Introduction

Dr. Odelu Vanga

Department of Computer Science and Engineering
Indian Institute of Information Technology
Sri City - 517646, Chittoor, AP, India

odelu.vanga@iiits.in
https://sites.google.com/site/odeluvanga/home

Monsoon-2021

Source of Content

Course Title	:	**Database Management Systems**
Textbook	:	**Database System Concepts**, Silberschatz, H. Korth and S. Sudarshan, McGraw-Hill Education, 6^{th} edition, 2010.

Source of Content

Course Title	:	**Database Management Systems**
Textbook	:	**Database System Concepts**, Silberschatz, H. Korth and S. Sudarshan, McGraw-Hill Education, 6^{th} edition, 2010.

Reference books:

R1. **Fundamentals of Database Systems**, R. Elmasri and S. B. Navathe, Pearson Education, 6^{th} edition, 2010.

R2. **An Introduction to Database Systems**,C. J. Date, A. Kannan, S. Swamynathan, 8^{th} edition, Pearson Education, 2006.

R3. **Principles of Database Systems**, J. D. Ullman, 2^{nd} edition, Galgotia Publications, 1999.

R4. **Database Management Systems**, R. Ramakrishnan and J. Gehrke, 3^{rd} edition, McGraw Hill Education, 2014.

University Data Files

STUDENT

Name	S_number	Class	Major
Smith	17	1	CSE
Brown	8	2	CSE

COURSE

C_name	C_number	Credits	Dept
ICS	CS1310	4	CSE
DS	CS3320	4	CSE
DM	MATH2410	3	MATH
DB	CS3380	3	CSE

SECTION

Section_ID	C_number	Sem	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

S_number	Section_ID	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

C_number	Prerequisite_no.
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

- Simple to operate
- Better local control

Drawbacks of Traditional File System

- Data Redundancy and Inconsistency
- Difficulty in Accessing Data
- Data Isolation
- Integrity Problems
- Atomicity Problems
- Concurrent-Access Anomalies
- Security Problems

Course Outline

- Module-01: Database System Concepts
  - Data Models
  - Schemas
  - Instances

Course Outline

- Module-01: Database System Concepts
  - Data Models
  - Schemas
  - Instances
- Module-02: Database System Architecture
  - Three-Schema Architecture
  - Data Independence
  - Database Languages and Interfaces
  - Centralized and Client/Server Architectures

Course Outline

- Module-01: Database System Concepts
  - Data Models
  - Schemas
  - Instances
- Module-02: Database System Architecture
  - Three-Schema Architecture
  - Data Independence
  - Database Languages and Interfaces
  - Centralized and Client/Server Architectures
- Module-03-Data Modeling
  - Entity-Relationship Diagram
  - Relational Model
  - Integrity Constraints
  - Data Manipulation Operations
  - Relational Algebra
  - Relational Calculus

Course Outline

- Module-04: SQL (Structured Query Language)
  - Data Definition and Data Types
  - Constraints, Queries, Insert, Delete, and Update Statements
  - Views, Stored Procedures and Functions
  - Database Triggers, SQL Injection.

Course Outline

- Module-04: SQL (Structured Query Language)
  - Data Definition and Data Types
  - Constraints, Queries, Insert, Delete, and Update Statements
  - Views, Stored Procedures and Functions
  - Database Triggers, SQL Injection.
- Module-05: Normalization for Relational Databases
  - Functional Dependencies
  - Normalization
  - Query Processing
  - Query Optimization Algorithms

Course Outline

- Module-04: SQL (Structured Query Language)
  - Data Definition and Data Types
  - Constraints, Queries, Insert, Delete, and Update Statements
  - Views, Stored Procedures and Functions
  - Database Triggers, SQL Injection.
- Module-05: Normalization for Relational Databases
  - Functional Dependencies
  - Normalization
  - Query Processing
  - Query Optimization Algorithms
- Module-06: Transaction Processing
  - Concurrency Control Techniques
  - Database Recovery Techniques
  - Object and Object-Relational Databases
  - Database Security and Authorization

Assessment Components (tentative)

Component	Weightage(%)	
MidSem	20%	
EndSem	40%	
Scheduled Quizzes (2)	10%	
Lab Assignments (10)	5%	
Lab Tests (2)	10%	
Lab End Exam	5%	

Assessment Components (tentative)

Component	Weightage(%)	
MidSem	20%	
EndSem	40%	
Scheduled Quizzes (2)	10%	
Lab Assignments (10)	5%	
Lab Tests (2)	10%	
Lab End Exam	5%	

Remark

Assessment components may change according to the institute norms.

Thank You

Basic Definitions:

Database: A collection related data

Data: known facts that can be recorded and having an implicit meaning

Mini-world: some part of real-world about which data is stored in a database.

University
- student
- grades
- transcripts

Database Management System (DBMS)

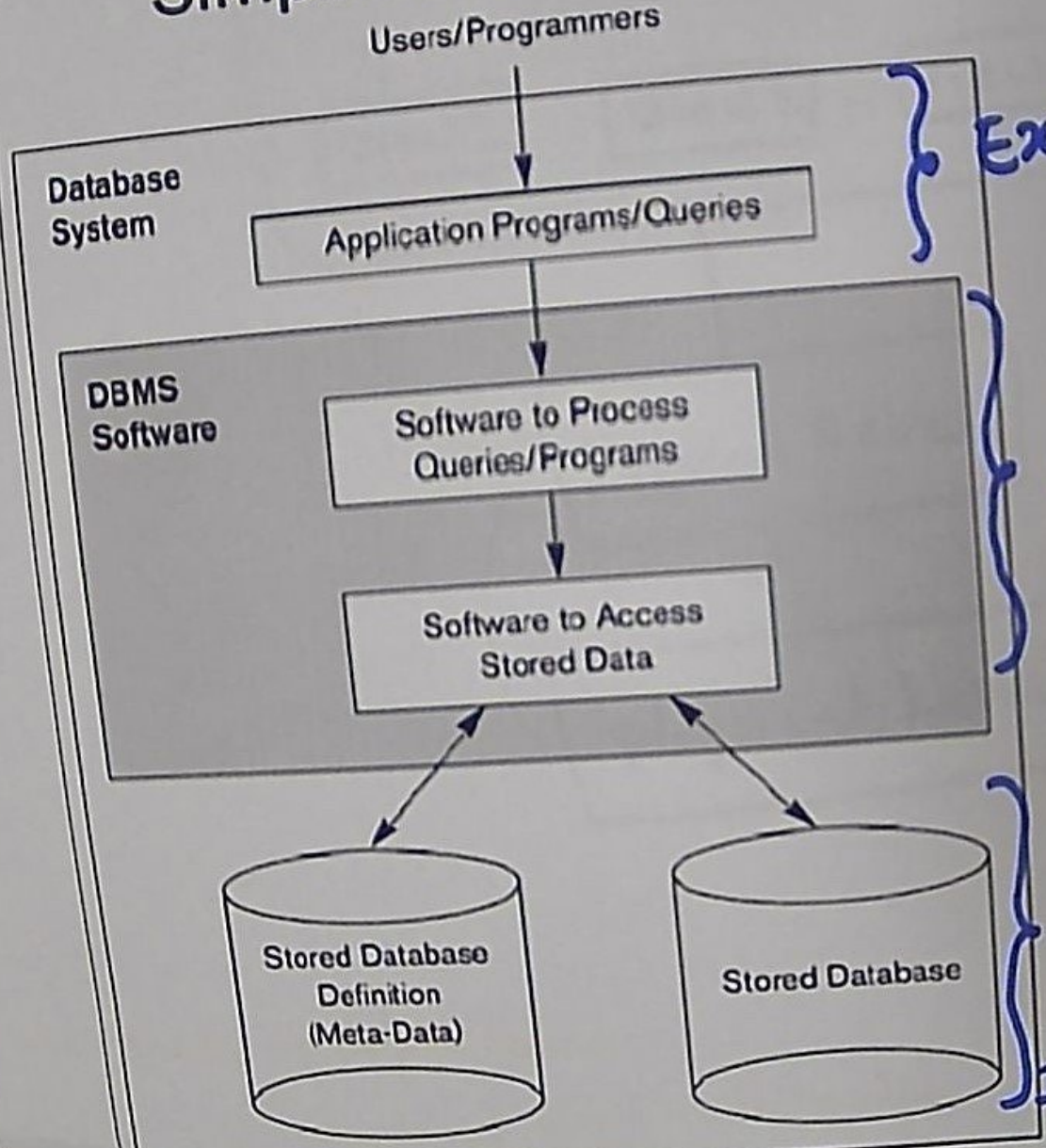
→ A software package / system to facilitate the creation and maintenance of a computerized database.

Database system:

→ The DBMS software together with the data itself.

→ applications.

Simplified database system environment



External level

Conceptual level

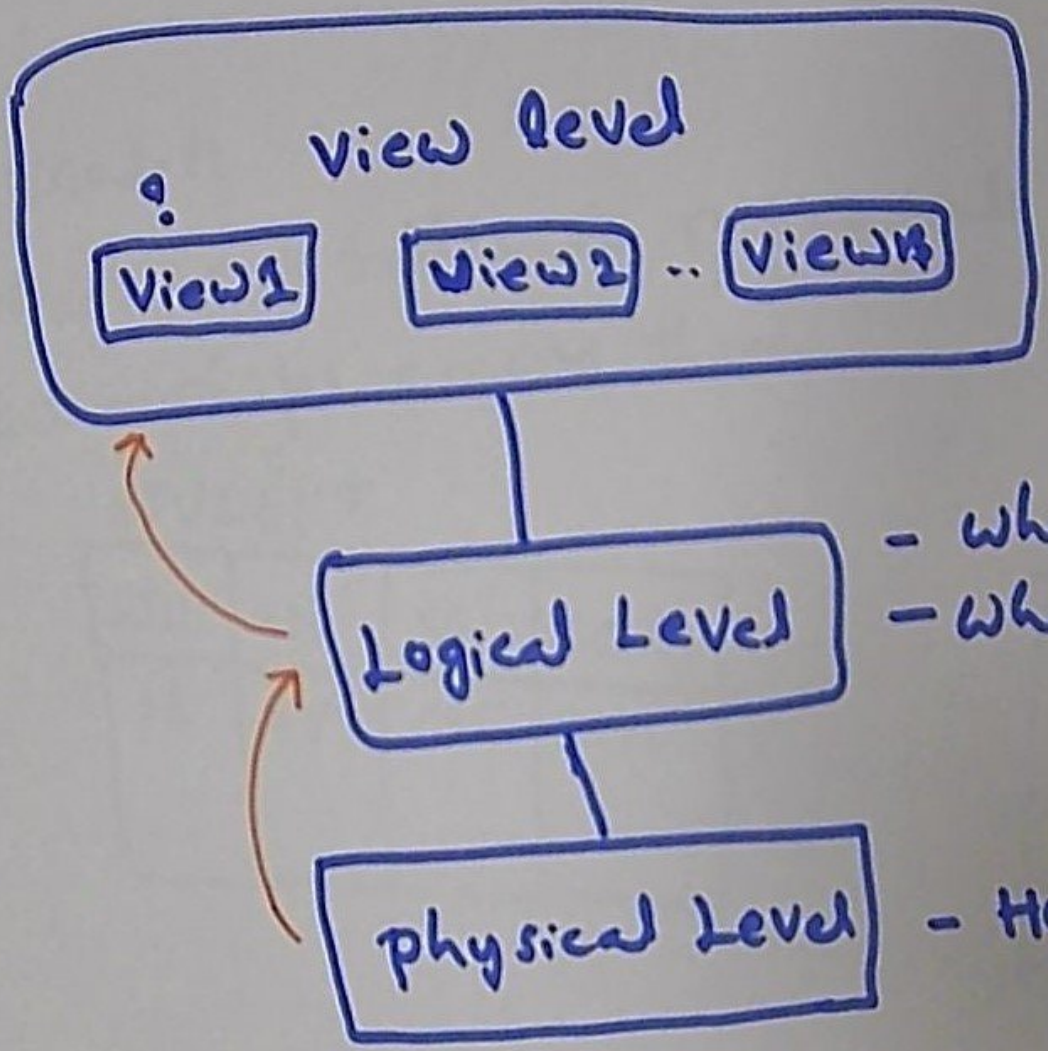
Student

The database definition or descriptive information is stored by the DBMS in the form of a database catalog or dictionary;

it is called **meta-data**

Internal

Data Abstraction



→ Data independence.

* physical data independence
* Logical data independence.

Data models:

* Relational models
  → A collection of tables to represent both data and relationships among them.

STUDENT

SID	SN	SG	Scity
01	xy	CSE	HYD

Relation

INSTRUCTOR

IID	IN	Isalary

Relation

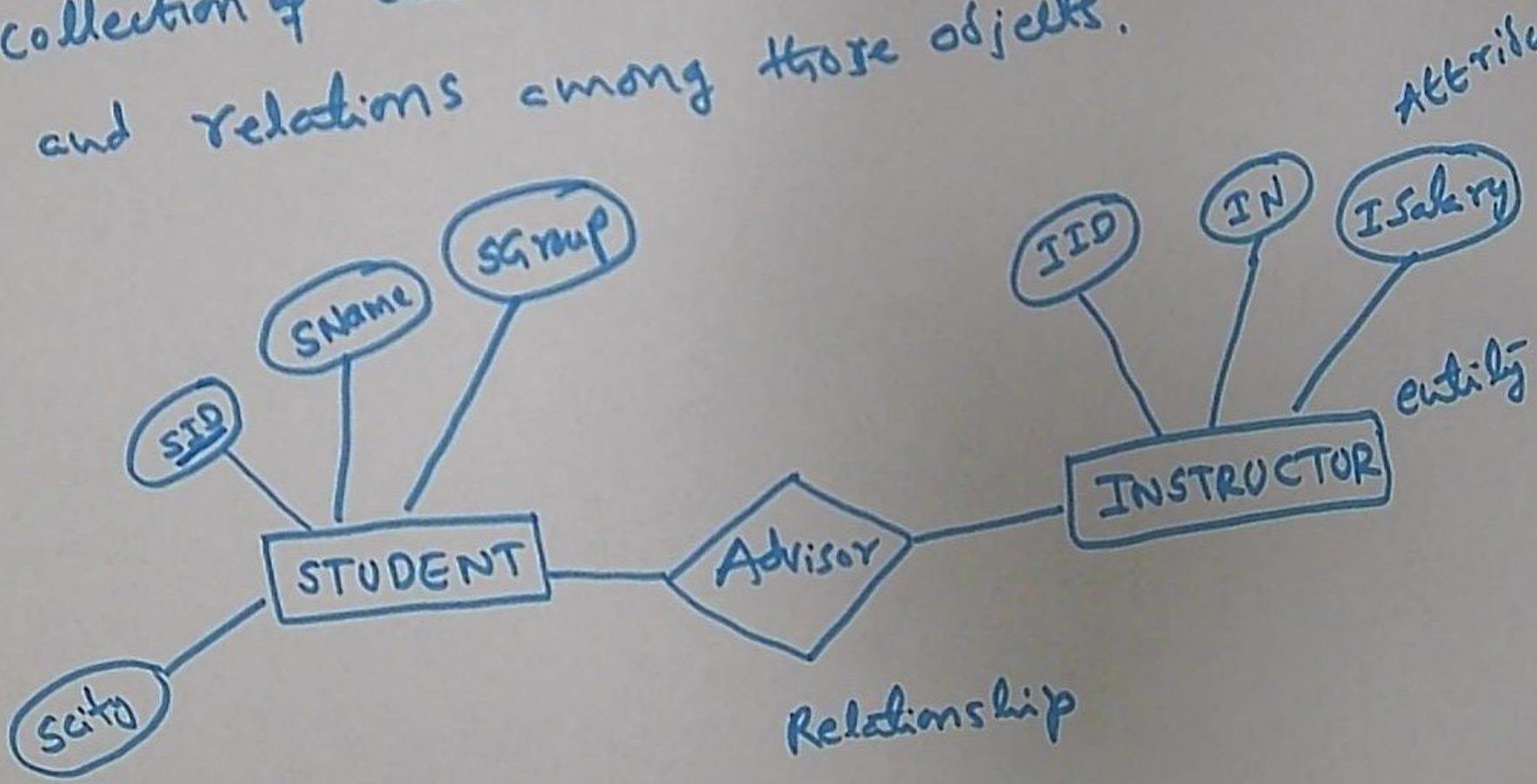
Advisor

PID	SID	IID

project advisor.

Entity - Relationship model (ER-model).

→ collection of basic object, called entities, and relations among those objects.



Schema and Instances :

Schema: A description of database

includes, database structure
datatypes
constrains

STUDENT

ID	SName	Phone
S1	Syam	900101
S2	XYZ	900201
S1	Ram	900201

STUDENT (ID, SName, Ph)

Schema construct: A component of the schema

(or) an object within the schema.

Ex: STUDENT, COURSE

Schema of University Database.

classroom (building, room_no, capacity)

dept (dept-name, building, budget)

course (c_id, title, dept_name, credits)

Instructor (ID, name, dept_name, salary)

section (c_id, sec_id, sem, year, building, room_no, time_slot_id)

teaches (ID, c_id, sec_id, sem, year)

student (ID, name, dept_name, tot_cred)

takes (ID, c_id, sec_id, sem, year, grade)

advisor (s_ID, I_ID)

time_slot (time_slot_id, day, start_time, end_time)

prereq (course_id, prereq_id)

Schema of Univ Database

classroom(<u>building</u>, <u>room_number</u>, capacity)
department(<u>dept_name</u>, building, budget)
course(<u>course_id</u>, title, dept_name, credits)
instructor(<u>ID</u>, name, dept_name, salary)
section(<u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, building, room_number, time_slot_id)
teaches(<u>ID</u>, <u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>)
student(<u>ID</u>, name, dept_name, tot_cred)
takes(<u>ID</u>, <u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, grade)
advisor(<u>s_ID</u>, i_ID)
time_slot(<u>time_slot_id</u>, <u>day</u>, <u>start_time</u>, end_time)
prereq(<u>course_id</u>, <u>prereq_id</u>)

classroom – relation name schema content.

classroom (building, room_number, capacity) – relation schema

classroom

building	room_num	capacity
A1	101	60
A3	301	50
A1	102	60

Attributes

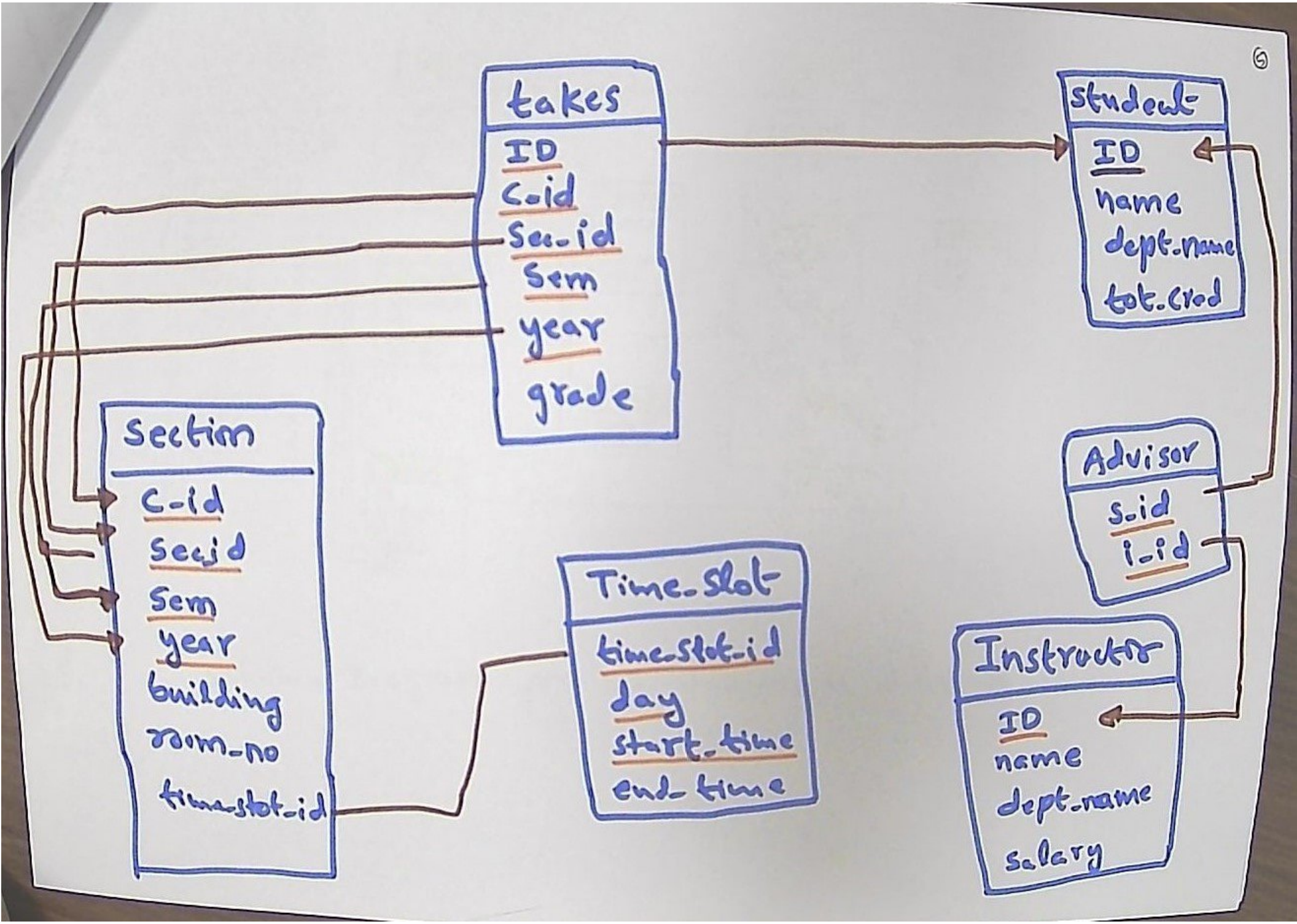
tuples / records

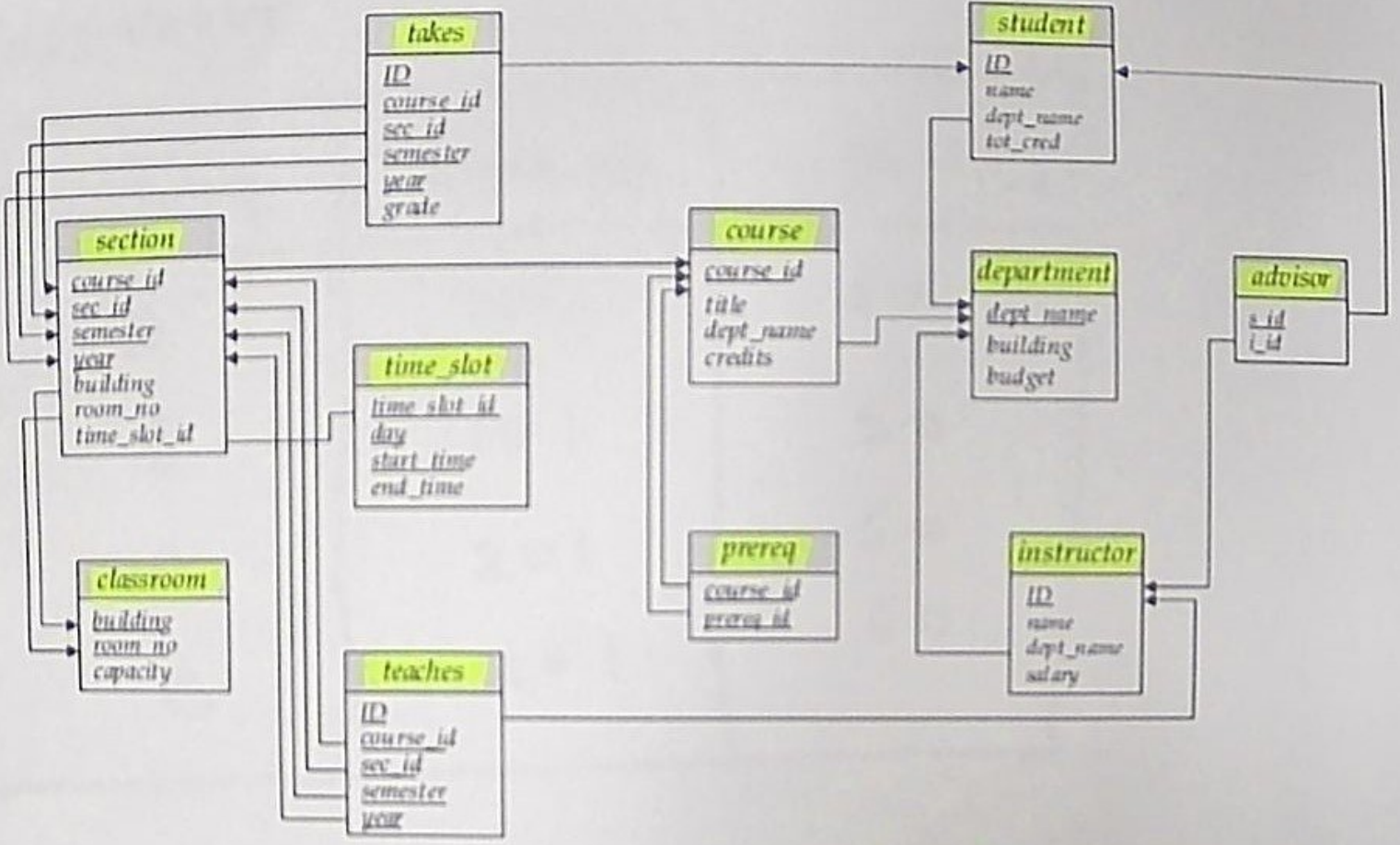
- → Insert record
- → update record
- → delete record

Manipulation operations

Schema Diagram:

→ An illustrative display of a database schema.


takes
ID
C_id
Sec_id
Sem
year
grade
Student
ID
name
dept_name
tot_cred
Section
C_id
Sec_id
Sem
year
building
room_no
time_slot_id
Time_Slot
time_slot_id
day
start_time
end_time
Advisor
S_id
I_id
Instructor
ID
name
dept_name
salary



Schema Diagram for the university database.

classroom

building	room_no	capacity
A	101	60
B	101	50
A	201	60
C	201	60

Database state: The actual data stored in a database at a particular moment of time.

Database instance:

Basic structure

* $A_1, A_2, \cdots A_n$ – n attributes
* Non-empty set D_i, $(1 \le i \le n)$, domain of attribute A_i
* Domain $D = D_1 \cup D_2 \cup \cdots \cup D_n$
* Finite set of attributes called a relation schema

$$R = \{A_1, A_2, \cdots A_n\}$$

* relation, r, is a subset of

$$D_1 \times D_2 \times \cdots \times D_n$$

Table / Relation

	A_1	A_2	...	A_n
t_1	a_{11}	a_{12}	...	a_{1n}
t_2	a_{21}	a_{22}	...	a_{2n}
	⋮			
t_k	a_{k1}	a_{k2}	...	a_{kn}

$t_1 = (a_{11}, a_{12}, \cdots, a_{1n})$

$t_2 = (a_{21}, a_{22}, \cdots a_{2n})$

$t_k = (a_{k1}, a_{k2}, \cdots a_{kn})$

$r(R) = r = \{t_1, t_2, \cdots t_k\}$

Ex:

classroom

	building	room_num	capacity
t_1	B1	101	60
t_2	B2	301	50
t_3	B1	102	60

classroom = (building, room_num, capacity)

$r = \{t_1, t_2, t_3\}$

Relation is unordered.

building domain = {B1, B2, B3, B4, B5}

room_num = {101, 102, 103, 201, 202, 203, ⋮ 501, 502, 503}

t_1 = (B1, 101, 60)

t_2 = (B2, 301, 50)

t_3 = (B1, 102, 60)

$R = \{A_1, A_2, \ldots A_n\}$

$r(R)$ – relation on R

$K = \{A_u, A_v, \ldots A_w\}$. $1 \le u \le v \le w \le n$

is a subset of R

Uniqueness property: for any $t_1 \neq t_2$ of r,

$\exists$ an attribute A_j of K such that

$$t_1(A_j) \neq t_2(A_j)$$

Minimal property: There is no proper subset $K' \subset K$

satisfying uniqueness property.

Def: Super key

A set of attributes K which can uniquely identify a tuple, is known as super key.

* Uniqueness property.

Def: Candidate key: A super key satisfy minimality property.

Def: primary key: → There may be more than one candidate keys for a relation. one of the candidate key is selected as primary.

Def prime attributes: The attributes which are part of candidate keys

STUDENT

	SID	SName	phn
t_1	S01	RAM	9490201
t_2	S02	RAM	9490301
t_3	S03	SUJIT	960400
t_4	S04	SURESH	NULL

Constraints: 1) UNIQUE

2) NOT NULL

Define primary key

Super : trivial SID, SName, phn

SID, SName

SID, phn

SID.

candidate : SID

primary : SID.

Query:

```
CREATE TABLE STUDENT (
    SID char(3) NOT NULL
    SName Varchar(10) NOT NULL
    phn int,
    PRIMARY KEY (SID)
);
```

Course

	SID	CID	CName
t_1	S01	C1	DBMS
t_2	S02	C2	CN
t_3	S01	C2	CN

Find Candidate Key.

Trivial — SID, CID, CName — Super key.

SID, CID

$\rightarrow t_1 \neq t_2 \;\exists\; SID \ni t_1(SID) \neq t_2(SID)$

$\rightarrow t_1 \neq t_3 \;\exists\; CID \ni t_1(CID) \neq t_2(CID)$

$\rightarrow t_2 \neq t_3 \;\exists\; SID \ni t_2(SID) \neq t_3(SID)$

IS there any other candidate key? $\exists$ no proper subset K' of {SID, CID} which satisfy uniqueness property.

Assume, for each course if you assign a unique course number.

SID, CName — candidate key.

Query:

```
CREATE TABLE COURSE (

        SID   char(3) NOT NULL

        CID   char(2) NOT NULL

        CName   Varchar(10),

        PRIMARY KEY (SID, CID)

);
```

```
CONSTRAINT

    PK_COURSE

    PRIMARY KEY

    (SID, CID)
```

optional.

subquery :

→ SET MEMBERSHIP

IN
NOT IN

Dept (dno, dname, building)

Student (id, name, phn, dno, fee)

$t_i(dno) \in \{D01, D02, D03\}$

Q: List dept with no students.

Dept

	dno	dname	building
t_1	D01	CSE	B01
t_2	D02	ECE	B02
t_3	D03	MEC	B02
t_4	D04	CIVIL	B03

Student

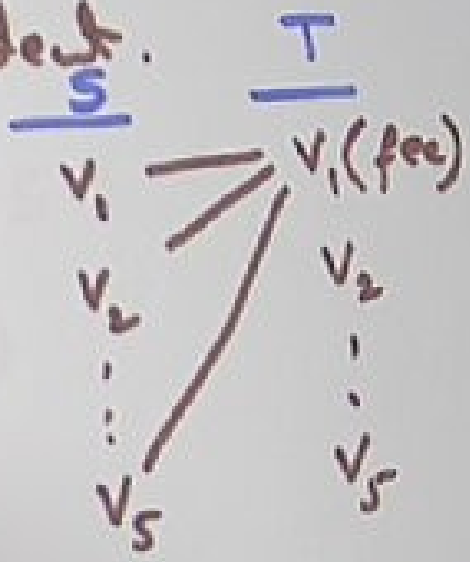
	pid	name	phn	dno	fee
v_1	S01	RAM	9441	D01	10
v_2	S02	SAM	9241	D01	5
v_3	S03	SYAM	9314	D02	20
v_4	S04	JANU	9315	D03	31
v_5	S05	VANI	9214	D03	15

```
Select distinct dname
from dept
where dno NOT IN ( select distinct dno
                   from student);
```

for every tuple $t_i(dno) \in \{D01, D02, D03\}$

SET COMPARISON

Q: List all students who payed fee greater than atleast one student.




```
select distnct T.name
from student as T, studat as S
where T.fee > S.fee
```

$S = \{10, 5, 20, 31, 15\}$

$x = 10$, $x > y$ *

Test for empty relations :

EXISTS
NOT EXISTS
} construct

" Relation A contains Relation B " $B \subseteq A$

→ NOT EXISTS (B except A)

Remark:

$x = \text{some } \{x_1, x_2 \ldots\}$

$x > \text{some } \{x_1, x_2 \ldots\}$

$x < \text{some } \{x_1, x_2, \ldots\}$

identical

$= \text{some}$? yes IN

$<> \text{some}$? No NOT IN

```
select name
from student
where fee > some ( select fee
                   from student) ;
```

: Find all students who have taken all courses offered in CSE.

```
select distinct s.id, s.name
from student as s
where not exists (
        (select c.id
         from course
         where dname = 'CSE')
        except
        ( select T.course-id
          from Takes as T
          where s.id = T.id)
);
```

c.id course-id

Entity-Relationship Model

Dr. Odelu Vanga

Indian Institute of Information Technology Sri City

http://www.iiits.ac.in/people/regular-faculty/dr-odelu-vanga/

Outline

- Entity Sets
- Relationship Sets
- Binary Relationship
- N-ary Relationship
- Examples:
    1. University
    2. Supply

Entity Sets

- ***Database***
  - a collection of **entities**
  - **relationship** among entities
- ***Entity***
  - An object that exists and distinguishable from other objects.

    Example: specific person, student, instructor, plant
- ***Attributes***

  Example: people have *names* and *addresses*
- ***Entity set***
  - A set of entities of the same type that share the same properties.

    Example: set of all persons, companies, trees, holidays

Entity Sets *customer* and *loan*

Customer ID	Customer Name	Customer Street	Customer City
321-12-3123	Jones	Main	Harrison
019-28-3746	Smith	North	Rye
677-89-9011	Hayes	Main	Harrison
555-55-5555	Jackson	Dupont	Woodside
244-66-8800	Curry	North	Rye
963-96-3963	Williams	Nassau	Princeton
335-57-7991	Adams	Spring	Pittsfield

customer

Loan Number	Amount
L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-19	500
L-11	900
L-16	1300

loan

Attributes

- An entity is represented by a set of attributes.

  Example: *customer = (customer-id,*

  customer-name,

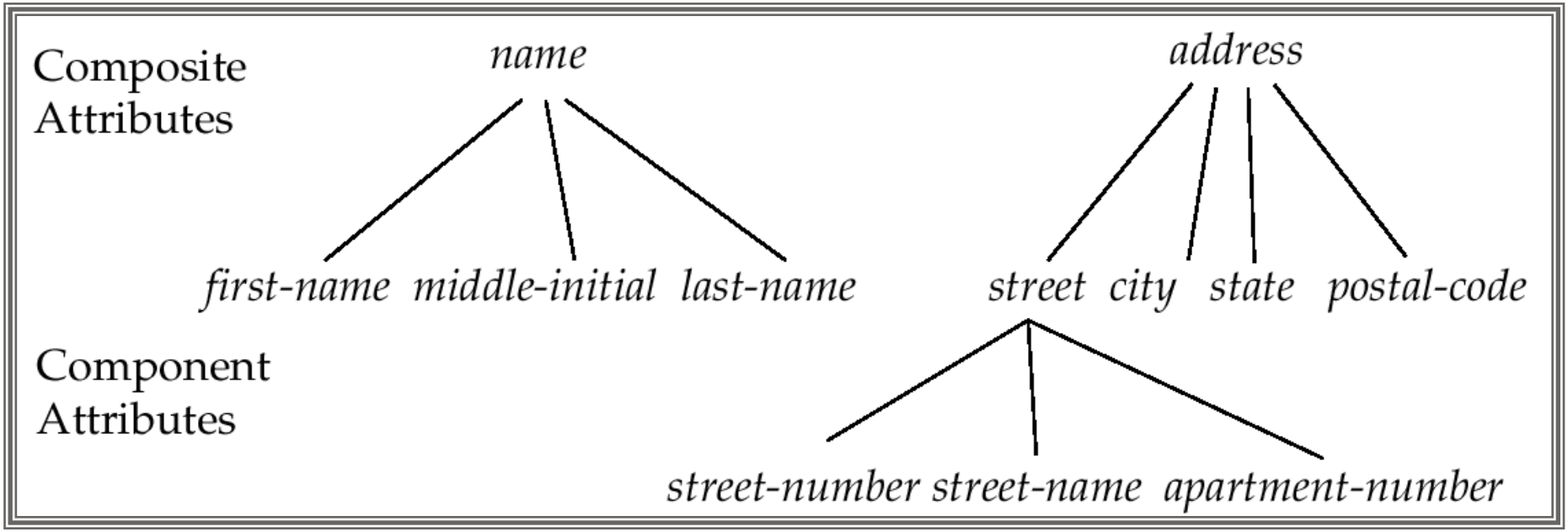
  customer-street,

  customer-city)

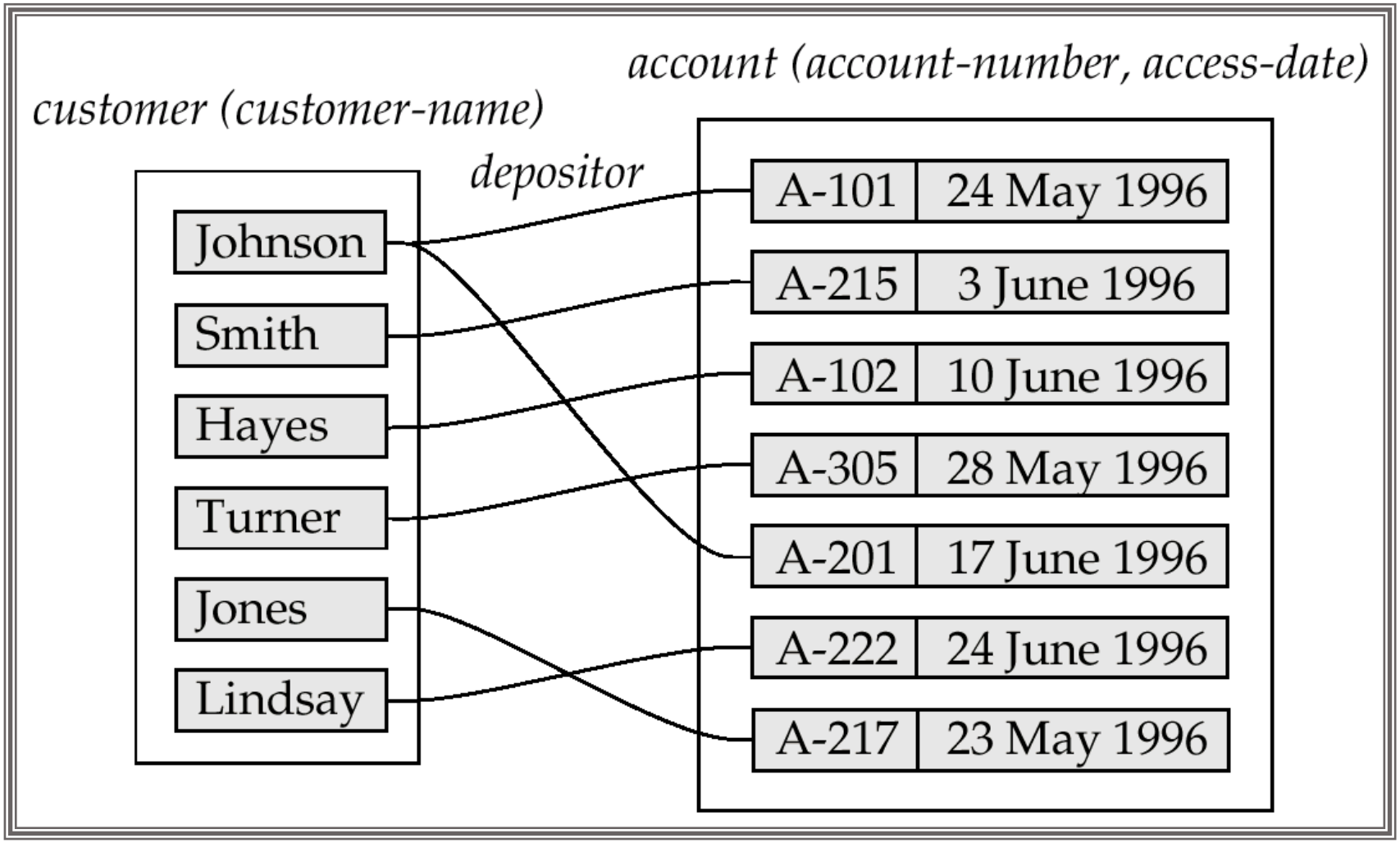
  loan = (loan-number, amount)

- ***Domain*** : Set of permitted values for each attribute
- Types of Attribute:
  - *Simple* and *composite* attributes.
  - *Single-valued* and *multi-valued* attributes
    - E.g. multivalued attribute: *phone-numbers*
  - *Derived* attributes
    - Can be computed from other attributes
    - E.g. ***age***, from given date of birth

Composite Attributes

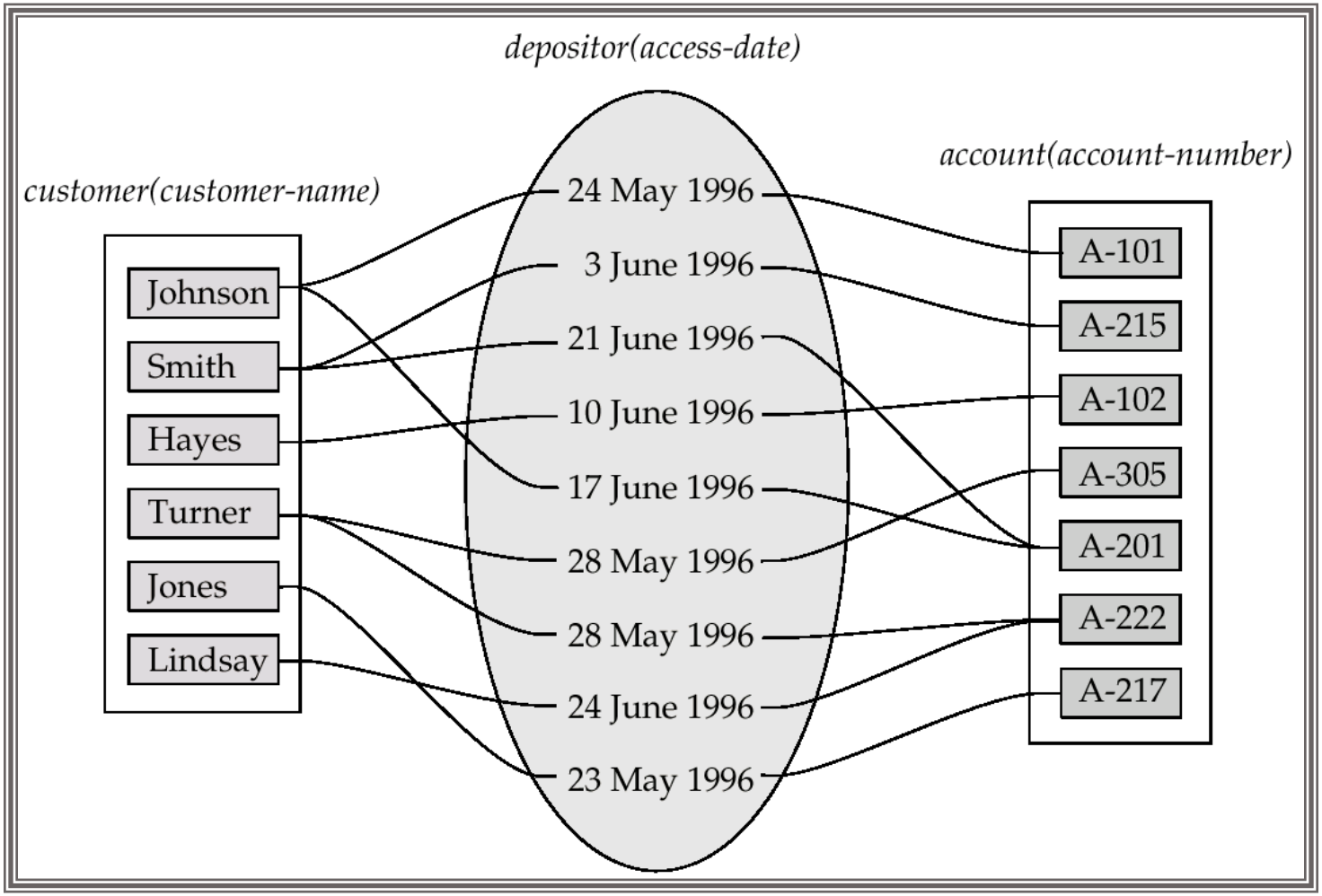


Relationship Sets



Relationship Sets (Cont.)

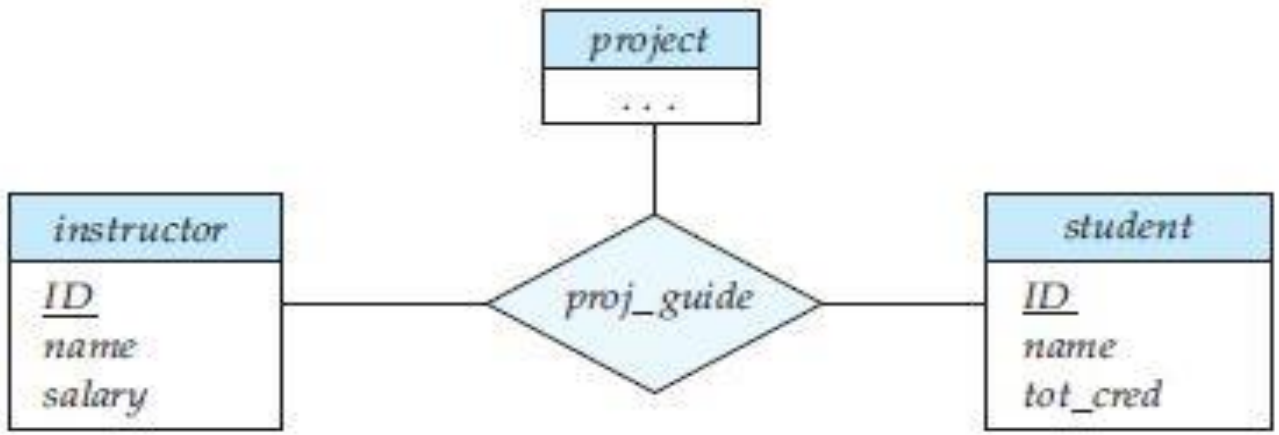
- An *attribute* can also be property of a relationship set.
- For instance, the *depositor* relationship set between entity sets *customer* and *account* may have the attribute *access-date*



Degree of a Relationship Set

- Refers to number of entity sets that participate in a relationship set.
- Relationship sets that involve two entity sets are *binary* (or degree two).
  - Generally, most relationship sets in a database system are binary.
- Relationship sets may involve more than two entity sets.

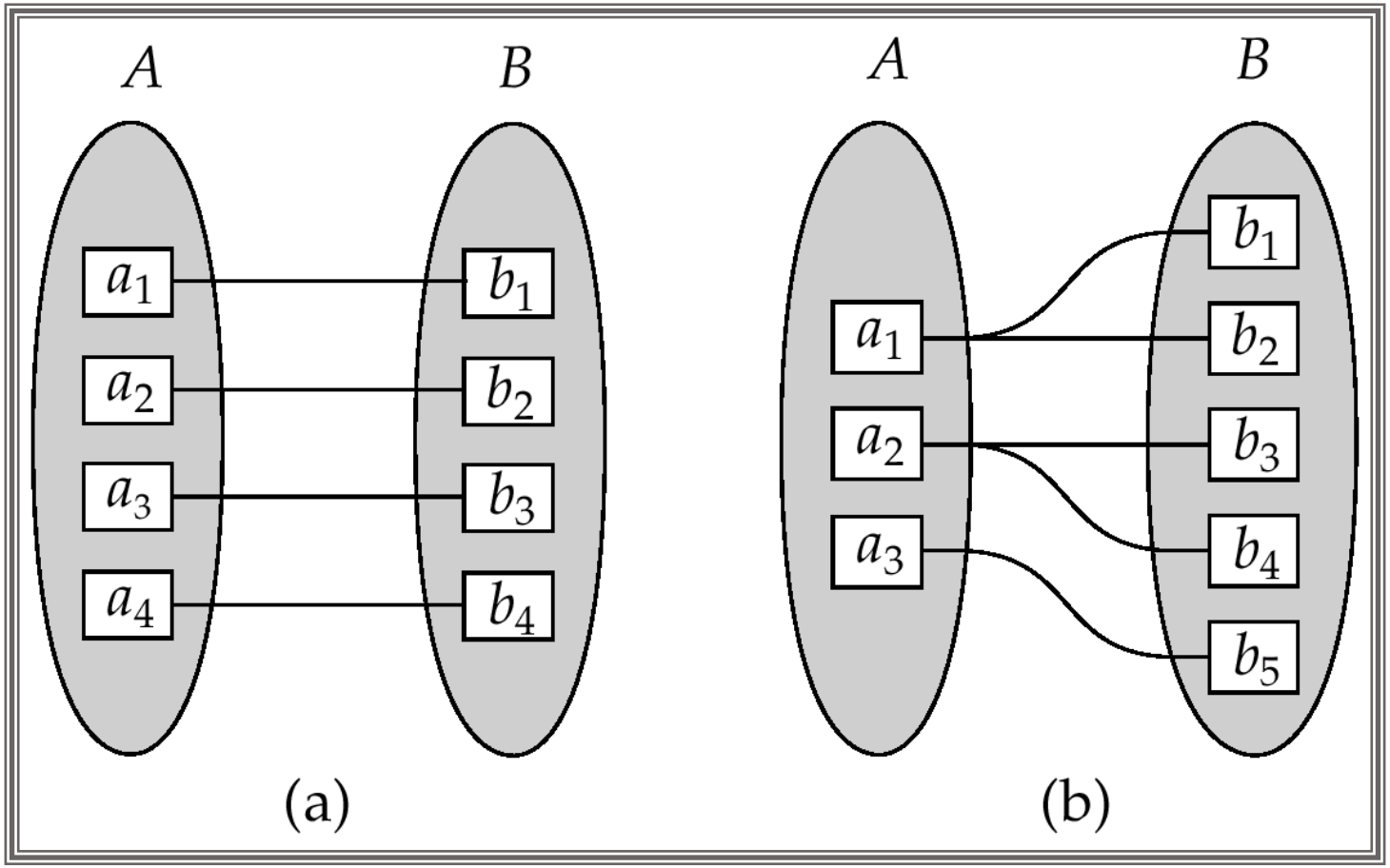
Ternary relationship



Mapping Cardinalities

- For a binary relationship set the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

Mapping Cardinalities

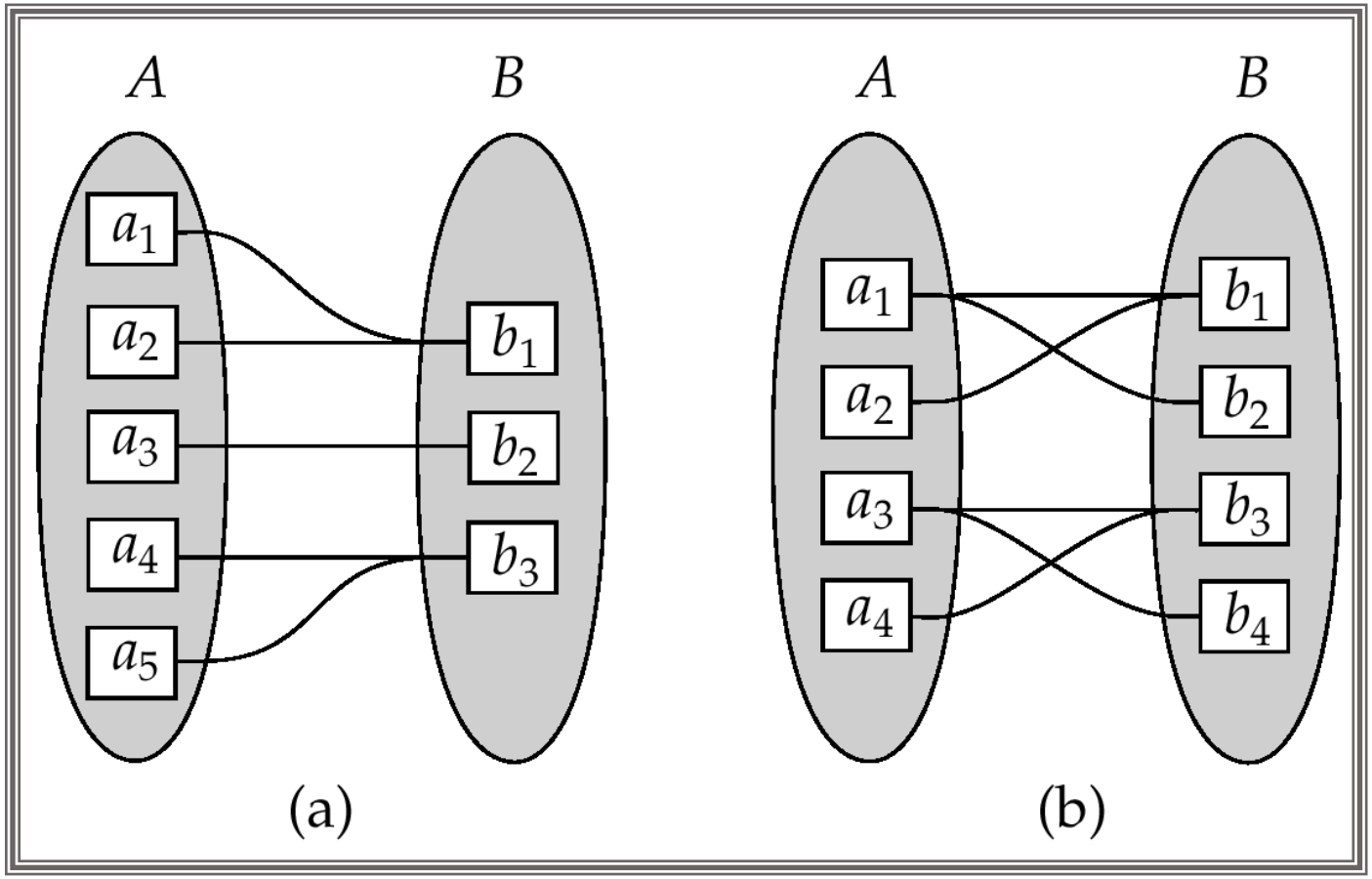


One to one

One to many

Note: Some elements in A and B may not be mapped to any elements in the other set

Mapping Cardinalities



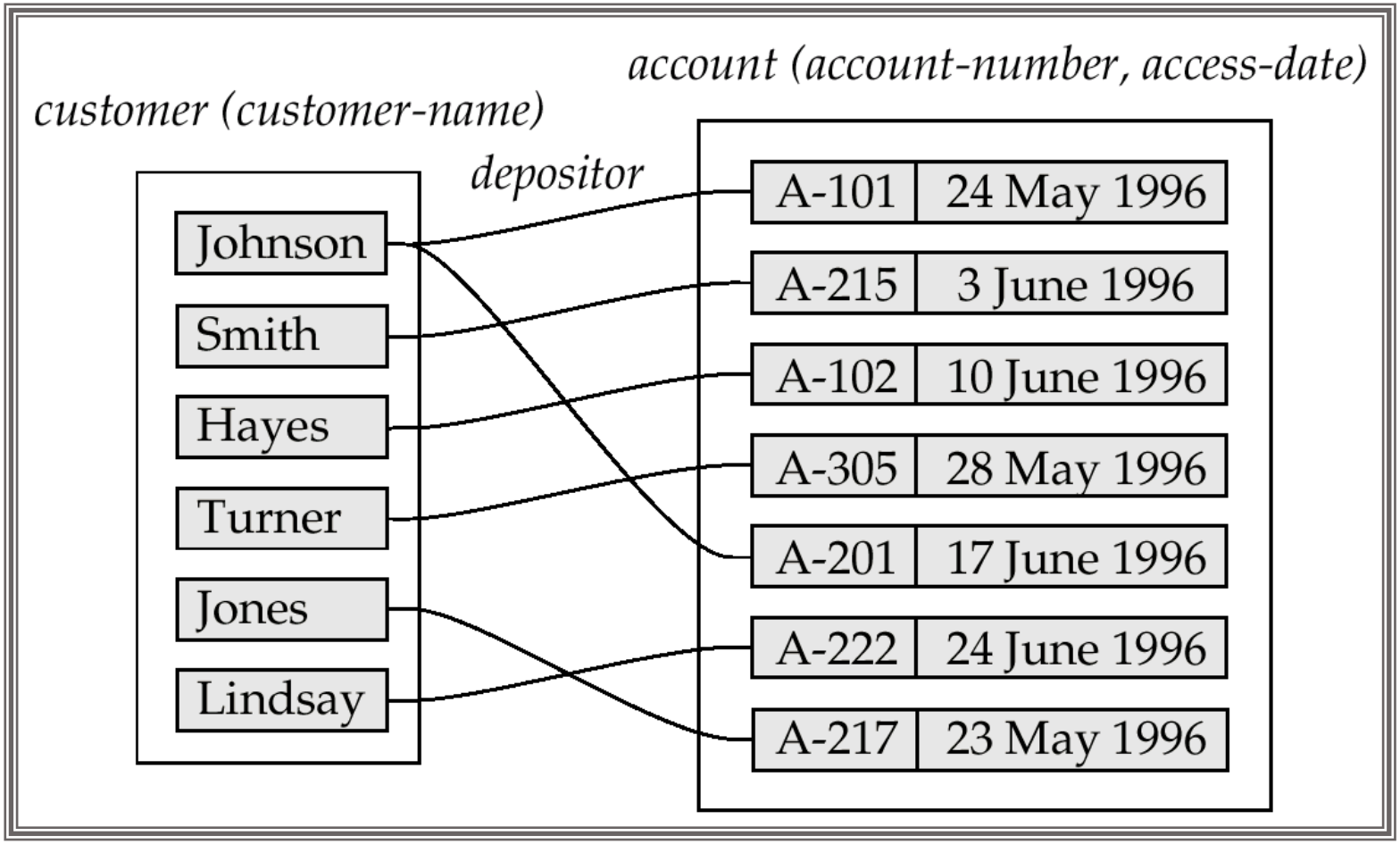
Many to one

Many to many

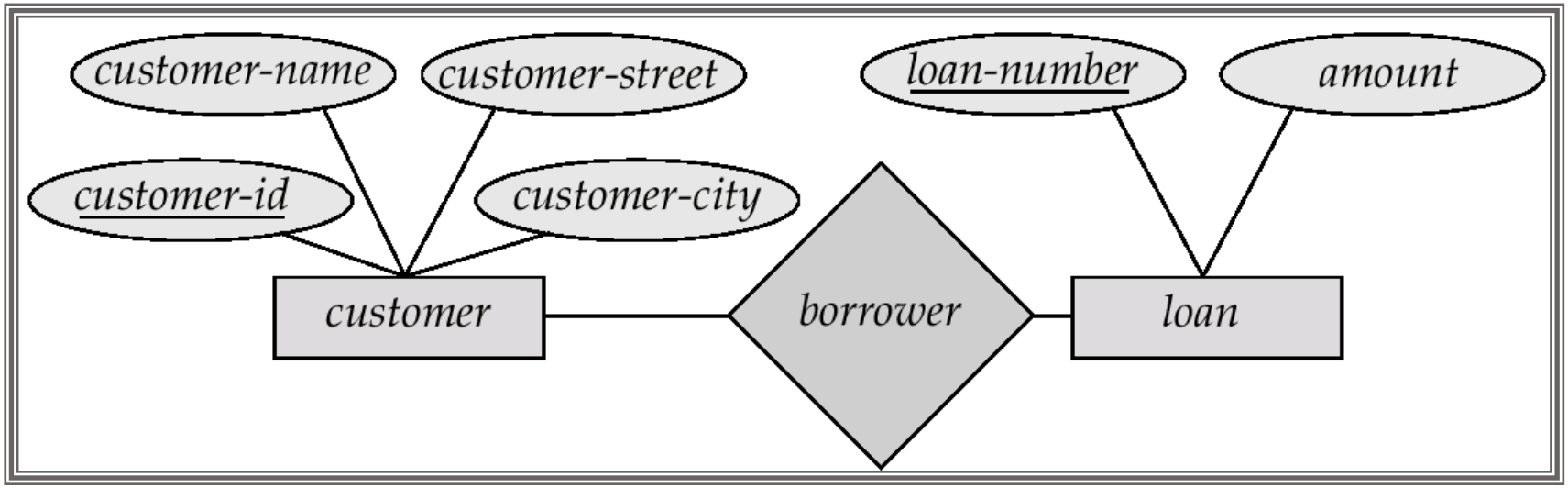
Note: Some elements in A and B may not be mapped to any elements in the other set

Mapping Cardinalities affect ER Design

- Can make *access-date* an attribute of account, instead of a relationship attribute, if each account can have only one customer
- That is, **the relationship from account to customer is many to one**, or equivalently, customer to account is one to many

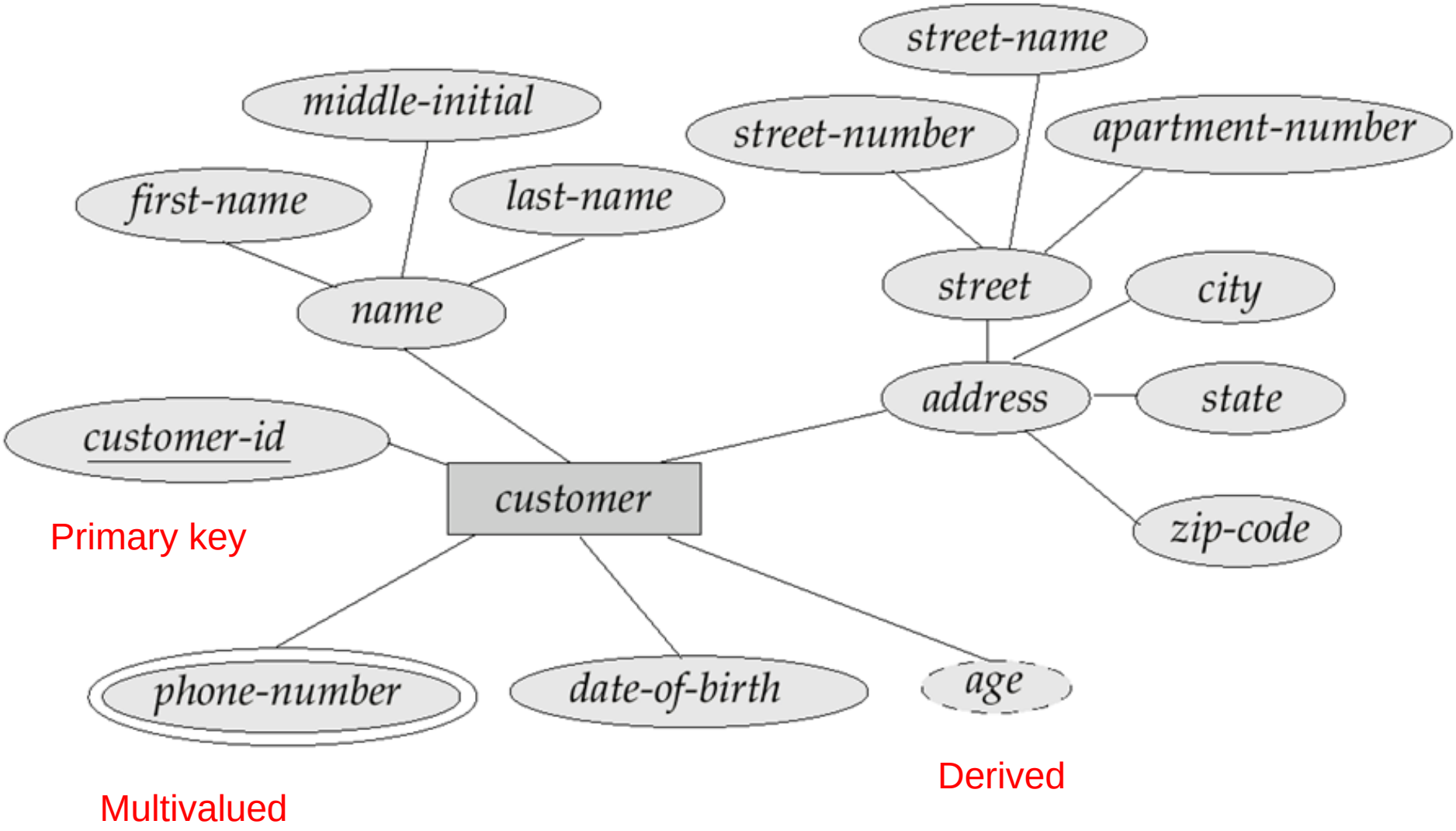


E-R Diagrams



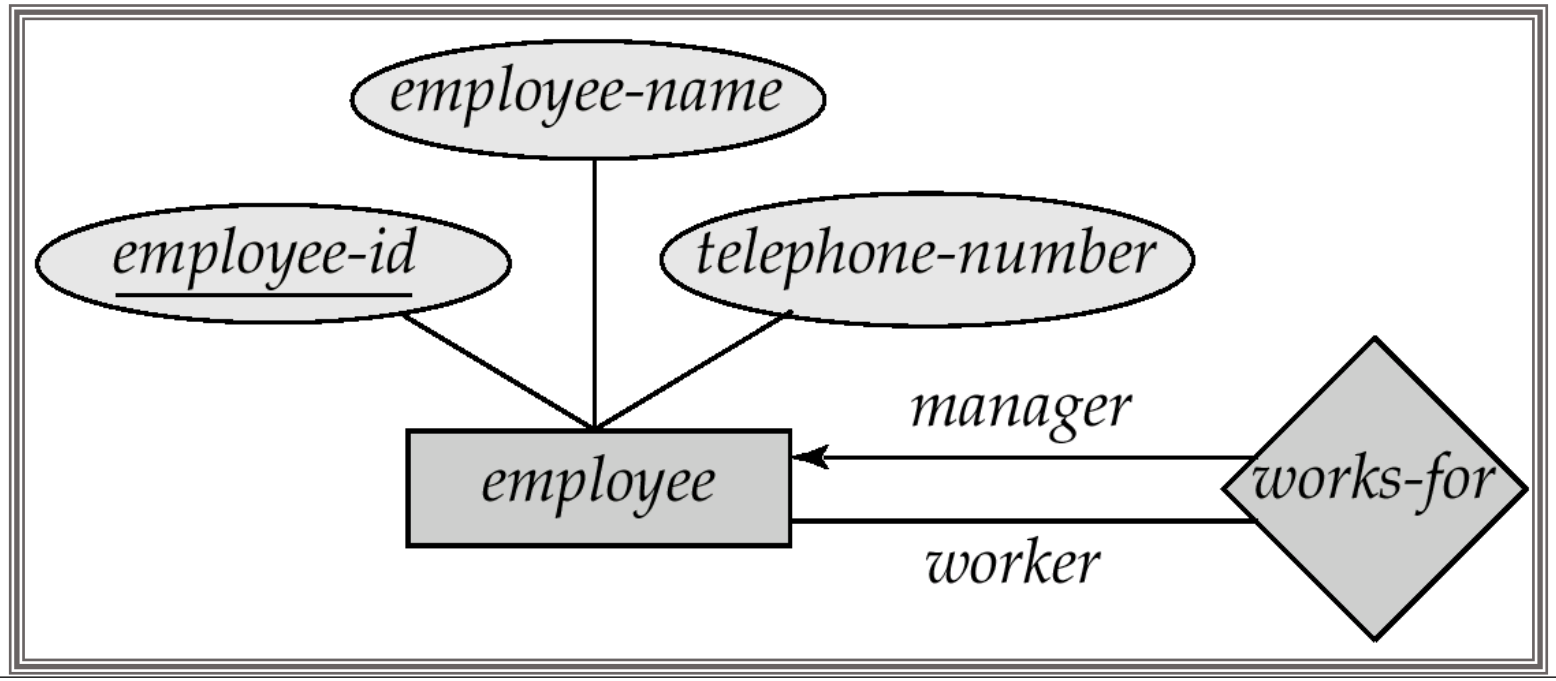
- **Rectangles** represent entity sets.
- **Diamonds** represent relationship sets.
- **Lines** link attributes to entity sets and entity sets to relationship sets.
- **Ellipses** represent attributes
  - **Double ellipses** represent multivalued attributes.
  - **Dashed ellipses** denote derived attributes.
- **Underline** indicates primary key attributes.

E-R Diagram With Composite, Multivalued, and Derived Attributes



Roles

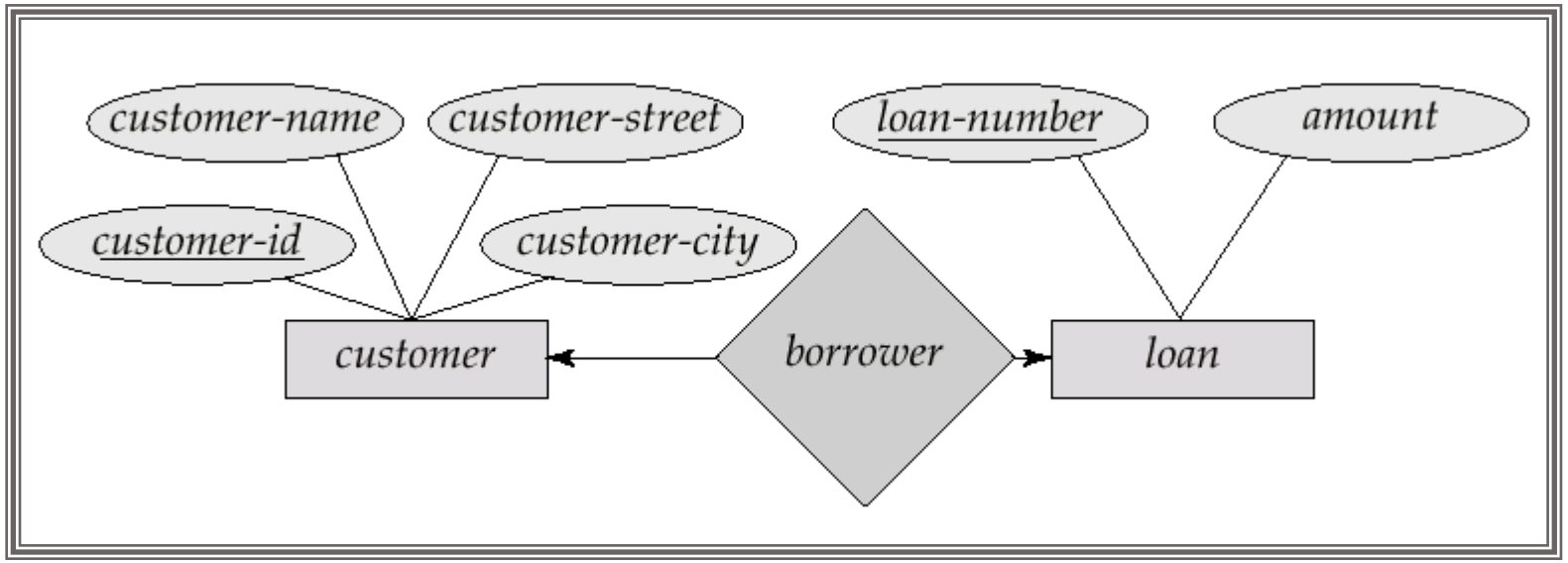
- Entity sets of a relationship need not be distinct
  - The labels "manager" and "worker" are called roles; they specify how employee entities interact via the works-for relationship set.
  - Roles are indicated in E-R diagrams by labeling the lines that connect diamonds to rectangles.
  - Role labels are optional, and are used to clarify semantics of the relationship



Cardinality Constraints

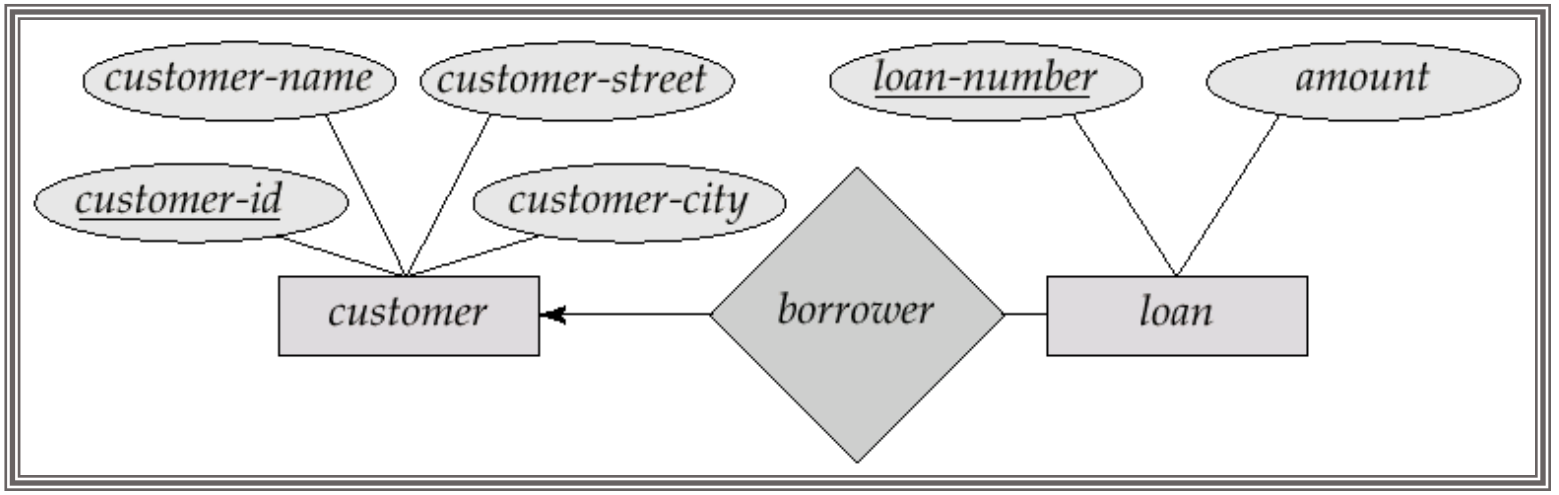
- We express cardinality constraints by drawing either a
  - directed line (→), signifying "one," or
  - an undirected line (—), signifying "many,"

  between the relationship set and the entity set.
- E.g.: One-to-one relationship:
  - A customer is associated with at most one loan via the relationship *borrower*
  - A loan is associated with at most one customer via *borrower*



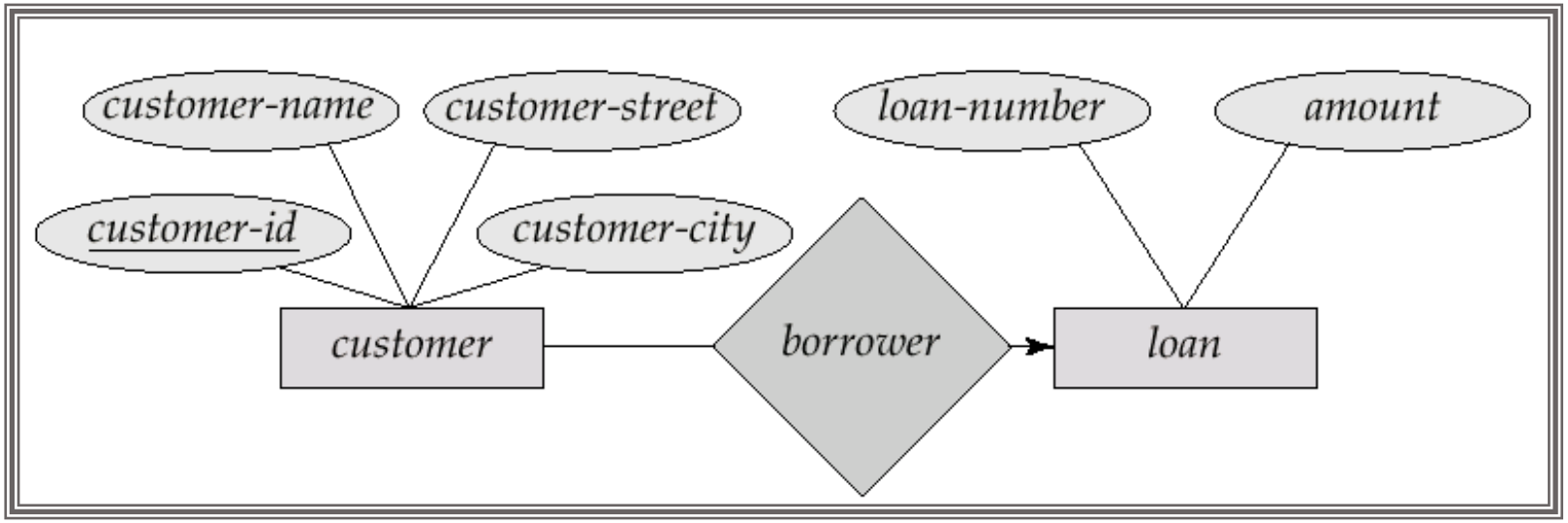
One-To-Many Relationship

- In the one-to-many relationship a loan is associated with at most one customer via *borrower*, a customer is associated with several (including 0) loans via *borrower*

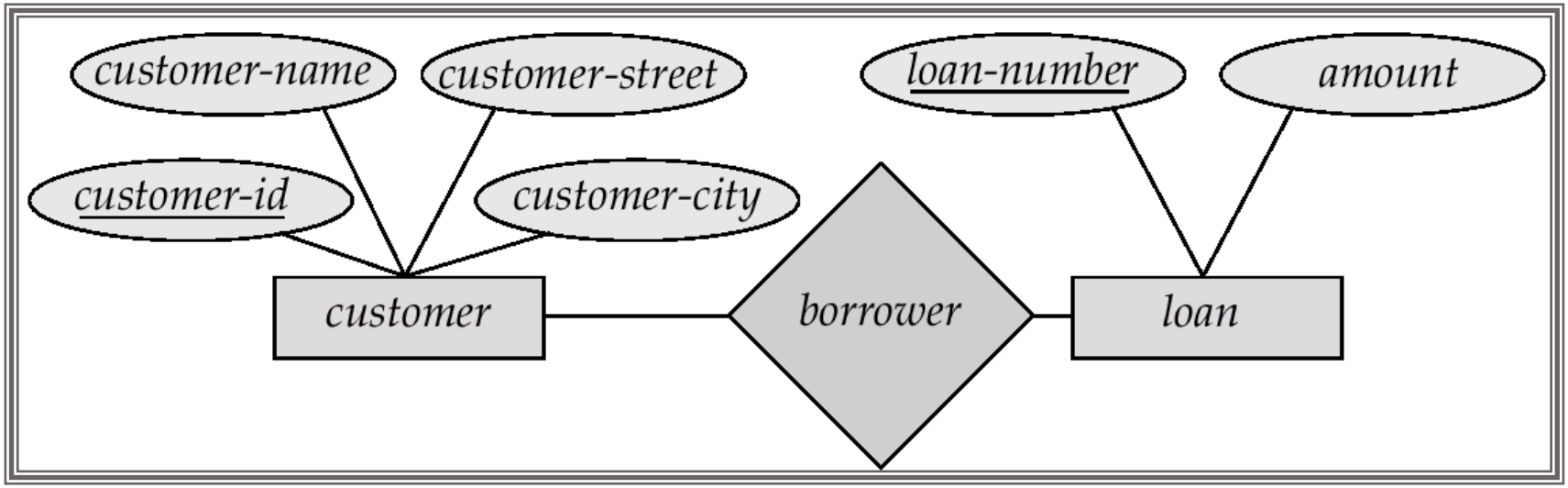


Many-To-One Relationships

- In a many-to-one relationship a loan is associated with several (including 0) customers via *borrower*, a customer is associated with at most one loan via *borrower*



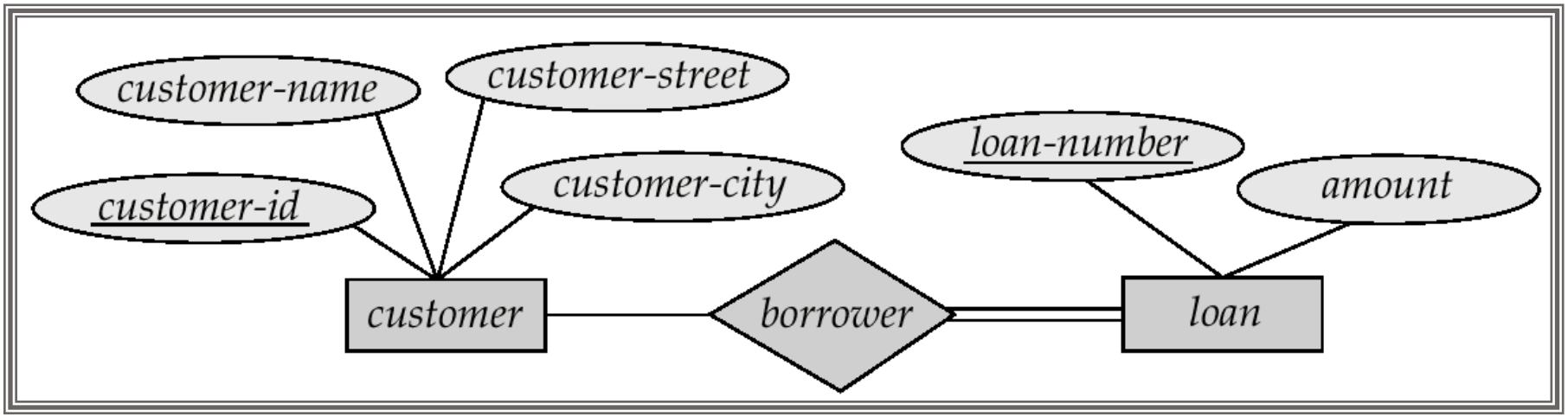
Many-To-Many Relationship



- A customer is associated with several (possibly 0) loans via borrower
- A loan is associated with several (possibly 0) customers via borrower

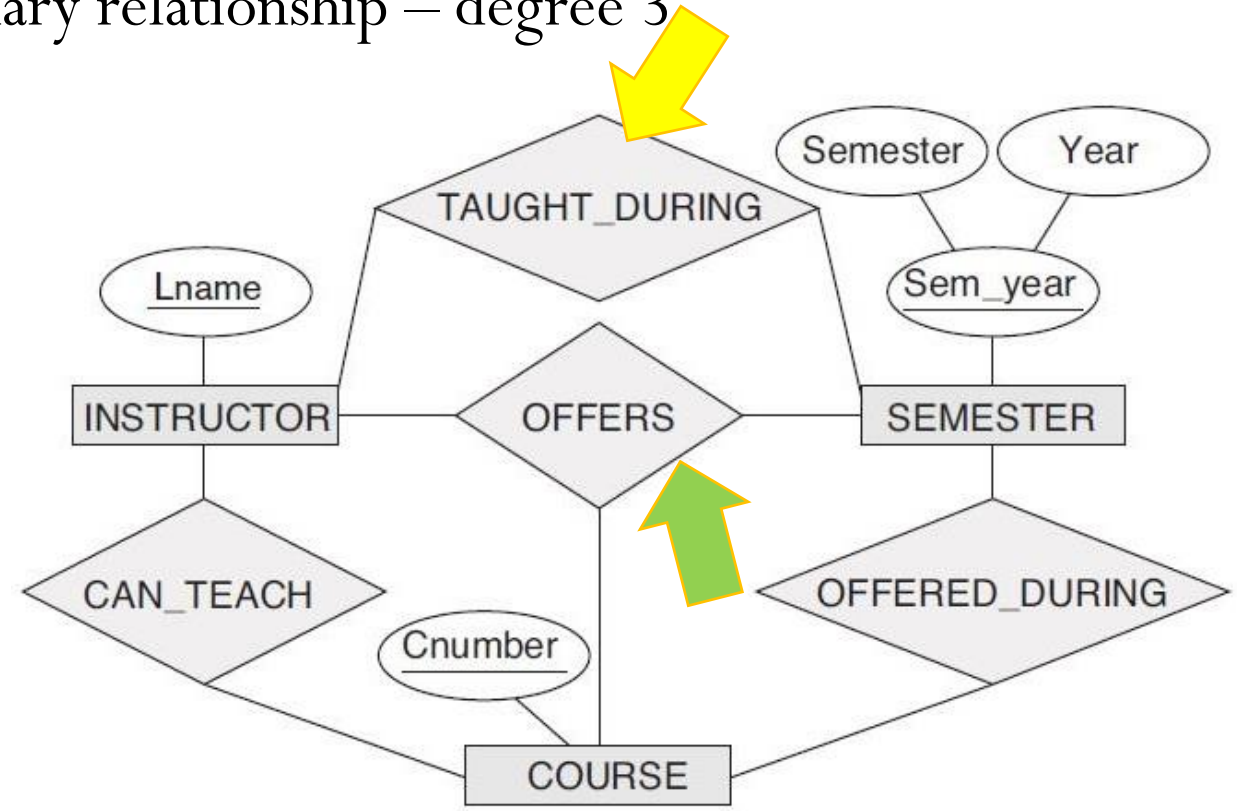
Participation of an Entity Set in a Relationship Set

- Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set
  - E.g. participation of *loan* in *borrower* is total
    - every loan must have a customer associated to it via borrower
- Partial participation: some entities may not participate in any relationship in the relationship set
  - E.g. participation of *customer* in *borrower* is partial

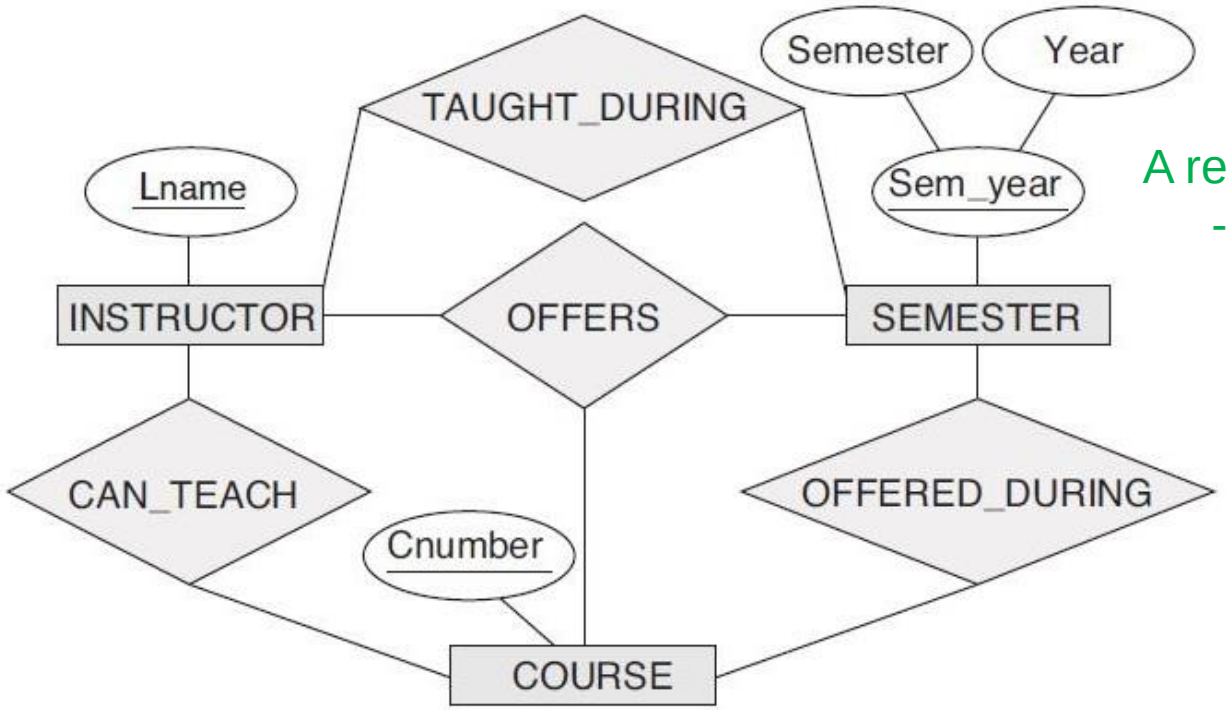


N-ary Relationship

- A relationship type *R* of degree *n* will have *n* edges in an ER diagram, one connecting *R* to each participating entity type.
- Binary relationship – degree 2
- Ternary relationship – degree 3



Ternary vs Binary Relationship



A relationship instance (*i*, *s*, *c*)
- INSTRUCTOR *i* offers
COURSE *c* during
SEMESTER *s*.

- A relationship instance (*i*, *s*, *c*) should not exist in OFFERS *unless* an instance (*i*, *s*) exists in TAUGHT_DURING, an instance (*s*, *c*) exists in OFFERED_DURING, and an instance (*i*, *c*) exists in CAN_TEACH.
- However, the reverse is not always true; we may have instances (*i*, *s*), (*s*, *c*), and (*i*, *c*) in the three binary relationship types with no corresponding instance (*i*, *s*, *c*) in OFFERS.

Ternary vs Binary Relationship

Instructor (i)	Semester (s)	Course (c)
Akhil	Sem-I	C1
Ram	Sem-II	C2
Ravi	Sem-III	C1
Akhil	Sem-III	C3

OFFERS

A relationship instance (*i*, *s*, *c*) – means INSTRUCTOR *i* offers COURSE *c* during SEMESTER *s*

- (Akhil, Sem-I, C1) ∈ OFFERS implies (exists)

  (Akhil, Sem-I) ∈ **TAUGHT_DURING**, ----------(1)

  (Sem-I, C1) ∈ **OFFERED_DURING**, and --------(2)

  (Akhil, C1) ∈ **CAN_TEACH** ----------(3)

Whether converse true ?

That is, given (1), (2), (3) can we say the relation (i, s, c) ?

Ternary vs Binary Relationship

Instructor (i)	Semester (s)	Course (c)
Akhil	Sem-I	C1
Ram	Sem-II	C2
Ravi	Sem-III	C1
Akhil	Sem-III	C3

OFFERS

A relationship instance (*i*, *s*, *c*) – means INSTRUCTOR *i* offers COURSE *c* during SEMESTER *s*

Whether converse is true ?

That is, given (1), (2), (3) can we say the relation (i, s, c) ?

Suppose

(Akhil, Sem-III) ∈ **TAUGHT_DURING**, ----------(1)

(Sem-III, C1) ∈ **OFFERED_DURING**, and --------(2)

(Akhil, C1) ∈ **CAN_TEACH** ----------(3)

implies (exists)

(Akhil, Sem-III, C1) ∈ **OFFERS ….?**

Ternary vs Binary Relationship

Instructor (i)	Semester (s)	Course (c)
Akhil	Sem-I	C1
Ram	Sem-II	C2
Ravi	Sem-III	C1
Akhil	Sem-III	C3

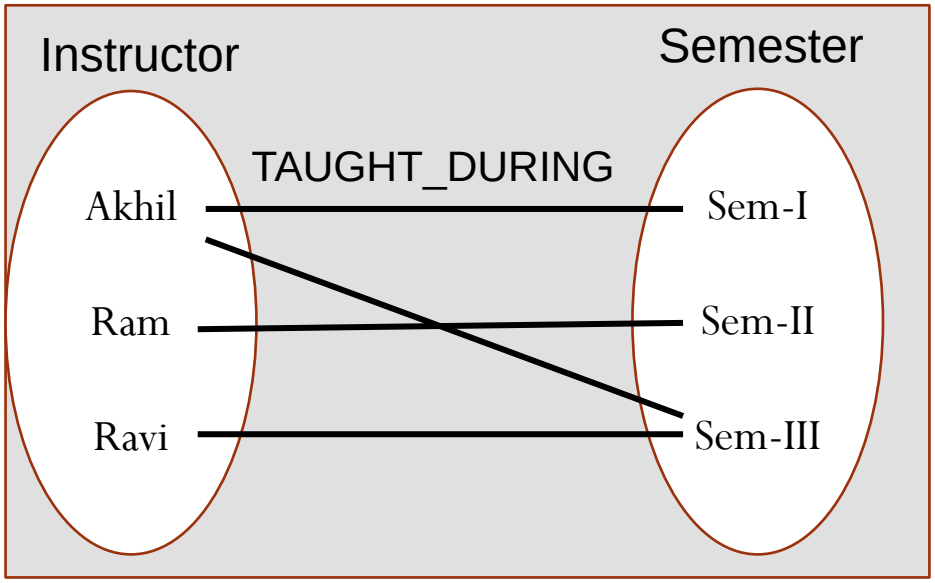
OFFERS

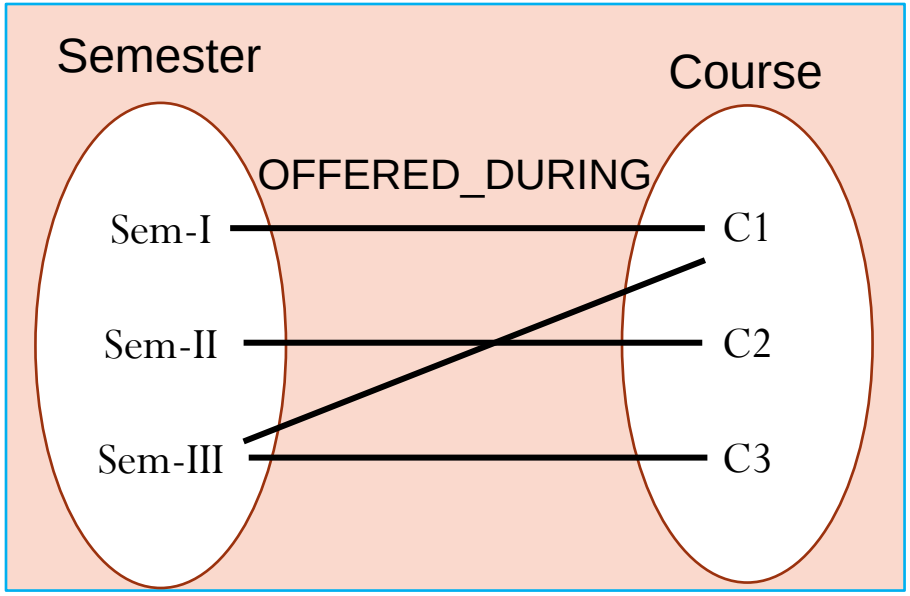
A relationship instance (*i*, *s*, *c*) – means INSTRUCTOR *i* offers COURSE *c* during SEMESTER *s*

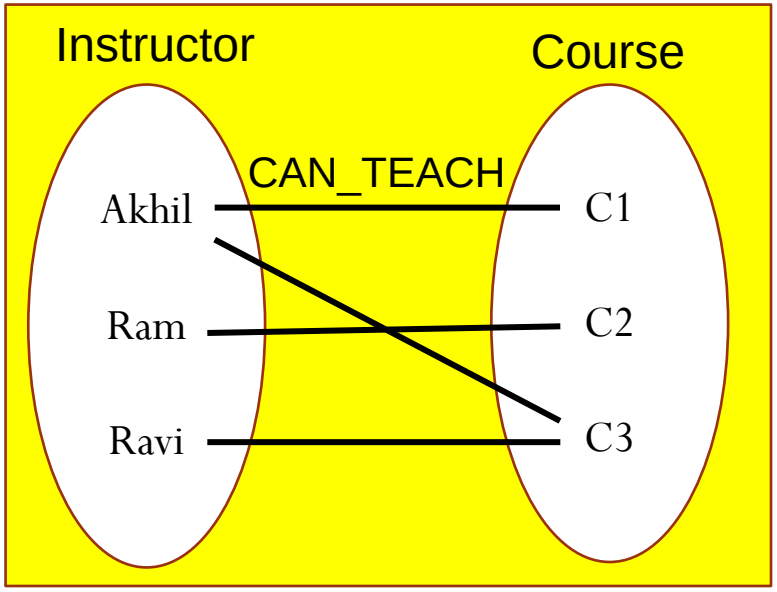
- Based on the meanings of relationships, we can infer the instances of
  - TAUGHT_DURING
  - OFFERED_DURING

  from the instances in OFFERS.
- But, we cannot infer the instances of CAN_TEACH.
- Therefore, TAUGHT_DURING and OFFERED_DURING are redundant, and can be left out.

Constraints on Ternary







(Akhil, Sem-III) ∈ **TAUGHT_DURING** ---(1)

(Sem-III, C1) ∈ **OFFERED_DURING** ----(2)

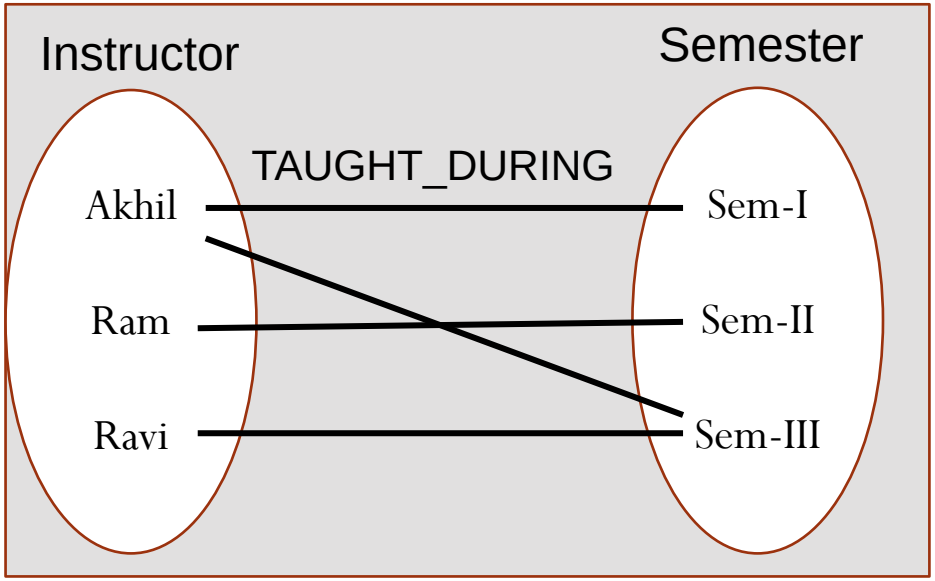
(Akhil, C1) ∈ **CAN_TEACH** ----------(3)

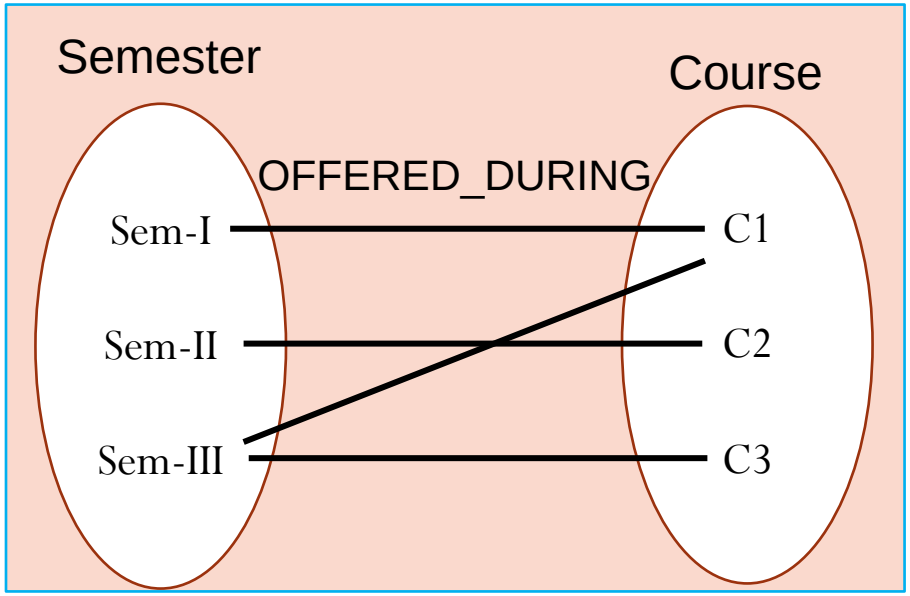
If CAN_TEACH relationship is 1:1

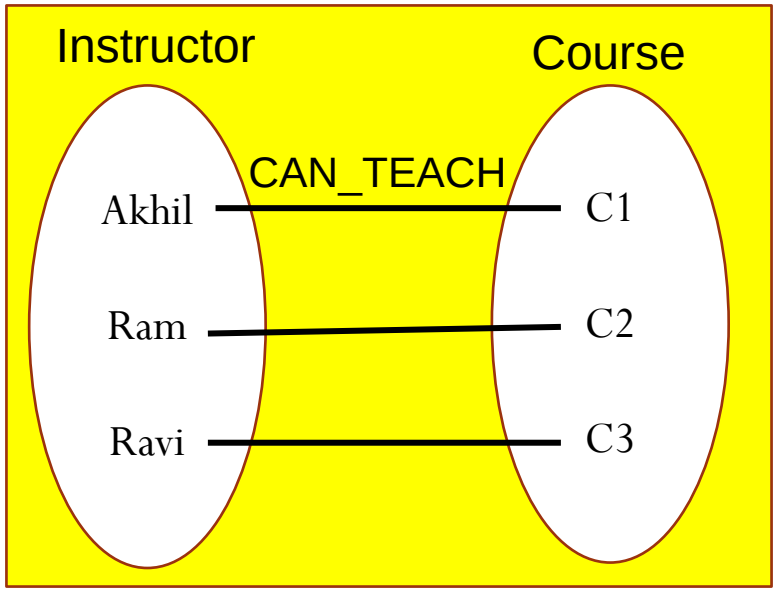
Then ternary can be left out

Because: (i,s), (i, c), (c, s) implies (i, s, c)

Constraints on Ternary







(Akhil, Sem-III) ∈ **TAUGHT_DURING** ---(1)

(Sem-III, C1) ∈ **OFFERED_DURING** ----(2)

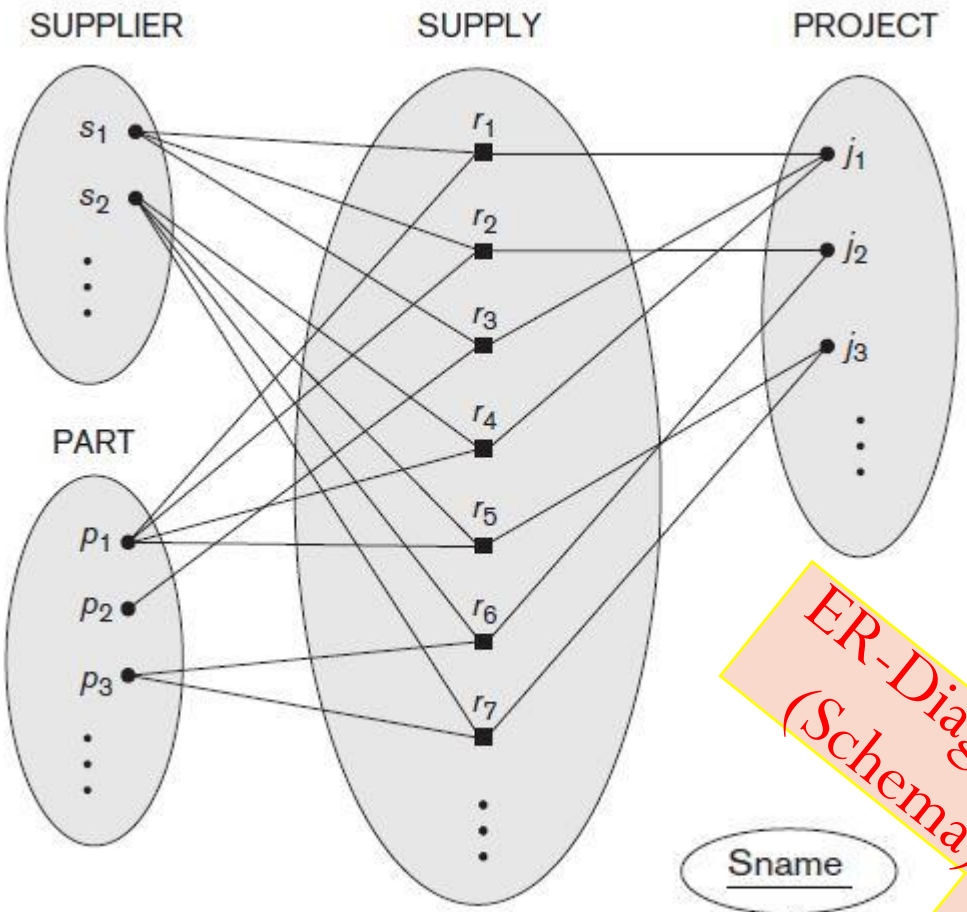
(Akhil, C1) ∈ **CAN_TEACH** ----------(3)

If CAN_TEACH relationship is 1:1

Then ternary can be left out

Because: (i,s), (i, c), (c, s) implies (i, s, c)

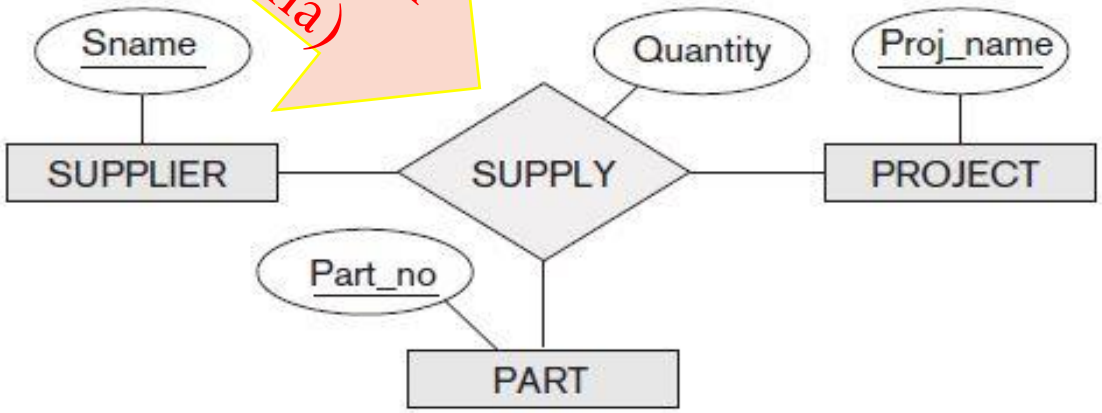
SUPPLY relation



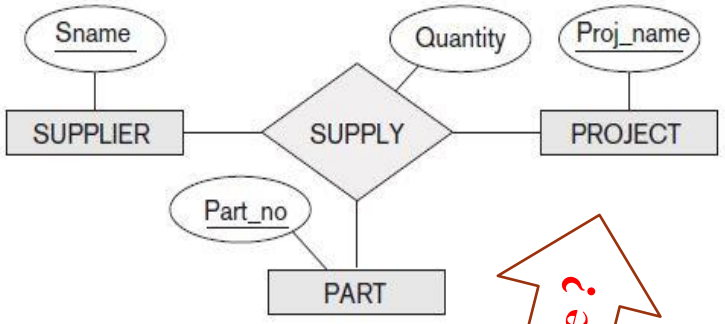
The relationship set of SUPPLY is a set of relationship instances (s, j, p) – that is,

- a SUPPLIER s who is currently
- - supplying a PART p
- - to a PROJECT j.

ER-Diagram (Schema)



SUPPLY relation

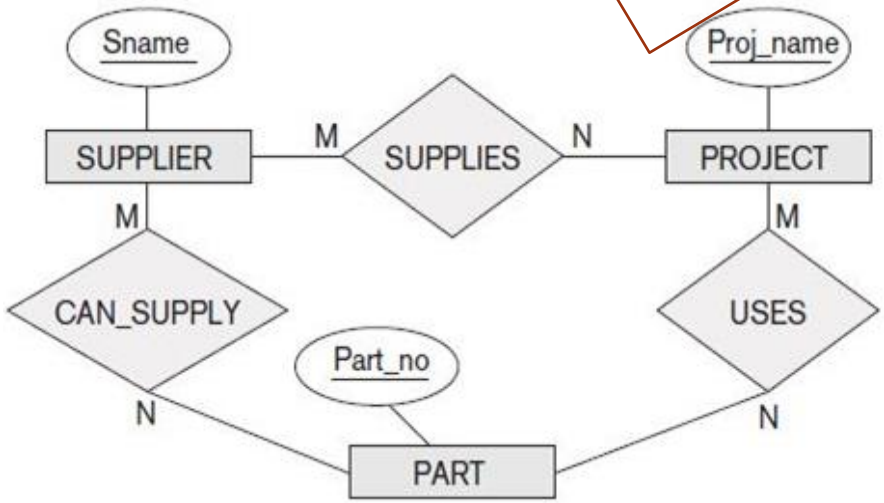


(a) Ternary Relation

The relationship set of SUPPLY is a set of relationship instances (s, j, p) – that is,

- a SUPPLIER s who is currently
- - supplying a PART p
- - to a PROJECT j.

Both same ?



(b) Binary Relation

- CAN_SUPPLY, between SUPPLIER and PART, includes an instance (s, p) whenever supplier s *can supply* part p (to any project);
- USES, between PROJECT and PART, includes an instance (j, p) whenever project j uses part p;
- SUPPLIES, between SUPPLIER and PROJECT, includes an instance (s, j) whenever supplier s supplies *some part* to project j.

THANKS

Entity-Relationship Model

Dr. Odelu Vanga

Indian Institute of Information Technology Sri City

http://www.iiits.ac.in/people/regular-faculty/dr-odelu-vanga/

Weak and Strong Entity Sets

- An entity set that does not have sufficient attributes to form a primary key is termed a **weak entity set**.
- An entity set that has a primary key is termed a **strong (regular) entity set**.

  course: with attributes (*course_id*, *title*, *credits*)

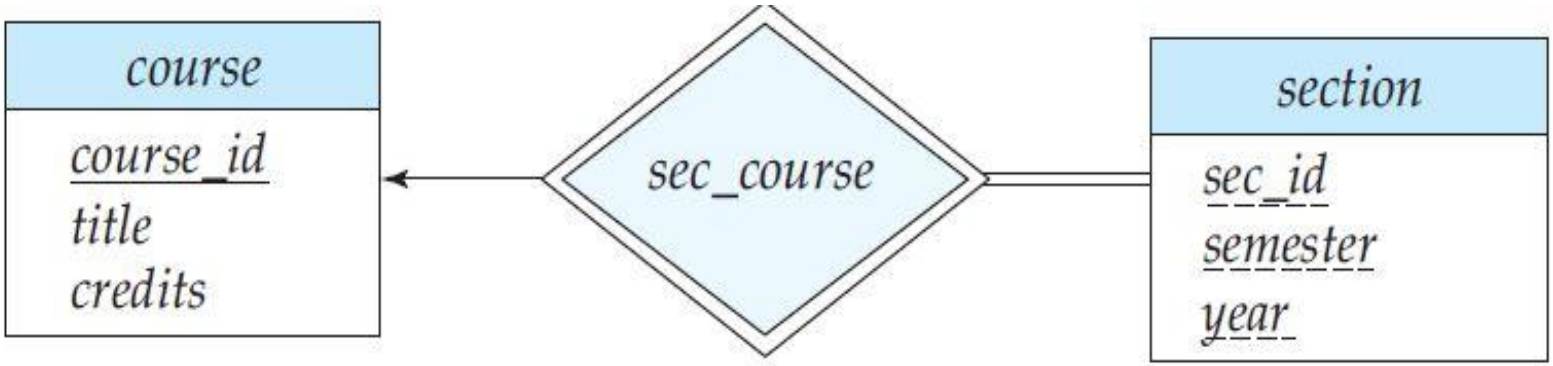
  section: with attributes (*course_id*, *sec_id*, *semester*, *year*)
- Suppose create a relationship-set *sec_course* between entity sets *section* and *course*.
- The primary key of a weak entity set is formed by the primary key of the identifying entity set, plus the weak entity set's discriminator (partial key).

Identifying or **owner entity set**

Identifying relationship

Existence dependent

Weak Entity Set



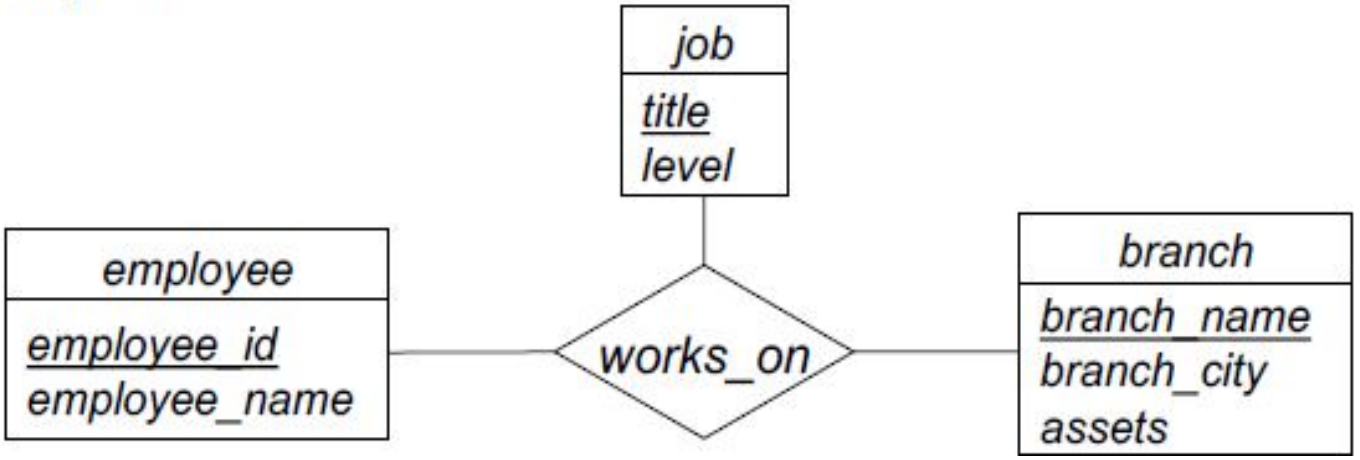
- A weak entity type always has a ***total participation constraint*** (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.

Whether every existence dependency results in a weak entity type ?

- **DRIVER_LICENSE entity cannot exist unless it is related to a PERSON entity, even though it has its own key (License_number) and hence is not a weak entity.**

N-ary Relationships

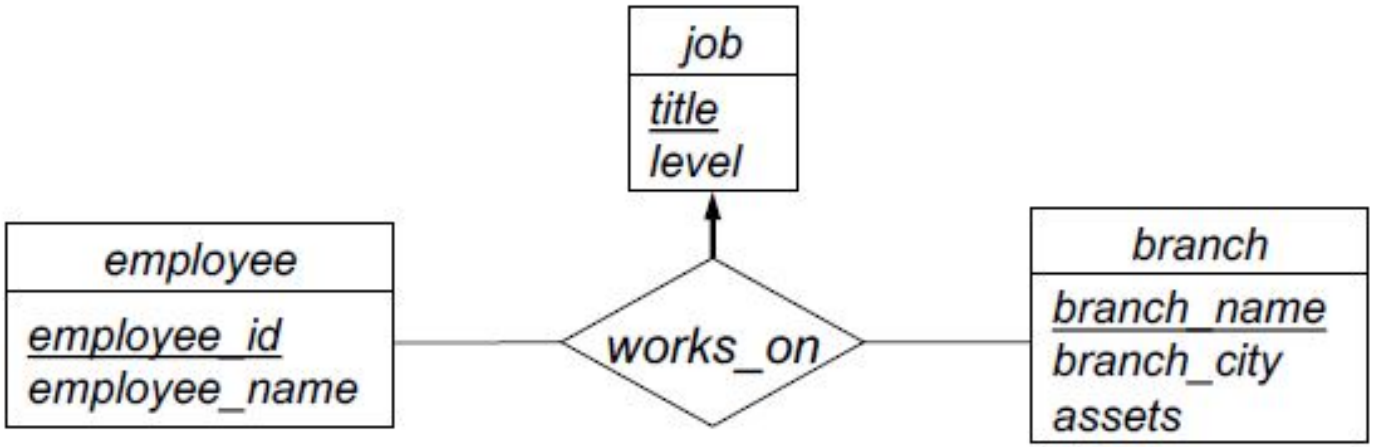
- We can specify relationships of degree > 2 in E-R model
- For example:



- Employees are assigned to jobs at various branches
- **Many-to-many mapping:** any combination of employee, job, and branch is allowed
- An employee can have several jobs at one branch

N-ary Mapping Cardinalities

- We can specify some mapping cardinalities on the relationships with degree > 2



- Each combination of employee and branch can only be associated with one job:
- Each employee can have only one job at each branch

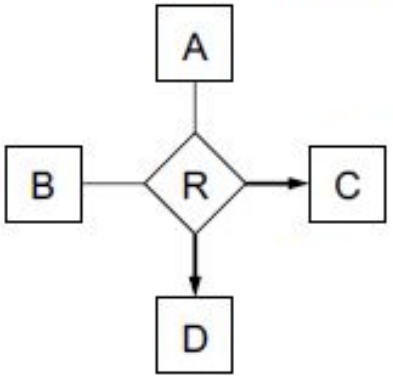
N-ary Mapping Cardinalities

- For degree > 2 relationships, we only allow at most one edge with an arrow.

  Reason: multiple arrows on N-ary relationship-set is ambiguous.
  - Interpreted in two ways.
- Relationship-set R associating entity-sets $A_1, A_2, \ldots, A_n$
  - No arrows on edges $A_1, A_2, \ldots, A_i$
  - Arrows are on edges to $A_{i+1}, A_{i+2}, \ldots, A_n$

- **Meaning 1** (the simpler one):
  - A particular combination of entities in $A_1, A_2, \ldots, A_i$ can be associated with at most one set of entities in $A_{i+1}, A_{i+2}, \ldots, A_n$
  - Primary key of R is union of primary keys from set $\{A_1, A_2, \ldots, A_i\}$

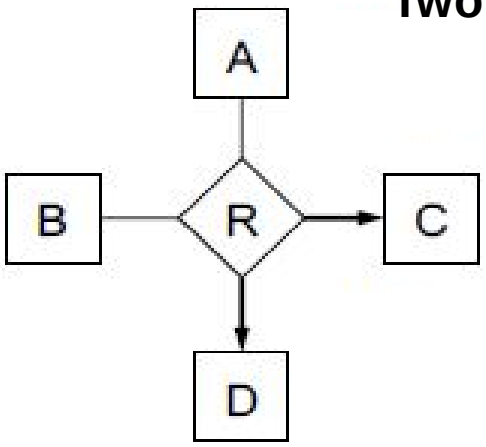
Primary key (a, b)



(a_1, b_2, c_3, d_1)
(a_2, b_3, c_1, d_2)
(a_1, b_3, c_2, d_2)
~~(a_1, b_2, c_3, d_1)~~

N-ary Mapping Cardinalities

- Relationship-set R associating entity-sets $A_1, A_2, \ldots, A_n$
  - No arrows on edges $A_1, A_2, \ldots, A_i$ and Arrows are on edges to $A_{i+1}, A_{i+2}, \ldots, A_n$
- **Meaning 2** (the insane one):
  - For each entity-set A_k $(i < k \leq n)$, a particular combination of entities from all other entity-sets can be associated with at most one entity in A_k
  - R has a candidate key for each arrow in N-ary relationship-set
  - For each k $(i < k \leq n)$, another candidate key of R is union of primary keys from entity-sets $\{A_1, A_2, \ldots, A_i\}$



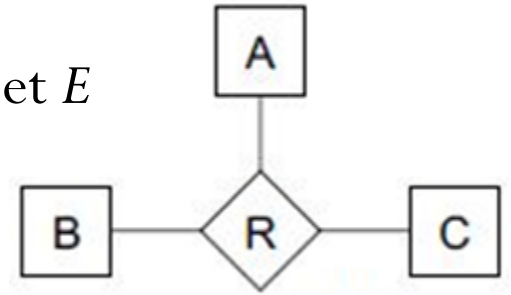
Two candidate key
(a, b, c)
(a, b, d)

1. (a_1, b_2, c_3, d_5)
2. (a_3, b_4, c_3, d_2)
3. (a_1, b_2, c_1, d_4)
4. (a_3, b_4, c_5, d_7)
5. ~~(a_1, b_2, c_3, d_6)~~
6. ~~(a_3, b_4, c_5, d_2)~~

Can these allowed in Meaning 1 ?

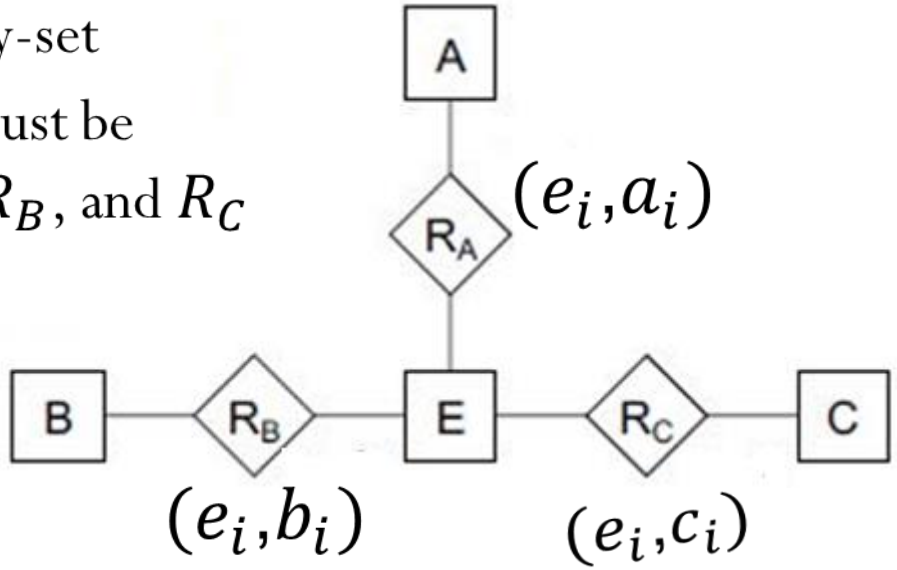
Binary vs. N-ary Relationships

- Often have only binary relationships in DB schemas
- For degree > 2 relationships, *could* replace with binary relationships
  - Replace N-ary relationship-set with a new entity-set *E*
    - Create an identifying attribute for *E*
    - e.g. an auto-generated ID value



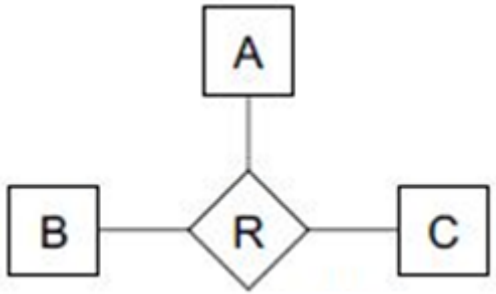
(a_i, b_i, c_i)

  - Create a relationship-set between *E* and each other entity-set
  - Relationships in *R* must be represented in R_A, R_B, and R_C



Binary vs. N-ary Relationships

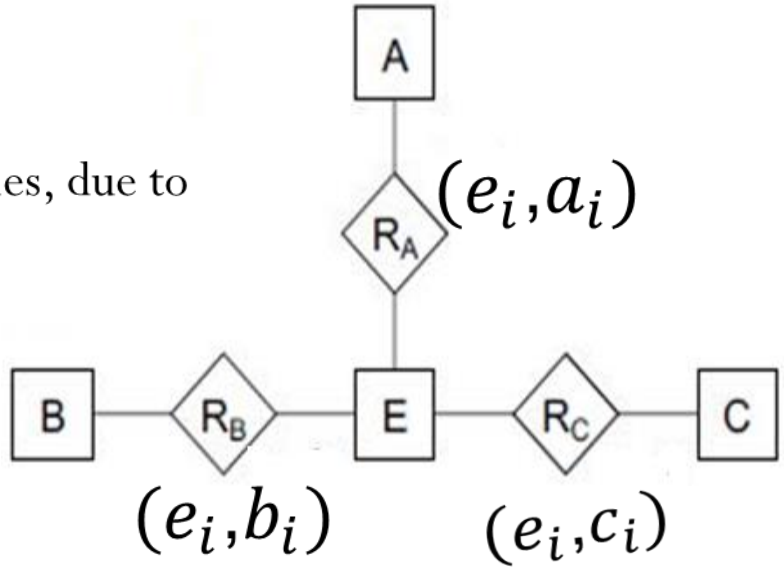
- **Are these representations identical?**
- **Example:** Want to represent a relationship between entities a_5, b_1 and c_2
  - How many relationships can we actually have between these three entities?



(a_i, b_i, c_i)

- **Ternary relationship set:**
  - Can only store one relationship between a_5, b_1 and c_2, due to primary key of R

- **Alternate approach:**
  - Can create many relationships between these entities, due to the entity-set E !
    - $(a_5, e_1), (b_1, e_1), (c_2, e_1)$
    - $(a_5, e_2), (b_1, e_2), (c_2, e_2)$
    - …



- **Can't constrain in exactly the same ways**

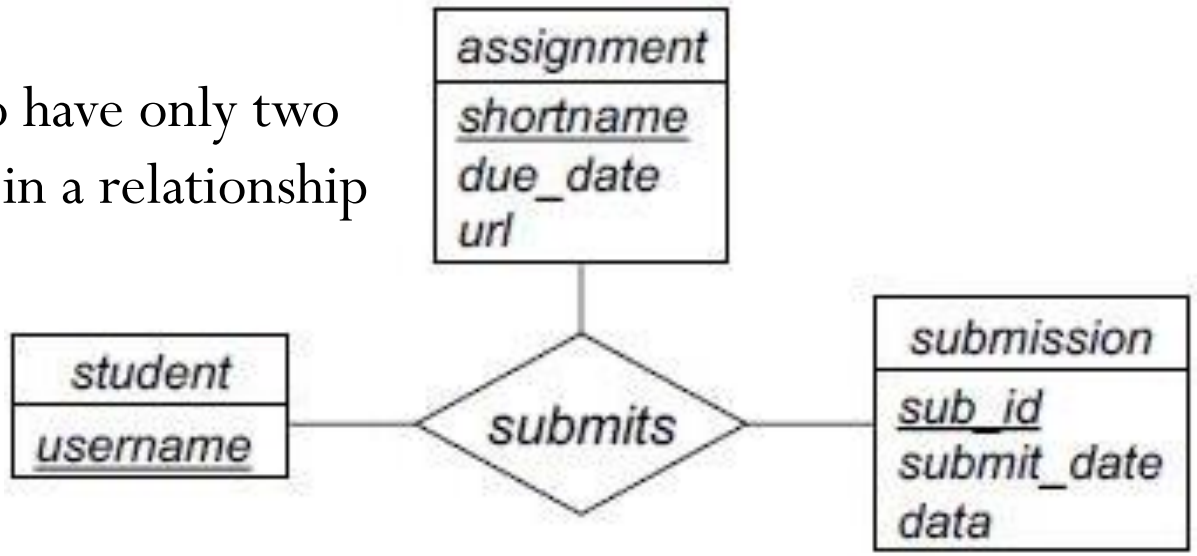
Binary vs. N-ary Relationships

- Using binary relationships is sometimes more intuitive for particular designs
- **Example:** office-equipment inventory database
  - Ternary relationship-set *inventory*, associating *department*, *machine*, and *vendor* entity-sets
- What if vendor info is unknown for some machines?
  - For ternary relationship, must use *null* values to represent missing vendor details
  - With binary relationships, can simply have no relationship between *machine* and *vendor*
- For cases like these, use binary relationships
  - If it makes sense to model as separate binary relationships, do it that way!

Course Database Example -1

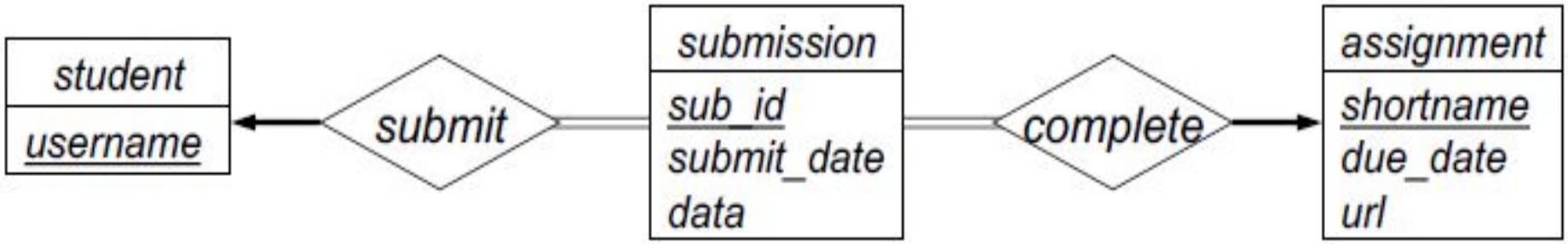
- **What about this case:**
  - Ternary relationship between *student*, *assignment*, and *submission*
  - Need to allow multiple submissions for a particular assignment, from a particular student

- **In this case, it could make sense to represent as a ternary relationship**
  - Doesn't make sense to have only two of these three entities in a relationship



Course Database Example - 2

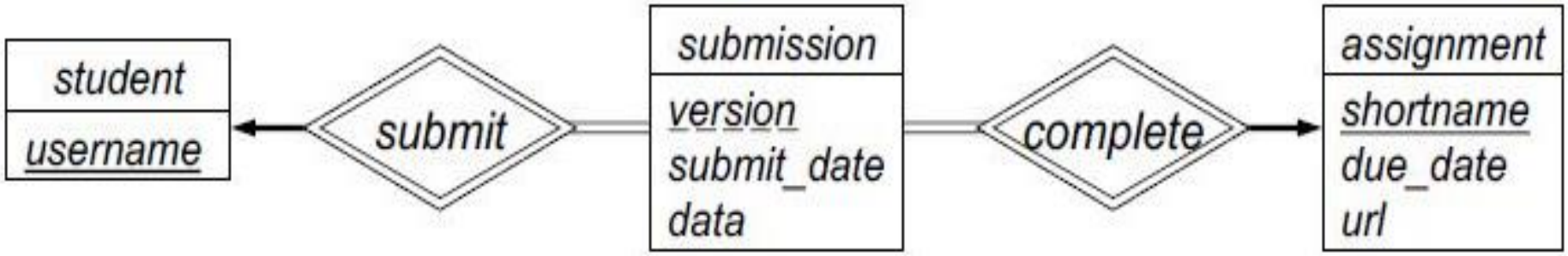
- Other ways to represent students, assignments and submissions?
- Can also represent as two binary relationships



- Note the total participation constraints!
  - Required to ensure that every *submission* has an associated *student*, and an associated *assignment*
  - Also, two one-to-many constraints

Course Database Example - 3

- Could even make *submission* a weak entity-set
  - Both *student* and *assignment* are identifying entities!

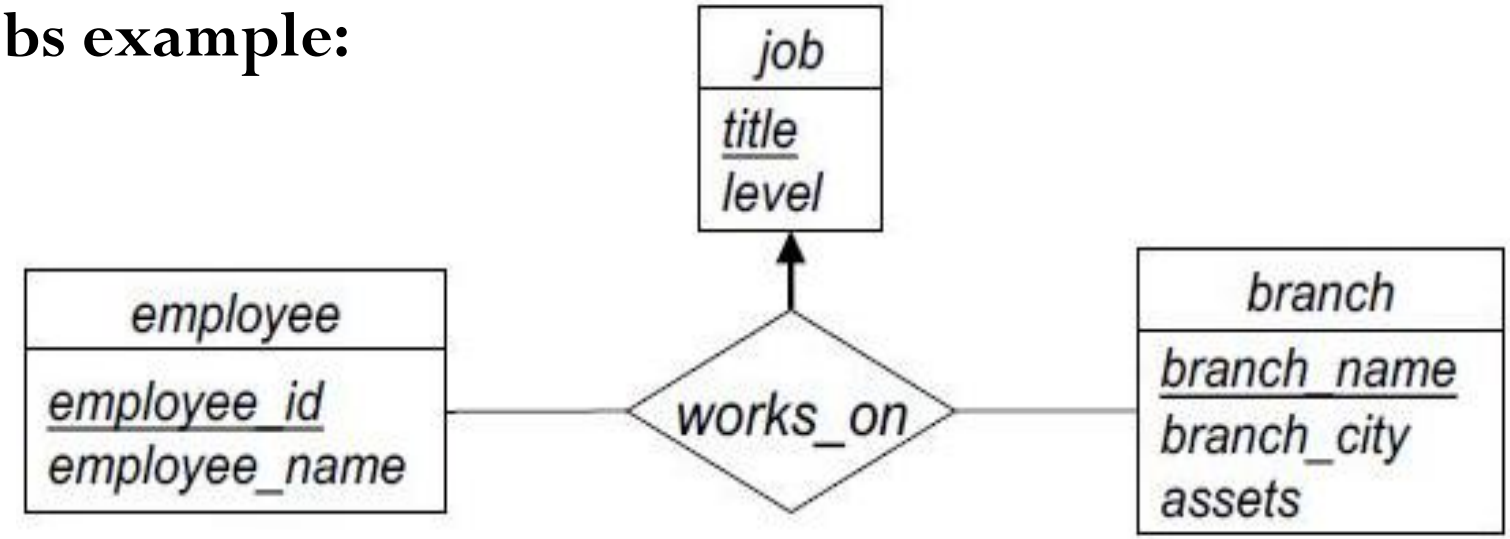


- **Discriminator for *submission* is version number**
- Primary key for *submission* ?
  - Union of primary keys from all owner entity-sets, plus discriminator

(*username*, *shortname*, *version*)

Binary vs. N-ary Relationships

- Sometimes ternary relationships are best
  - Clearly indicates all entities involved in relationship
  - Only way to represent certain constraints!
- **Bank jobs example:**



- **Each (*employee*, *branch*) pair can have only one job**
- **Simply cannot construct the same constraint using only binary relationships.**

Reason ?

E-R Model and Real Databases

- For E-R model to be useful, need to be able to convert diagrams into an implementation schema
- Turns out to be very easy to do this!
  - Big overlaps between E-R model and relational model
  - Biggest difference is E-R composite/multivalued attributes, vs. relational model atomic attributes
- Three components of conversion process:
  - Specify schema of the relation itself
  - Specify primary key on the relation
  - Specify any foreign key references to other relations

Strong Entity-Sets

- Strong entity-set E with attributes $a_1, a_2, \ldots, a_n$
  - Assume simple, single-valued attributes for now
- Create a relation schema with same name E, and same attributes $a_1, a_2, \ldots, a_n$
- Primary key of relation schema is same as primary key of entity-set
  - Strong entity-sets require no foreign keys to other things
- Every entity in E is represented by a tuple in the corresponding relation

Entity-Set Examples

- Geocache location E-R diagram:
  - Entity-set named *location*

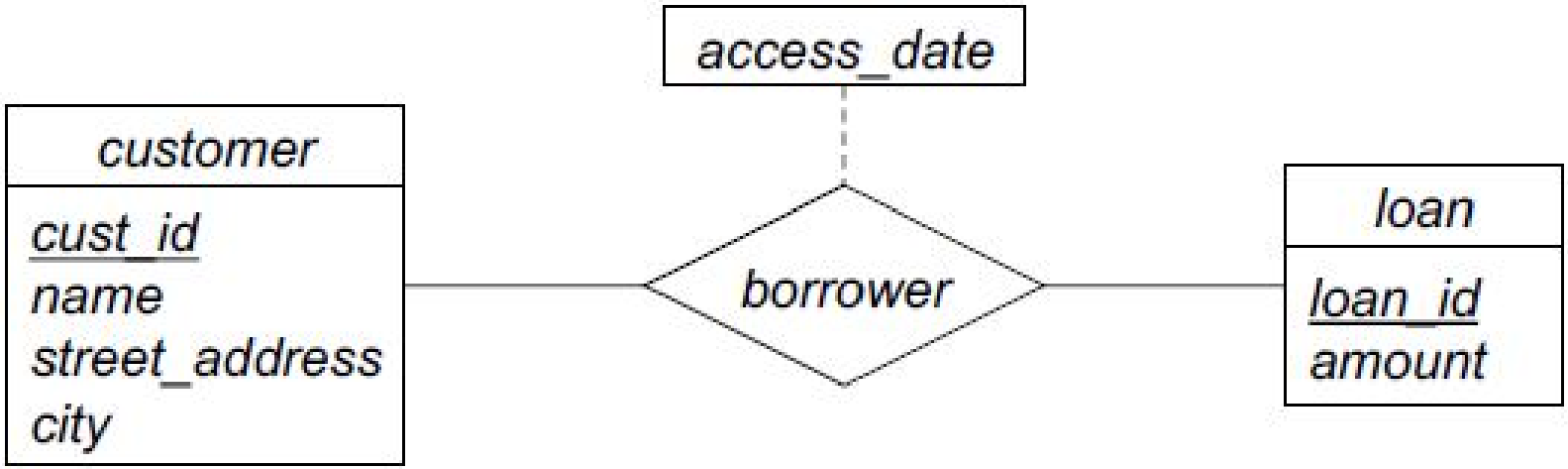
location
latitude
longitude
description
last_visited

- Convert to relation schema:

location(*latitude*, *longitude*, *description*, *last_visited*)

Entity-Set Examples - 2

- **E-R diagram for customers and loans:**



THANK YOU

Entity-Relationship Model

Dr. Odelu Vanga

Indian Institute of Information Technology Sri City

http://www.iiits.ac.in/people/regular-faculty/dr-odelu-vanga/

Weak and Strong Entity Sets

- An entity set that does not have sufficient attributes to form a primary key is termed as a **weak entity set**.
- An entity set that has a primary key is termed a **strong (regular) entity set**.

  course: with attributes (*course_id*, *title*, *credits*)

  section: with attributes (*course_id*, *sec_id*, *semester*, *year*)
- Suppose create a relationship-set *sec_course* between entity sets *section* and *course*.
- The primary key of a weak entity set is formed by the primary key of the identifying entity set, plus the weak entity set's discriminator (partial key).

Identifying or **owner entity set**

Identifying relationship

Existence dependent

Strong Entity-Sets

- Strong entity-set E with attributes $a_1, a_2, \ldots, a_n$
  - Assume simple, single-valued attributes for now
- Create a relation schema with same name E, and same attributes $a_1, a_2, \ldots, a_n$
- Primary key of relation schema is same as primary key of entity-set
  - Strong entity-sets require no foreign keys to other things
- Every entity in E is represented by a tuple in the corresponding relation

Entity-Set Examples

- Geocache location E-R diagram:
  - Entity-set named *location*

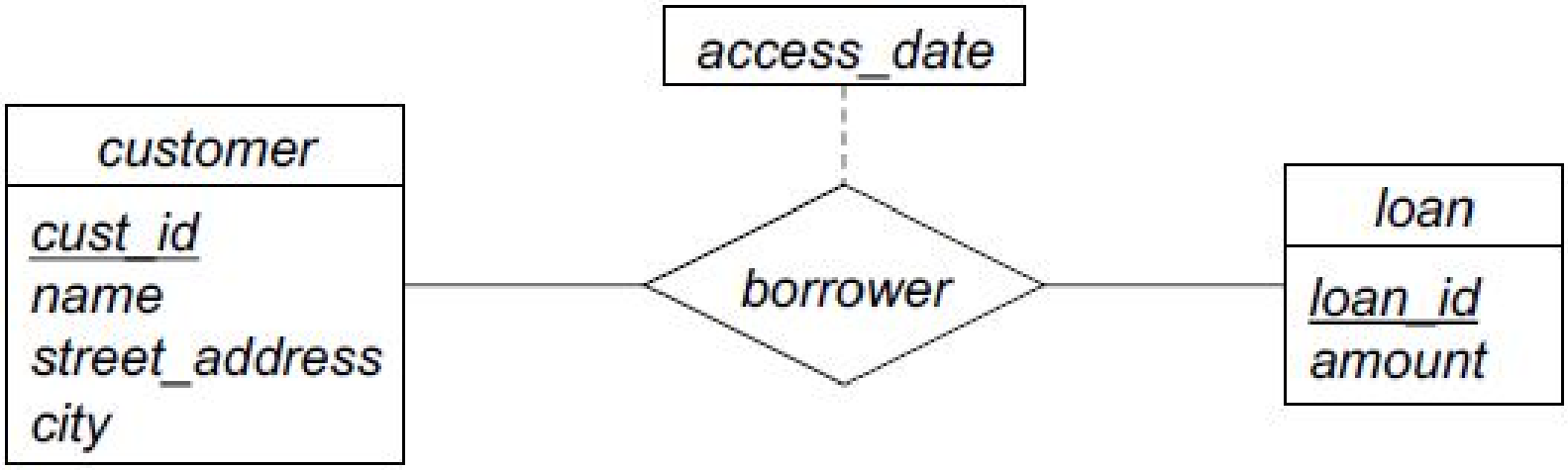
location
latitude
longitude
description
last_visited

- Convert to relation schema:

location(*latitude*, *longitude*, *description*, *last_visited*)

Entity-Set Examples - 2

- **E-R diagram for customers and loans:**



- Convert *customer* and *loan* entity-sets:

  customer(*cust_id*, *name*, *street_address*, *city*)

  loan(*loan_id*, *amount*)

Relationship-Sets

- Relationship-set R
  - For now, assume that all participating entity-sets are strong entity-sets
  - $a_1, a_2, \ldots, a_m$ is the union of all participating entity-sets' primary key attributes
  - $b_1, b_2, \ldots, b_n$ are descriptive attributes on R (if any)
- Relational model schema for R is:
  - $\{a_1, a_2, \ldots, a_m\} \cup \{b_1, b_2, \ldots, b_n\}$
- $\{a_1, a_2, \ldots, a_m\}$ is a super-key, but not necessarily a candidate key
  - Primary key of R depends on R's mapping cardinality

Relationship-Sets: Primary Keys

- For binary relationship-sets:
  - e.g. between strong entity-sets *A* and *B*
  - If many-to-many mapping:
    - Primary key of relationship-set is union of all entity-set primary keys

$$primary\_key(A) \cup primary\_key(B)$$

- If one-to-one mapping:
  - Either entity-set's primary key is acceptable

$$primary\_key(A), \text{ or } primary\_key(B)$$

  - **Enforce both candidate keys in DB schema!**

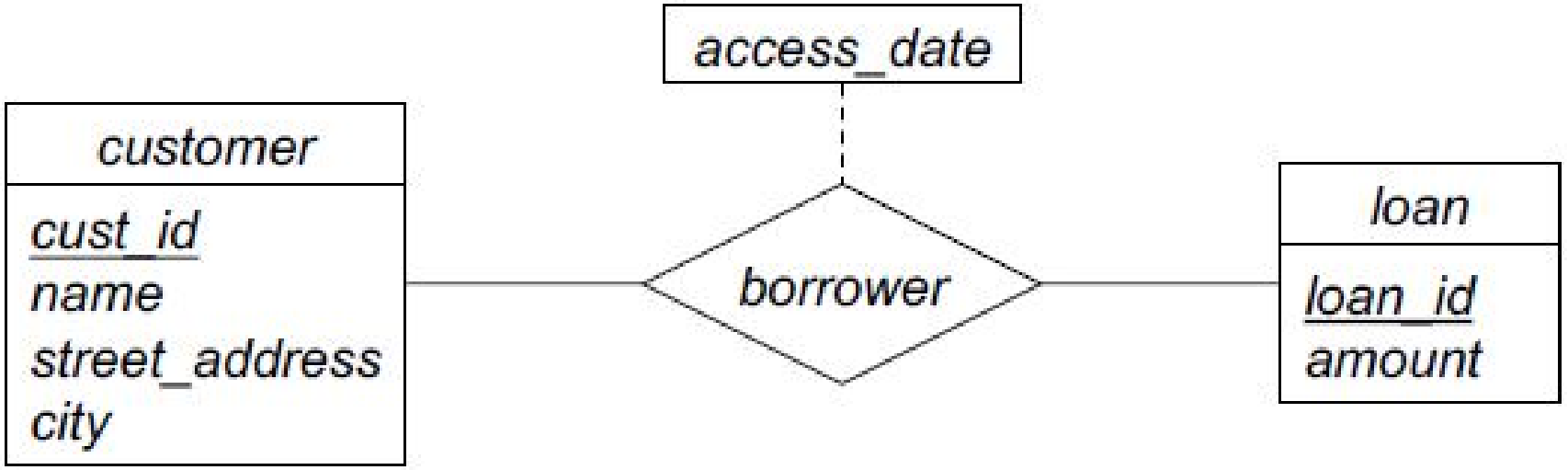
Relationship-Sets: Primary Keys

- For many-to-one or one-to-many mappings:
  - e.g. between strong entity-sets *A* and B
  - Primary key of entity-set on "many" side is primary key of relationship
- **Example:** relationship *R* between *A* and *B*
  - One-to-many mapping, with *B* on "many" side
  - Schema contains *primary_key*(*A*) ∪ *primary_key*(*B*), plus any descriptive attributes on *R*
  - *primary_key*(*B*) is primary key of *R*
    - Each $a \in A$ can map to many $b \in B$
    - Each value for *primary_key*(*B*) can appear only once in *R*

Relationship-Sets: Foreign Keys

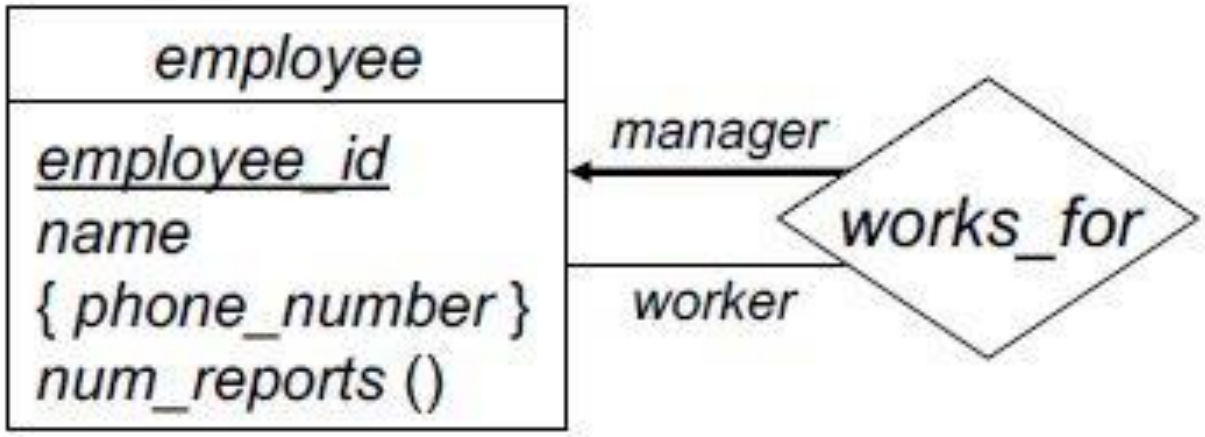
- Relationship-sets associate entities in entity-sets
  - We need foreign-key constraints on relation schema for R !
- For each entity-set E_i participating in R :
  - Relation schema for R has a foreign-key constraint on E_i relation, for $primary\_key(E_i)$ attributes
- Relation schema notation doesn't provide mechanism for indicating foreign key constraints
  - Don't forget about foreign keys and candidate keys!
    - Making notes on your relational model schema is a very good idea
  - Can specify both foreign key constraints and candidate keys in the SQL DDL

Relationship-Sets: Example - 1



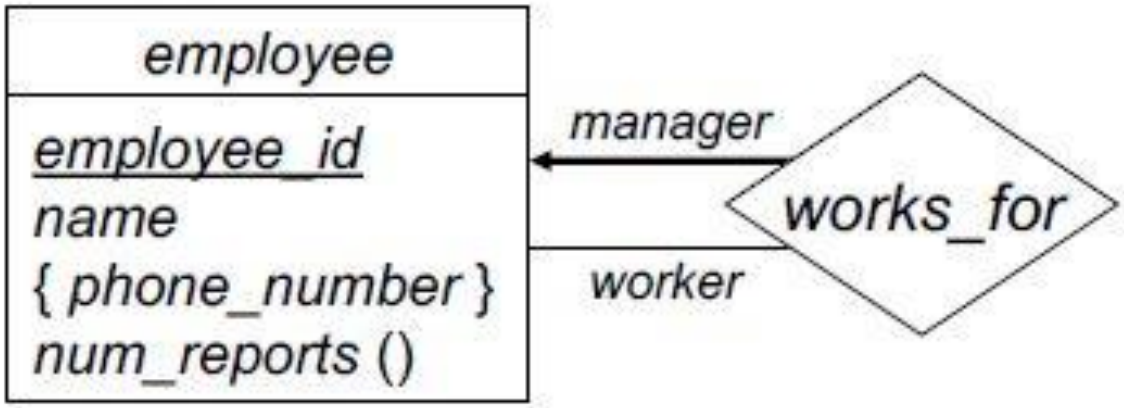
- **Relation schema for *borrower*:**
  - Primary key of *customer* is *cust_id*
  - Primary key of *loan* is *loan_id*
  - Descriptive attribute *access_date*
  - *borrower* mapping cardinality is many-to-many
  - **Result:** *borrower*(*cust_id*, *loan_id*, *access_date*)

Relationship-Sets: Example - 2



- In cases like this, must use roles to distinguish between the entities involved in the relationship-set
  - *employee* participates in *works_for* relationship-set twice
  - Can't create a schema (*employee_id*, *employee_id*) !
- Change names of key-attributes to distinguish **roles**
  - e.g. (*manager_employee_id*, *worker_employee_id*)
  - e.g. (*manager_id*, *employee_id*)

Relationship-Sets: Example - 2



- Relation schema for *employee* entity-set:
  - (For now, ignore *phone_number* and *num_reports*…)

    employee(*employee_id*, *name*)
- Relation schema for *works_for*:
  - One-to-many mapping from *manager* to *worker*
  - "Many" side is used for primary key
  - **Result:** *works_for*(*employee_id*, *manager_id*)

THANK YOU

Relational Model

Dr. Odelu Vanga

Indian Institute of Information Technology Sri City

http://www.iiits.ac.in/people/regular-faculty/dr-odelu-vanga/

Basic Structure

- Formally, given sets $D_1, D_2, \ldots D_n$, a *relation r* is a subset of $D_1 \times D_2 \times \ldots \times D_n$
  That is, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$, where $a_i \in D_i$
- 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*

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

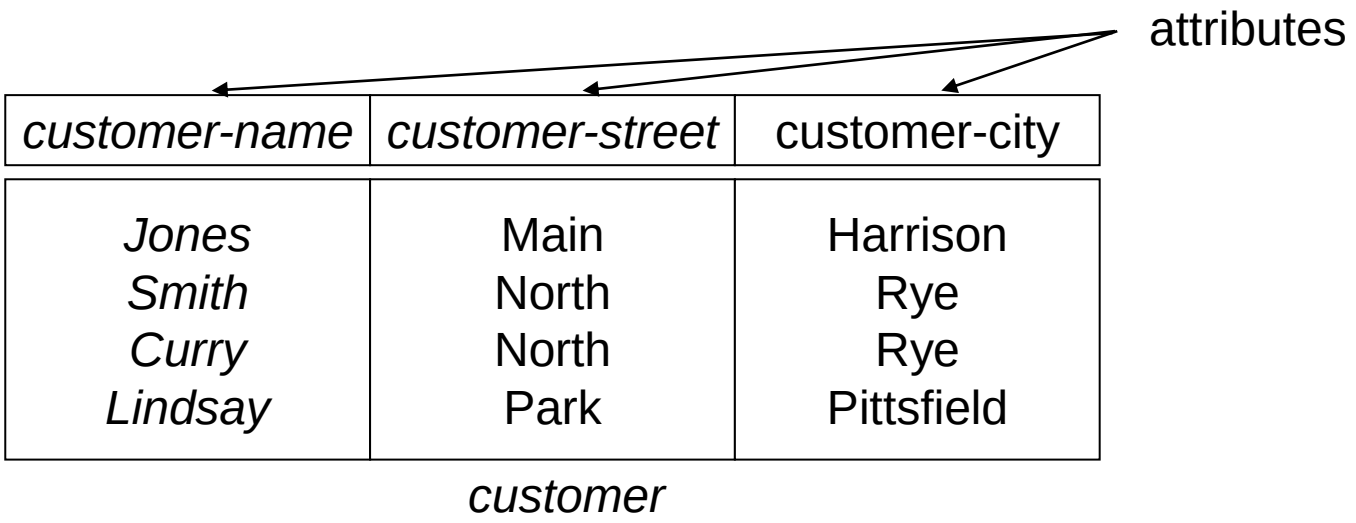
customer

Relation Schema

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

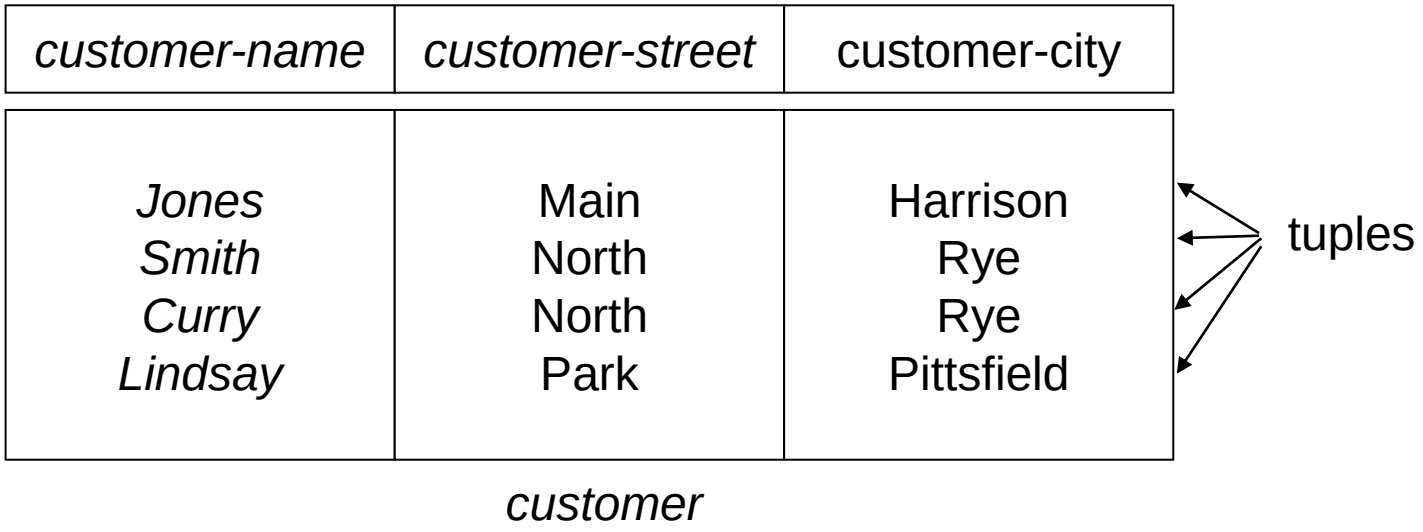
  E.g. *Customer-schema* = (*customer-name, customer-street, customer-city*)
- $r(R)$ is a *relation* on the *relation schema* R

  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)

- **Normalization theory, deals with how to design relational schemas**

Banking Example

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

Assume branch-name is unique

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

Assume customer-name is unique

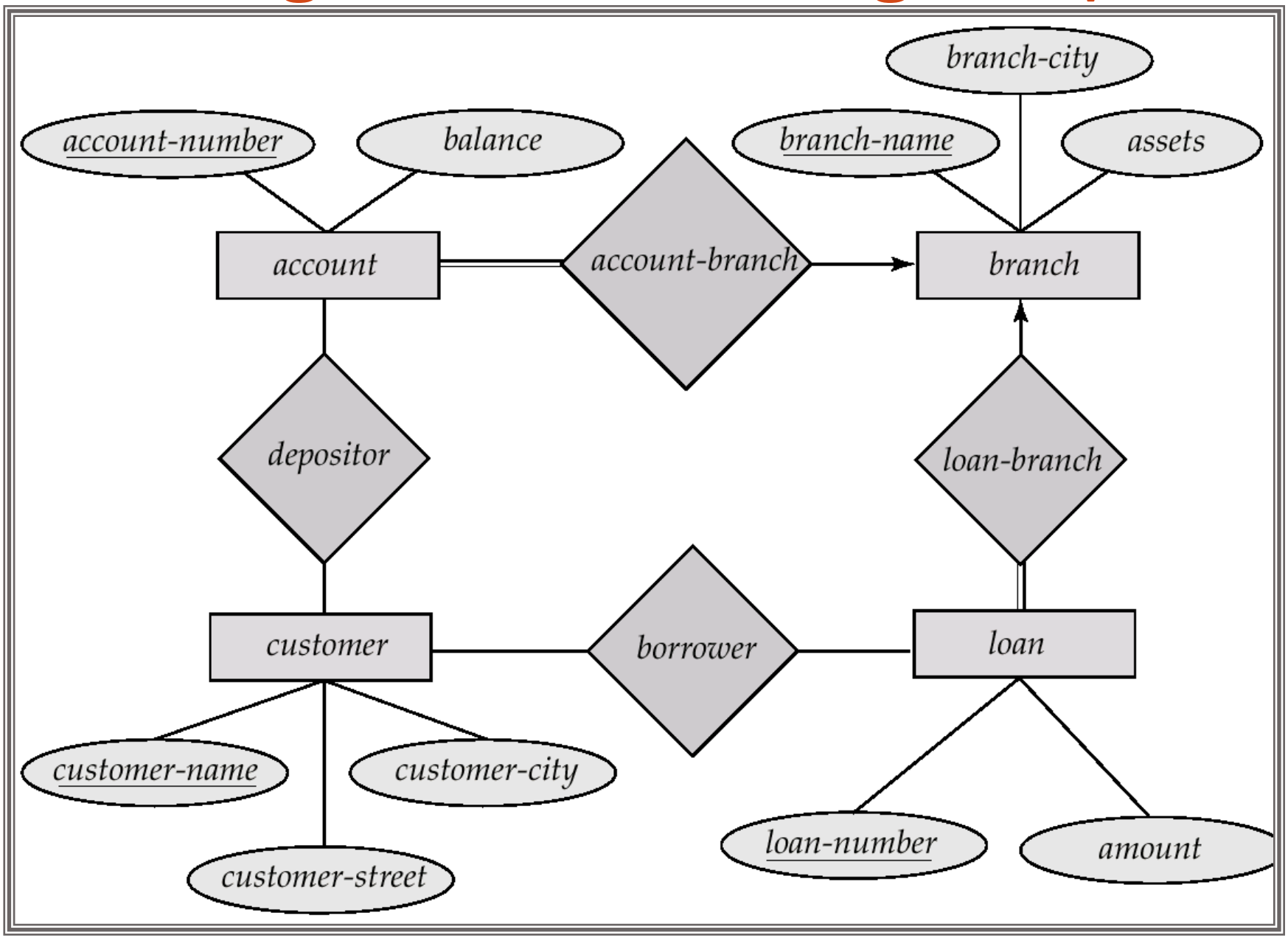
account (*account-number*, *branch-name*, *balance*)

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

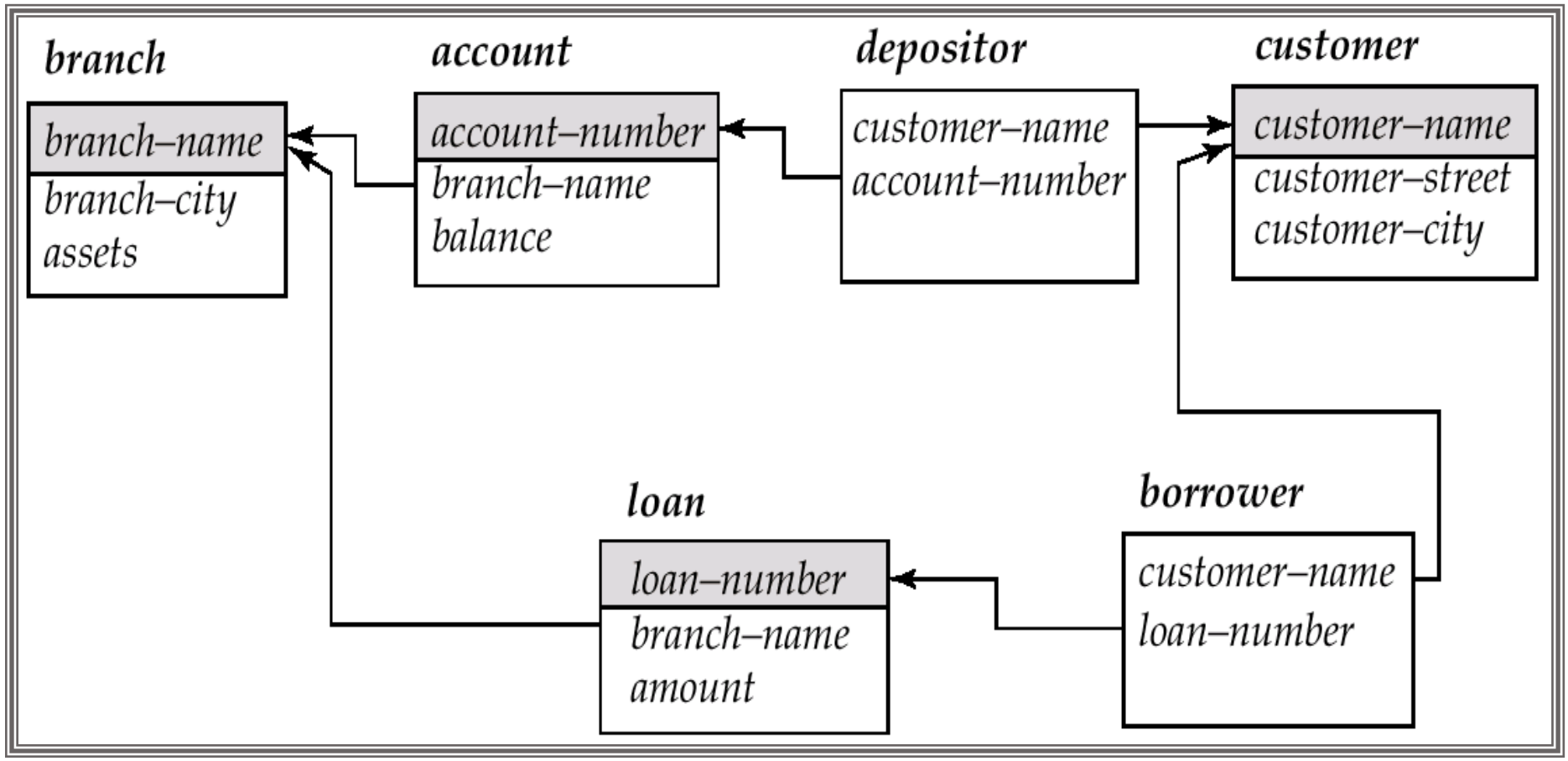
depositor (*customer-name*, *account-number*)

borrower (*customer-name*, *loan-number*)

E-R Diagram for the Banking Enterprise



Schema Diagram for the Banking Enterprise



Relational Algebra

- Six basic operators
  - select
  - project
  - union
  - set difference
  - Cartesian product
  - rename

- **The operators take two or more relations as inputs and give a new relation as a result.**

Select Operation

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

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

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:

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

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

- $\sigma_{A\neq B \wedge C < D}(r)$

A	B	C	D
α	β	5	7

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*

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

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

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

A	B
α	10
α	20
β	30
β	40

- 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

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

A	C
α	1
β	23

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

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

Basic Structure

- SQL is based on set and relational operations with certain modifications and enhancements
- A typical SQL query has the form:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  - A_i - represent attributes
  - r_i - represent relations
  - P - a predicate.
- This query is equivalent to the relational algebra expression.

$$\Pi_{A1, A2, \ldots, An}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

- The result of an SQL query is a relation.

The select Clause

- The **select** clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.
- Find the names of all branches in the *loan* relation

  select *branch-name*
  from *loan*

  In the "pure" relational algebra syntax, the query would be:

  $$\Pi_{\text{branch-name}}(loan)$$

- An asterisk in the select clause denotes "all attributes"

  select *
  from *loan*

- NOTE: SQL does not permit the '-' character in names, so you would use, for example, *branch_name* instead of *branch-name* in a real implementation.
- NOTE: SQL names are case insensitive, meaning you can use upper case or lower case. You may wish to use upper case in places where we use bold font.

The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after **select**.
- Find the names of all branches in the *loan* relations, and remove duplicates

  select distinct *branch-name*
  from *loan*

- The keyword **all** specifies that duplicates not be removed.

  select all *branch-name*
  from *loan*

The select Clause (Cont.)

- The **select** clause can contain arithmetic expressions involving the operation, +, –, *, and /, and operating on constants or attributes of tuples.
- The query:

  select *loan-number, branch-name, amount* * 100
  from *loan*

  would return a relation which is the same as the *loan* relations, except that the attribute *amount* is multiplied by 100.

The where Clause

- The **where** clause corresponds to the selection predicate of the relational algebra. If consists of a predicate involving attributes of the relations that appear in the **from** clause.
- The find all loan number for loans made a the Perryridge branch with loan amounts greater than $1200.

  select *loan-number*
  from *loan*
  where *branch-name* = 'Perryridge' **and** *amount* > 1200

- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- Comparisons can be applied to results of arithmetic expressions.

The where Clause (Cont.)

- SQL Includes a **between** comparison operator in order to simplify **where** clauses that specify that a value be less than or equal to some value and greater than or equal to some other value.
- Find the loan number of those loans with loan amounts between \$90,000 and \$100,000 (that is, ≥\$90,000 and ≤\$100,000)

select *loan-number*
from *loan*
where *amount* **between** 90000 **and** 100000

Set Operations

- The set operations **union, intersect,** and **except** operate on relations and correspond to the relational algebra operations $\cup$, $\cap$, $-$.
- Each of the above operations automatically eliminates duplicates; to retain all duplicates use the corresponding multiset versions **union all, intersect all** and **except all.**
- Suppose a tuple occurs m times in r and n times in s, then, it occurs:
  - $m + n$ times in r **union all** s
  - $\min(m,n)$ times in r **intersect all** s
  - $\max(0, m - n)$ times in r **except all** s

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., ith column of r deals with the same type of values as does the ith column of s)
- E.g. to find all customers with either an account or a loan

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

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

Find all customers who have a loan, an account, or both:

(select *customer-name* **from** *depositor*)
union
(select *customer-name* **from** *borrower)*

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

Compatibility of R1 and R2

- arity(R1) = arity(R2)
- the corresponding attribute domains in R1 and R2 are the same

Tuple arity: the number of values in the sequence (including nulls)

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*:

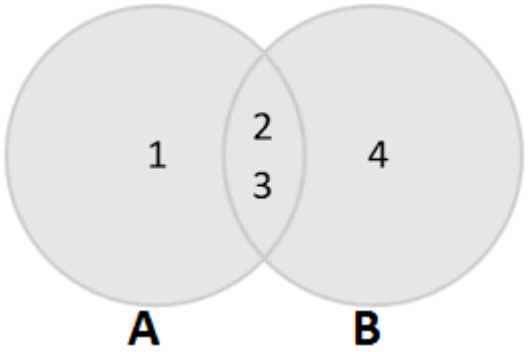
A	*B*
α	1
β	1

Find all customers who have an account but no loan.

(**select** *customer-name* **from** *depositor*)
except
(select *customer-name* **from** *borrower)*

OR

```
select customer_name
from depositor
where customer_name not in
        (select customer_name
          from borrower);
```



Cartesian-Product Operation

- Notation r x s
- Defined as:

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

- 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.

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	19	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

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	19	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

from Clause - Cartesian

- The **from** clause corresponds to the Cartesian product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.
- Find the Cartesian product *borrower x loan*

select *

from *borrower, loan*

- Find the name, loan number and loan amount of all customers having a loan at the Perryridge branch.

select *customer-name, borrower.loan-number, amount*
from *borrower, loan*
where *borrower.loan-number* = *loan.loan-number* **and**
branch-name = 'Perryridge'

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 $A1, A2, \ldots, An$.

The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:

  old-name **as** *new-name*

- Find the name, loan number and loan amount of all customers; rename the column name *loan-number* as *loan-id*.

  select *customer-name, borrower.loan-number* **as** *loan-id, amount*
  from *borrower, loan*
  where *borrower.loan-number* = *loan.loan-number*

Rename - Tuple Variables

- Tuple variables are defined in the **from** clause via the use of the **as** clause.
- Find the customer names and their loan numbers for all customers having a loan at some branch.

  select *customer-name, B.loan-number, L.amount*
  from *borrower* **as** *B, loan* **as** *L*
  where *B.loan-number* = *L.loan-number*

- Find the names of all branches that have greater assets than some branch located in Brooklyn.

  select distinct *T.branch-name*
  from *branch* **as** *T, branch* **as** *S*
  where *T.assets* > *S.assets* **and** *S.branch-city* = 'Brooklyn'

Ordering the Display of Tuples

- List in alphabetic order the names of all customers having a loan in Perryridge branch

  select distinct *customer-name*
  from *borrower, loan*
  where *borrower.loan-number* = *loan.loan-number* **and**
  branch-name = 'Perryridge'
  order by *customer-name*

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - E.g. **order by** *customer-name* **desc**

Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result.
- ***Multiset*** - versions of some of the relational algebra operators, given multiset relations r_1 and r_2:
  1. If there are c_1 copies of tuple t_1 in r_1, and t_1 satisfies selections $\sigma_{\theta,}$, then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$.
  2. For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$, where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1.
  3. If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, there are c_1 x c_2 copies of the tuple $t_1.t_2$ in r_1 x r_2

Duplicates (Cont.)

- Example: Suppose multiset relations r_1 (A, B) and r_2 (C) are as follows:

$$r_1 = \{(1, a)\ (2, a)\} \qquad r_2 = \{(2), (3), (3)\}$$

- Then $\Pi_B(r_1)$ would be $\{(a), (a)\}$, while $\Pi_B(r_1)$ x r_2 would be

$$\{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)\}$$

- SQL duplicate semantics:

select $A_1,_, A_2, \ldots, A_n$
from $r_1, r_2, \ldots, r_m$
where P

is equivalent to the ***multiset*** version of the expression:

$$\Pi_{A1,, A2, \ldots, An}(\sigma_P\ (r_1 \text{ x } r_2 \text{ x } \ldots \text{ x } r_m))$$

Example Queries

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

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

(select *customer-name* **from** *depositor*) **union (select** *customer-name* **from** *borrower)*

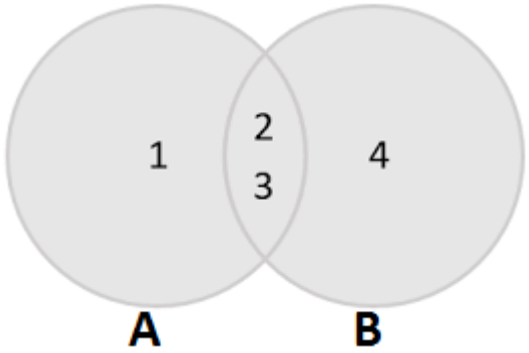
- Find the names of all customers who have a loan and an account at bank.

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

(**select** *customer-name* **from** *depositor*) **intersect (select** *customer-name* **from** *borrower)*

OR

```
select customer_name
from depositor
where customer_name  in
        (select customer_name
         from borrower);
```



THANK YOU

Relational Model

Dr. Odelu Vanga

Indian Institute of Information Technology Sri City

http://www.iiits.ac.in/people/regular-faculty/dr-odelu-vanga/

Additional Operations

- Set intersection
- Natural join
- Division

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

A	B
α	1
α	2
β	3

r

A	B
α	2
β	3

s

$r \cap s$

A	B
α	2
β	3

Natural-Join Operation

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

  $R = (A, B, C, D)$

  $S = (E, B, D)$

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

- $r \bowtie s$ is defined as:

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

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

- $r \bowtie s$ is defined as:

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

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

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	∈

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

- $r \bowtie s$ is defined as:

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

Natural Join - Example Queries

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

- Query :

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

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

Division Operation

- 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

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

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

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

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

Division Operation – Example

Relations *r*, *s*:

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

r

B
1
2

s

$R = \{A, B\}$

$S = \{B\}$

$R\text{-}S = \{A\}$

$r \div s$:

A
α

(α) (α)

(1) (α, 1)
(2) (α, 2)

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

Complete the process ?

Division Operation – Example

Relations *r*, *s*:

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

r

B
1
2

s

$R = \{A, B\}$

$S = \{B\}$

$R\text{-}S = \{A\}$

$r \div s$:

A
α

(β) (β)

(1) (β, 1)
(2) (β, 2)

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

Division Operation – Example

Relations *r*, *s*:

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

r

B
1
2

s

$R = \{A, B\}$

$S = \{B\}$

$R\text{-}S = \{A\}$

$r \div s$:

A
α
β

(β) (β)

(1) (β, 1)
(2) (β, 2)

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

Division Operation – Example

Relations *r*, *s*:

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

r

B
1
2

s

$R = \{A, B\}$

$S = \{B\}$

$R\text{-}S = \{A\}$

$r \div s$:

A
α
β

(γ) (γ)

(1) (γ, 1)

(2) (γ, 2) ✗

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

Division Operation – Example

Relations *r*, *s*:

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

r

B
1
2

s

$R = \{A, B\}$

$S = \{B\}$

$R\text{-}S = \{A\}$

$r \div s$:

A
α
β

(δ) (δ)

(1) (2)

(δ, 1) (δ, 2)

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

Division Operation – Example

Relations *r*, *s*:

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

r

B
1
2

s

$R = \{A, B\}$

$S = \{B\}$

$R\text{-}S = \{A\}$

$r \div s$:

A
α
β

(α, 1) (α, 2) (∈, 1) (∈, 2)

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

Division Operation – Example

Relations *r*, *s*:

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

r

B
1
2

s

$R = \{A, B\}$

$S = \{B\}$

$R\text{-}S = \{A\}$

$r \div s$:

A
α
β

Result

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

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*: **?**

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

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

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

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.

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

THANK YOU

Relational Model

Dr. Odelu Vanga

Indian Institute of Information Technology Sri City

http://www.iiits.ac.in/people/regular-faculty/dr-odelu-vanga/

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 true, then y is true**

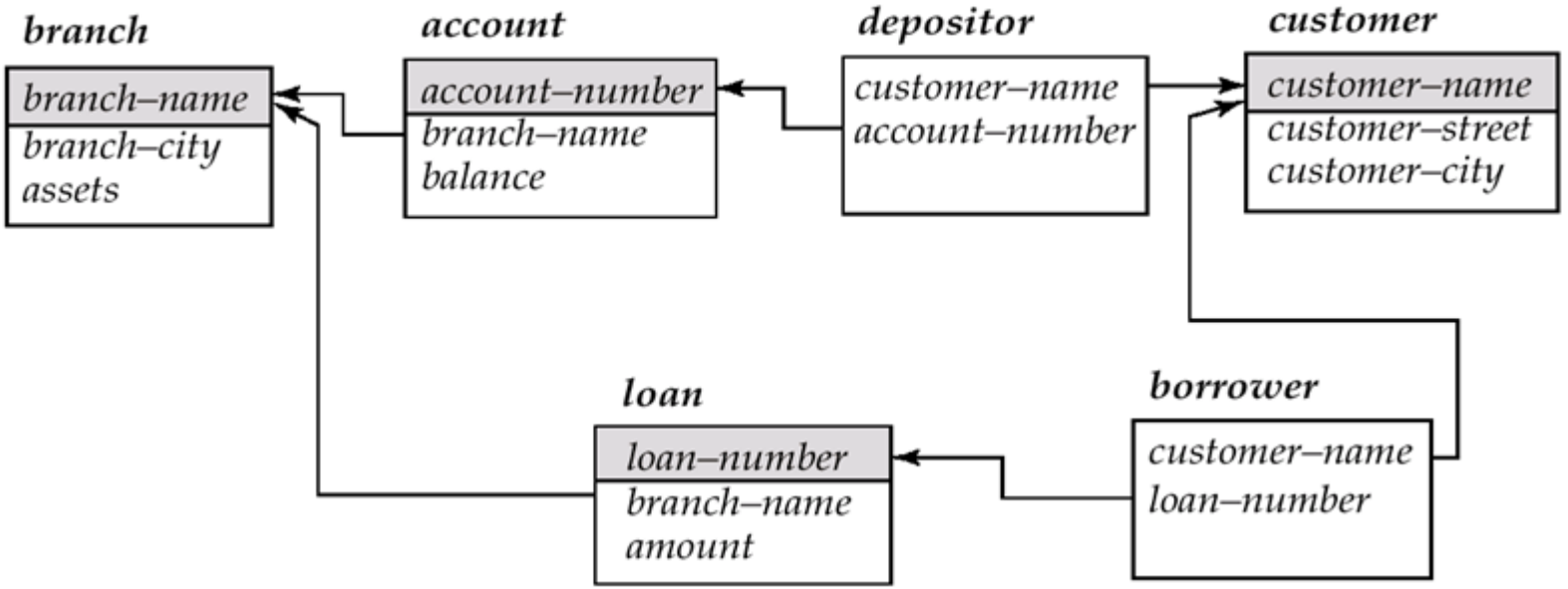
$$x \Rightarrow y \equiv \neg x \text{ v } y$$

for example: **x = 2 implies x + 3 = 5**

5. Set of quantifiers:
   - $\exists\, t \in r\, (Q(t)) \equiv$ "there exists" a tuple t in relation r such that predicate $Q(t)$ is true
   - $\forall t \in r\, (Q(t)) \equiv$ "for all" tuples t in relation r such that predicate $Q(t)$ is true

Banking Example

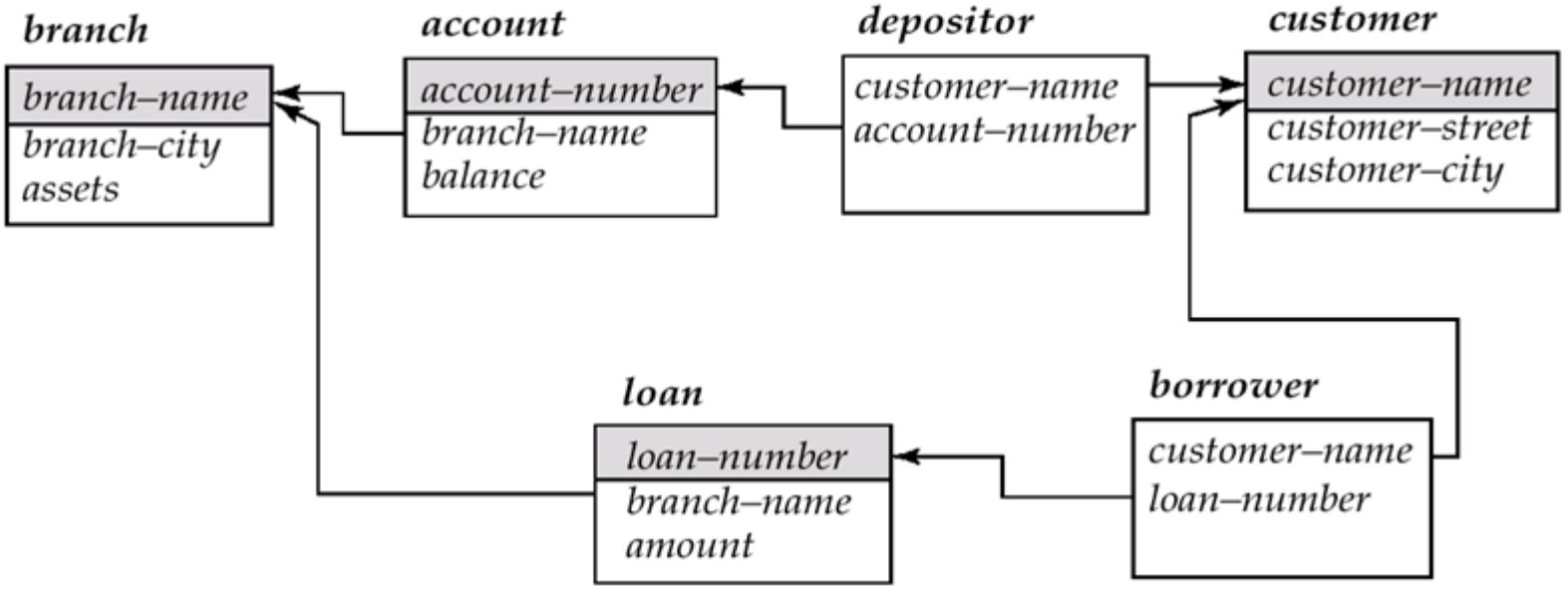
- *branch (branch-name, branch-city, assets)*
- *customer (customer-name, customer-street, customer-city)*
- *account (account-number, branch-name, balance)*
- *loan (loan-number, branch-name, amount)*
- *depositor (customer-name, account-number)*
- *borrower (customer-name, loan-number)*



Example Queries

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

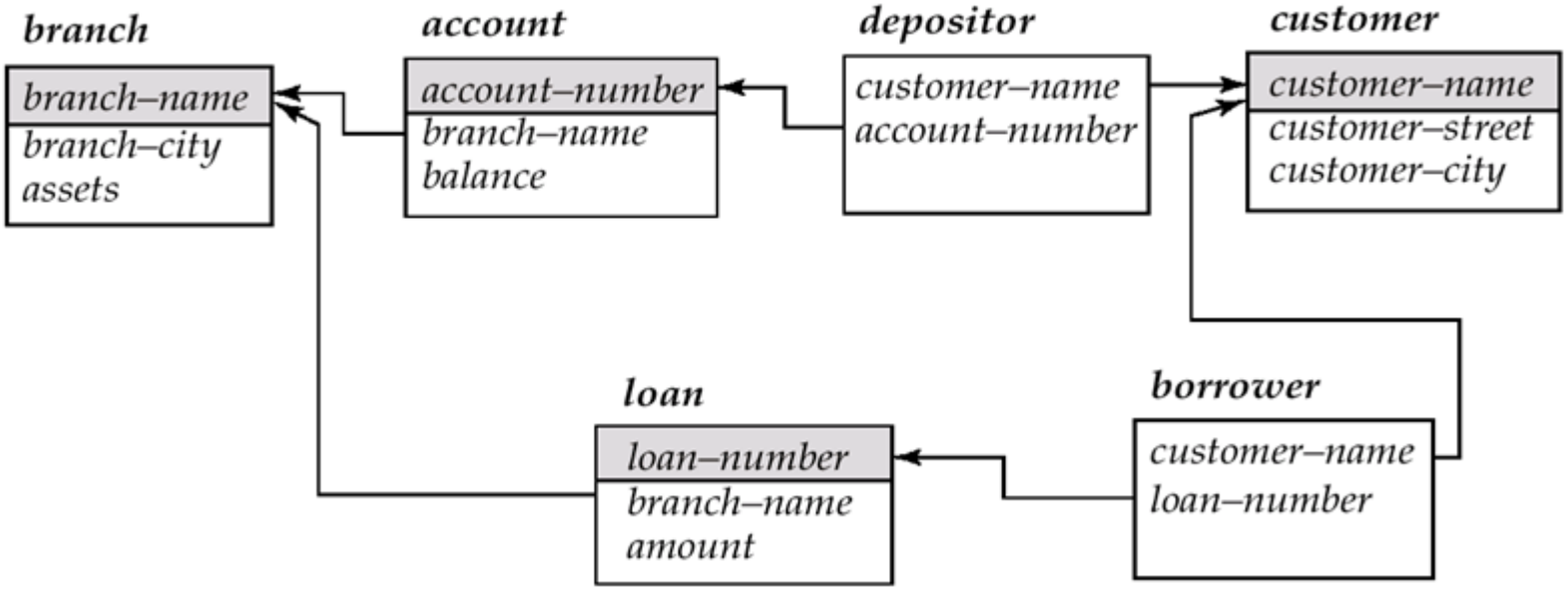
$$\{t \mid t \in loan \wedge t\,[amount] > 1200\}$$



Example Queries

- Find the names of all customers having a loan, an account, or both at the bank

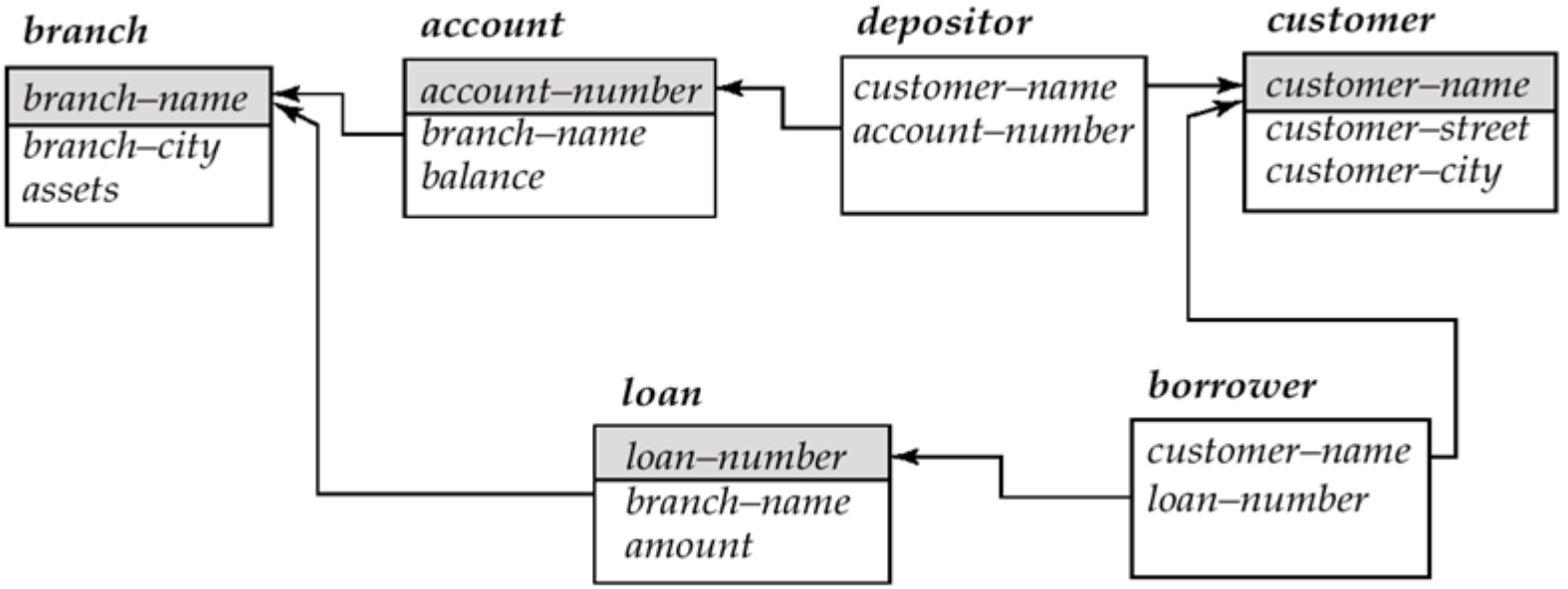
$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name]) \vee \exists u \in depositor(t[customer\text{-}name] = u[customer\text{-}name])$$



Example Queries

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

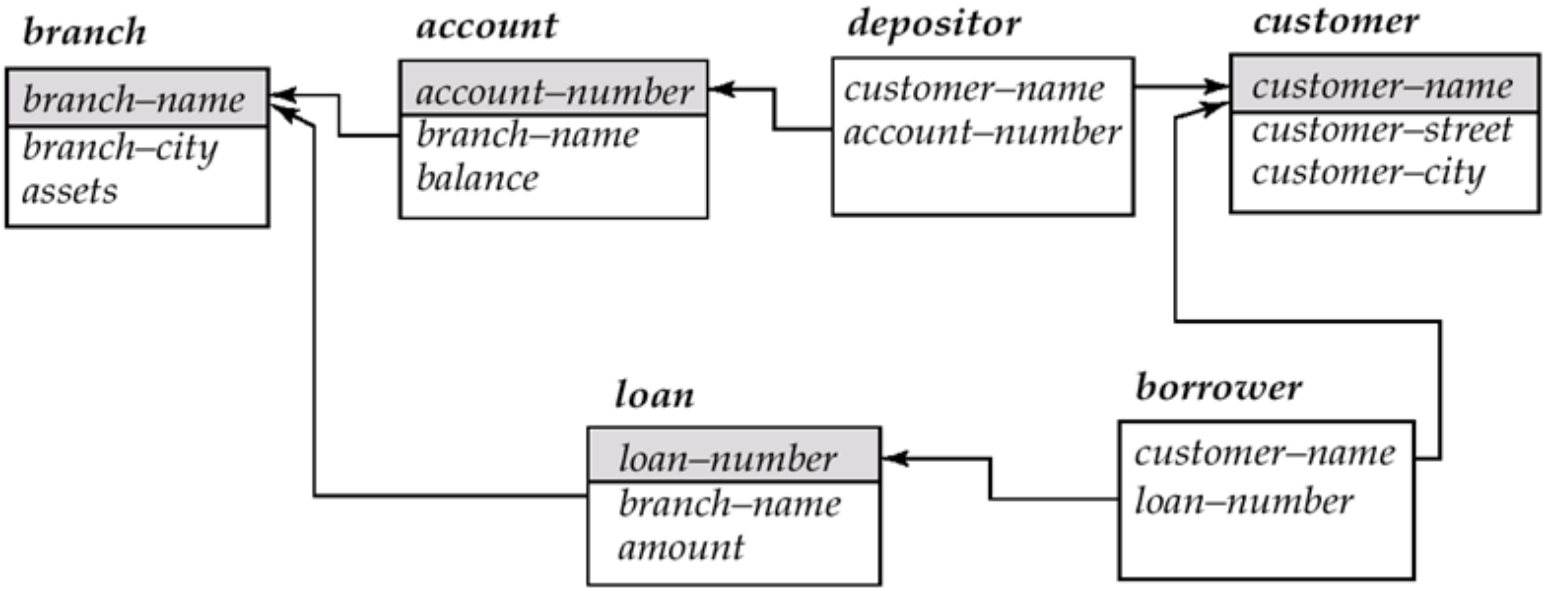
$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name]) \wedge \exists u \in depositor(t[customer\text{-}name] = u[customer\text{-}name])$$



Example Queries

- Find the names of all customers having a loan at the Perryridge branch

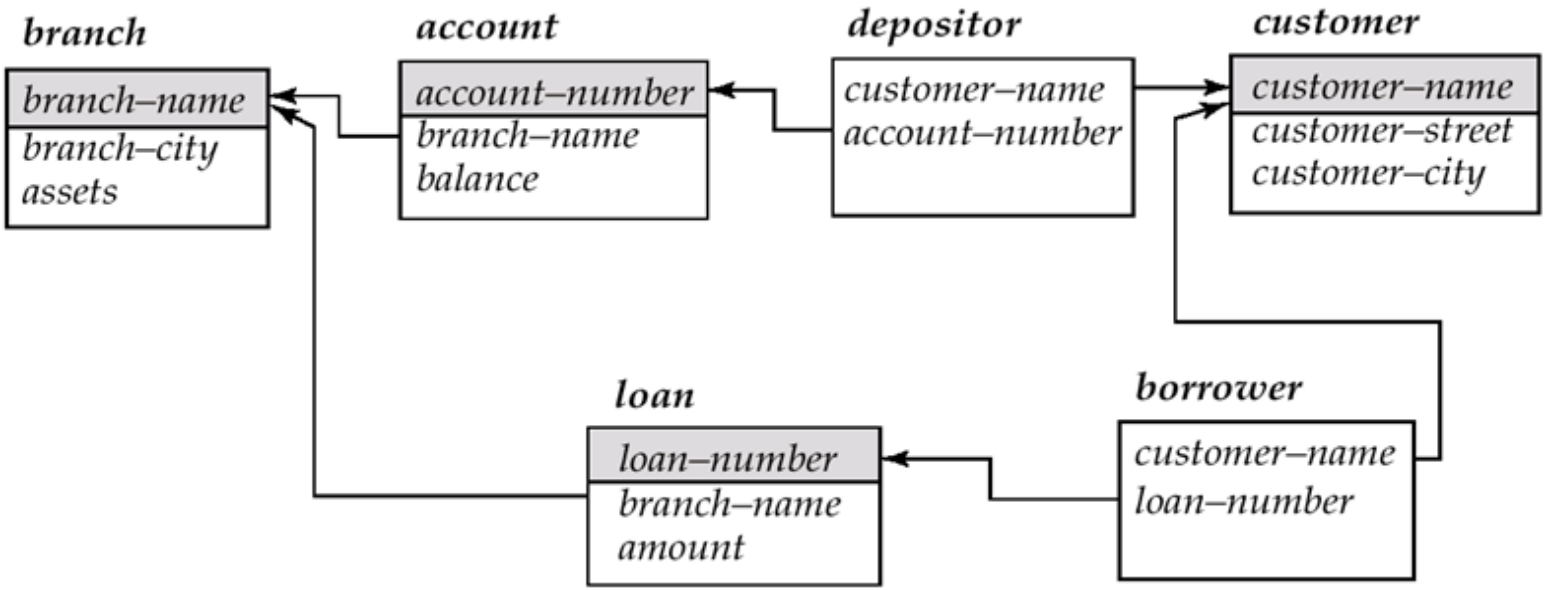
$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name] \wedge \exists u \in loan(u[branch\text{-}name] = \text{“Perryridge”} \wedge u[loan\text{-}number] = s[loan\text{-}number]))\}$$



Example Queries

- Find the names of all customers who have a loan at the Perryridge branch, but no account at any branch of the bank

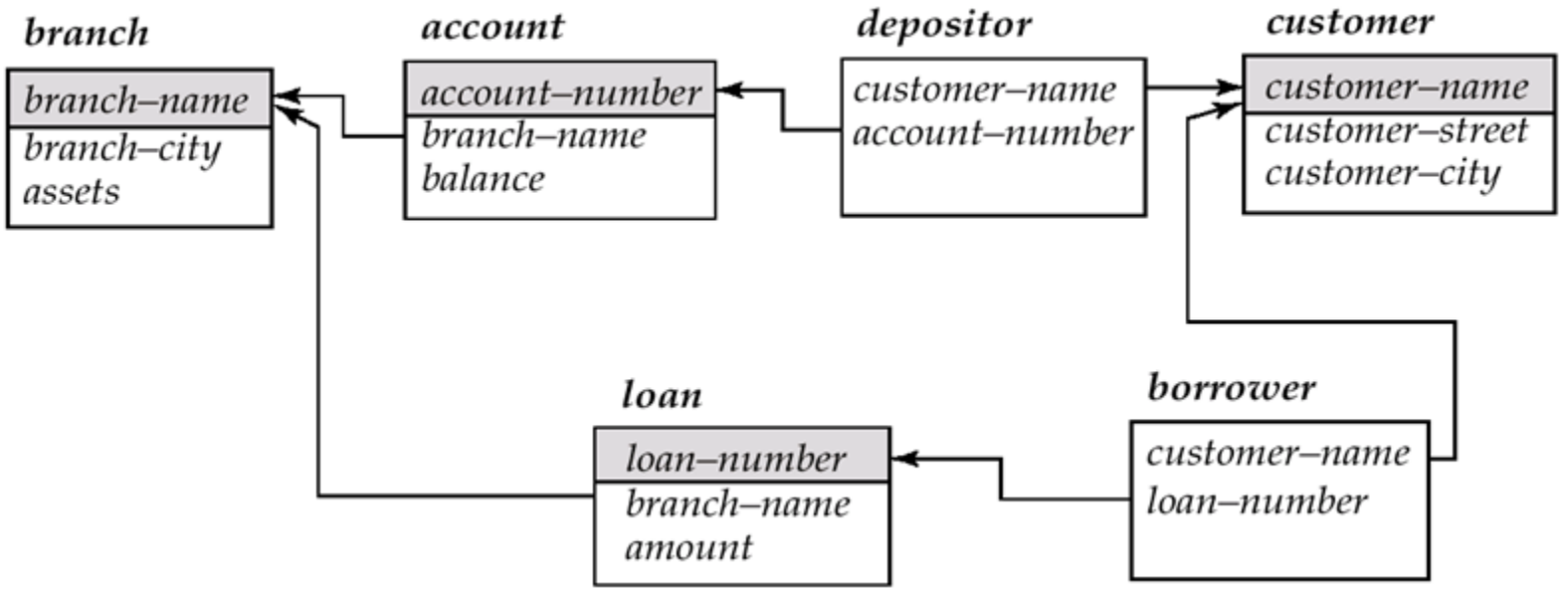
$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[\text{customer-name}] \wedge \exists u \in loan(u[branch\text{-}name] = \text{“Perryridge”} \wedge u[loan\text{-}number] = s[\text{loan-}number])) \wedge \textbf{not } \exists v \in depositor\ (v[customer\text{-}name] = t[\text{customer-name}])\}$$



Example Queries

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$\{t \mid \exists\, c \in \text{customer}\ (t[\text{customer-name}] = c[\text{customer-name}]) \wedge$$
$$\forall\, s \in \textit{branch}(s[\textit{branch-city}] = \text{“Brooklyn”} \Rightarrow$$
$$\exists\, u \in \textit{account}\ (\ s[\textit{branch-name}] = u[\text{branch-name}]$$
$$\wedge \exists\, s \in \textit{depositor}\ (s[\textit{account-number}] = u[\text{account-number}]$$
$$\wedge\ t[\textit{customer-name}] = s[\text{customer-name}]))))\}$$



Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example, $\{t \mid \neg\, t \in r\}$ results in an infinite relation if the domain of any attribute of relation r is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions.
- An expression $\{t \mid P(t)\}$ in the tuple relational calculus is *safe* if every component of t appears in one of the relations, tuples, or constants that appear in P

Domain Relational Calculus

- 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

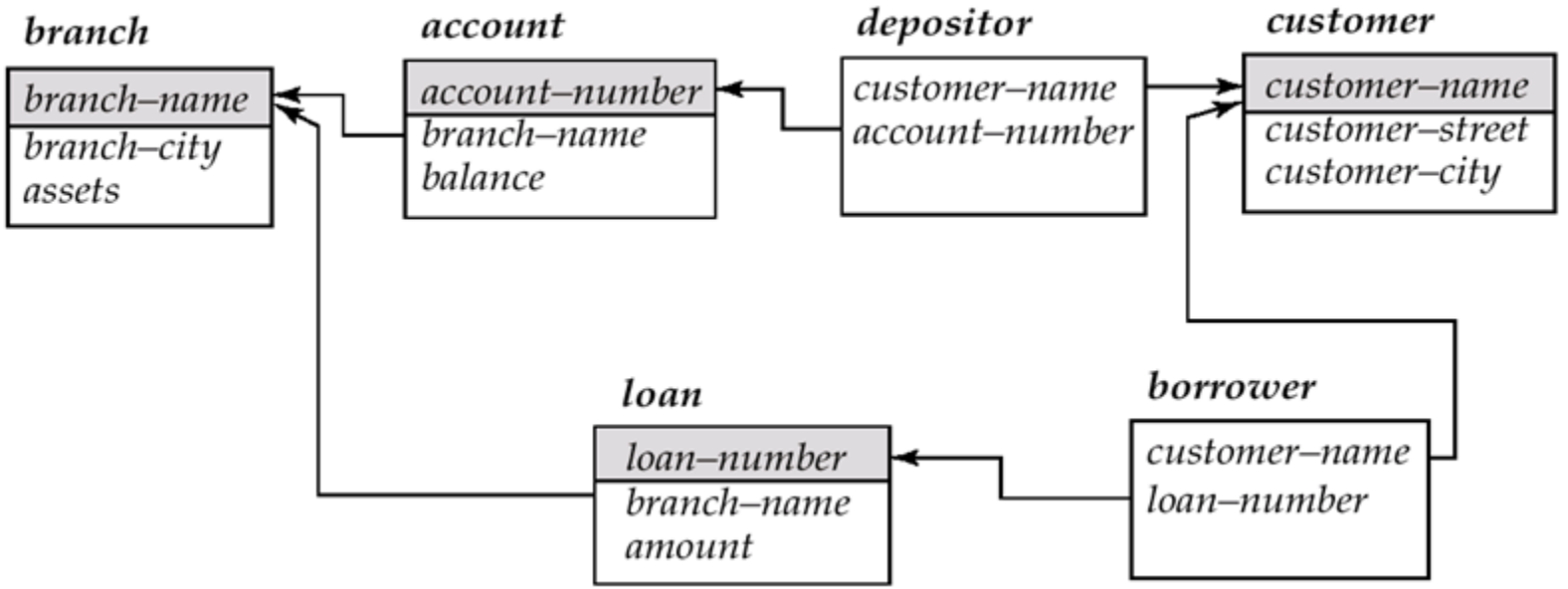
Example Queries

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

  $\{< l, b, a > \mid < l, b, a > \in loan \wedge a > 1200\}$

- Find the names of all customers who have a loan of over $1200

  $\{< c > \mid \exists\, l, b, a\, (< c, l > \in borrower \wedge < l, b, a > \in loan \wedge a > 1200)\}$



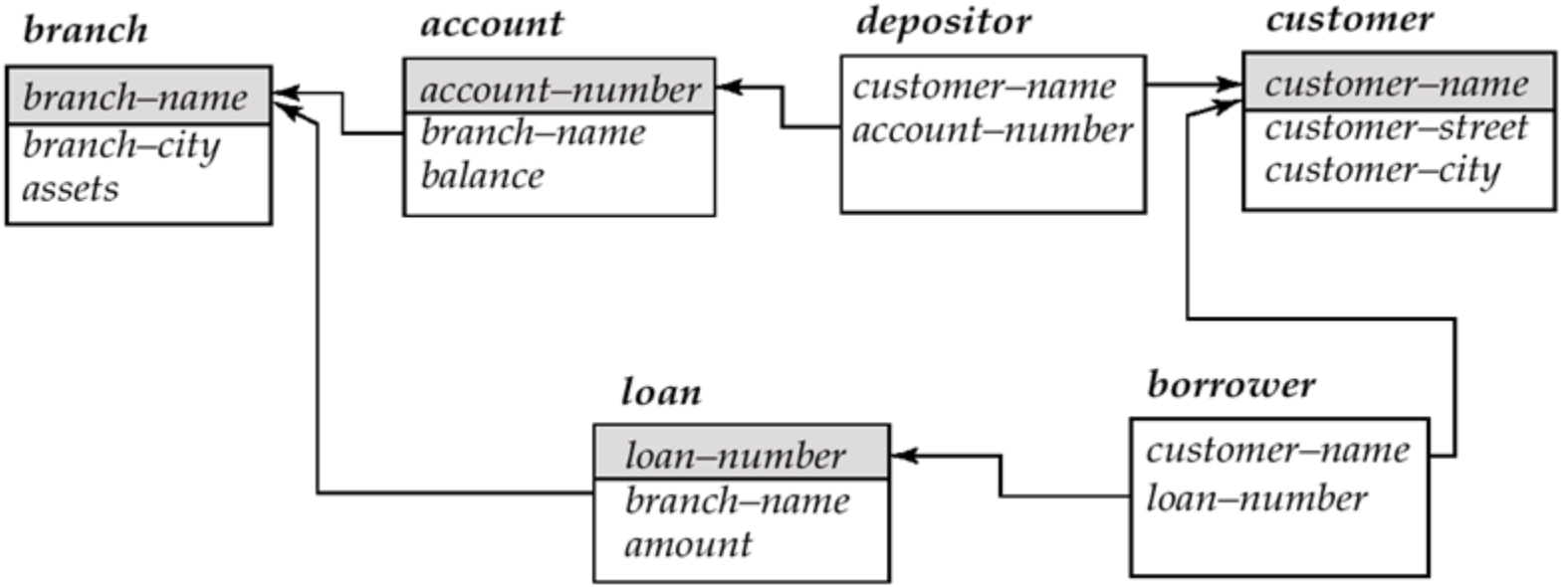
Example Queries

- Find the names of all customers who have a loan from the Perryridge branch and the loan amount:

  $\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge \exists b(< l, b, a > \in loan \wedge b = \text{“Perryridge”}))\}$

  or

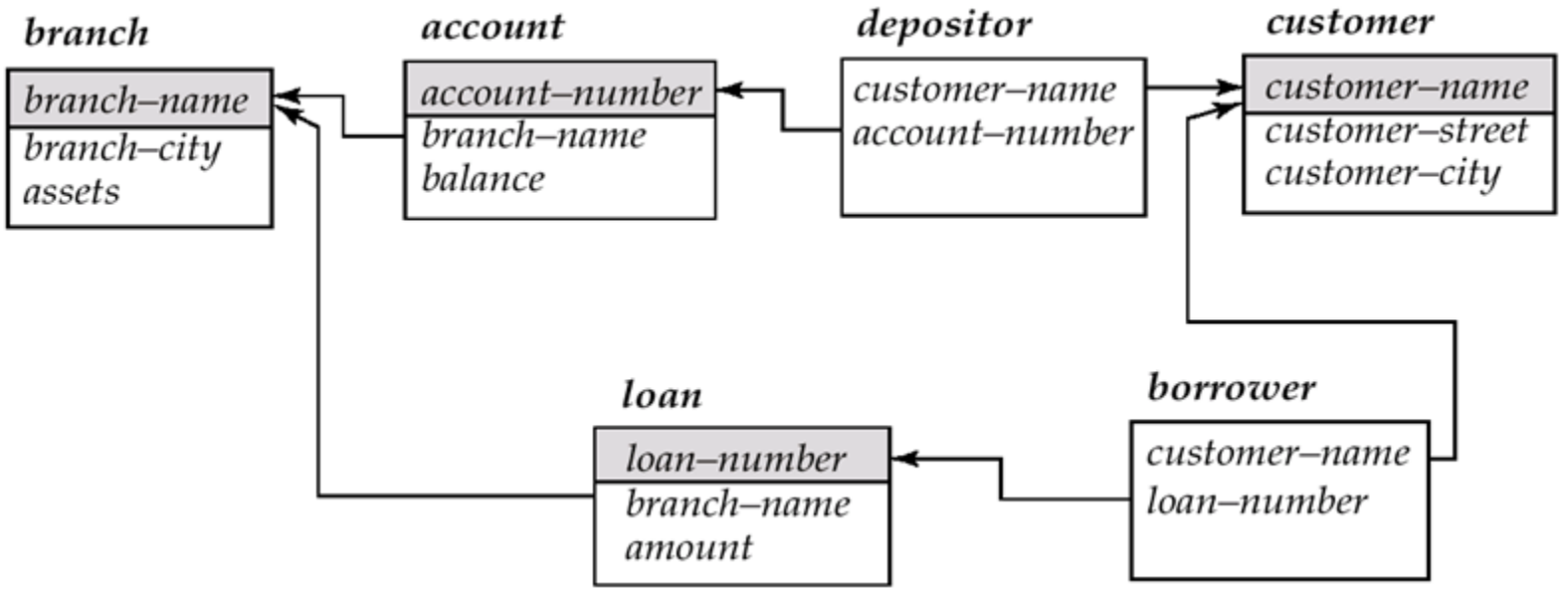
  $\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge < l, \text{“Perryridge”}, a > \in loan)\}$



Example Queries

- Find the names of all customers having a loan, an account, or both at the Perryridge branch:

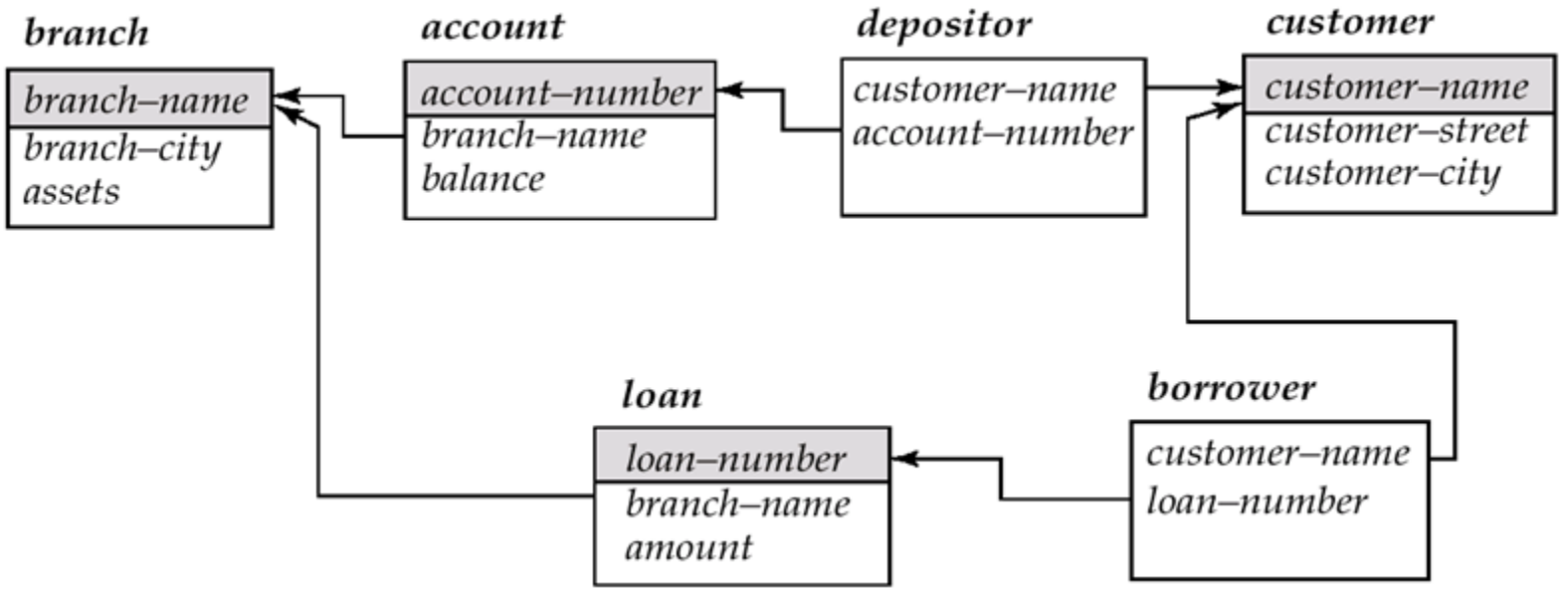
$$\{< c > \mid \exists\, l\, (\{< c, l > \in \textit{borrower} \wedge \exists\, b, a(< l, b, a > \in \textit{loan} \wedge b = \text{“Perryridge”})) \vee \exists\, a(< c, a > \in \textit{depositor} \wedge \exists\, b, n(< a, b, n > \in \textit{account} \wedge b = \text{“Perryridge”}))\}$$



Example Queries

- Find the names of all customers who have an account at all branches located in Brooklyn:

$$\{< c > \mid \exists\, n\, (< c, s, n > \in \text{customer}) \wedge$$
$$\forall\, x,y,z(< x, y, z > \in \textit{branch} \wedge y = \text{“Brooklyn”}) \Rightarrow$$
$$\exists\, a,b\, (< a, x, b > \in \textit{account} \wedge < c,a > \in \textit{depositor})\}$$



Safety of Expressions

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

It is safe if all of the following hold:

1. All values that appear in tuples of the expression are values from *dom*(P) (that is, the values appear either in P or in a tuple of a relation mentioned in P).
2. For every "there exists" sub-formula of the form $\exists\ x\ (P_1(x))$, the sub-formula is true if an only if there is a value of x in *dom*(P_1) such that $P_1(x)$ is true.
3. For every "for all" sub-formula of the form $\forall$x $(P_1\ (x))$, the sub-formula is true if and only if $P_1(x)$ is true for all values x from *dom* (P_1).

THANK YOU

Relational Model

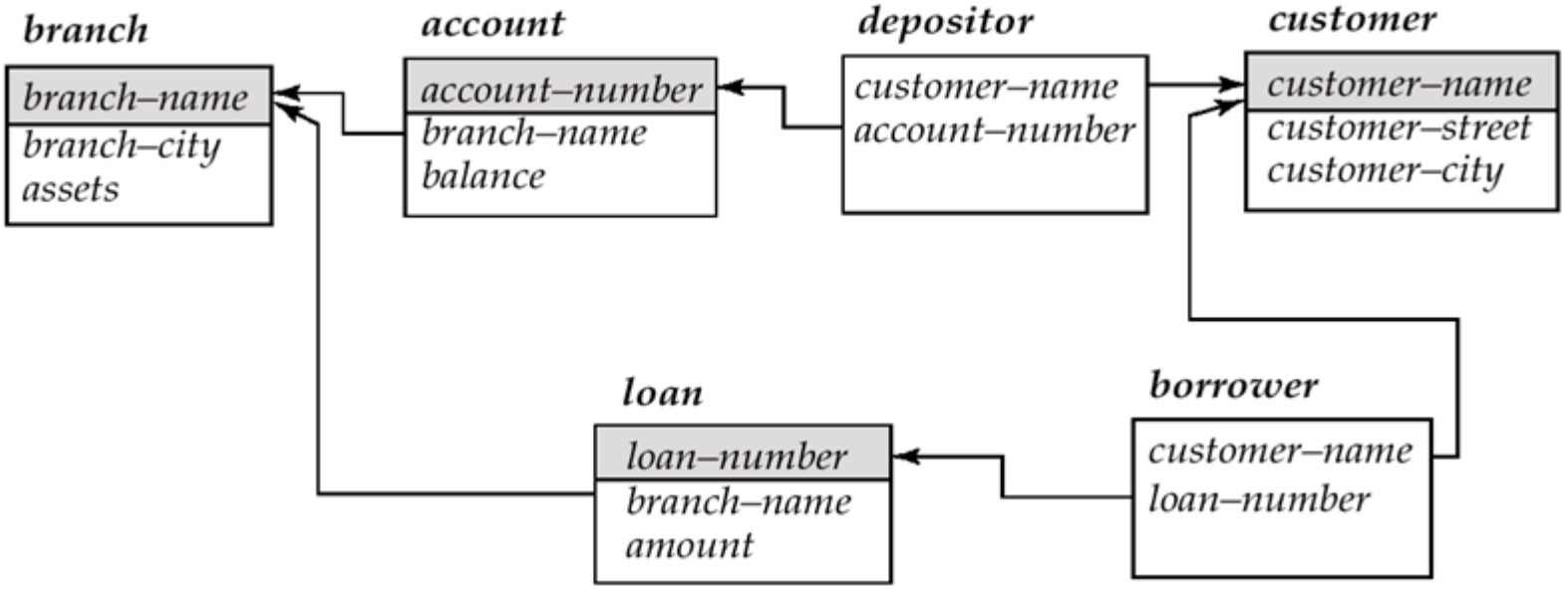
Dr. Odelu Vanga

Indian Institute of Information Technology Sri City

Extended Relational-Algebra-Operations

- Set membership
- Test for Empty Relations
- Generalized Projection
- Outer Join
- Aggregate Functions

set membership



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

  select distinct *customer-name*
  from *borrower*
  where *customer-name* **in (select** *customer-name*
  from depositor)

- Find all customers who have a loan at the bank but do not have an account at the bank

  select distinct *customer-name*
  from *borrower*
  where *customer-name* **not in (select** *customer-name*
  from *depositor)*

Definition of Some Clause

- F <comp> **some** $r \Leftrightarrow \exists\ t \in r\ \ s.t.$ (F <comp> t)
  Where <comp> can be: $<, \leq, >, =, \neq$

(5 < **some** | 0 / 5 / 6 |) = true (read: 5 < some tuple in the relation)

(5 < **some** | 0 / 5 |) = false

(5 = **some** | 0 / 5 |) = true

(5 ≠ **some** | 0 / 5 |) = true (since 0 ≠ 5)

(= **some**) ≡ **in**
However, (≠ **some**) ≢ **not in**

Set Comparison

- Find all branches that have greater assets than some branch located in Brooklyn.

```
select distinct  T.branch-name
from branch as T, branch as S
where  T.assets > S.assets and
          S.branch-city = 'Brooklyn'
```

- Same query using > **some** clause

```
select branch-name
from branch
where assets > some
          (select assets
           from branch
                    where branch-city = 'Brooklyn')
```

Definition of all Clause

- F <comp> **all** $r \Leftrightarrow \forall\ t \in r$ (F <comp> t)

(5< **all** | 0 / 5 / 6) = false

(5< **all** | 6 / 10) = true

(5 = **all** | 4 / 5) = false

(5 ≠ **all** | 4 / 6) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠ **all**) ≡ **not in**

However, (= **all**) ≠ **in**

Example Query

- Find the names of all branches that have greater assets than all branches located in Brooklyn.

```
select branch-name
from branch
where assets > all
    (select assets
    from branch
    where branch-city = 'Brooklyn')
```

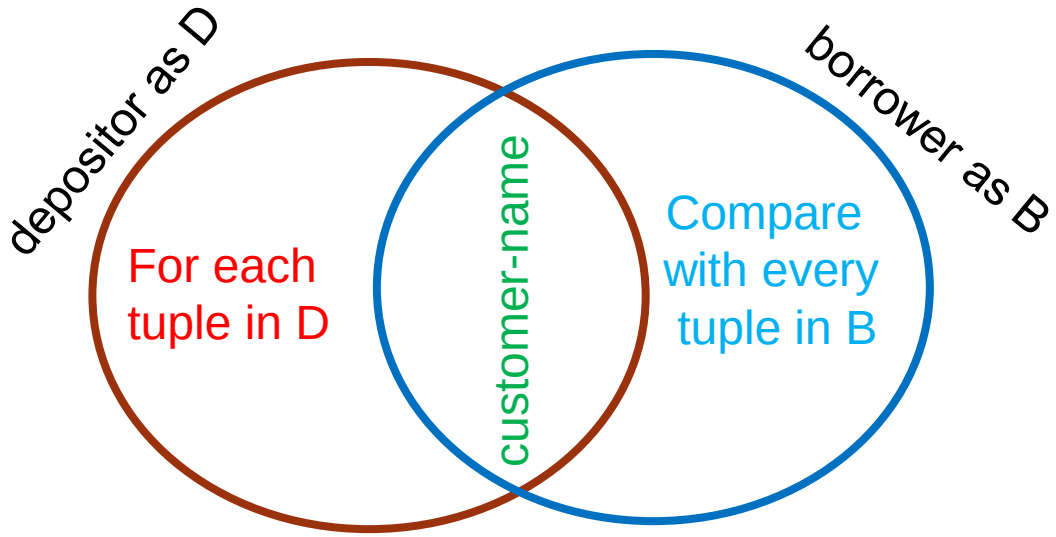
Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

Test for Empty Relations

- Find all customer names who have both a loan and an account.

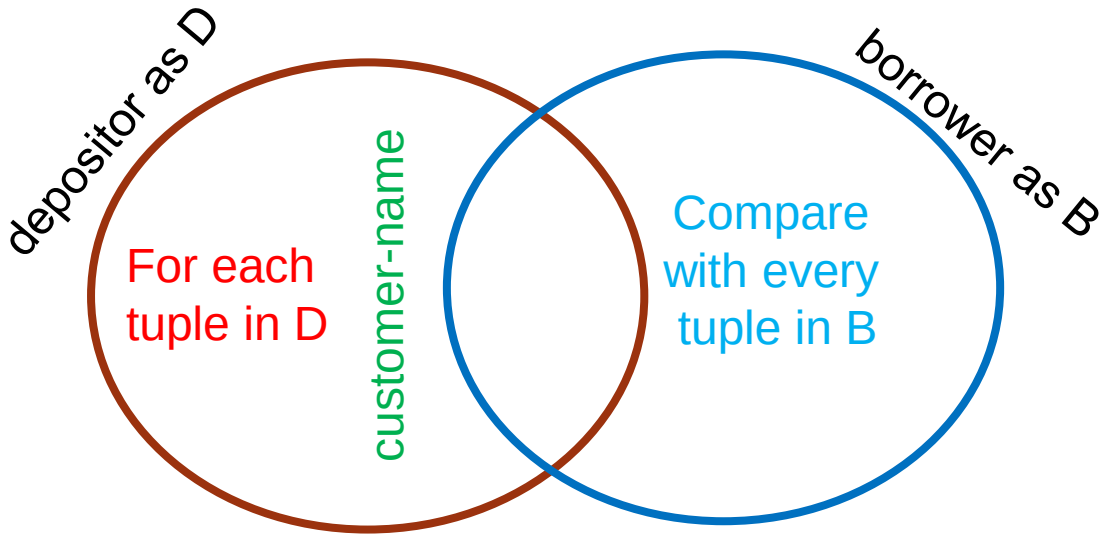
```
select customer-name
from depositor as D
where exists (select *
                from borrower as B
                where D.customer-name = B.customer-name)
```



Test for Empty Relations

- Find all customer names who have an account but no loan.

```
select customer-name
from depositor as D
where not exists (select *
                  from borrower as B
                  where D.customer-name = B.customer-name)
```



Correlated Nested Query

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

```
select distinct S.customer-name
from depositor as S
where not exists (
        (select branch-name
        from branch
        where branch-city = 'Brooklyn')
       except
        (select R.branch-name
        from depositor as T, account as R
        where T.account-number = R.account-number and
              S.customer-name = T.customer-name))
```

For each customer S, check

All branches at Brooklyn

Branches where customer S has an account

- (Schema used in this example)
- **Note that $X - Y = \emptyset \Leftrightarrow X \subseteq Y$**
- *Note:* Cannot write this query using = **all** and its variants

Example Query

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

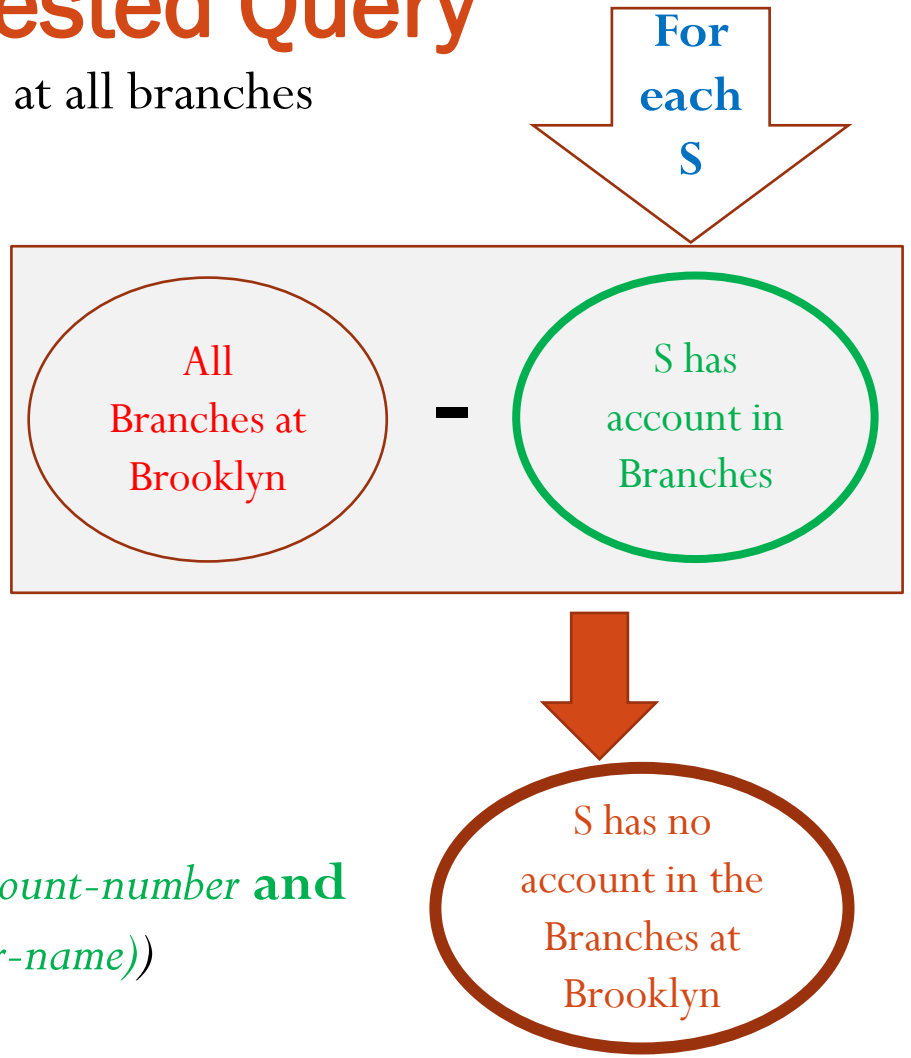
```
select distinct S.customer-name
from depositor as S
where not exists (
        (select branch-name
        from branch
        where branch-city = 'Brooklyn')
       except
        (select R.branch-name
        from depositor as T, account as R
        where T.account-number = R.account-number and
              S.customer-name = T.customer-name))
```

- (Schema used in this example)
- **Note that $X - Y = \emptyset \Leftrightarrow X \subseteq Y$**
- *Note:* Cannot write this query using = **all** and its variants

Correlated Nested Query

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

```
select distinct S.customer-name
from depositor as S
where not exists (
    (select branch-name
     from branch
     where branch-city = 'Brooklyn')
     except
     (select R.branch-name
      from depositor as T, account as R
       where T.account-number = R.account-number and
             S.customer-name = T.customer-name))
```

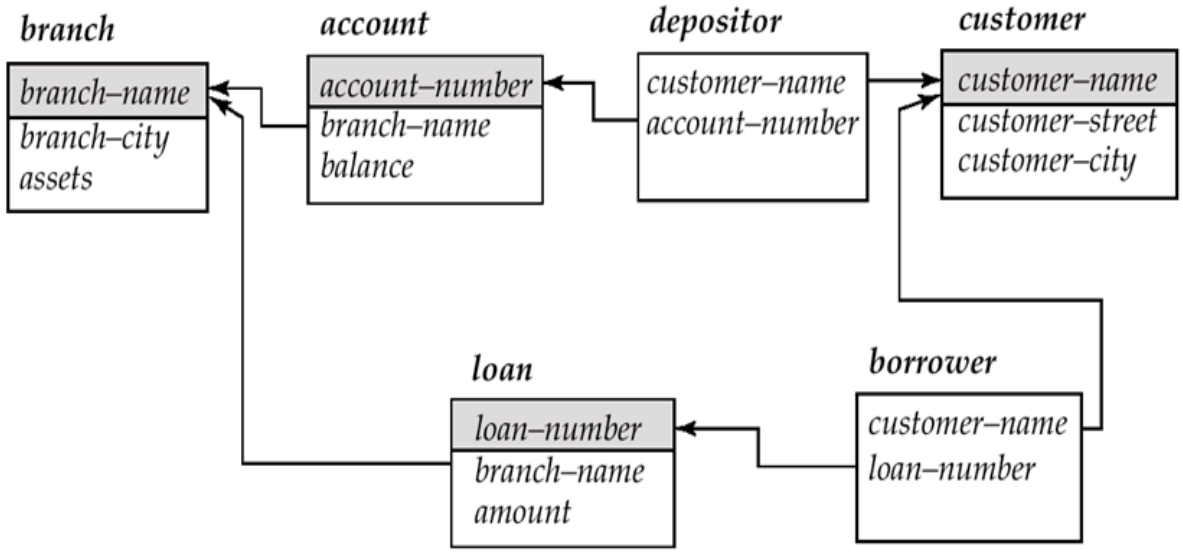


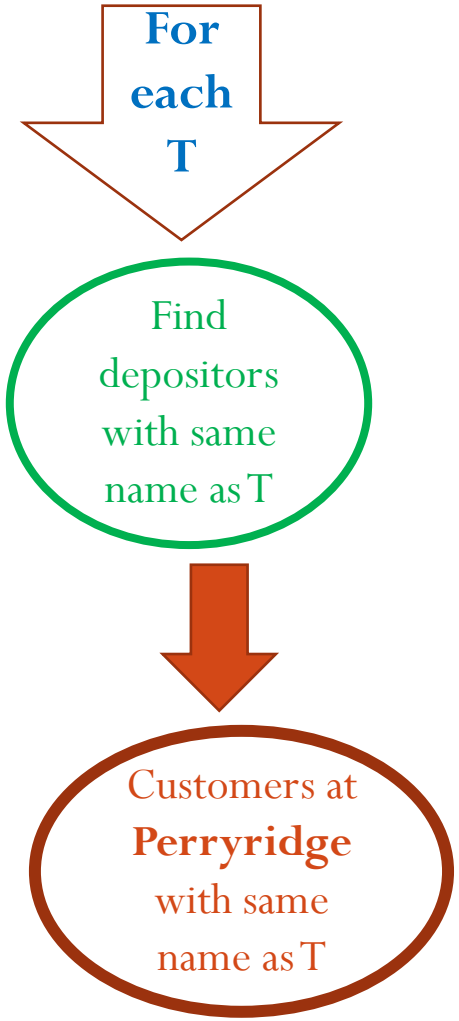
- Note that $X - Y = \emptyset \Leftrightarrow X \subseteq Y$
- *Note:* Cannot write this query using = **all** and its variants

Test for Absence of Duplicate Tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
- Find all customers who have at most one account at the Perryridge branch.

```
select T.customer-name
from depositor as T
where unique (
        select R.customer-name
        from account, depositor as R
        where T.customer-name = R.customer-name and
                R.account-number = account.account-number and
                account.branch-name = 'Perryridge')
```

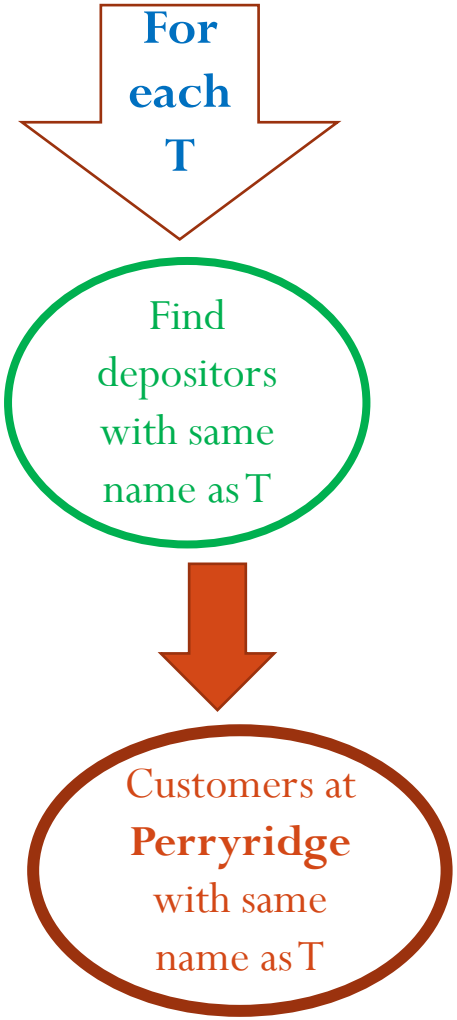




Example Query

- Find all customers who have at least two accounts at the Perryridge branch.

```
select distinct T.customer-name
from depositor as T
where not unique (
        select R.customer-name
        from account, depositor as R
        where T.customer-name = R.customer-name and
              R.account-number = account.account-number and
              account.branch-name = 'Perryridge')
```



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.

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

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

  - E is any relational-algebra expression
  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (it 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

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

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

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

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

Aggregate Functions (Cont.)

For convenience, we permit renaming as part of aggregate operation

$$\textit{branch-name}\ \mathcal{G}\ \textbf{\textit{sum}}(\textit{balance})\ \textbf{\textit{as}}\ \textit{sum-balance}\ (\textit{account})$$

Examples - Joined Relations

- 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

Note: borrower information missing for L-260

and loan information missing for L-155

Inner 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

- **Inner Join**

  loan $\bowtie$ *Borrower*

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

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

- **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

- 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

- **Right Outer Join**

$loan \mathbin{⟖} borrower$

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

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

- **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

CPQ-02

Joined Relations

- Join operations take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- Join condition – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- Join type – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join Types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using (A_1, A_2, ..., A_n)

Joined Relations – Datasets for Examples

- 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

Note: borrower information missing for L-260
and loan information missing for L-155

Joined Relations – Examples

loan **inner join** *borrower* **on**
loan.loan-number = borrower.loan-number

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

loan **left inner join** *borrower* **on**
loan.loan-number = borrower.loan-number

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

Joined Relations – Examples

loan **natural inner join** *borrower*

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

loan **natural right outer join** *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	null	null	Hayes

Joined Relations – Examples

loan **full outer join** *borrower* **using** *(loan-number)*

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

Find all customers who have either an account or a loan (but not both) at the bank.

select *customer-name*
from (*depositor* **natural full outer join** *borrower*)
where *account-number* **is** *null* **or** *loan-number* **is** *null*

Null Values

- *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
- 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*
  - 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*

THANK YOU