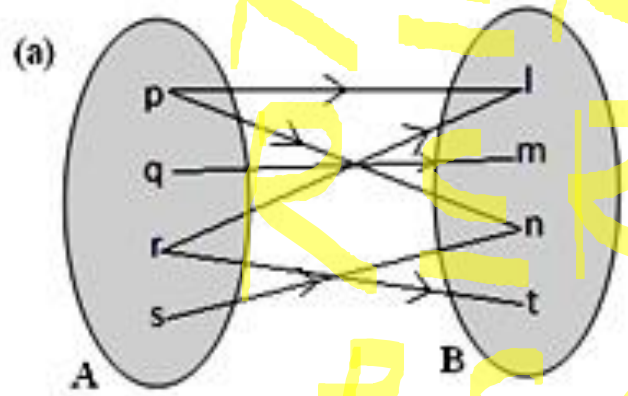


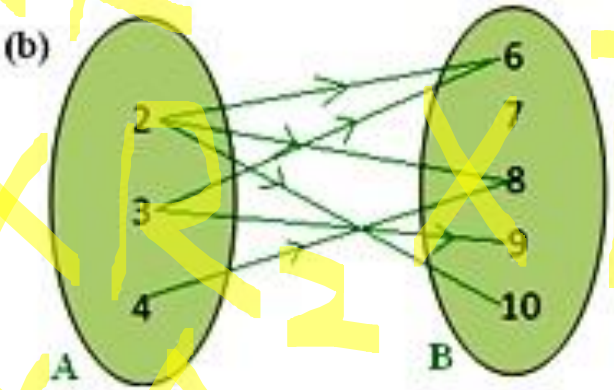


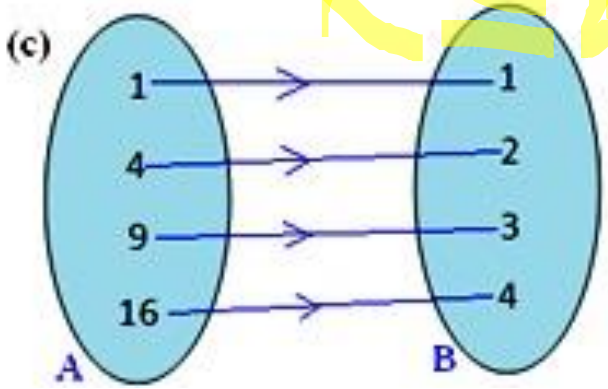


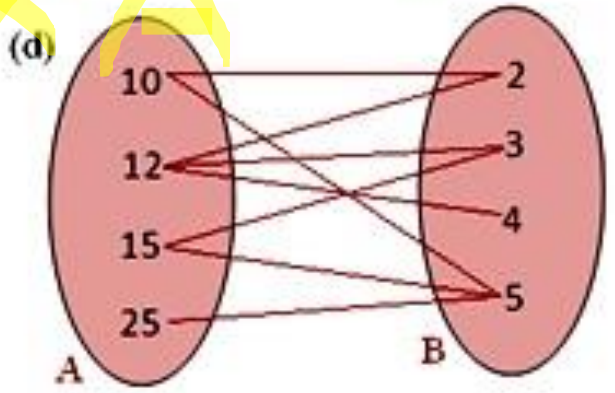


What is Relation?









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

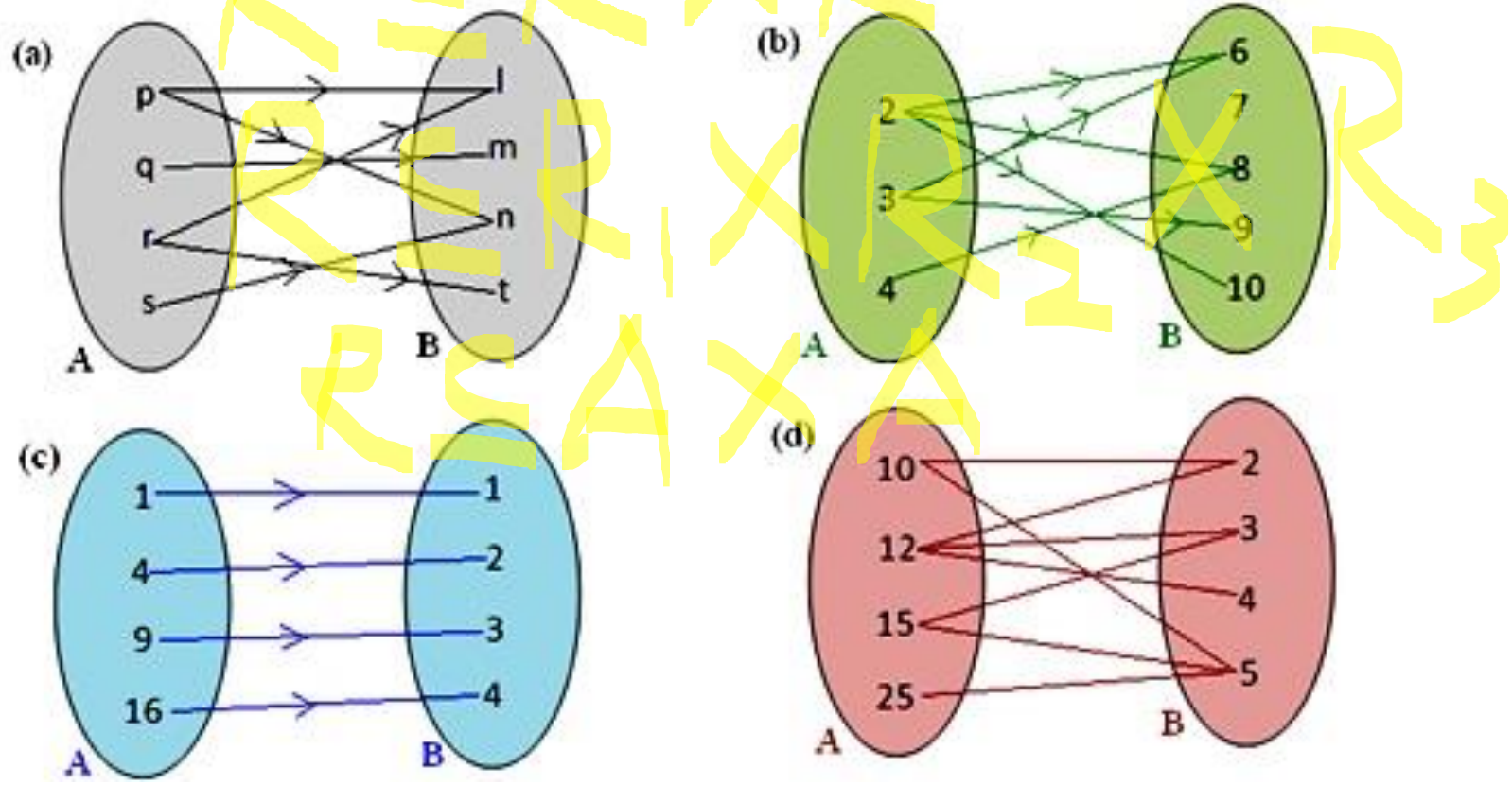
s

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

What is Relation?

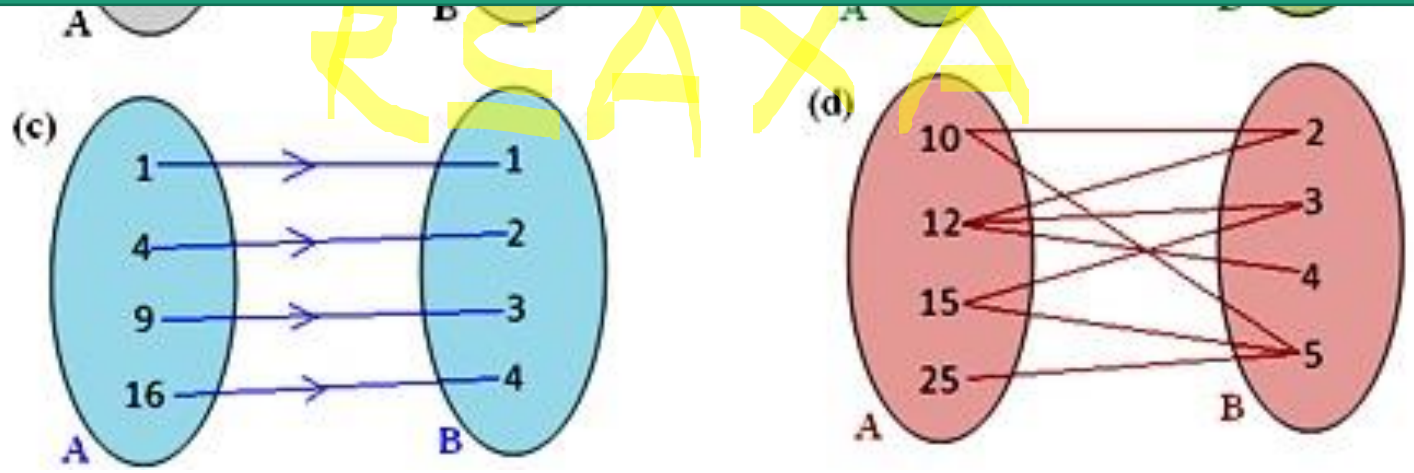
Relation is the subset of Cartesian product of sets



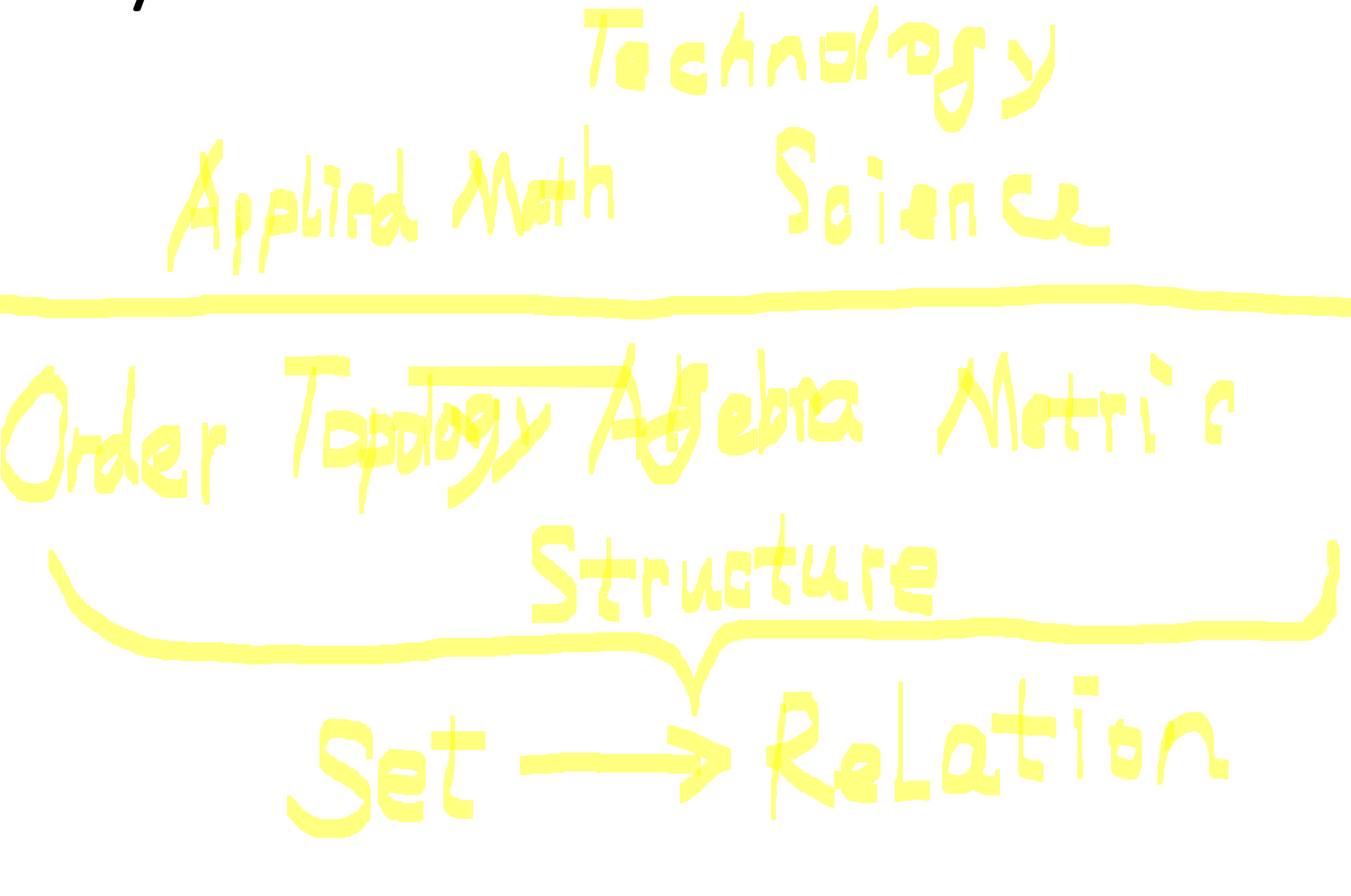
What is Relation?

Relation is the subset of Cartesian product of sets

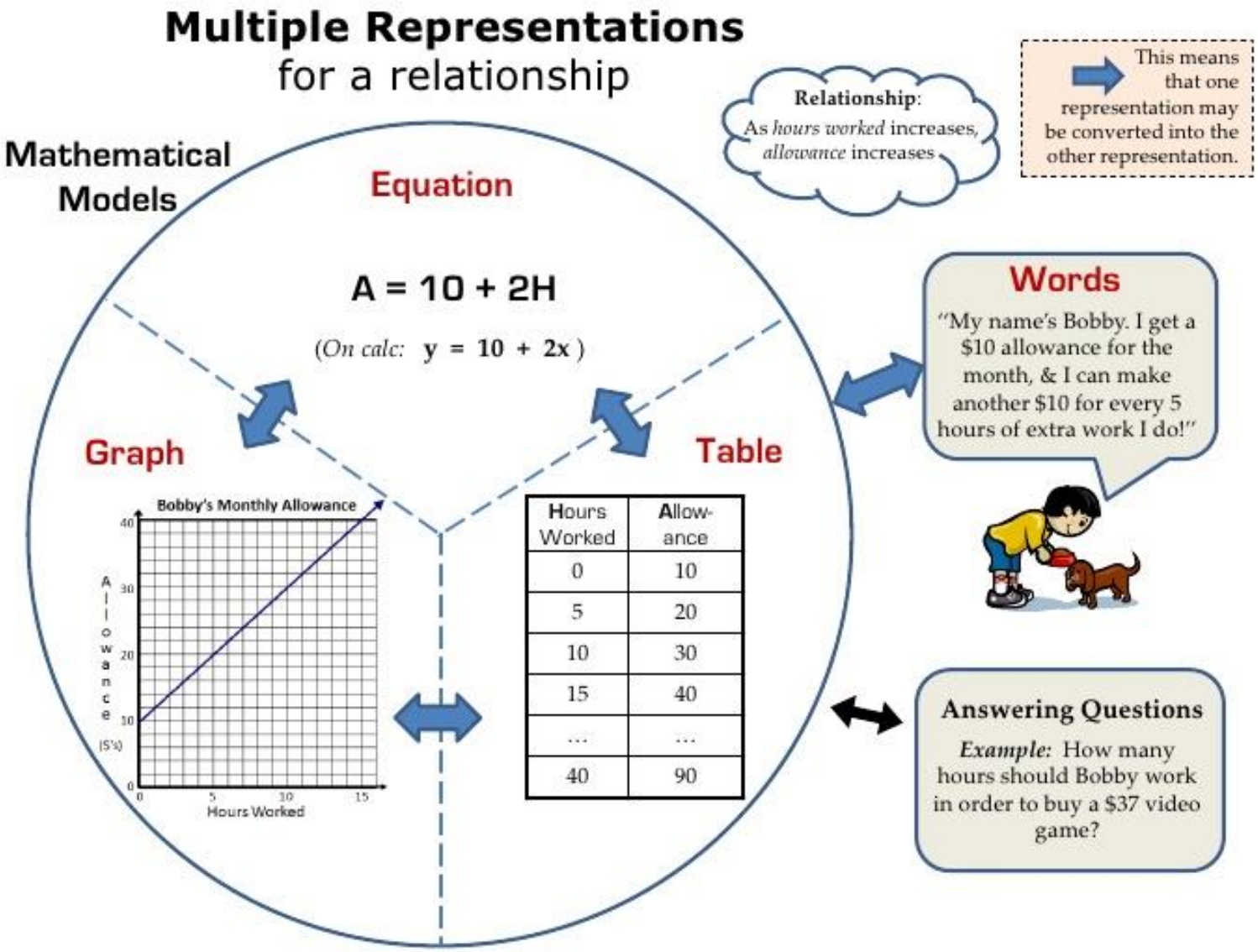
1. **Relation is a set**
2. **Relation of relations is a set, produced by the Cartesian product of relations(sets)**



Why Relation?



What is Relation?



Chapter 2: Relational Model

- **Data Model and Relational Model**
  - Data Structure, Operation, Constraint
- **Relational Data Model and Relational Databases**
  - Fundamental Relational-Algebra-Operations
  - Additional Relational-Algebra-Operations
  - Extended Relational-Algebra-Operations
- **Null Values**
- **Modification of the Database**

- **Given sets D_1, D_2, …. D_n, relation *r* is a subset of**

  D_1 x D_2 x … x D_n

  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, …}
    /* Set of all customer names */
  - *customer_street* = {Main, North, Park, …} /* set of all street names*/
  - *customer_city* = {Harrison, Rye, Pittsfield, …} /* set of all city names */

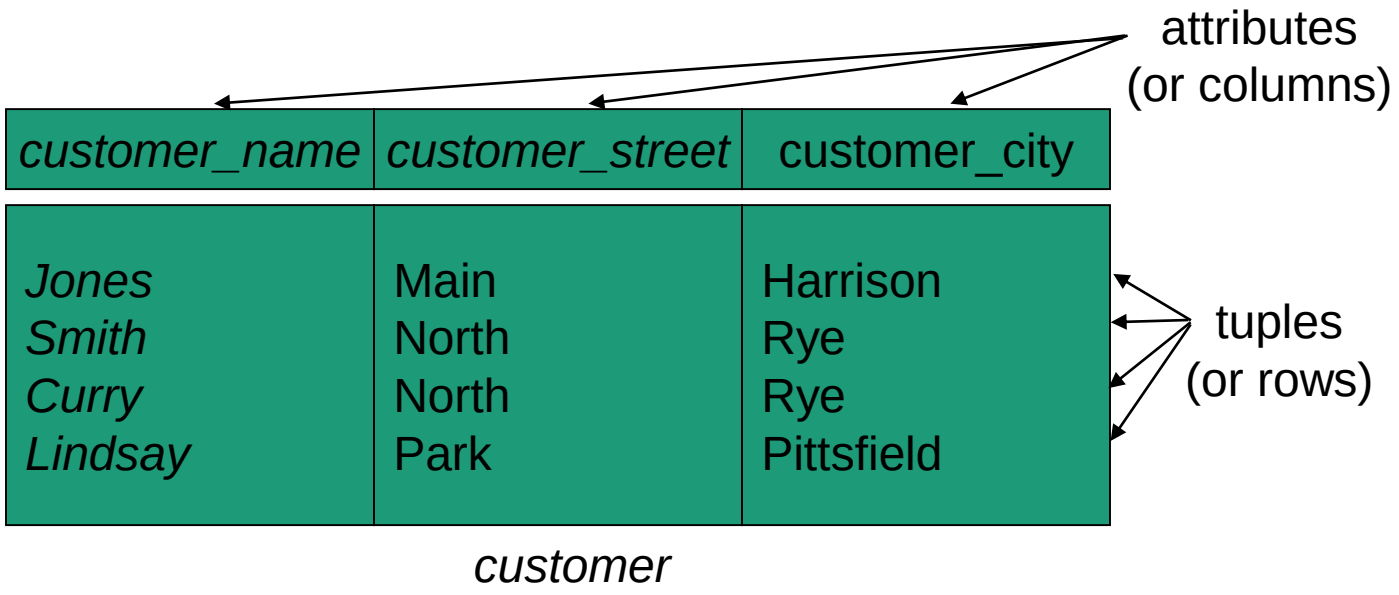
  Then *r* = { (Jones, Main, Harrison),
  (Smith, North, Rye),
  (Curry, North, Rye),
  (Lindsay, Park, Pittsfield) }

  is a relation over

  customer_name x customer_street x customer_city

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**



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. the value of an attribute can be an account number, but cannot be a set of account numbers
- Domain is said to be atomic if all its members are 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***

  Example:

  Customer_schema = (*customer_name, customer_street, customer_city*)

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

Example:

customer (Customer_schema)

Relations are Unordered

Order of tuples is irrelevant (tuples may be stored in an arbitrary order)

Example: *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 is broken up into relations storing one part of the information

  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.,if two customers own an account (What gets repeated?)
  - **the need for null values**
    - e.g., to represent a customer without an account
- Normalization theory (Chapter 7) 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

Keys

- Let $K \subseteq R$
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - by "possible *r* " we mean a relation *r* that could exist in the enterprise we are modeling.
  - Example: {*customer_name, customer_street*} and

    {*customer_name*}

    are both superkeys of *Customer*, if no two customers can possibly have the same name
    - In real life, an attribute such as ***customer_id*** would be used instead of ***customer_name*** to uniquely identify customers, but we omit it to keep our examples small, and instead assume customer names are unique.

Keys (Cont.)

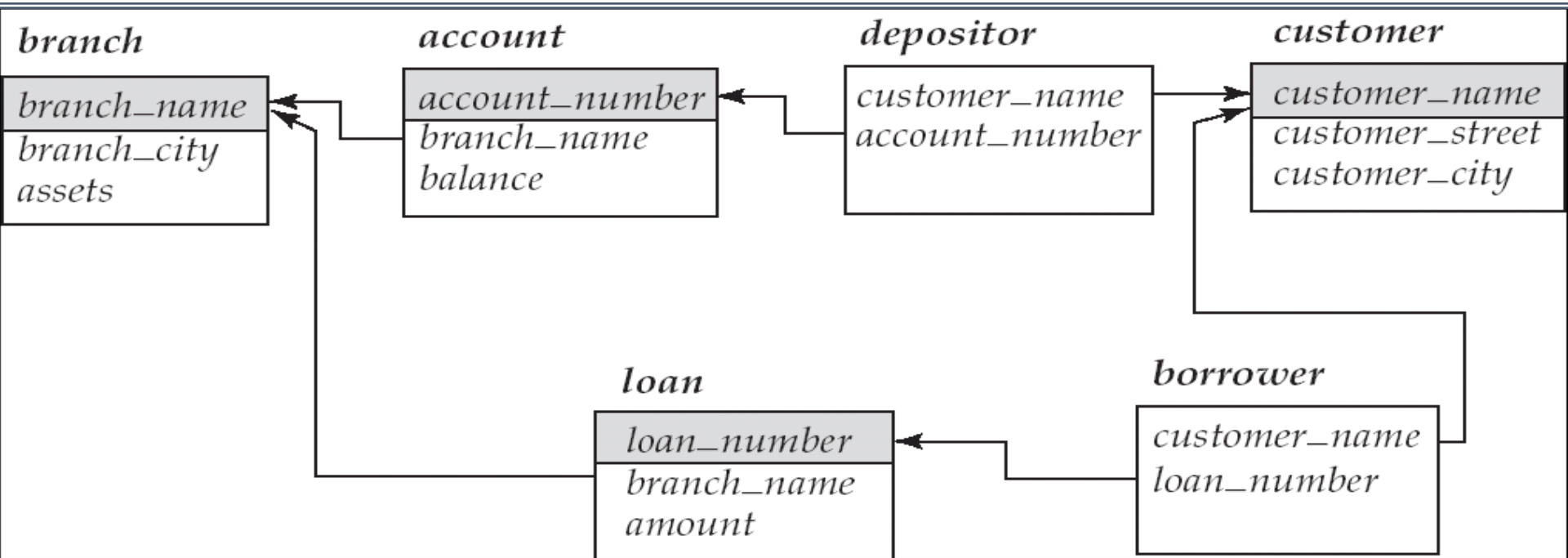
- *K* is a **candidate key** if *K* is minimal

  Example: {*customer_name*} is a candidate key for *Customer*, since it is a superkey and no subset of it is a superkey.

- **Primary key:** a candidate key chosen as the principal means of identifying tuples within a relation
  - Should choose an attribute whose value never, or very rarely, changes.
  - E.g. email address is unique, but may change

Foreign Keys

- A relation schema may have an attribute that corresponds to the primary key of another relation. The attribute is called a **foreign key**.
  - E.g. *customer_name* and *account_number* attributes of *depositor* are foreign keys to *customer* and *account* respectively.
  - Only values occurring in the primary key attribute of the **referenced relation** may occur in the foreign key attribute of the **referencing relation.**
- **Schema diagram**



Query Languages

- **Language** in which user requests information from the database.
- Categories of languages
  - Procedural
  - Non-procedural, or declarative
- “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: σ
  - project: $\prod$
  - 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.

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:

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

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:

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

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:

$$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *branch_name* attribute of *account*

$$\Pi_{account\_number,\ balance}\ (account)$$

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** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- Example: to find all customers with either an account or a loan

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

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	10	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

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

  returns the result of expression *E* under the name *X*, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

Banking Example

branch (branch_name, branch_city, assets)

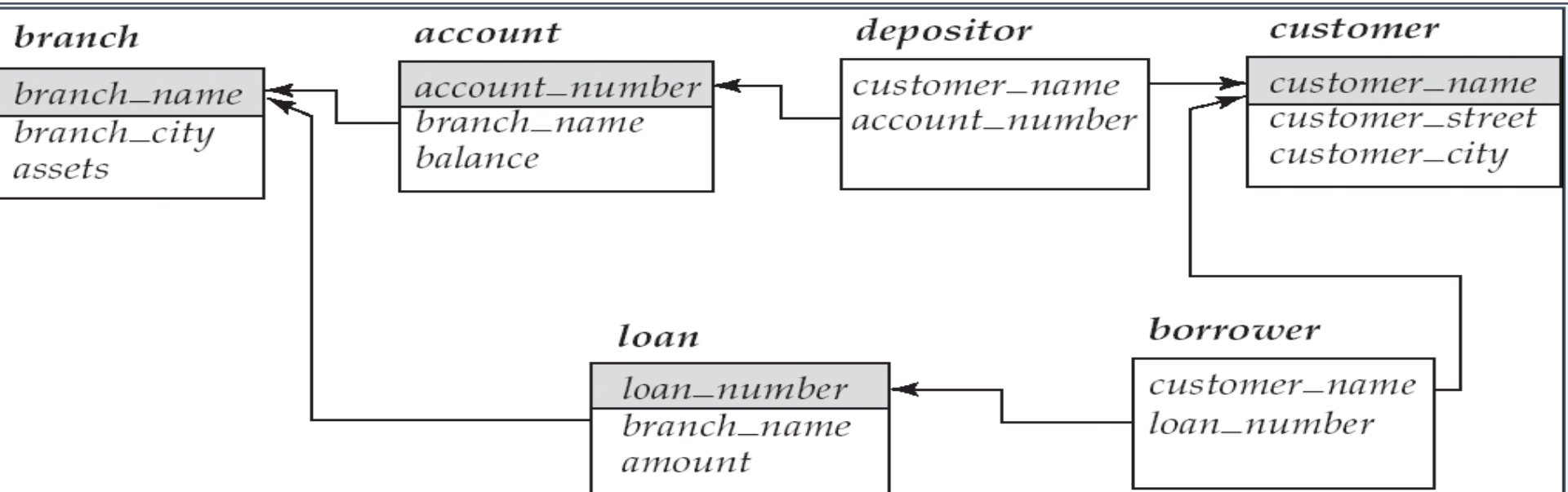
customer (customer_name, customer_street, customer_city)

account (account_number, branch_name, balance)

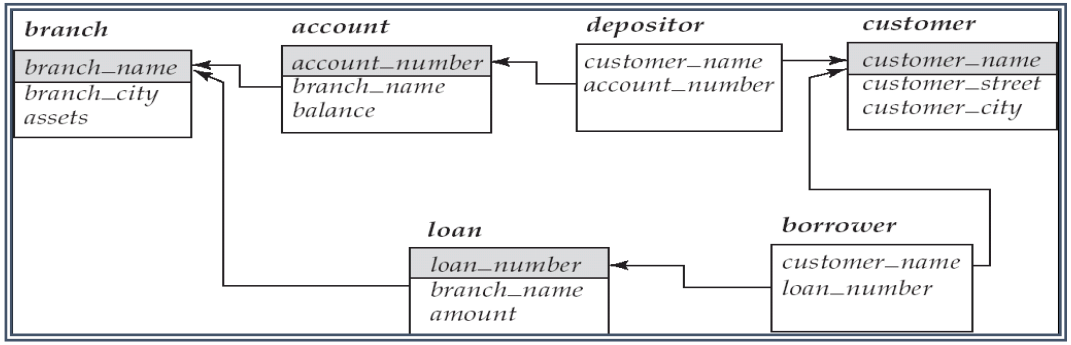
loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)



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

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

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

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

Example Queries

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

$$\Pi_{customer\_name}\,(\sigma_{branch\_name=\text{“Perryridge”}}$$

$$(\sigma_{borrower.loan\_number\,=\,loan.loan\_number}(borrower\ x\ loan)))$$

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.

$$\Pi_{customer\_name}\,(\sigma_{branch\_name\,=\,\text{“Perryridge”}}$$

$$(\sigma_{borrower.loan\_number\,=\,loan.loan\_number}(\text{borrower x loan}))\ -$$

$$\Pi_{customer\_name}(\text{depositor})$$

Example Queries

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

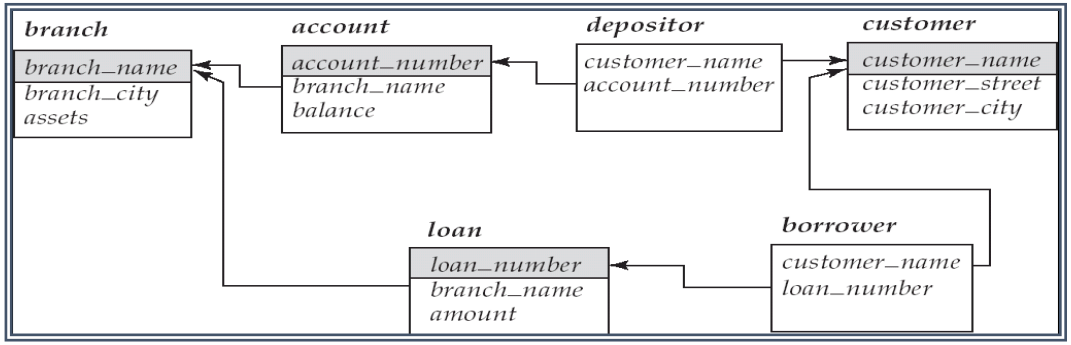
Query 1

$$\Pi_{\text{customer\_name}} (\sigma_{\text{branch\_name} = \text{“Perryridge”}} (\sigma_{\text{borrower.loan\_number} = \text{loan.loan\_number}} (\text{borrower x loan})))$$

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
  - Strategy:
    - Find those balances that are *not* the largest
      - Rename *account* relation as *d* so that we can compare each account balance with all others
    - Use set difference to find those account balances that were *not* found in the earlier step.
  - 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 x 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.

- 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 ⋈ 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

- Notation: $r \div s$
- 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 ÷ s is a relation on schema

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

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

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

Division Operation – Example

Relations *r, s*:

A	*B*
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ϵ	6
ϵ	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 $S \subseteq R$

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

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

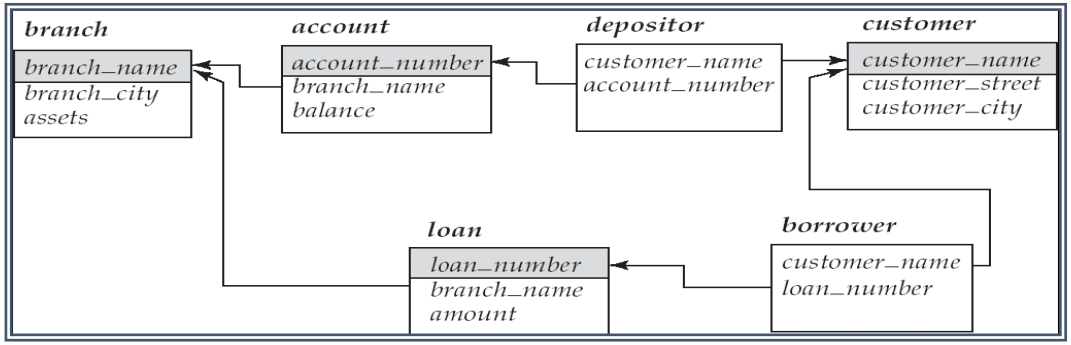
Assignment Operation

- 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

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

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
  - May use variable in subsequent expressions.

Bank Example Que



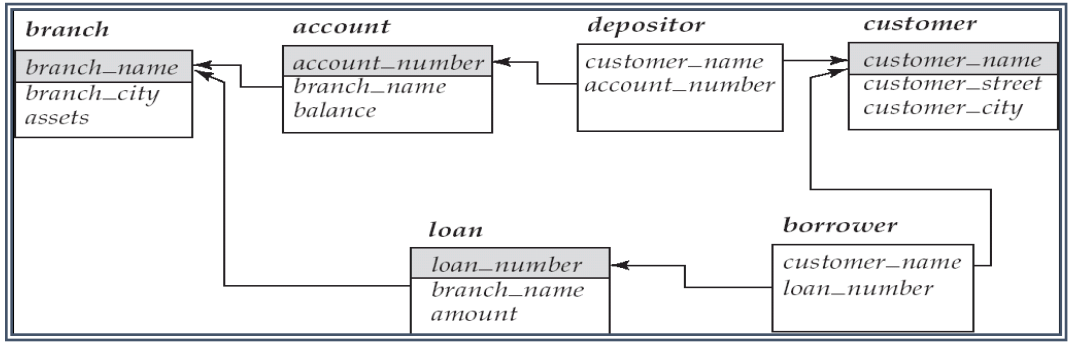
Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer\_name}(borrower) \cap \Pi_{customer\_name}(depositor)$$

- Find the name of all customers who have a loan at the bank and the loan amount

$$\Pi_{customer\_name,\ loan\_number,\ amount}(borrower \bowtie loan)$$

Bank Example Queries



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

  Query 1

$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{"Downtown"}}(depositor \bowtie account)) \cap$$

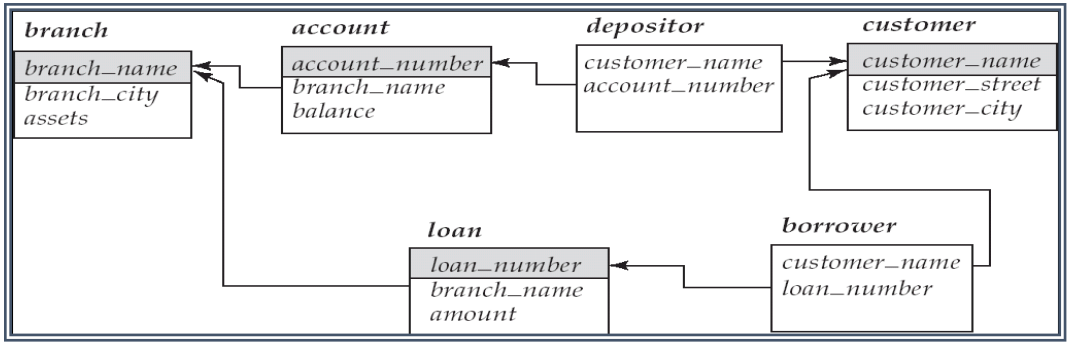
$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{"Uptown"}}(depositor \bowtie account))$$

  Query 2

$$\Pi_{customer\_name,\ branch\_name}(depositor \bowtie account) \div \rho_{temp(branch\_name)}(\{(\text{"Downtown"}), (\text{"Uptown"})\})$$

  Note that Query 2 uses a constant relation.

Bank Example Queries



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

$$\prod_{customer\_name,\ branch\_name} (depositor \bowtie account) \div \prod_{branch\_name} (\sigma_{branch\_city = \text{“Brooklyn”}} (branch))$$

Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions
- Outer Join

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, …, F_n are are arithmetic expressions involving constants and attributes in the schema of *E*.
- Given relation *credit_info(customer_name, limit, credit_balance),* find how much more each person can spend:

$$\Pi_{customer\_name,\ limit - credit\_balance}\ (credit\_info)$$

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

$$_{G_1,G_2,\ldots,G_n}\vartheta_{F_1(A_1),F_2(A_2,\ldots,F_n(A_n)}(E)$$

  E is any relational-algebra expression
  - G_1, G_2 …, G_n is a 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 Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{G}_{\text{sum(c)}}(\text{r})$

sum(*c*)
27

Aggregate Operation – Example

- 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

$$branch\_name\ \mathcal{G}_{\mathbf{sum}(balance)}\ (account)$$

branch_name	**sum**(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Functions (Cont.)

- 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

$$branch\_name\ \mathcal{G}\ \textbf{sum}(balance)\ \textbf{as}\ sum\_balance\ (account)$$

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 comparisons involving *null* are (roughly speaking) **false** by definition.
    - We shall study precise meaning of comparisons with nulls later

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

- 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 $\overset{\frown}{\bowtie}$ *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 (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)

Null Values

- Comparisons with null values return the special truth value: *unknown*
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- 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*
  - In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*

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.

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

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

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

Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{branch\_city = \text{“Needham”}}(account \bowtie branch)$$

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

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

$$account \leftarrow account - r_2$$

$$depositor \leftarrow depositor - r_3$$

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.

$$account \leftarrow account \cup \{(\text{“A-973”}, \text{“Perryridge”}, 1200)\}$$

$$depositor \leftarrow depositor \cup \{(\text{“Smith”}, \text{“A-973”})\}$$

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.

$$r_1 \leftarrow (\sigma_{branch\_name = \text{“Perryridge”}}(borrower \bowtie loan))$$

$$account \leftarrow account \cup \prod_{loan\_number,\ branch\_name,\ 200}(r_1)$$

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

Updating

- A mechanism to change a value in a tuple without charging *all* values in the tuple
- Use the generalized projection operator to do this task $r \leftarrow \prod_{F_1, F_2, \ldots, F_l}(r)$

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

Update Examples

- Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \prod_{account\_number, branch\_name, balance * 1.05} (account)$$

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

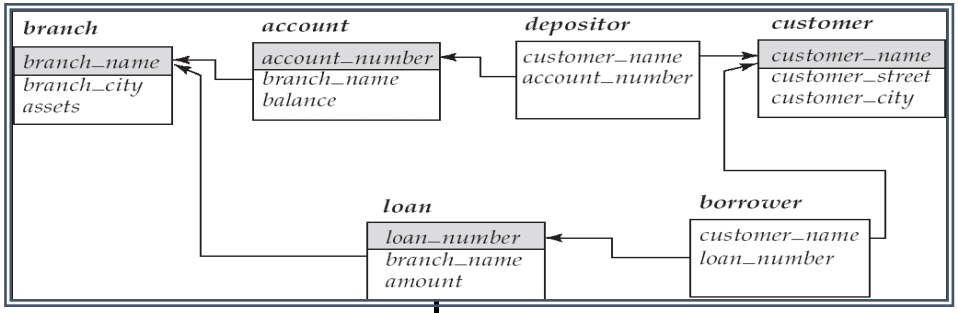
$$account \leftarrow \prod_{account\_number, branch\_name, balance * 1.06} (\sigma_{BAL > 10000} (account)) \cup \prod_{account\_number, branch\_name, balance * 1.05} (\sigma_{BAL \leq 10000} (account))$$

Reading

A relational model of data for large shared data banks

Summary

- Data-→**Data model-**→relational model--→other models
- One of the earliest pioneering works in modelling information systems has been done **by Young and Kent (1958)**, who argued for "**a precise and abstract way of specifying the informational and time charateristics of a data processing problem“**.
- A next step in IS modelling was taken by CODASYL, an IT industry consortium formed in 1959, who essentially aimed at the same thing as Young and Kent: **the development of "a proper structure for machine independent problem definition language, at the system level of data processing“**
- Two famous database models, the network data model and the hierarchical data model, were proposed during this period of time". Towards the end of the 1960s **Edgar F. Codd** worked out his theories of data arrangement, and proposed the **relational model** for database management based on **first-order predicate logic.**



Explicit Relation

Semi-structured Data

Structured Data

Unstructured Data

Entailed Relation

Pattern

Query

IR/IE

Mining\Analysis\Monitoring\Reasoning

Entailed/Implicit Relation

(no pattern)

Persistent Data

Trancient Data

Centralized

Decentralized

Home work

- Provide a relational data model on course, student and teacher in a university

End of Chapter 2