[DBP 10-2] 9th May 2023

Database

(Database Design – Conceptual & Logical)

Spring, 2023

Jaewook Byun

Ph.D., Assistant Professor, Department of Software, Sejong University

jwbyun@sejong.ac.kr

https://sites.google.com/view/jack-dfpl/home

https://www.youtube.com/channel/UC988e-Y8nto0LXVae0aqaOQ

Schedule

Week	Title		
1	Course Overview and Environment Setting	Environment Setting and Revisiting Java	-
2	Introduction to Database		CH1
3	Relational Model	Relational Algebra	CH2
4	Database Language: SQL (DDL, DML, DQL, DCL)		CH3-5
5	Database Language: SQL (DDL, DML, DQL, DCL)		CH3-5
6	Database Language: SQL (DDL, DML, DQL, DCL)		CH3-5
7	Database Language: SQL (DDL, DML, DQL, DCL)		CH3-5
8	Midterm Examination		
9	Physical Database Design: Indexing		CH12
10	Physical Database Design: Indexing	Conceptual Database Design – E-R Data Model	CH12, CH6
11	Conceptual Database Design – E-R Data Model		CH6
12	Logical Database Design 1 – Schema Mapping		CH6
13	Logical Database Design 2 – Normalization		CH7
14	Query Processing and Optimization, or View (TBD)		CH13-14, CH3-5
15	Final Examination		

Subject to change

Calendar

- 수업일수는 요일별 15주 이상이며 수업 결손이 발생하지 않도록 진행

요 일 별 수업일수	월	화	수	목	금
	16일	15일	16일	16일	15일

나. 수업주차 : 한 주차는 목요일부터 수요일까지 임

수업주차	기간	수업주차	기간
1주차	03.02.(목) ~ 03.08.(수)	9주차	04.27.(목) ~ 05.03.(수)
2주차	03.09.(목) ~ 03.15.(수)	10주차	05.04.(목) ~ 05.10.(수)
3주차	03.16.(목) ~ 03.22.(수)	11주차	05.11.(목) ~ 05.17.(수)
4주차	03.23.(목) ~ 03.29.(수)	12주차	05.18.(목) ~ 05.24.(수)
5주차	03.30.(목) ~ 04.05.(수)	13주차	05.25.(목) ~ 05.31.(수)
6주차	04.06.(목) ~ 04.12.(수)	14주차	06.01.(목) ~ 06.07.(수)
7주차	04.13.(목) ~ 04.19.(수)	15주차	06.08.(목) ~ 06.14.(수)
8주차 (중간고사)	04.20.(목) ~ 04.26.(수)	16주차 (기말고사)	06.15.(목) ~ 06.21.(수)

Calendar

2023년 3월. March

	S	M	T	W	T	F	S
1				1	2	3	4
2	5	6	7	8	9	10	11
3	12	13	14	15	16	17	18
4	19	20	21	22	23	24	25
5	26	27	28	29	30	31	

- 2(목) 1학기 개강
- 3(금) - 8(수) 수강신청 과목 확인 및 변경
- 6(월) - 15(수) 교직신청
- 24(금) - 28(화) 수강신청과목 철회

2023년 4월. April

	S	M	T	W	T	F	S
5							1
6	2	3	4	5	6	7	8
7	9	10	11	12	13	14	15
8중간	16	17	18	19	20	21	22
9	23	24	25	26	27	28	29
10	30						

- 20(목) - 26(수) 1학기 중간고사
- 27(목) - 5.1(월) 1학기 중간고사 성적 입력

IEEE ICDE 2023, Anaheim, California, US

Calendar

2023년 5월. May

	S	M	T	W	T	F	S
10		1	2	3	4	5	6
11	7	8	9	10	11	12	13
12	14	15	16	17	18	19	20
13	21	22	23	24	25	26	27
14	28	29	30	31			

- 2(화) - 7(일) 1학기 중간고사 성적 열람 및 정정
- 4(목) - 30(화) 복수·부전공, 연계융합전공 신청
- 5(금) 창립 83주년 기념휴일 (창립일 : 1940. 5. 20)
- 29(월) - 31(수) 하계 계절학기 수강신청

Calendar

2023년 6월. June

	S	M	T	W	T	F	S
14					1	2	3
	4	5	6	7	8	9	10
15기말	11	12	13	14	15	16	17
	18	19	20	21	22	23	24
	25	26	27	28	29	30	

▪ 9(금) - 26(월)	1학기 강의평가
▪ 15(목) - 21(수)	1학기 기말고사 및 수업결손 보충
▪ 22(목) - 26(월)	1학기 기말고사 성적 입력
▪ 22(목)	하계방학 시작 및 계절학기 개강
▪ 27(화) - 7.3(월)	1학기 기말고사 성적 열람 및 정정

Calendar

2023년 7월. July

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

- 4(화) - 5(수)
- 24(월) - 30(일)

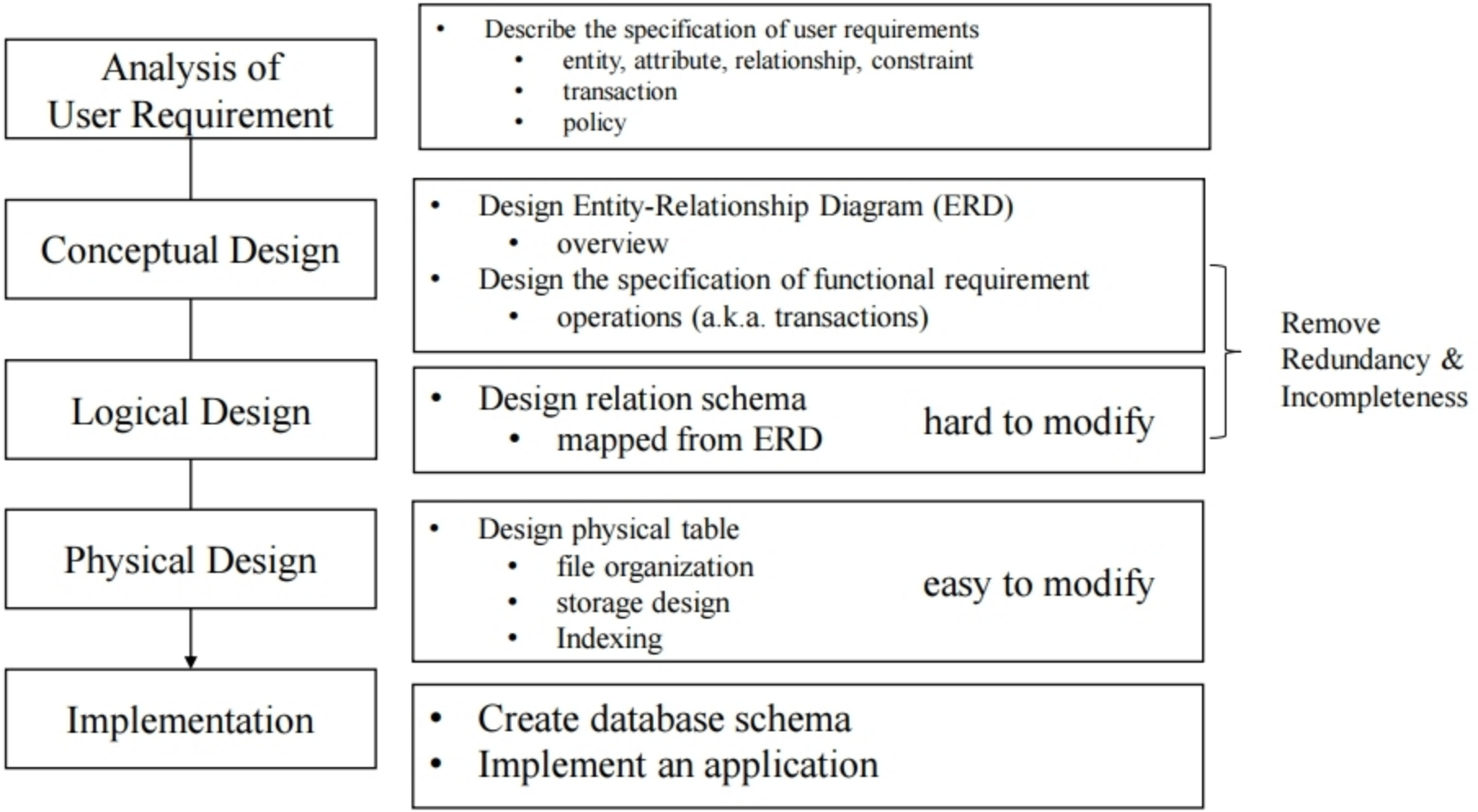
1학기 기말고사 성적마감

2학기 복학, 휴학 신청

Table of Contents

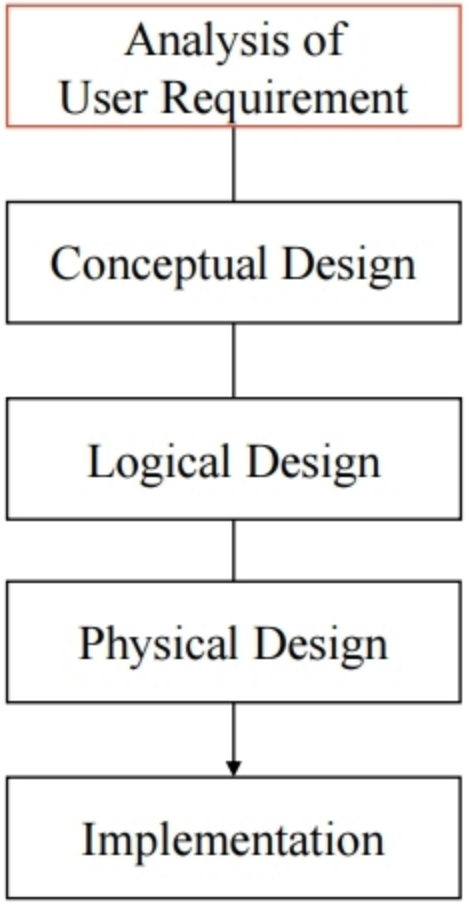
Database Design

- Lifecycle



Database Design

- Example: Lifecycle



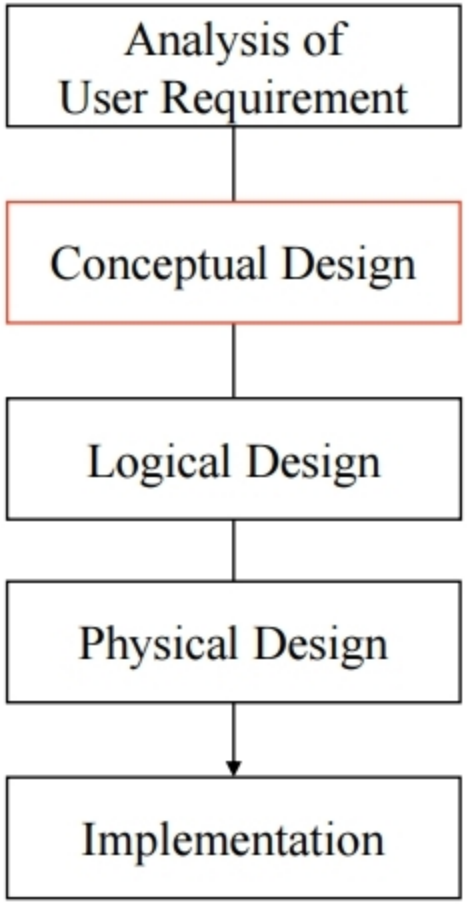
The specification of user requirements

- Bank consists of multiple branch
- A branch is in a specific city
- A branch is identified by its name
- A customer has its unique identifier, customer_id
- Bank knows name, city, address of each customer
- A customer may have an account or a loan
- A customer has his/her loan officer and personal banker

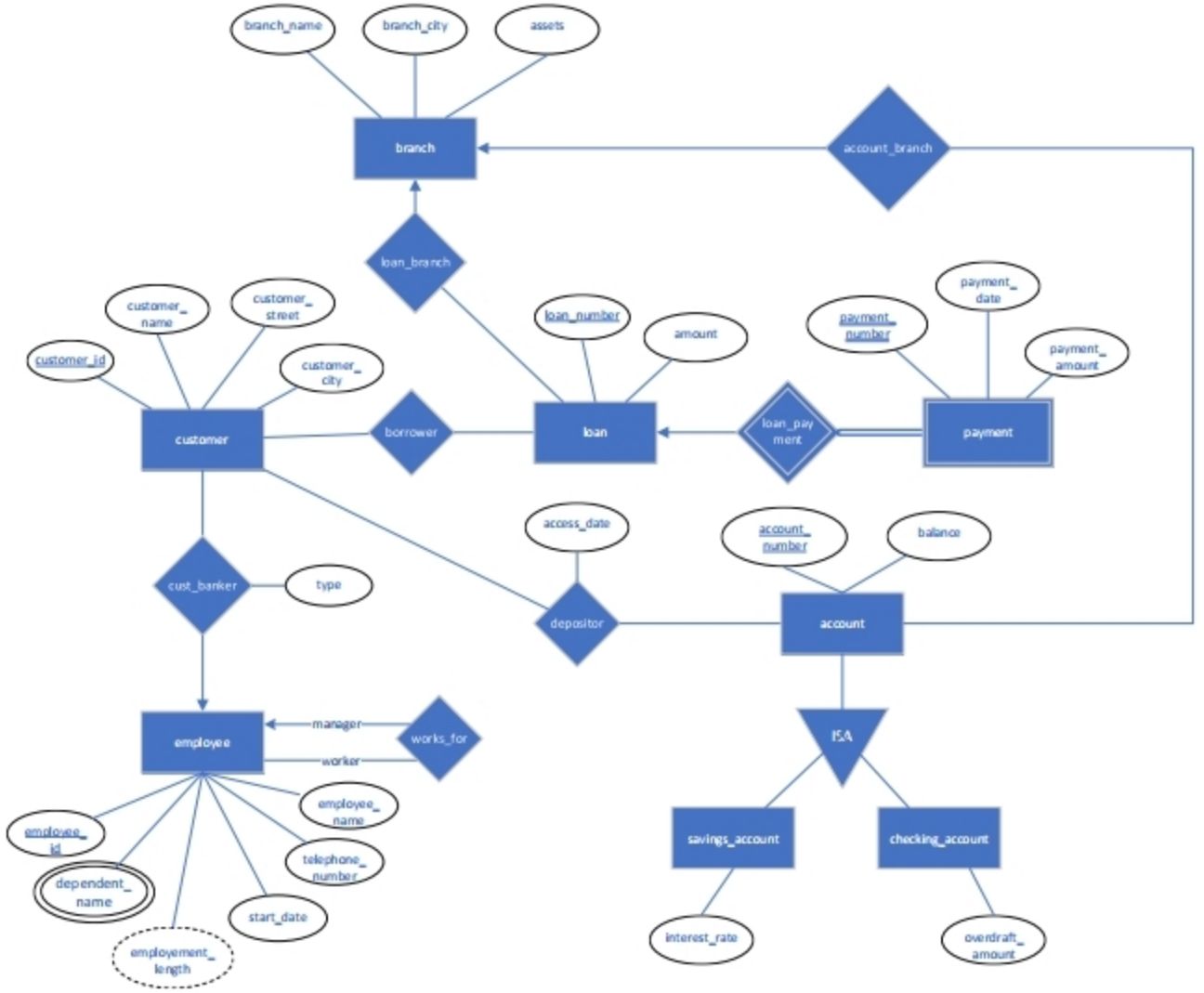
…

Database Design

- Example: Lifecycle

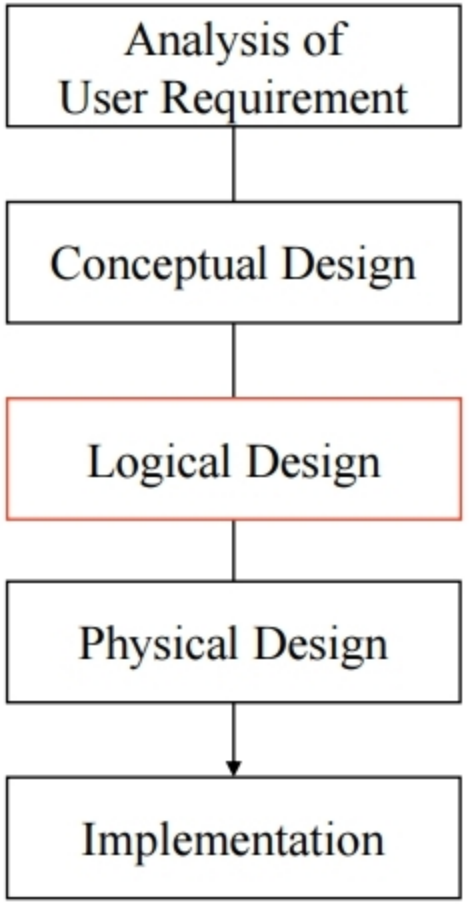


[E-R Diagram of bank organization]



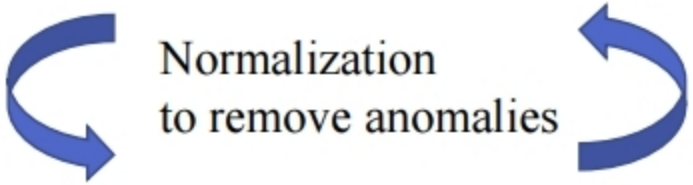
Database Design

- Example: Lifecycle



[Relation schema of bank organization]

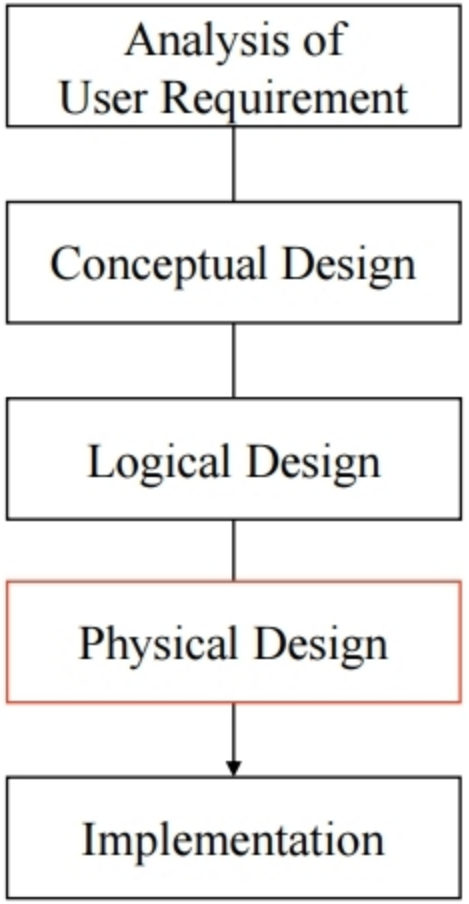
branch = (branch_name, branch_city, assets)
customer = (customer_id, customer_name, customer_street, customer_city)
loan = (loan_number, amount)
account = (account_number, balance)
employee = (employee_id, employee_name, telephone_number, start_date)
dependent_name = (employee_id, dname)
account_branch = (account_number, branch_name)
loan_branch = (loan_number, branch_name)
borrower = (customer_id, loan_number)
depositor = (customer_id, account_number)
cust_banker = (customer_id, employee_id, type)
works_for = (worker_employee_id, manager_employee_id)
payment = (loan_number, payment_number, payment_date, payment_amount)
savings_account = (account_number, interest_rate)
checking_account = (account_number, overdraft_amount)



Database Design

- Example: Lifecycle

loan = (loan_number, amount)



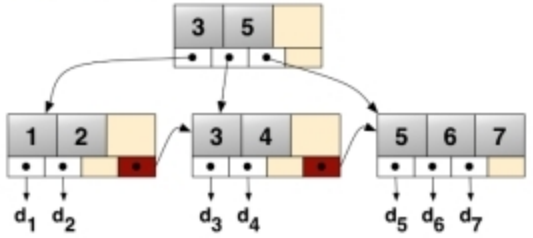
Data type
Value distribution
Service
Access rate
Indexing

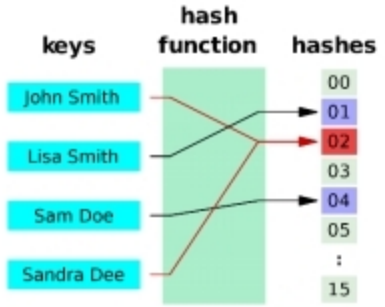
→

Low response time
Low storage requirement
High throughput

SELECT * FROM loan WHERE loan_number = 'L-11';

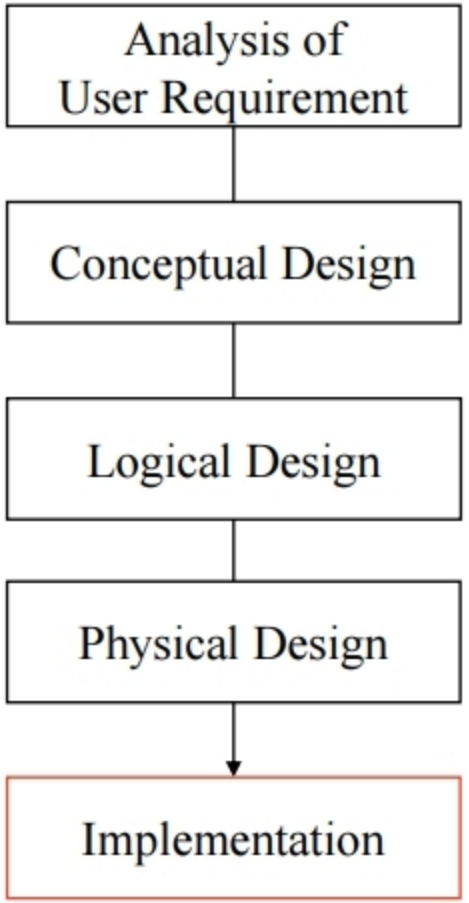
VARCHAR(50) vs. ENUM
B+Tree vs. Hash Index





Database Design

- Example: Lifecycle



loan = (loan_number, amount)

```
CREATE TABLE loan (
        loan_number VARCHAR(50) PRIMARY KEY,
        amount INTEGER
)
```

```
CREATE INDEX ln_idx ON loan (loan_number);
```

```
INSERT INTO loan VALUES ('L-11', 900);
```

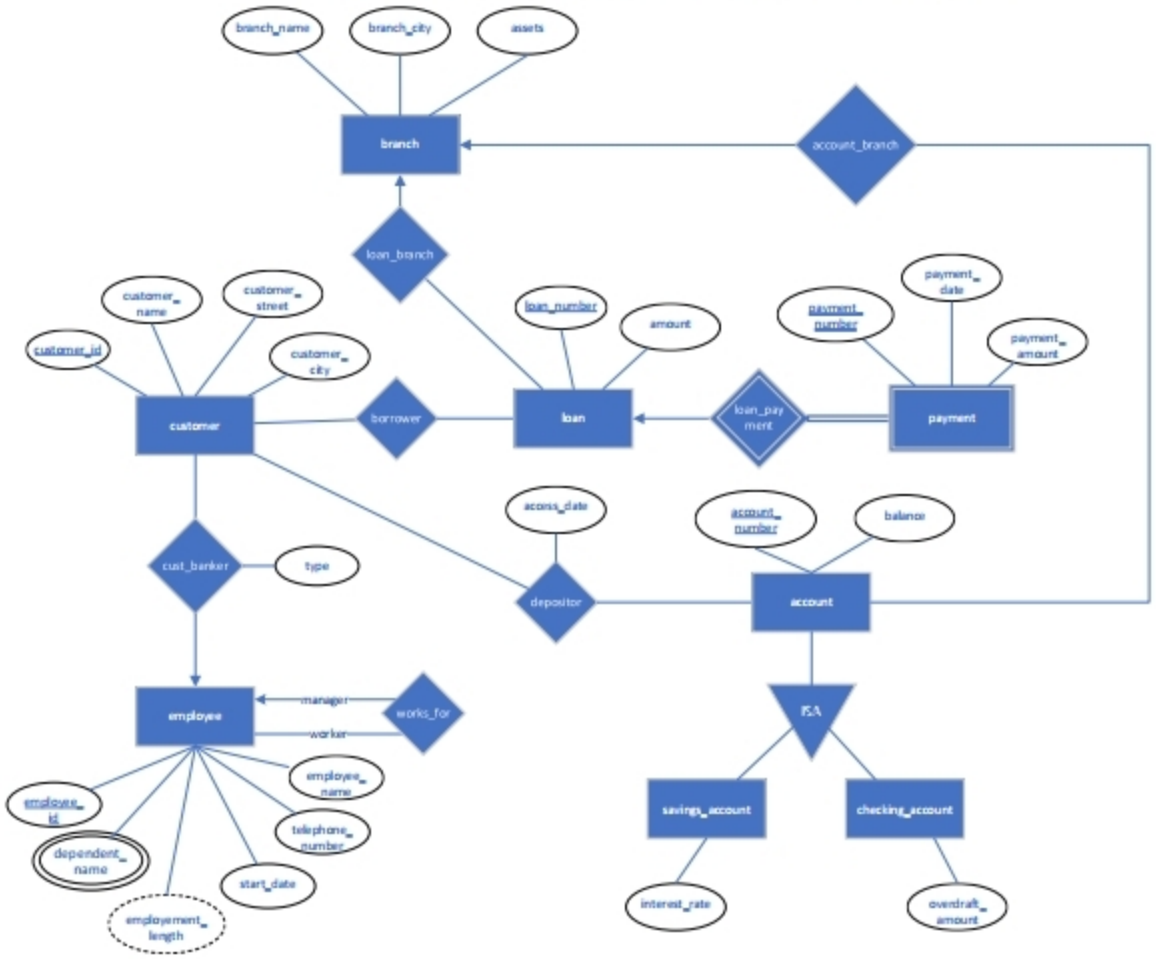
```
SELECT * FROM loan WHERE loan_number = 'L-11';
```

with Transaction processing in an application

Conceptual Database Design

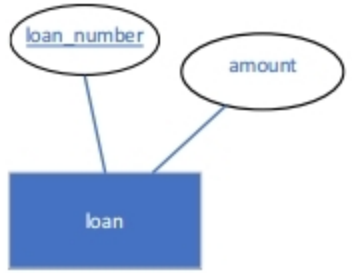
- Entity-Relationship (E-R) data model
  - As one of semantic data model, E-R data model represents a semantic of data
  - defines enterprise schema
  - consists of
    - entity set
    - relationship set
    - attribute

[E-R Diagram of bank organization]



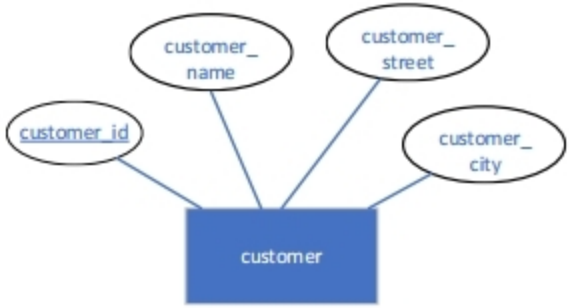
Conceptual Database Design

- Entity-Relationship (E-R) data model
  - Entity
    - an abstraction for a tangible or intangible concept that is distinguishable from other entities
    - depicted as a rectangle in E-R Diagram (ERD)
      - e.g., ‘Jones’ customer entity
    - described by attribute values
      - e.g., ‘Jones’ customer_name
  - Entity Type
    - Describes a type of similar entities
    - e.g., customer
  - Entity Set
    - A set of entities in an entity type
      - (e.g., a set of ‘Hayes’, ‘Johnson’, ‘Jones’)
  - Attributes
    - describes properties of an entity type
    - Type
      - Key attribute, Composite attribute, Multivalued attribute, Derived attribute



Conceptual Database Design

- Entity-Relationship (E-R) data model



a customer **Entity**, Jones

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

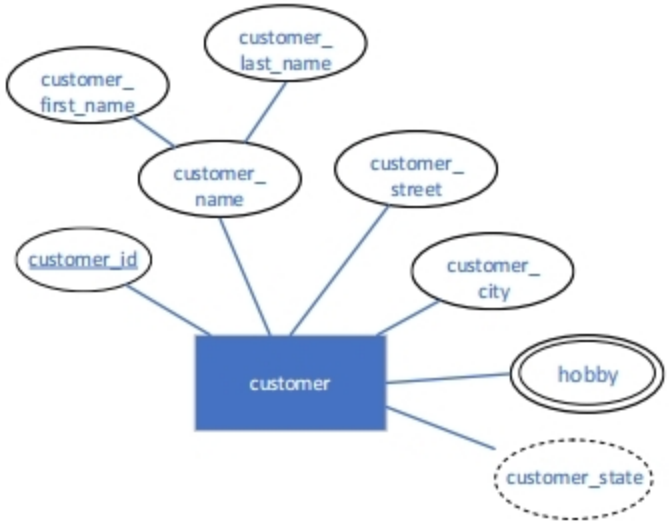
customer_id **Attribute**
its **value** is 244-66-8800

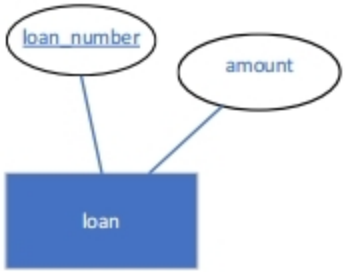
An Entity Set of
Customer **Entity Type**

Conceptual Database Design

- Entity-Relationship (E-R) data model

321-12-3123	Jones Kim	Main	Harrison	{‘baseball’, ‘soccer’}

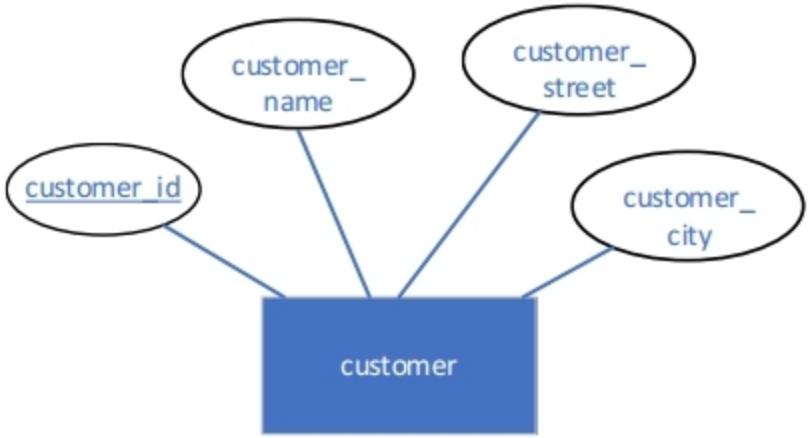




- **key attribute**: uniquely identify an entity (underlined)
  - e.g., 321-12-3123
- **non-key attribute**: all the attributes except key attribute(s)
  - Jones Kim, Main, Harrison, {‘baseball’, ‘soccer’}
- **simple attribute**: an atomic non-decomposable attribute
  - e.g., 321-12-3123, Main, Harrison
- **composite attribute**: a combination of other attributes
  - e.g., Jones Kim = Jones and Kim
- **single-valued attribute**: an attribute has one value
  - e.g., 321-12-3123, Main, Harrison
- **multi-valued attribute**: an attribute could have multiple values including zero (double oval)
  - e.g., {‘baseball’, ‘soccer’}
- **derived attribute:** an attribute that can be derived from other attributes of its entity type.
  - e.g., customer_state can be derived from customer_city
- **null-valued attribute**: an attribute could have ‘null’ value

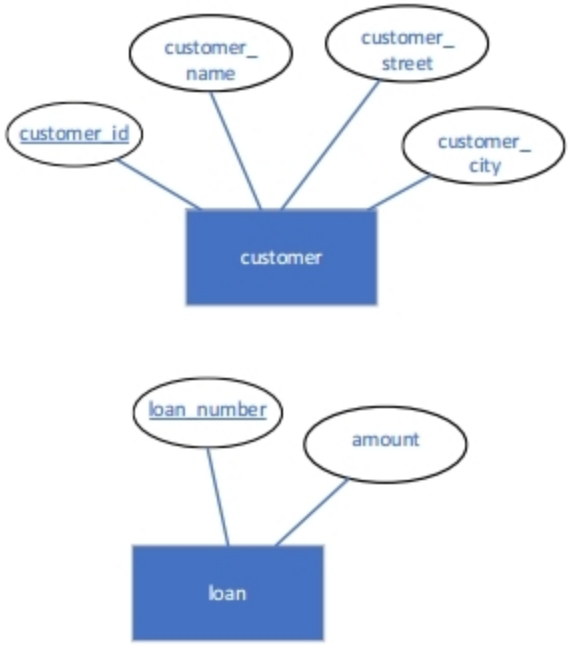
Conceptual Database Design

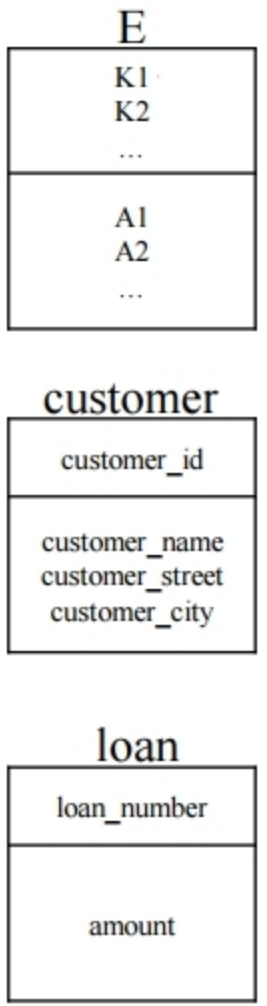
- Entity-Relationship (E-R) data model
  - Key of Entity
    - uniquely identify an entity
    - super key
      - {customer_id}
      - {customer_id, customer_name}
      - {customer_name, customer_street}
      - {customer_name, customer_street, customer_city}
      - etc.
  - Candidate Key
    - a minimal super key
      - {customer_id}
      - {customer_name, customer_street}
  - Primary Key
    - a selected candidate key



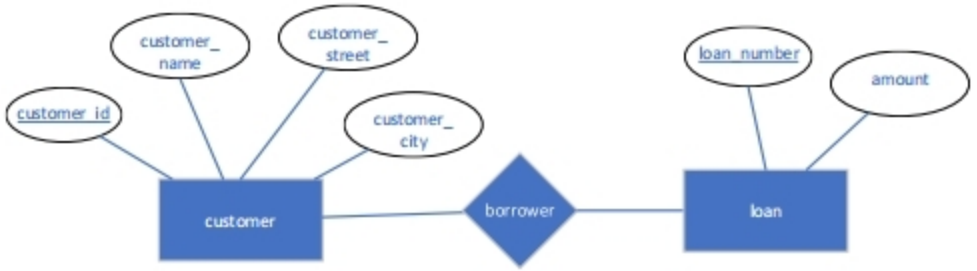
Conceptual Database Design

- Entity-Relationship (E-R) data model
  - Entity
    - Alternative representation





Conceptual Database Design



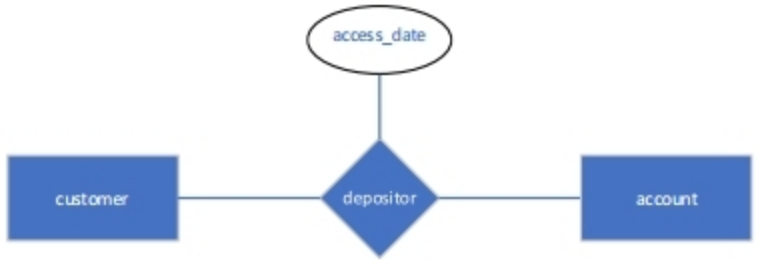
- Entity-Relationship (E-R) data model
  - Relationship
    - An association between several entities
    - "Jones" customer entity **participate in** "Jones borrows L-17" borrower relationship
  - Relationship Set
    - A set of relationships in a relationship type
- Relationship Type
  - Describes a type of similar entities
  - E.g., borrower

				relationship		
321-12-3123	Jones	Main	Harrison		L-17	1000
019-28-3746	Smith	North	Rye		L-23	2000
677-89-9011	Hayes	Main	Harrison		L-15	1500
555-55-5555	Jackson	Dupont	Woodside		L-14	1500
244-66-8800	Curry	North	Rye		L-19	500
963-96-3963	Williams	Nassau	Princeton		L-11	900
335-57-7991	Adams	Spring	Pittsfield		L-16	1300

[A Relationship Set of borrower Relationship Type]

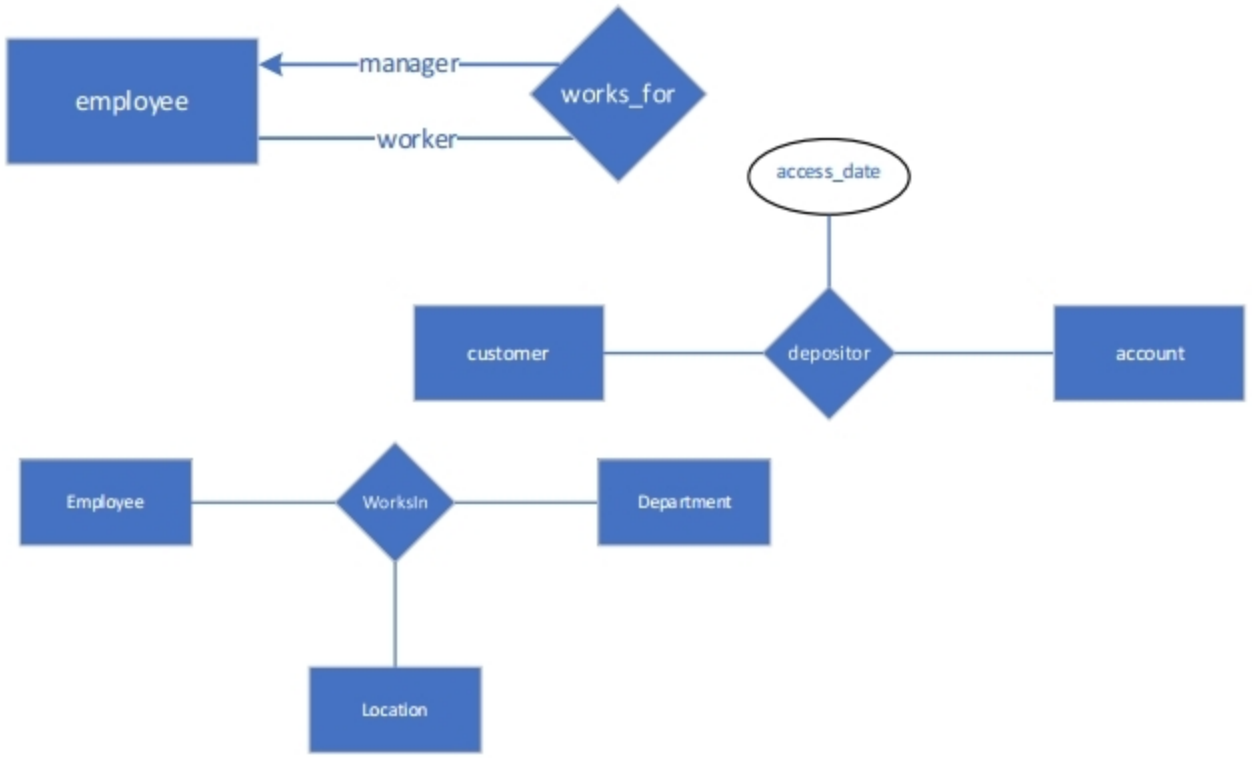
Conceptual Database Design

- Entity-Relationship (E-R) data model
  - Relationship
    - Recursive
      - employee could be worker or manager
    - Role
      - usually, implicit
      - a role of entity in a participation
      - e.g., worker, manager
    - Descriptive Attribute
      - describes a relationship



Conceptual Database Design

- Entity-Relationship (E-R) data model
  - Relationship
    - Degree of Relationship
      - the number of entity types that participate in a relationship type
      - Unary
      - Binary
      - Ternary
      - N-ary

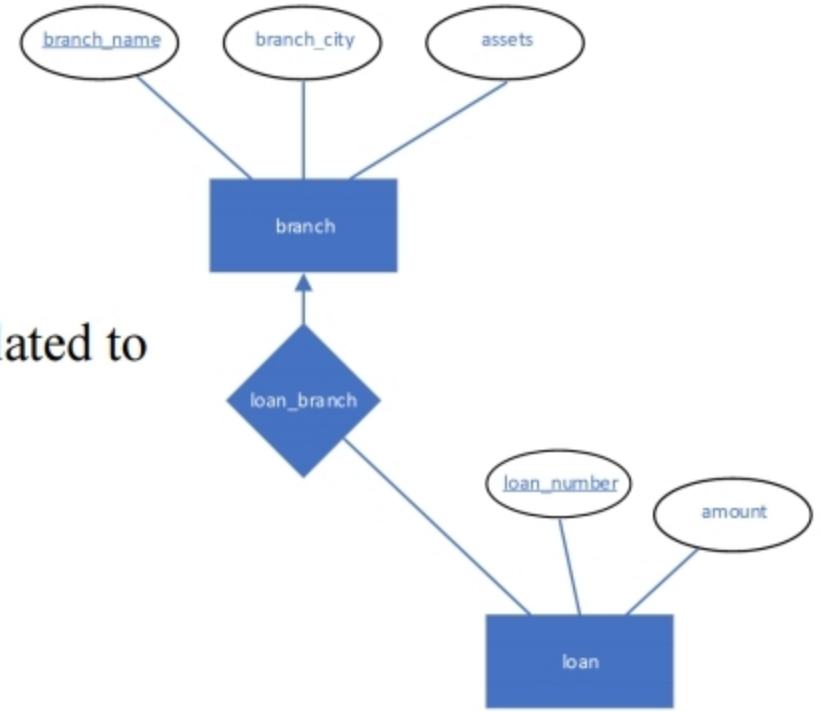


Conceptual Database Design

- Entity-Relationship (E-R) data model
  - Relationship
    - Mapping Cardinality (or Cardinality Ratio)
      - specifies how many instances of an entity in one entity can related to instances of another entity
    - 1:1

    - n:1

    - m:n



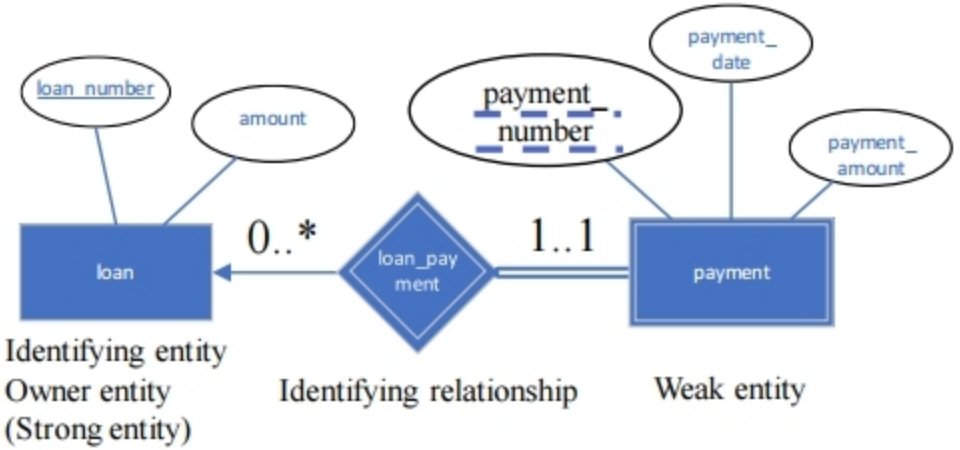
Which one occurs once?

loan_branch relation

loan_number	branch_name
L-11	Round Hill
L-14	Downtown
L-15	Perryridge
L-16	Perryridge
L-17	Downtown
L-23	Redwood
L-93	Mianus

Conceptual Database Design

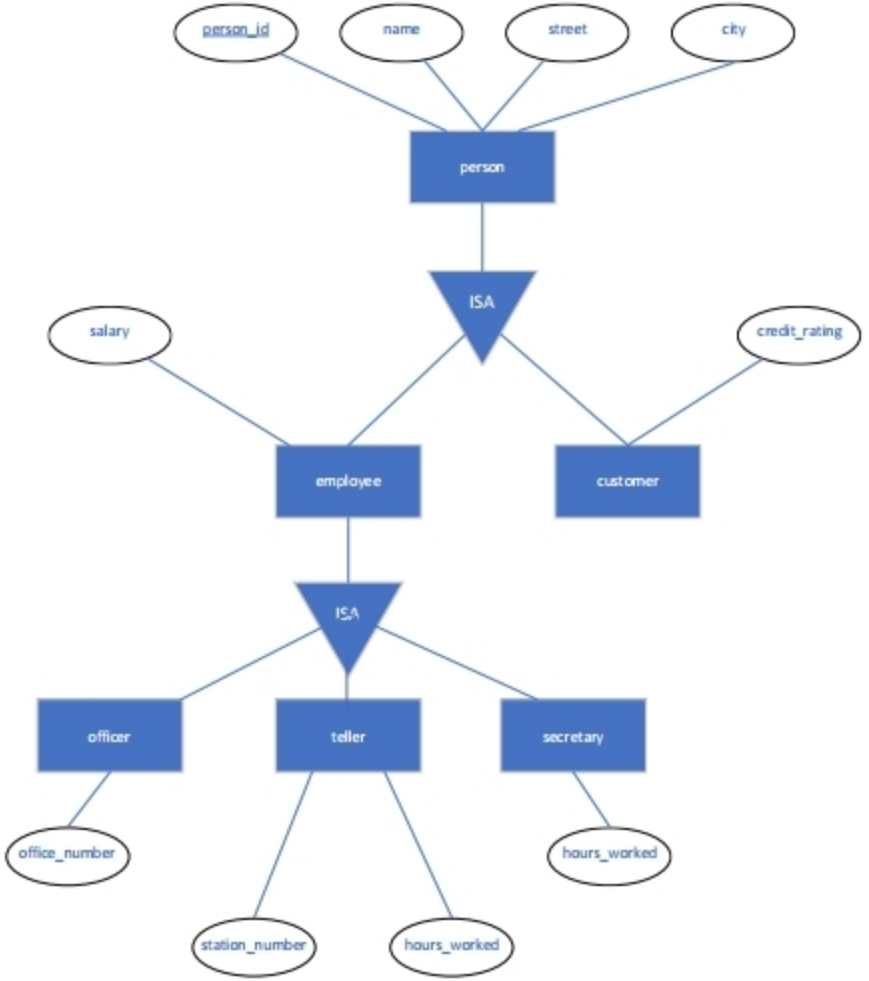
- Entity-Relationship (E-R) data model
  - Weak Entity (double rectangle)
    - an entity that cannot be uniquely identified by its attributes alone
    - become meaningful when it participates in **identifying** entities or **owner** entities
    - discriminator or partial key (dashed underline)
      - an attribute distinguishes a weak entity instance from the others
      - e.g., serial number from 1
    - making primary key
      - combine a primary key of identifying entity and discriminator
      - {loan_number, payment_number}
  - Participation Constraint
    - total participation (double line)
      - all the payment entity instances participate in loan_payment relationship
    - partial participation
      - a part of loan entity instances participate in loan_payment relationship



a loan could have 0 to Infinite payments
a payment should be one of loans

Conceptual Database Design

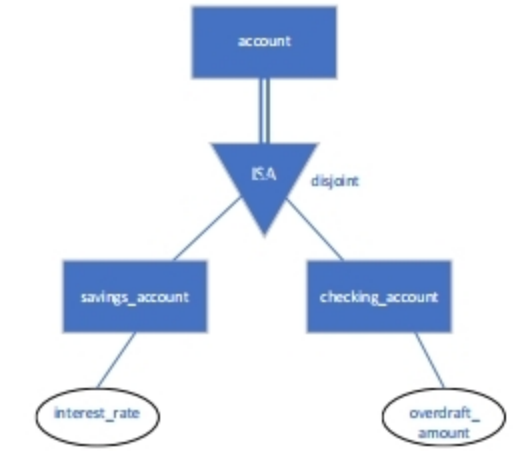
- Entity-Relationship (E-R) data model
  - Specialization & Generalization



- Customer is a **specialization** of Person
  - who could have more information
  - who inherits person_id, name, street, city attribute of Person
  - who is Person
  - who is a sub-class of Person
- Person is a **generalization** of Customer and Employee
  - who has a subset of common inf. of its specializations (sub-classes)
  - who is a super-class of Customer and Employee
- Single Inheritance
  - An entity could have at most one generalization
  - e.g., Java
- Multiple Inheritance
  - An entity could have multiple generalizations

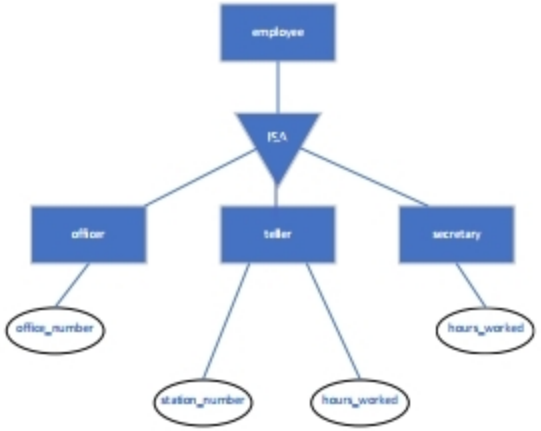
Conceptual Database Design

- Entity-Relationship (E-R) data model
  - Specialization & Generalization



- **Disjointness Constraint** in Single generalization
- Savings_account is Account but account could not be Savings_account
- If account should be either savings_account or checking_account,
- Account is **disjoint generalization** which
  - An account could be at most included in one of its sub-classes
- If employee could have two or more duty such as officer and teller
- employee is **overlapping generalization (default)**

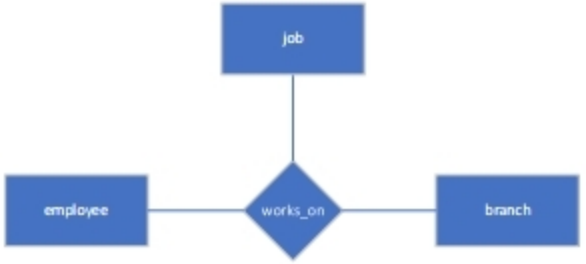
No correlation (total-disjoint, partial-disjoint, total-overlapping, partial-overlapping possible)

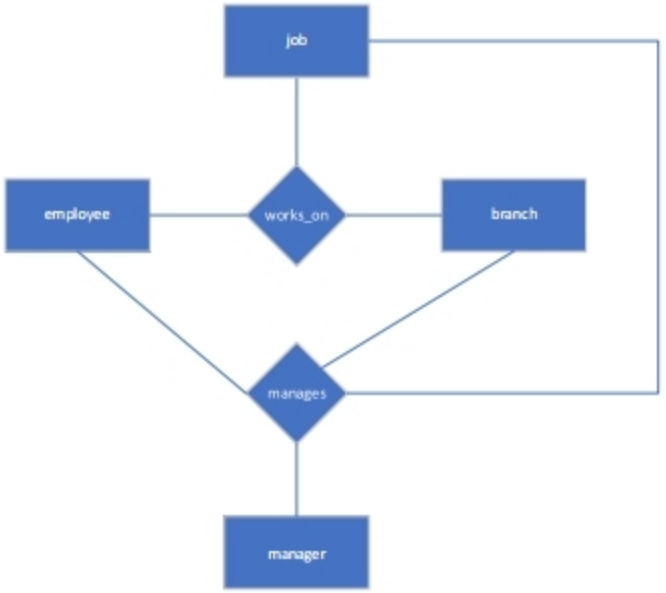


- **Completeness Constraint**
- If all the account instance should be included in savings_account or checking account,
  - **Total specialization (double line)**
- If some employee could not be included in officer, teller, secretary,
  - **Partial specialization (default)**

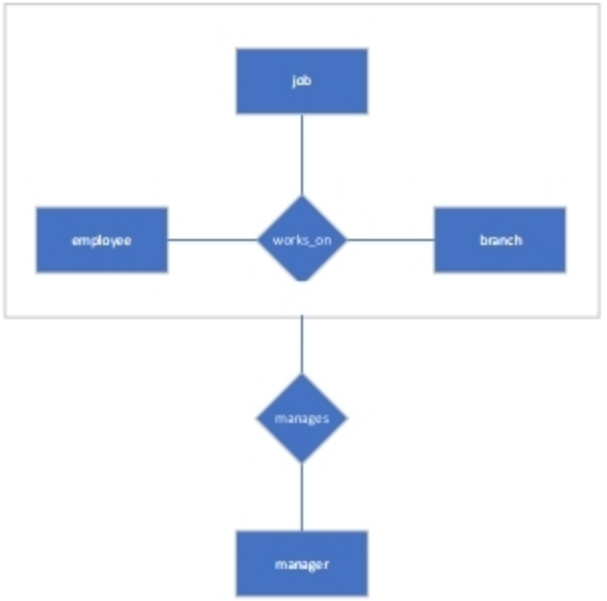
Conceptual Database Design

- Entity-Relationship (E-R) data model
  - Aggregation
    - There is no direct way to represent a relationship between relationships
    - e.g., what if a manager manages a fact that an employee of a branch did a specific job



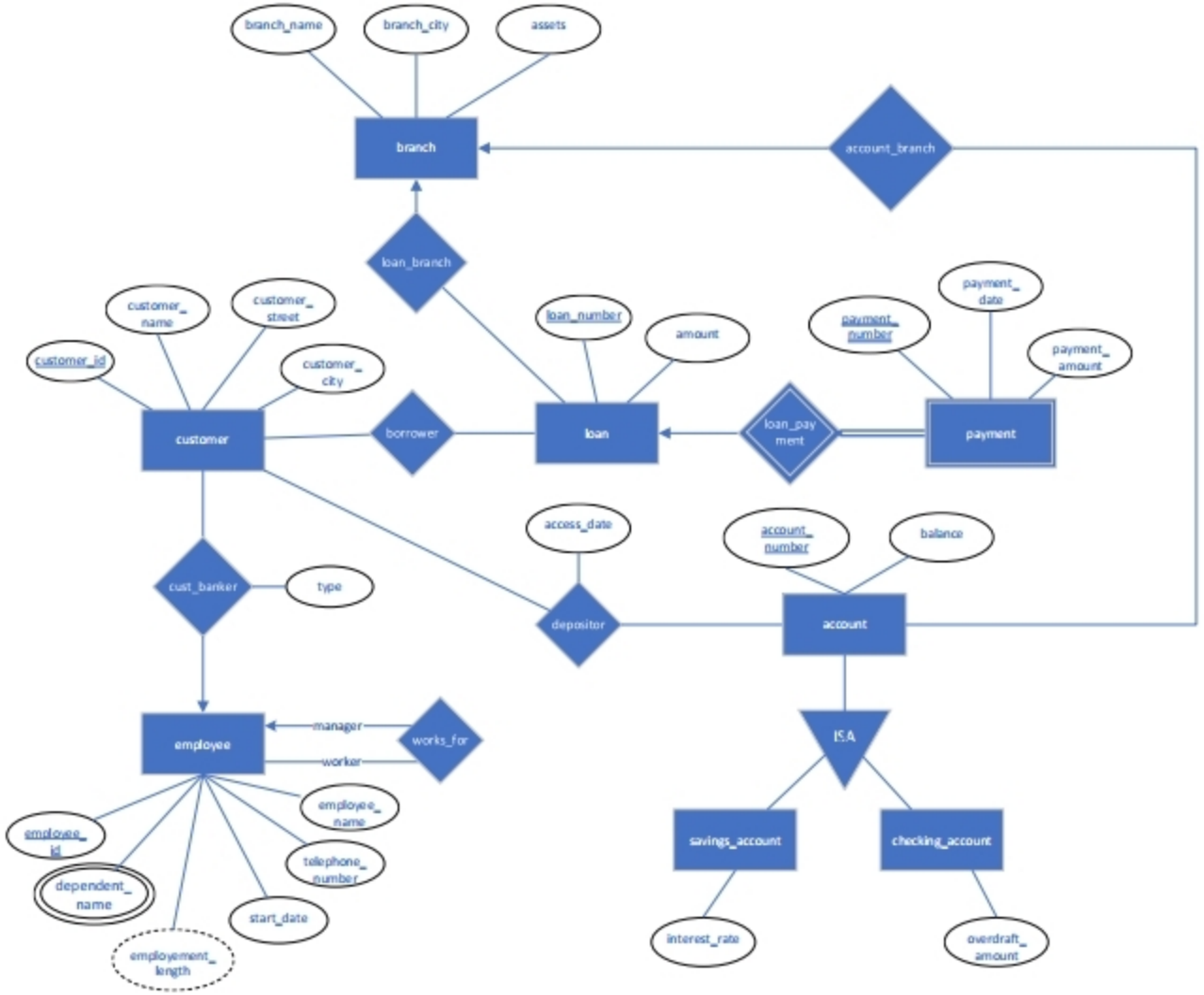


aggregation



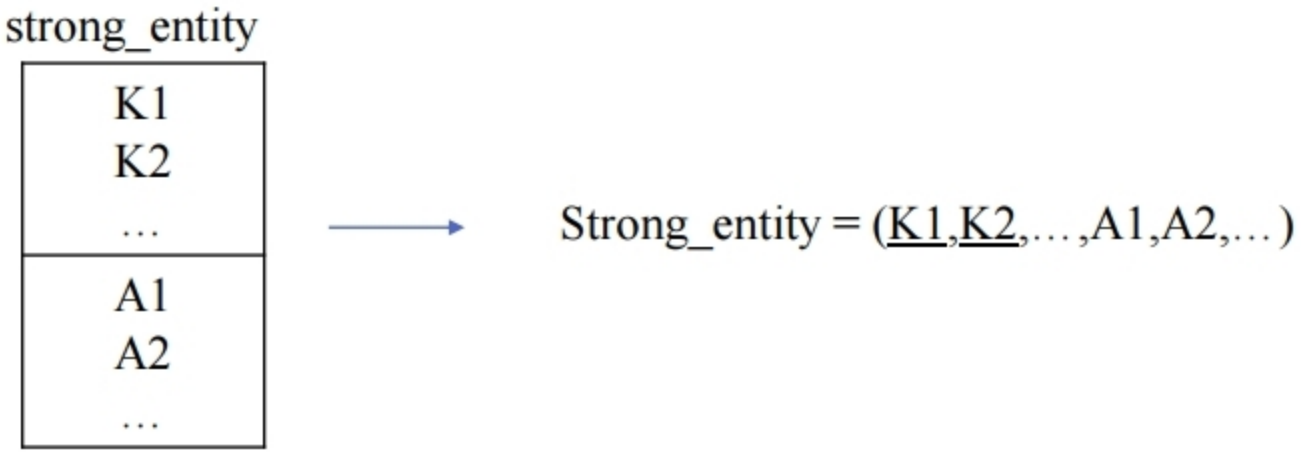
Mapping E-R data model to Relations

- Mapping E-R data model to Database Tables
  - e.g., Bank Organization



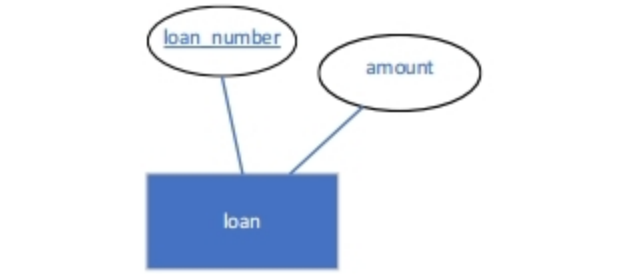
Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Strong Entity
    - Mapping a strong entity to a relation
      - key attributes
      - non-key attributes

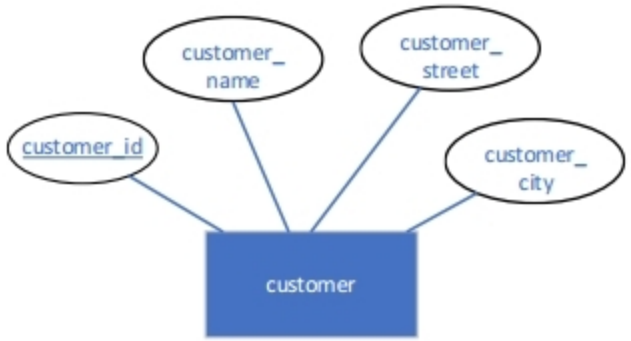


Mapping E-R data model to Relations

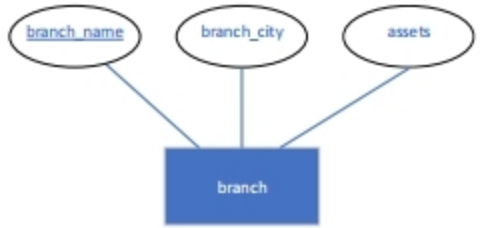
- Mapping E-R data model to relations
  - Strong Entity
    - Mapping a strong entity to a relation



Loan?



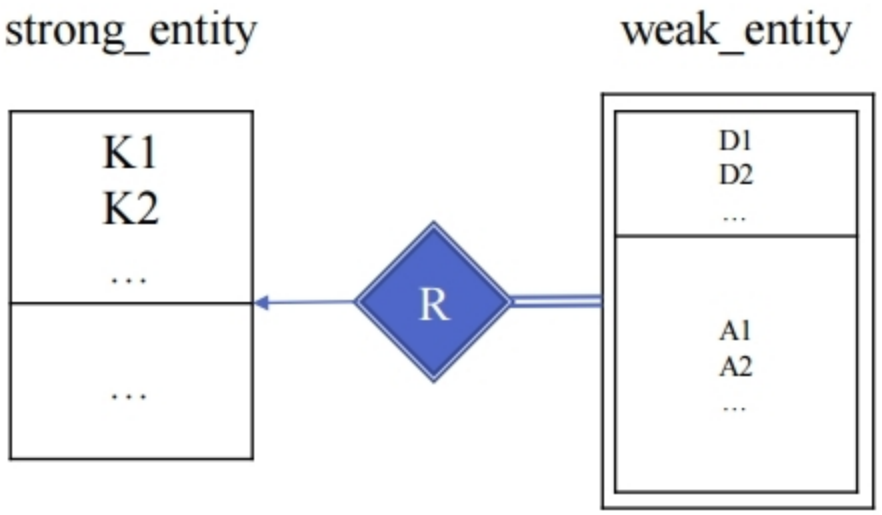
Customer?



Branch?

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Weak Entity
    - Mapping a weak entity to a relation

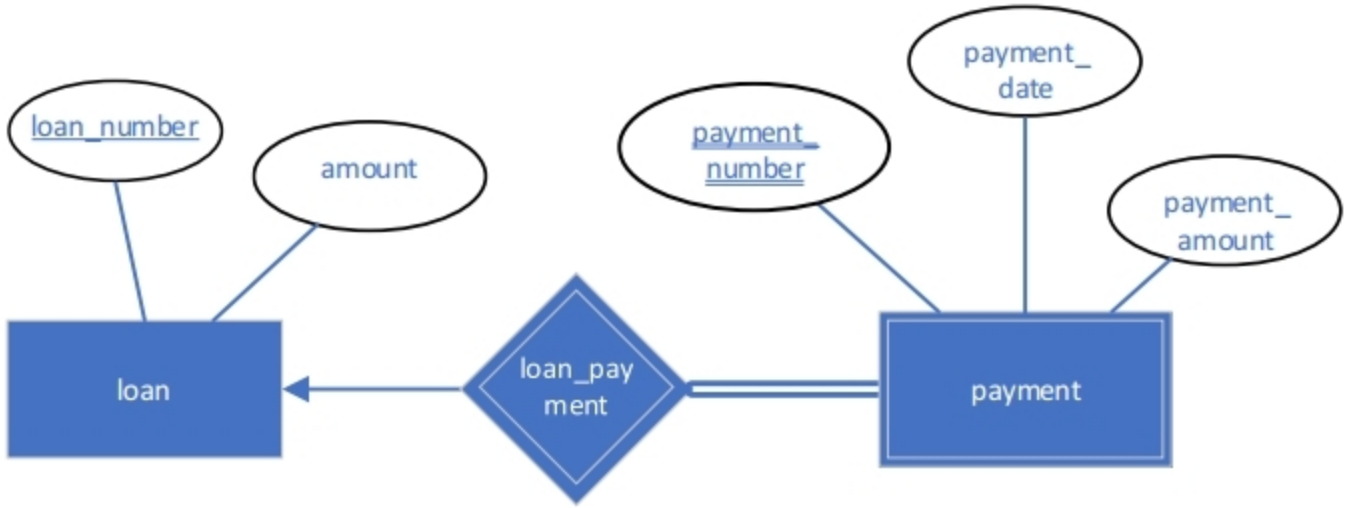


Weak_entity = (<u>K1,K2,…, D1,D2,…,</u>A1,A2,…)

Primary Key: Primary key of owner entity ∪ Discriminators of weak entity
Foreign Key: Primary key of owner entity

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Weak Entity
    - Mapping a weak entity to a relation



Payment?

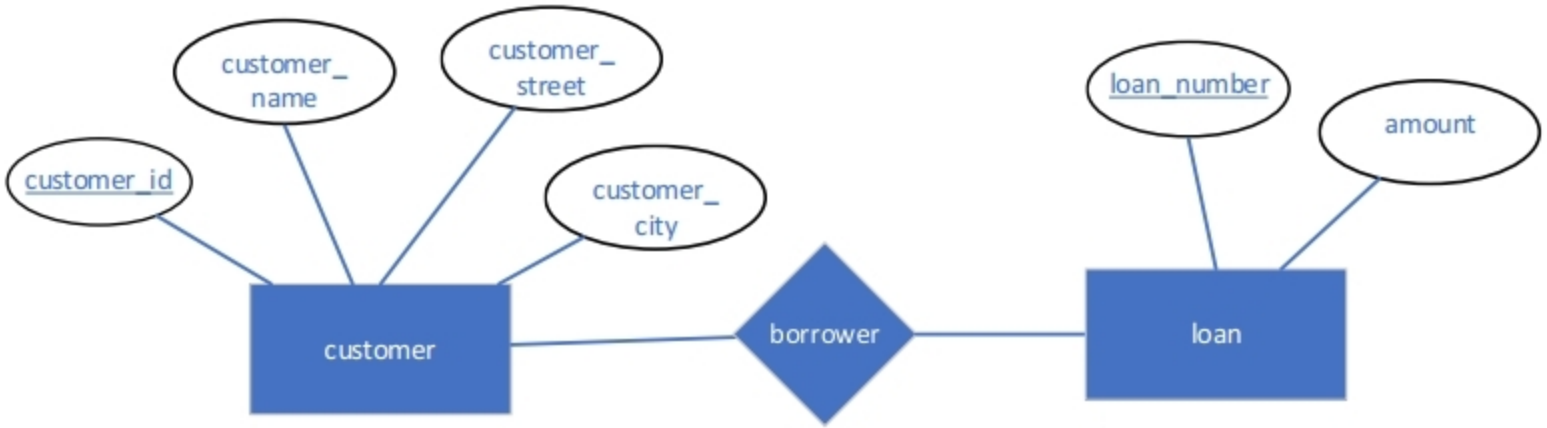
Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship
    - Binary N:M relationship
      - (ka1,ka2,…,kb1,kb2,…)

      A = (ka1,ka2,…,aa1,aa2,…) B = (kb1,kb2,…,ab1,ab2,…)
    - Binary 1:1 relationship
      - (ka1,ka2,…,kb1,kb2,…) or (ka1,ka2,…,kb1,kb2,…)
    - Binary N:1 relationship (assuming A is N)
      - (ka1,ka2,…,kb1,kb2,…)
    - Unary N:1 relationship
      - (ka1,ka2,…,kac1,kac2,…) (ka**c**N is a new attribute for kaN)
    - N-ary relationship without arrow
      - Primary Key: union of primary keys
    - N-ary relationship with one arrow
      - Primary Key: union of primary keys of entities without arrow

Mapping E-R data model to Relations

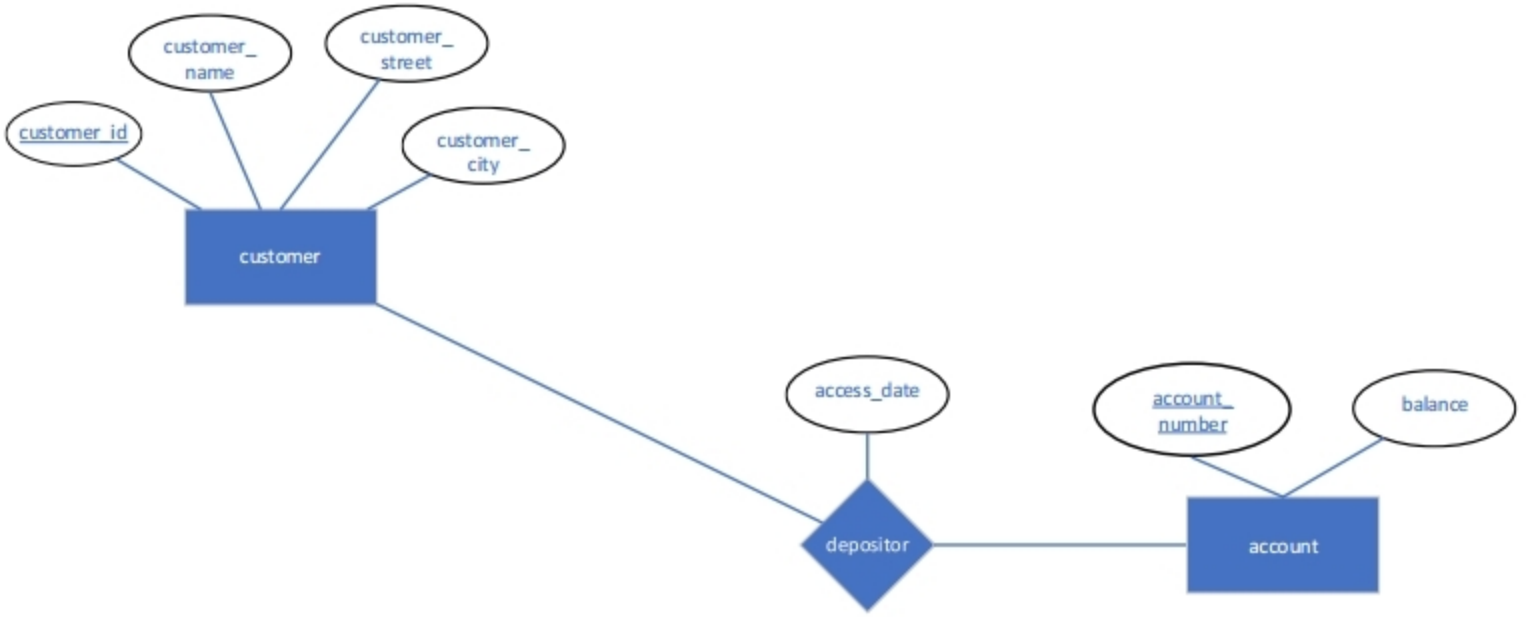
- Mapping E-R data model to relations
  - Relationship – Binary M:N relationship



Borrower?

Mapping E-R data model to Relations

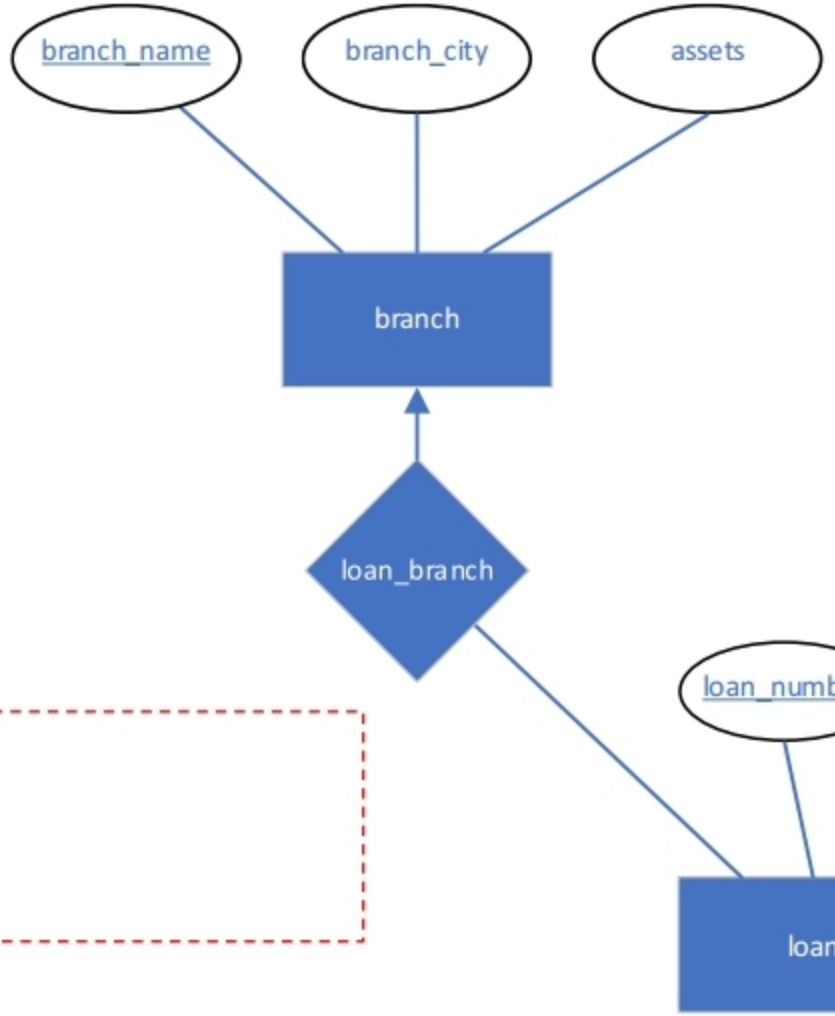
- Mapping E-R data model to relations
  - Relationship – Binary M:N relationship with attribute



Depositor?

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship – Binary N:1 relationship



loan_branch relation

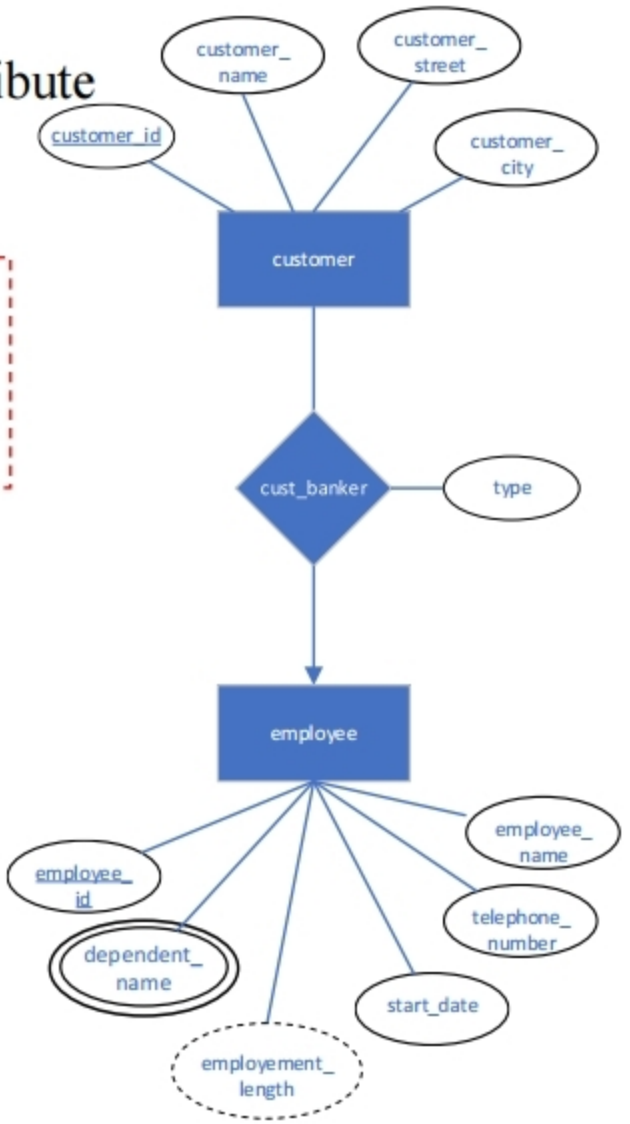
loan_number	branch_name
L-11	Round Hill
L-14	Downtown
L-15	Perryridge
L-16	Perryridge
L-17	Downtown
L-23	Redwood
L-93	Mianus

Loan_branch?

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship - Binary N:1 relationship with descriptive attribute

Cust_banker?



$\pi_{customer\_id,employee\_id}$(cust_banker)

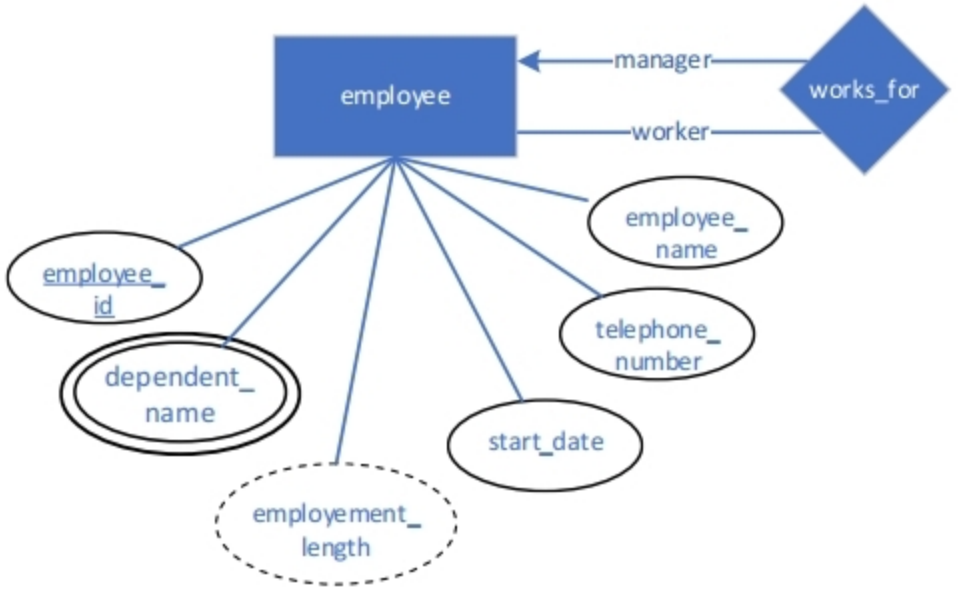
customer	customer_street
Adams	Coyote
Brooks	Smith
Curry	Coyote
Glenn	Williams
Hayes	Williams
Johbnson	Rabbit
Jones	Smith
Lindsay	Williams
Smith	Rabbit
Turner	Smith
Williams	Rabbit

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship - Unary N:1 relationship

Unary N:1 relationship

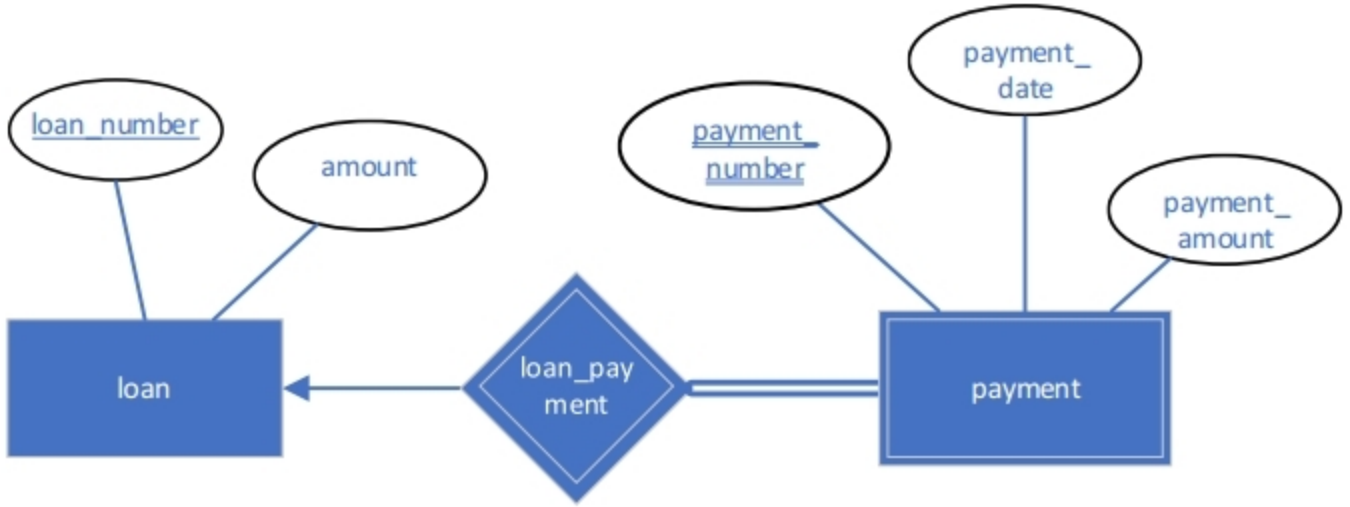
(ka1,ka2,…,kac1,kac2,…) (kacN is a new attribute for kaN)



Works_for?

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship – Binary n:1 relationship with Weak entity

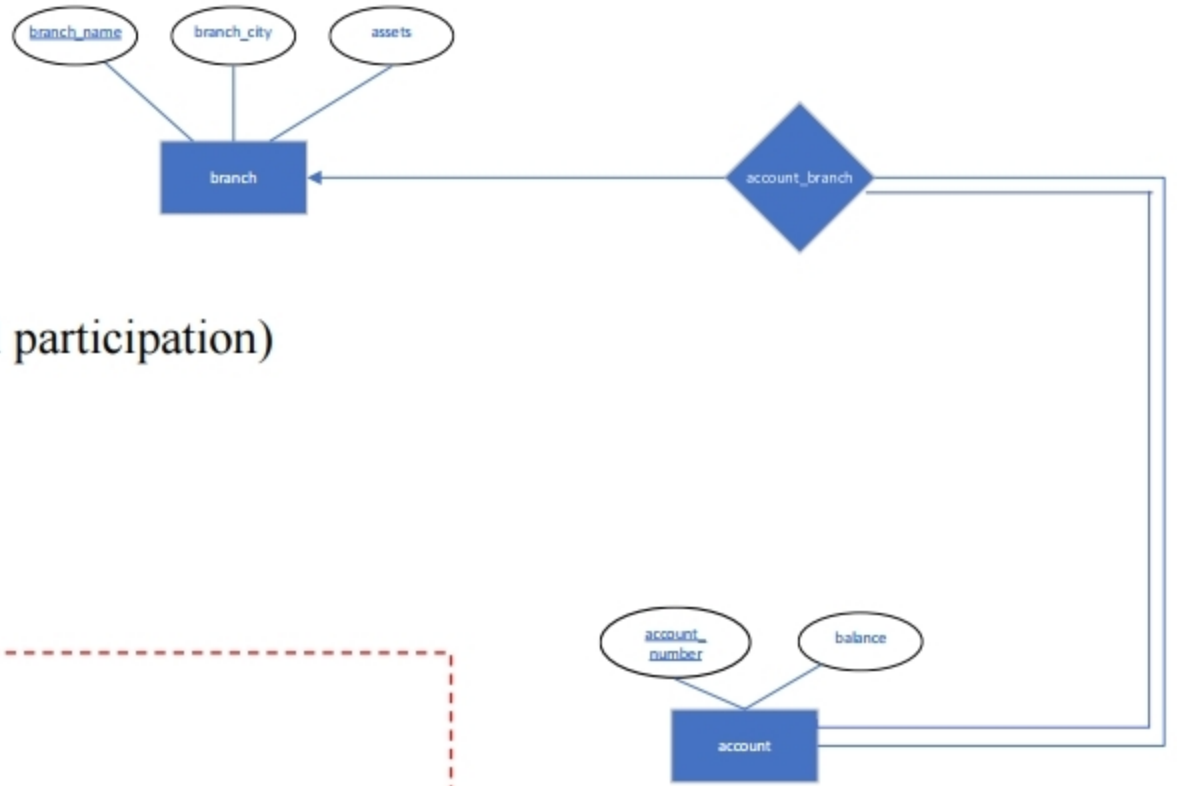


Unnecessary

Payment (loan_number, payment_number, payment_date, payment_amount)
Primary key: {loan_number, payment_number}
Foreign key: {loan_number}

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship – Binary n:1 relationship with total participation



Branch, Account, Account_branch? (assuming not total participation)

Mapping E-R data model to Relations

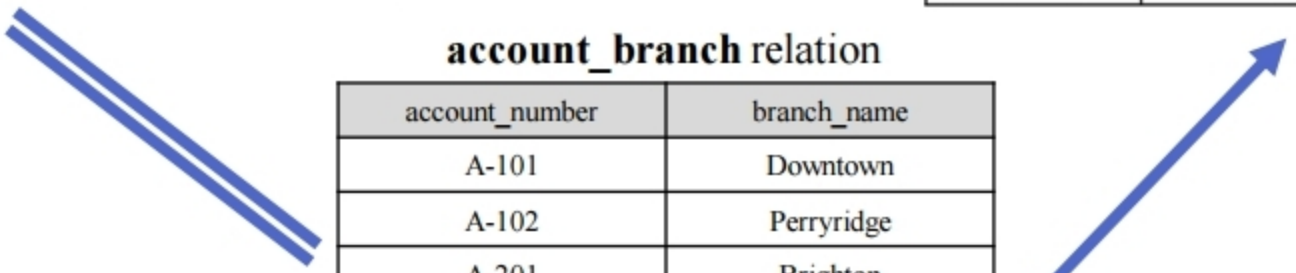
- Mapping E-R data model to relations
  - Relationship – Binary n:1 relationship with total participation

account relation

account_number	balance
A-101	500
A-102	400
A-201	900
A-215	700
A-217	750
A-222	700
A-305	350

branch **relation**

branch_name	branch_city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000



account_branch relation

account_number	branch_name
A-101	Downtown
A-102	Perryridge
A-201	Brighton
A-215	Mianus
A-217	Brighton
A-222	Redwood
A-305	Round Hill

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship – Binary n:1 relationship with total participation
    - Merging account and account_branch relation to remove redundancy
    - Total participation

account relation

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

branch **relation**

branch_name	branch_city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

Account?

Branch = (<u>branch_name</u>, branch_city, assets)

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship – Binary 1:1 relationship (partial participation)

male relation

male_number	?
101	?
102	?
201	?
202	?

marriage relation

male_number	female_number
101	x
102	y

female relation

female_number	?
x	?
y	?
z	?
p	?

Marriage?

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship – Binary 1:1 relationship (one directional total participation)

engine relation

engine_number	created_date
101	2021-03-01
102	2021-03-02
201	2021-03-03
202	2021-03-04

assembling relation

engine_number	car_number
101	x
102	y
201	z

car relation

car_number	created_date
x	2021-03-02
y	2021-03-03
z	2021-03-04

Assembling?

Enough?

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship – Binary 1:1 relationship (one directional total participation)

engine relation

engine_number	created_date
101	2021-03-01
102	2021-03-02
201	2021-03-03
202	2021-03-04

car relation (updated)

car_number	created_date	engine_number
x	2021-03-02	101
y	2021-03-03	102
z	2021-03-04	201

Car?

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship – Binary 1:1 relationship (two directional total participation)

engine relation

engine_number	created_date
101	2021-03-01
102	2021-03-02
201	2021-03-03

car relation (updated)

car_number	created_date	engine_number
x	2021-03-02	101
y	2021-03-03	102
z	2021-03-04	201

Car?

Enough?

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship – Binary 1:1 relationship (two directional total participation)

car relation (updated again)

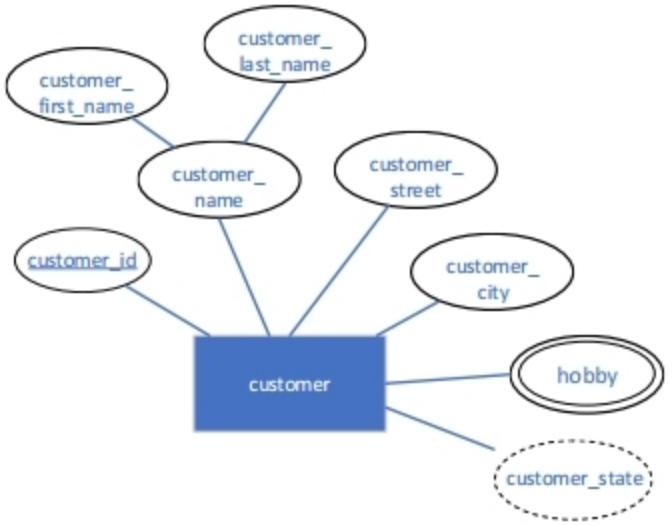
car_number	car_created_date	engine_number	engine_created_date
x	2021-03-02	101	2021-03-01
y	2021-03-03	102	2021-03-02
z	2021-03-04	201	2021-03-03

Car?

Mapping E-R data model to Relations

- Mapping E-R data model to relations

321-12-3123	Jones | Kim	Main	Harrison	{‘baseball’, ‘soccer’}

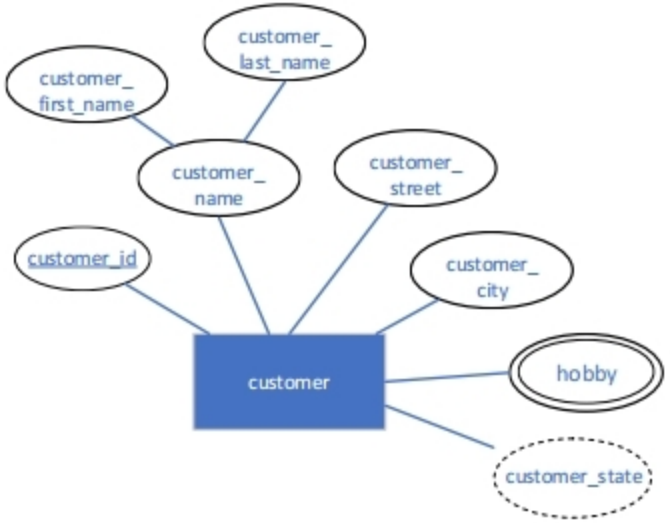


- **composite attribute**: a combination of other attributes
  - e.g., Johns Kim = Jones and Kim
- **multi-valued attribute**: an attribute could have multiple values including zero (double oval)
  - e.g., {‘baseball’, ‘soccer’}

Mapping E-R data model to Relations

- Mapping E-R data model to relations

321-12-3123	Jones Kim	Main	Harrison	{'baseball', 'soccer'}



- **composite attribute**: a combination of other attributes
  - e.g., Johns Kim = Jones and Kim

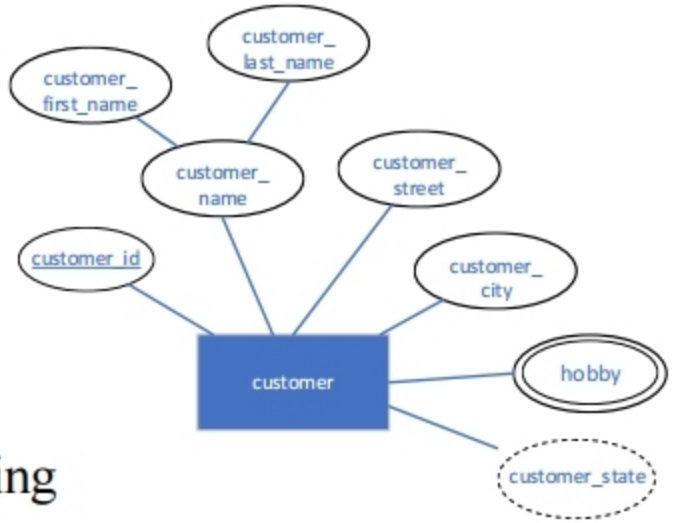
  - customer_name consists of *customer_first_name* and *customer_last_name*
  - include **only** italic to a relation

  - Customer?

Mapping E-R data model to Relations

- Mapping E-R data model to relations

321-12-3123	Jones | Kim	Main	Harrison	{'baseball', 'soccer'}



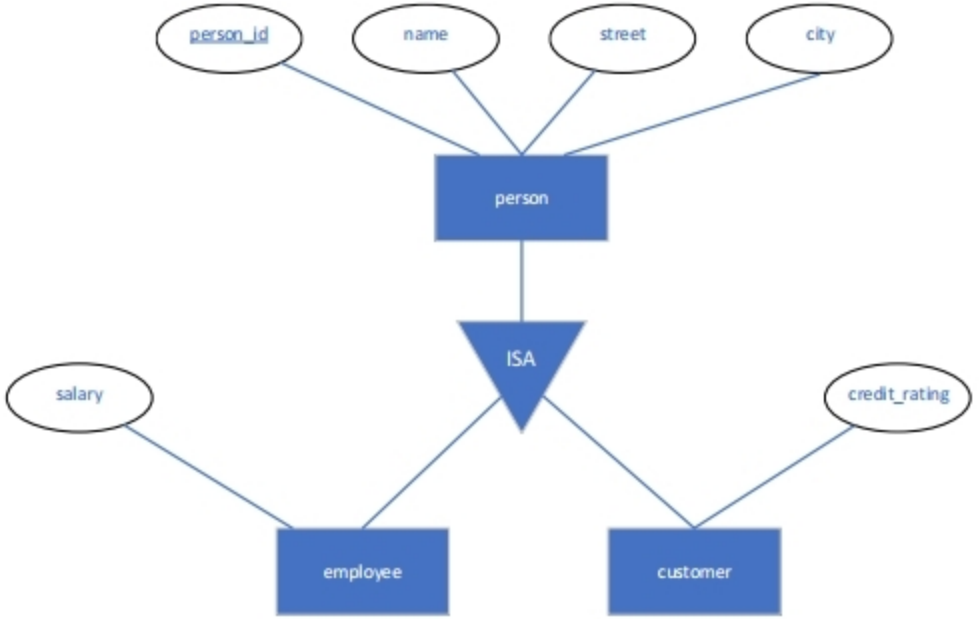
- **multi-valued attribute**: an attribute could have multiple values including zero (double oval)
  - e.g., {'baseball', 'soccer'}

- create one more relation with a primary key of its entity and attribute name such like

FOREIGN KEY(customer_id) REFERENCES customer(customer_id)

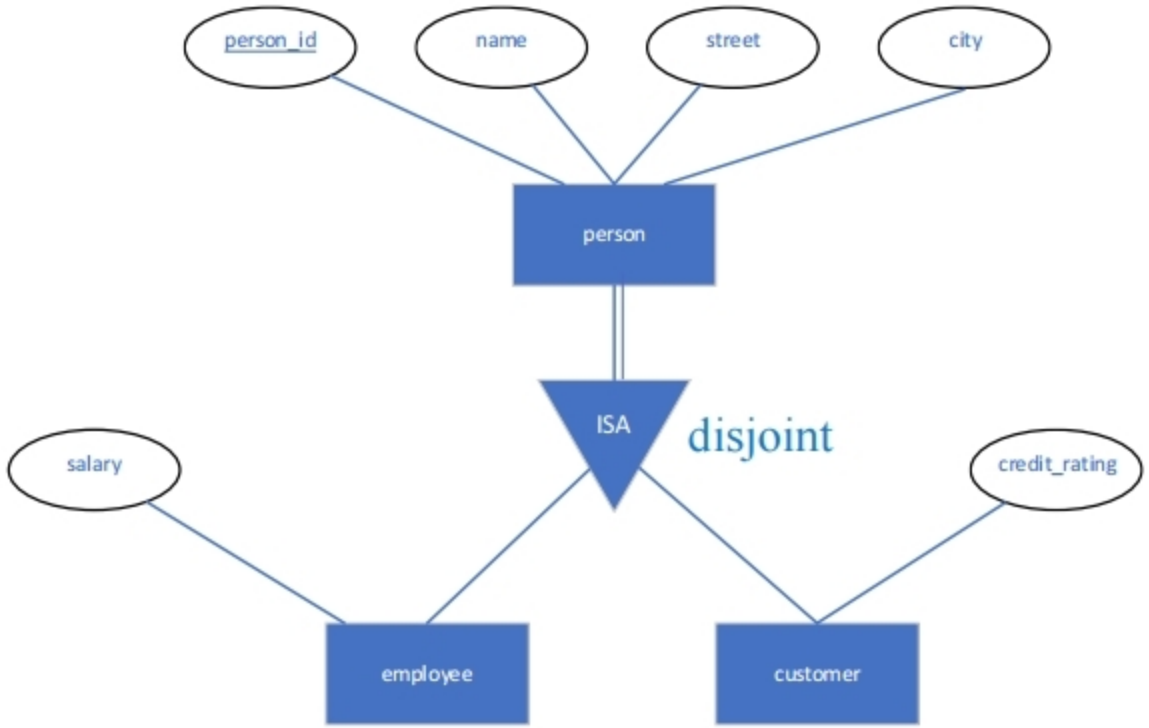
Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Specialization & Generalization
    - **Generally**, specializations set the primary key of its generalization with foreign key constraint
      - Person = (person_id, name, street, city)
      - Employee = (person_id, salary)
        - FOREIGN KEY (person_id) REFERENCES person (person_id)
      - Customer = (person_id, credit_rating)
        - FOREIGN KEY (person_id) REFERENCES person (person_id)



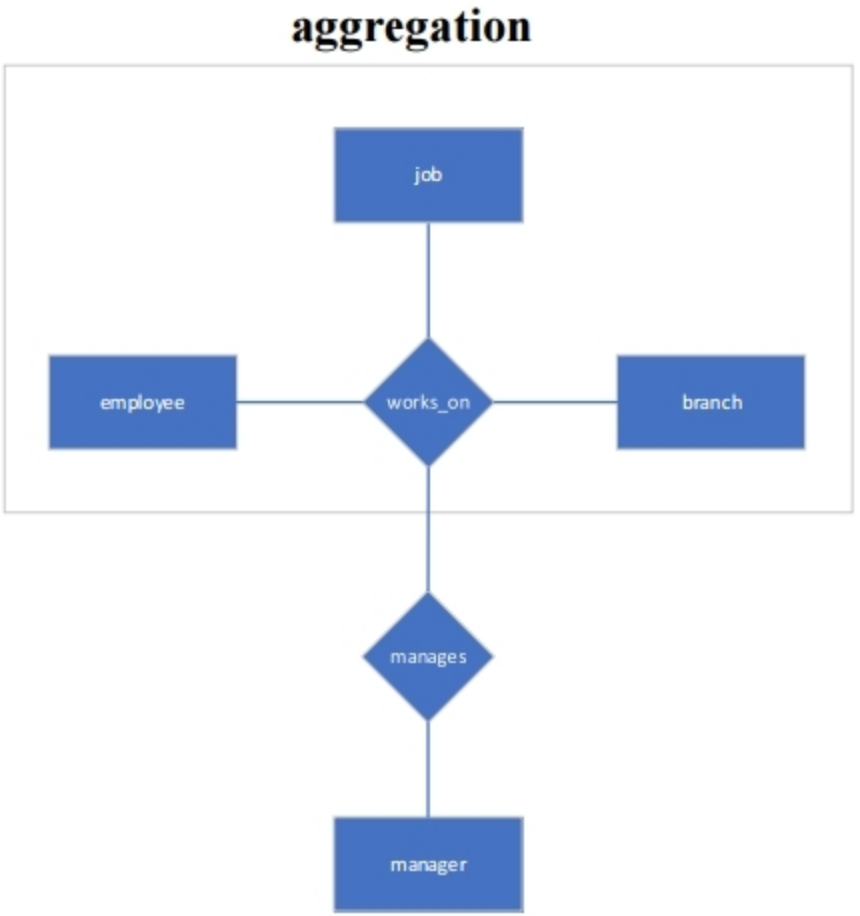
Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Specialization & Generalization
    - **For the total-disjoint specialization**, Specializations inherit all the attributes of its generalization and set its primary key
      - Employee = (person_id, name, street, city, salary)
      - Customer = (person_id, name, street, city, credit_rating)



Mapping E-R data model to Relations

- Mapping E-R data model to relations
  - Relationship between an entity and aggregation
    - The relationship has a primary key of the entity and non-key attributes of the primary keys of the relationship of the aggregation

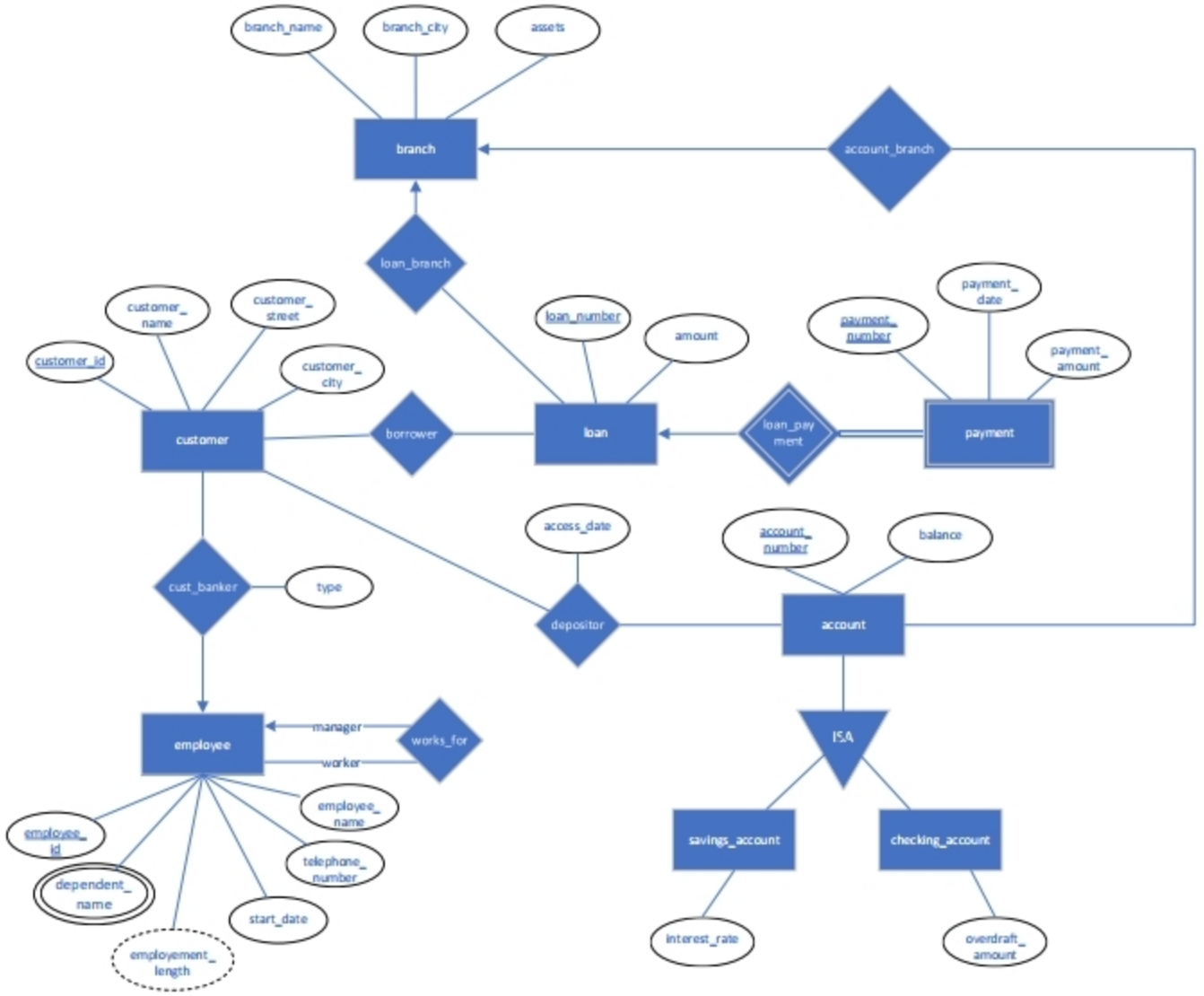


- The relationship (manages) has a primary key of the entity (manager) and non-key attributes of the primary keys of the relationship (works_on) of the aggregation

- Manager = (manager_id)
- Works_on = (employee_id, job_id, branch_id)
- Manages = (manager_id, employee_id, job_id, branch_id)

Mapping E-R data model to Relations

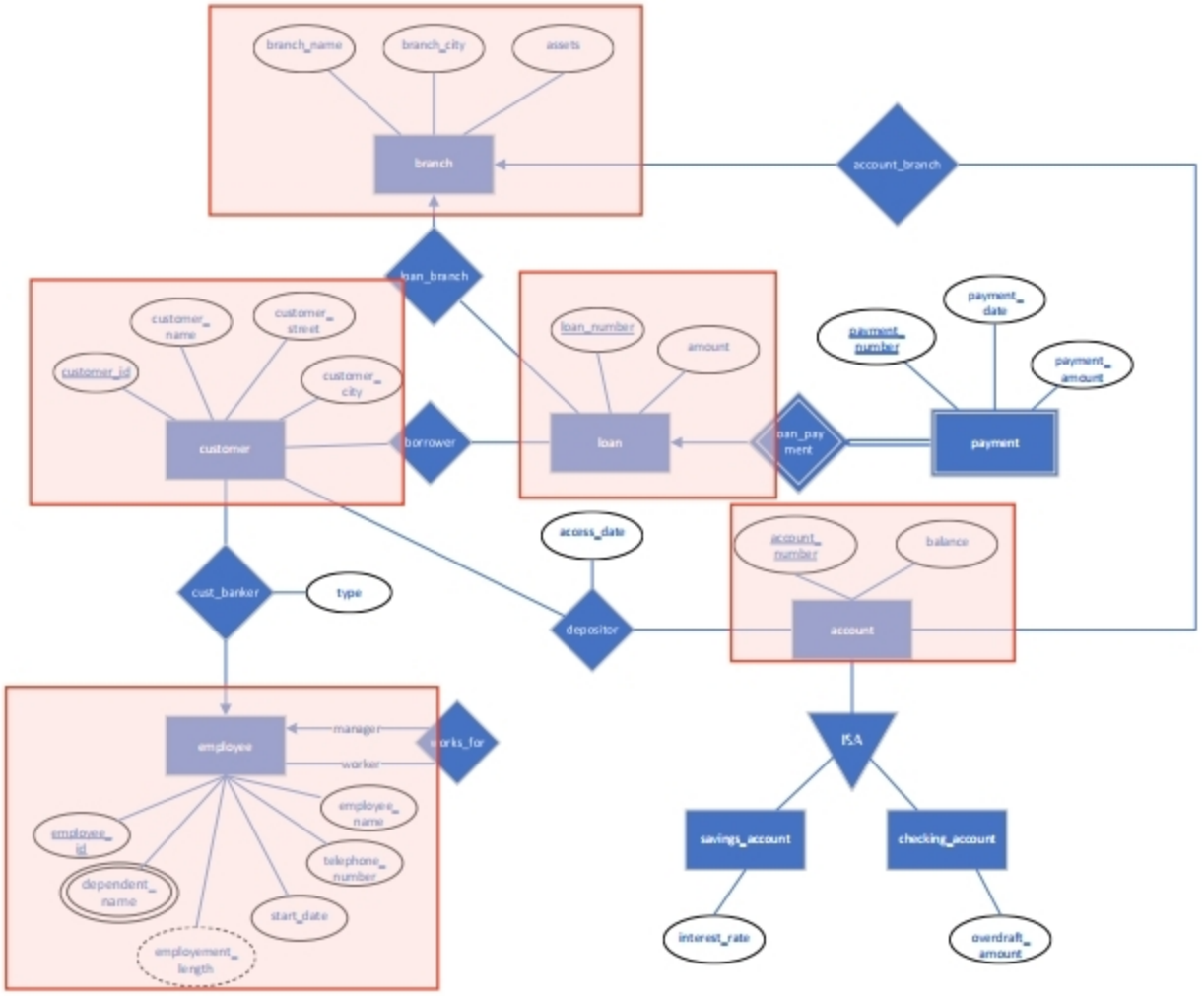
- Mapping E-R data model to relations



Mapping E-R data model to Relations

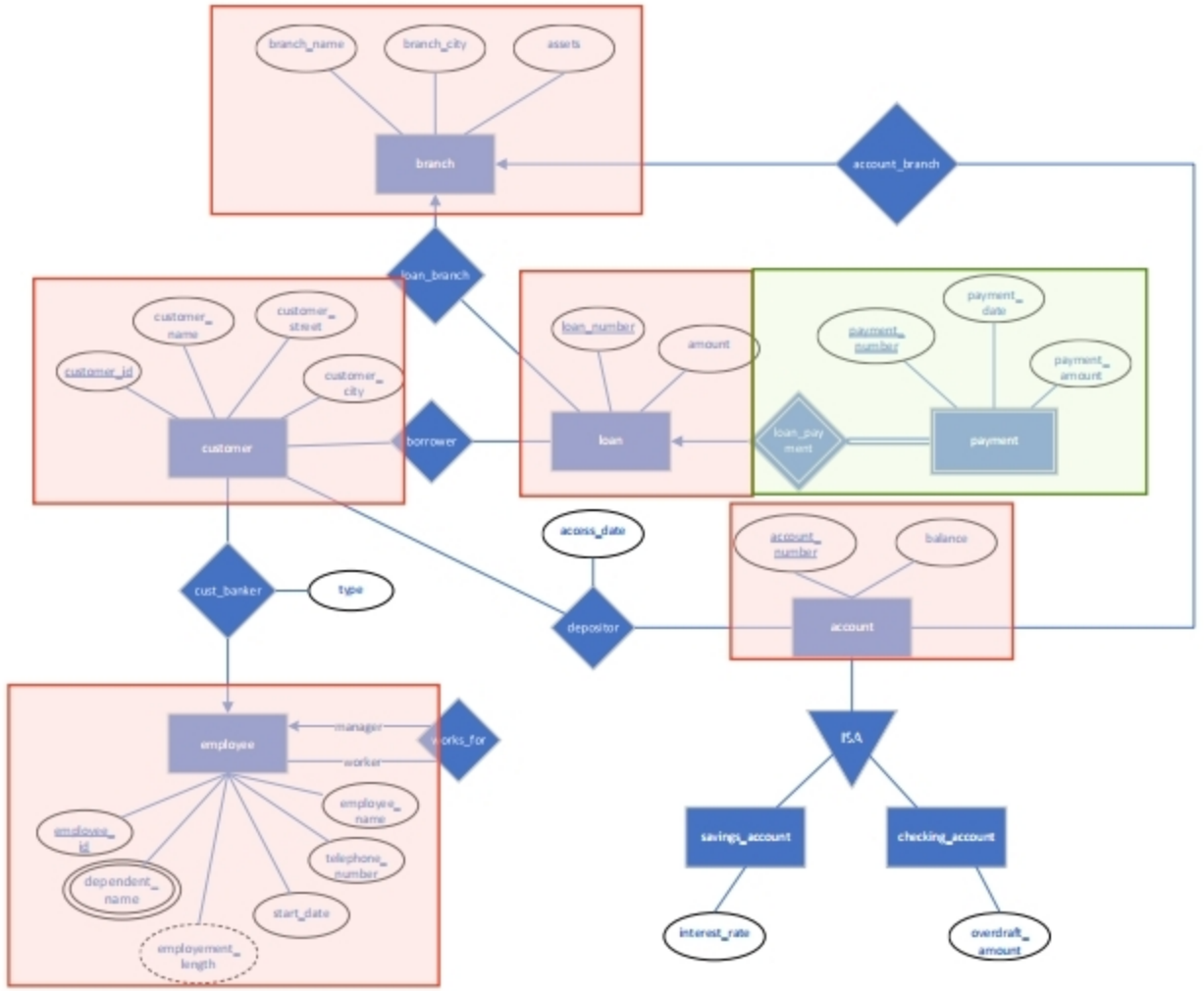
- Mapping E-R data model to relations
  1. Strong Entity

branch = (branch_name, branch_city, assets)
loan = (loan_number, amount)
customer = (customer_id, customer_name, customer_street, customer_city)
employee = (employee_id, employee_name, telephone_number, start_date)
account = (account_number, balance)
dependent_name = (employee_id, dependent_name)



Mapping E-R data model to Relations

- Mapping E-R data model to relations
  2. Weak Entity

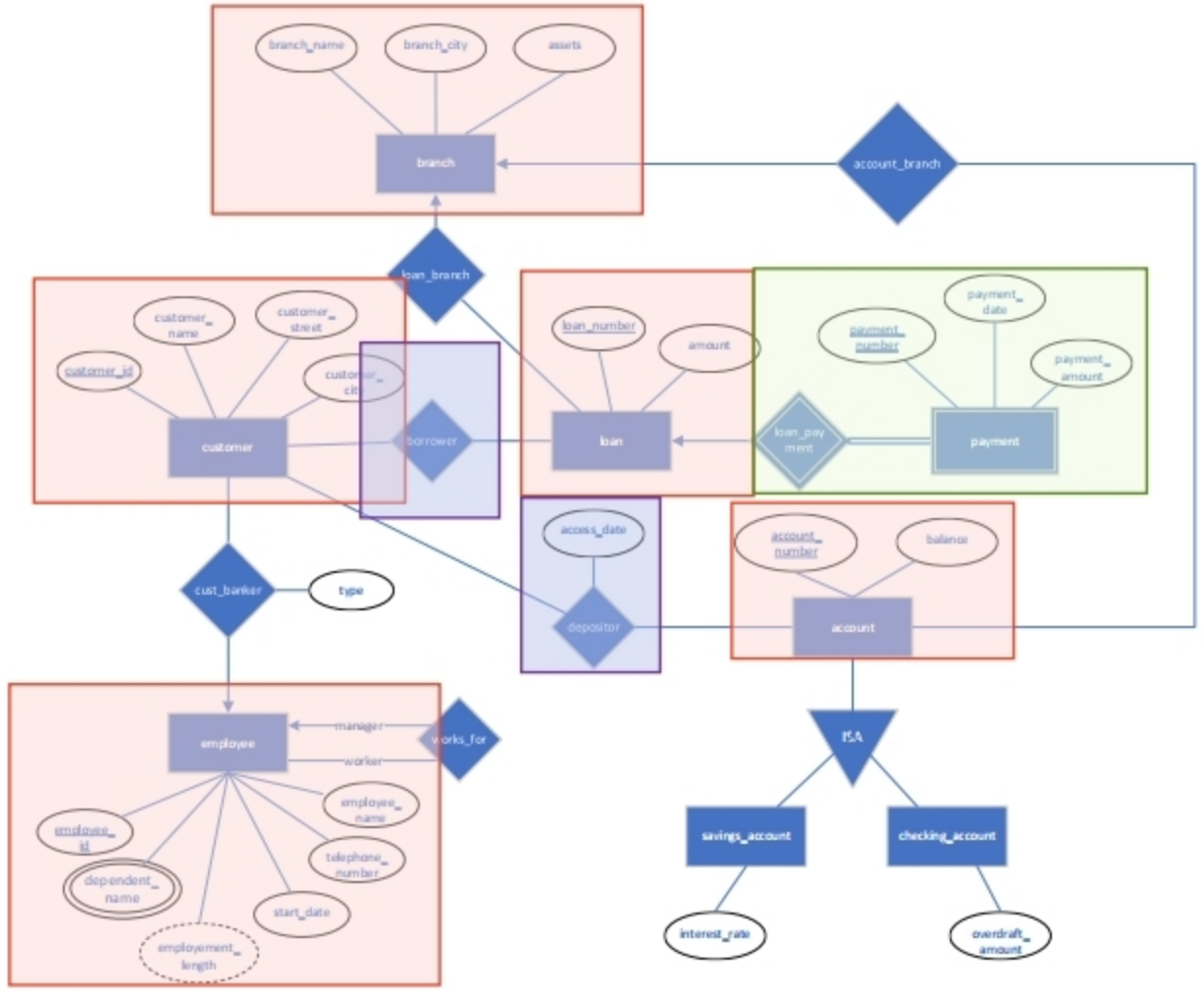


branch = (branch_name, branch_city, assets)
loan = (loan_number, amount)
customer = (customer_id, customer_name, customer_street, customer_city)
employee = (employee_id, employee_name, telephone_number, start_date)
account = (account_number, balance)
dependent_name = (employee_id, dependent_name)

payment = (loan_number, payment_number, payment_date, payment_amount)

Mapping E-R data model to Relations

- Mapping E-R data model to relations
  3. N:M Relation



branch = (branch_name, branch_city, assets)
loan = (loan_number, amount)
customer = (customer_id, customer_name, customer_street, customer_city)
employee = (employee_id, employee_name, telephone_number, start_date)
account = (account_number, balance)
dependent_name = (employee_id, dependent_name)

payment = (loan_number, payment_number, payment_date, payment_amount)

borrower = (customer_id, loan_number)
depositor = (customer_id, account_number, access_date)

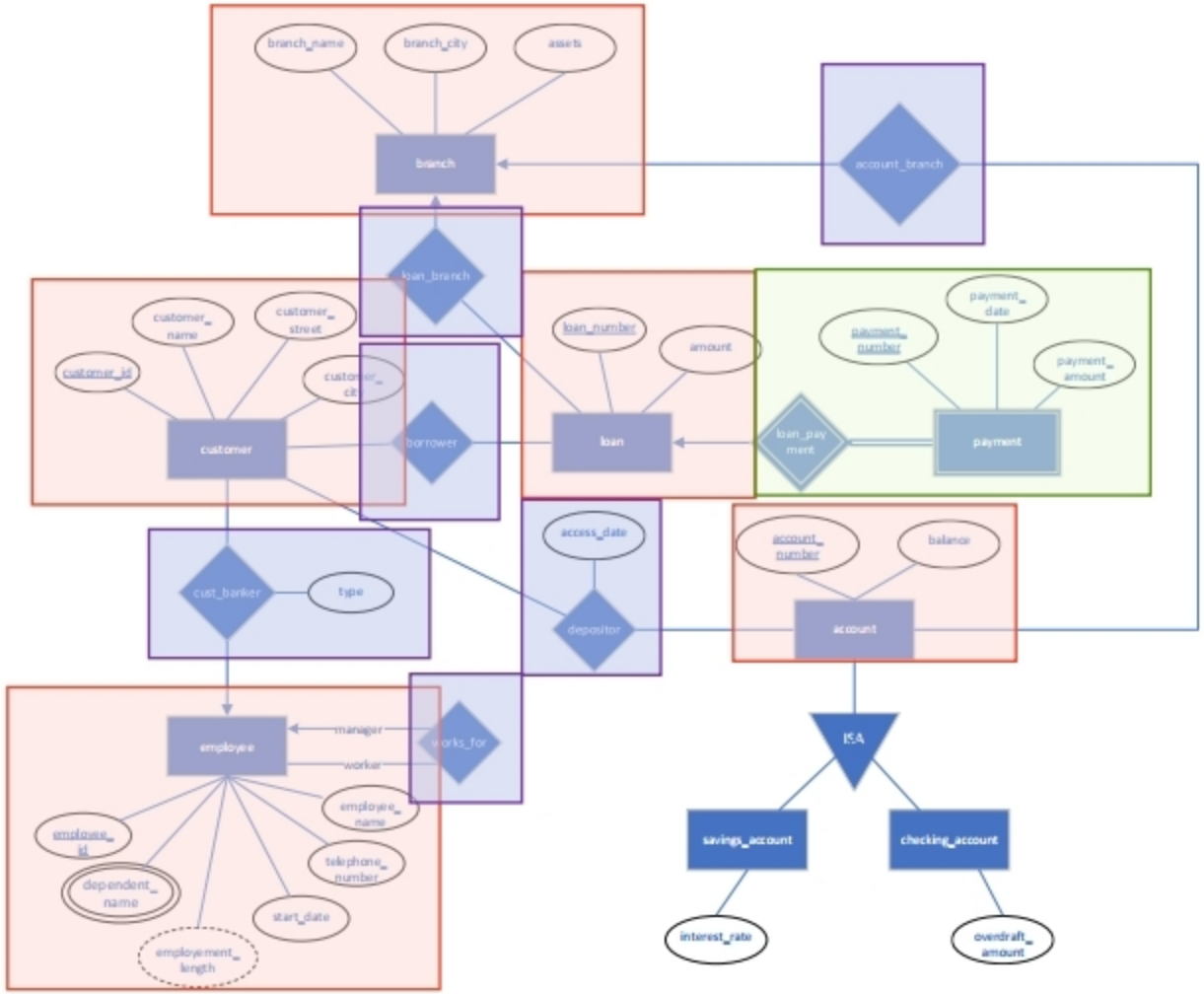
Mapping E-R data model to Relations

- Mapping E-R data model to relations
  4. N:1 Relation

branch = (branch_name, branch_city, assets)
loan = (loan_number, amount)
customer = (customer_id, customer_name, customer_street, customer_city)
employee = (employee_id, employee_name, telephone_number, start_date)
account = (account_number, balance)
dependent_name = (employee_id, dependent_name)

payment = (loan_number, payment_number, payment_date, payment_amount)

borrower = (customer_id, loan_number)
depositor = (customer_id, account_number, access_date)
loan_branch = (loan_number, branch_name)
account_branch = (account_number, branch_name)
cust_banker=(customer_id, employee_id, type)
works_for = (employee_id, manager_employee_id)



Mapping E-R data model to Relations

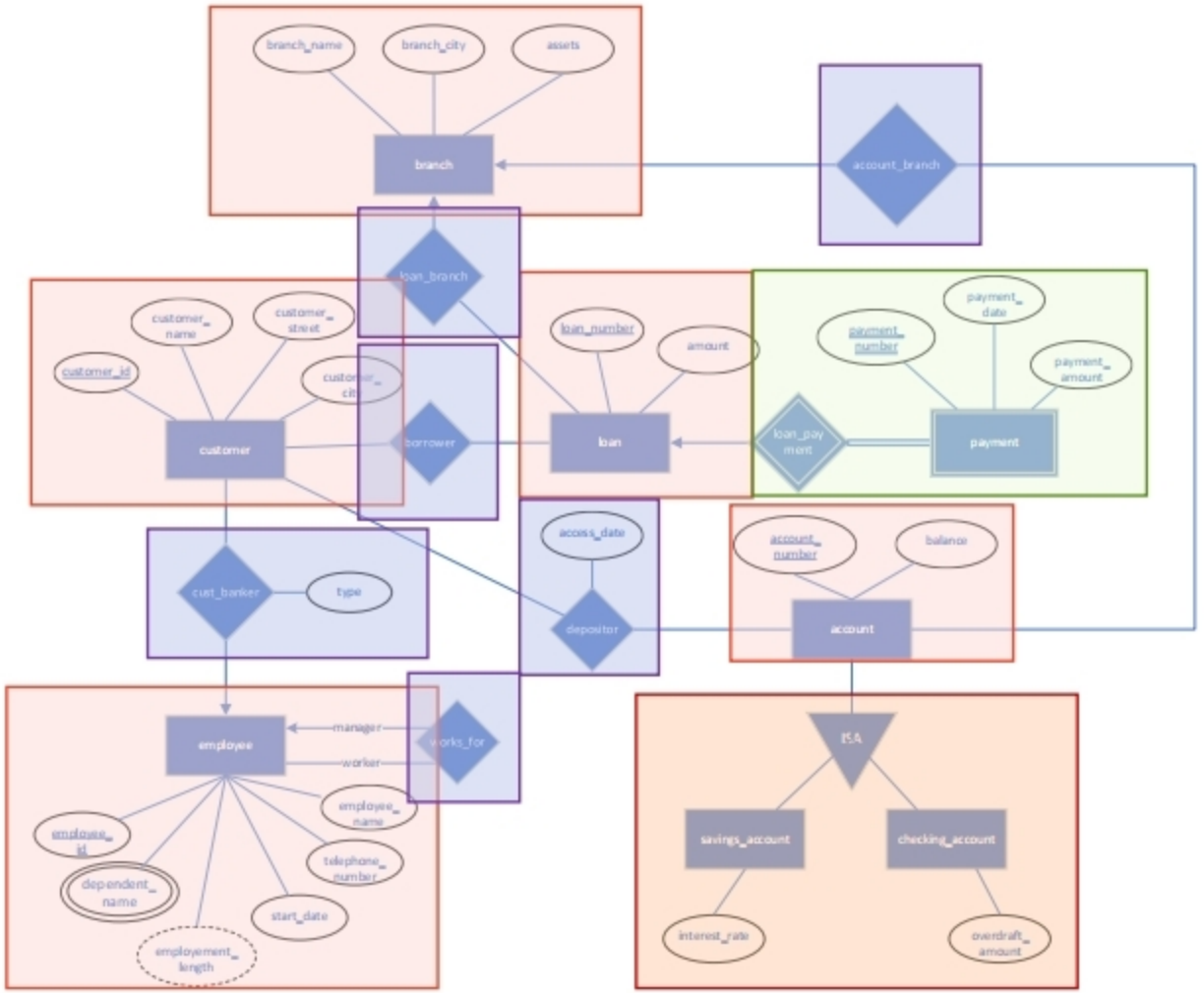
- Mapping E-R data model to relations
  5. Specialization & Generalization

branch = (branch_name, branch_city, assets)
loan = (loan_number, amount)
customer = (customer_id, customer_name, customer_street, customer_city)
employee = (employee_id, employee_name, telephone_number, start_date)
account = (account_number, balance)
dependent_name = (employee_id, dependent_name)

payment = (loan_number, payment_number, payment_date, payment_amount)

borrower = (customer_id, loan_number)
depositor = (customer_id, account_number, access_date)
loan_branch = (loan_number, branch_name)
account_branch = (account_number, branch_name)
cust_banker=(customer_id, employee_id, type)
works_for = (employee_id, manager_employee_id)

savings_account = (account_number, interest_rate)
checking_account = (account_number, overdraft_amount)



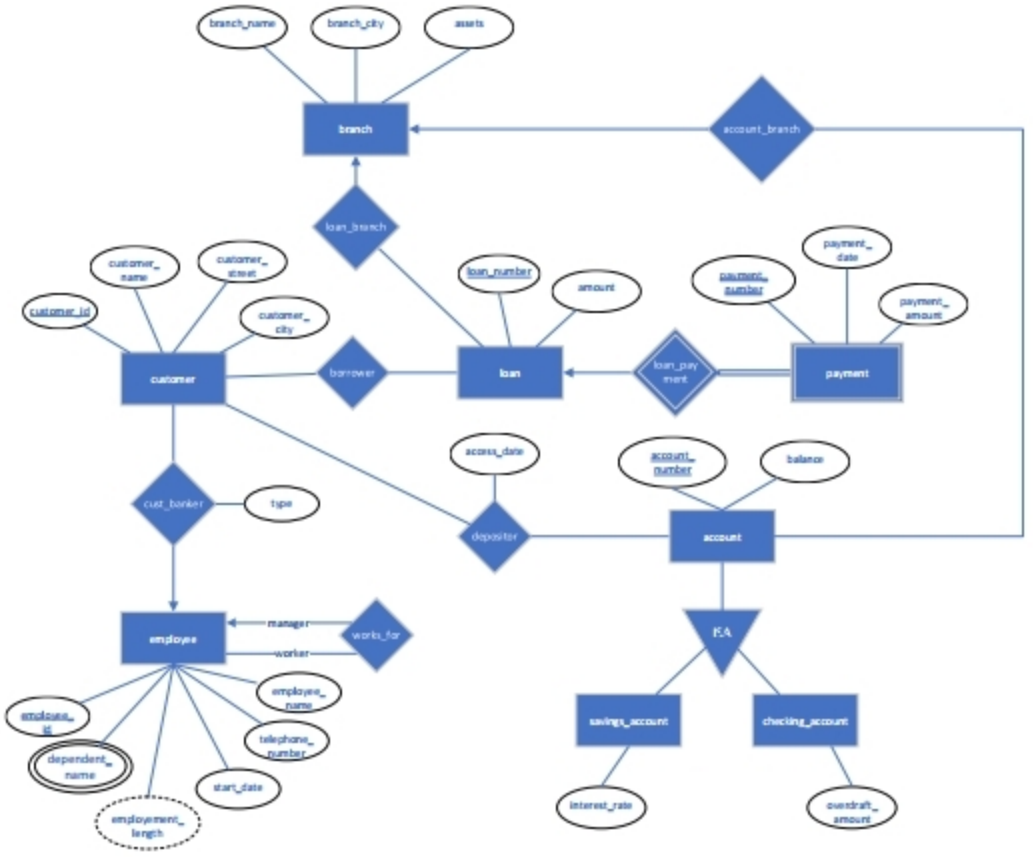
Introduction to normalization

- Relations
  - branch = (branch_name, branch_city, assets)
  - loan = (loan_number, amount)
  - customer = (customer_id, customer_name, customer_street, customer_city)
  - employee = (employee_id, employee_name, telephone_number, start_date)
  - account = (account_number, balance)
  - dependent_name = (employee_id, dependent_name)

  - payment = (loan_number, payment_number, payment_date, payment_amount)

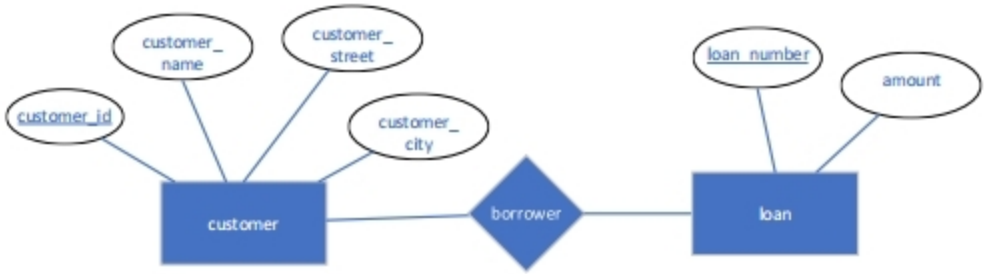
  - borrower = (customer_id, loan_number)
  - depositor = (customer_id, account_number, access_date)
  - loan_branch = (loan_number, branch_name)
  - account_branch = (account_number, branch_name)
  - cust_banker=(customer_id, employee_id, type)
  - works_for = (employee_id, manager_employee_id)

  - savings_account = (account_number, interest_rate)
  - checking_account = (account_number, overdraft_amount)



Can we make it better?

Introduction to normalization



- Less number of relations with joining relations: case 1
  - **loan = (loan_number, amount)**
  - **borrower = (customer_id, loan_number)**
  - customer = (customer_id, customer_name, customer_street, customer_city)
  - →
  - **borrower_loan = (customer_id, loan_number, amount)**
  - customer = (customer_id, customer_name, customer_street, customer_city)

loan relation

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

⋈

borrower relation

customer_id	loan_number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

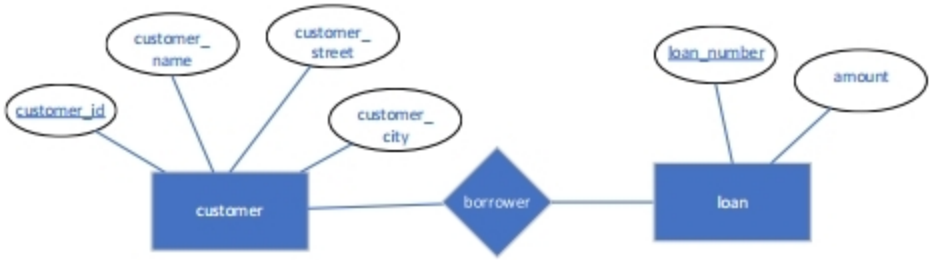
=

loan ⋈ borrower

customer_name	loan_number	amount
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-11	900
Smith	L-23	2000
Williams	L-17	1000

Introduction to normalization

- Less number of relations with joining relations: case 1
  - If James, Anthony, Jordan are co-owner they have a new loan #L-100 of $10000 and loan and borrower has n:m relationship
    - **Insert** three tuples
  - Yielding some problems
    - Data redundancy at insertion
    - Data inconsistency at update



loan relation

loan_number	amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500
L-100	10000

$\bowtie$

borrower relation

customer_name	loan_number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17
James	L-100
Anthony	L-100
Jordan	L-100

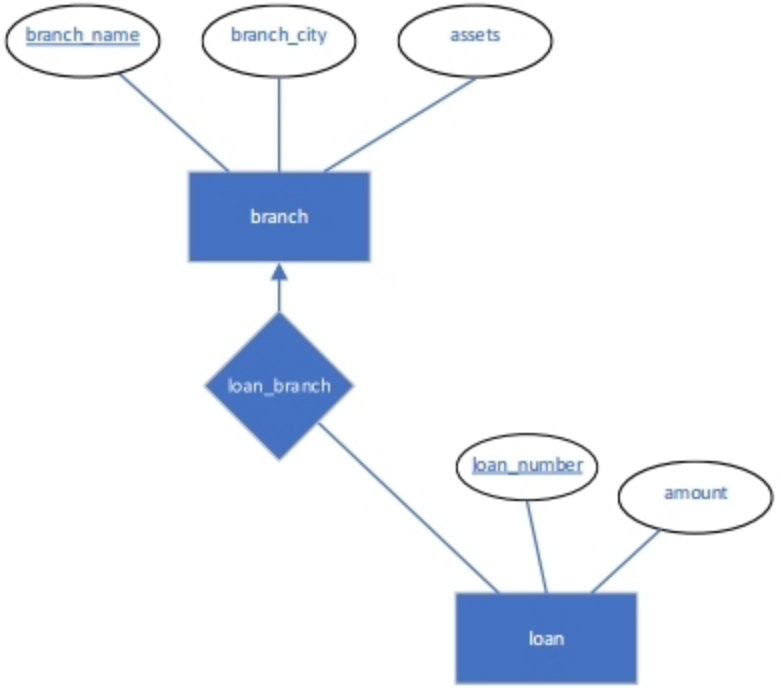
=

$loan \bowtie borrower$

customer_name	loan_number	amount
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-11	900
Smith	L-23	2000
Williams	L-17	1000
James	L-100	10000
Anthony	L-100	10000
Jordan	L-100	10000

Introduction to normalization

- Less number of relations with joining relations: case 2
  - **loan = (loan_number, amount)**
  - **loan_branch = (loan_number, branch_name)**
  - branch = (branch_name, branch_city, assets)
  - →
  - **loan_amt_br = (loan_number, amount, branch_name)**
  - branch = (branch_name, branch_city, assets)



loan relation

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

⋈

loan_branch **relation**

loan_number	branch_name
L-11	Brighton
L-14	Brighton
L-15	Brighton
L-16	North Town
L-17	North Town
L-23	North Town
L-93	Redwood

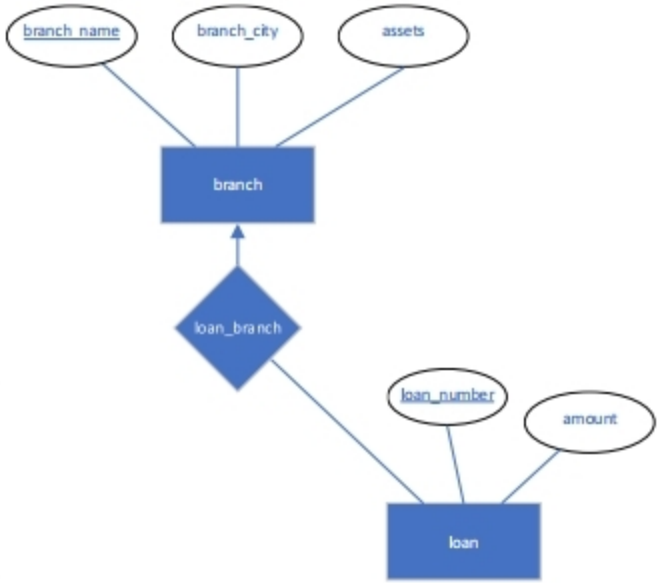
=

$loan \bowtie loan\_branch$

loan_number	amount	branch_name
L-11	900	Brighton
L-14	1500	Brighton
L-15	1500	Brighton
L-16	1300	North Town
L-17	1000	North Town
L-23	2000	North Town
L-93	500	Redwood

Introduction to normalization

- Less number of relations with joining relations: case 2
  - If 'L-100' loan is in 'Springfield' branch (new record) and loan & loan_branch is N:1 relationship
    - Insert one tuple
    - No redundancy
    - but **null** could be problematic



Lesson:
When two relation is merged,
the relationship between primary keys are important

loan relation

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

⋈

loan_branch **relation**

loan_number	branch_name
L-11	Brighton
L-14	Brighton
L-15	Brighton
L-16	North Town
L-17	North Town
L-23	North Town
L-93	Redwood
L-100	Springfield

=

$loan \bowtie loan\_branch$

loan_number	amount	branch_name
L-11	900	Brighton
L-14	1500	Brighton
L-15	1500	Brighton
L-16	1300	North Town
L-17	1000	North Town
L-23	2000	North Town
L-93	500	Redwood
L-100	**null**	Springfield

Anomalies

Introduction to normalization

- Data Anomaly
  - Yielded by
    - Poor design
    - e.g., Data Redundancy
    - e.g., if inter-related information is stored in one table
  - Type
    - Deletion Anomaly
    - Insertion Anomaly
    - Update Anomaly

course table

student_id	course_id	grade	semester
100	C413	A	4
100	E412	A	4
200	C123	B	3
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	4
400	C324	A	3
400	C413	B	3
400	E412	C	4
500	C312	B	2

Introduction to normalization

- Data Anomaly
  - Deletion Anomaly
    - Delete the fact that 200 takes C123 and get B (partial removal)
      - leads to the removal of (200, 'C123', 'B', 3)
      - cascading

course table

student_id	course_id	grade	semester
100	C413	A	4
100	E412	A	4
200	C123	B	3
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	4
400	C324	A	4
400	C413	B	4
400	E412	C	4
500	C312	B	2

course table

student_id	course_id	grade	semester
100	C413	A	4
100	E412	A	4
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	4
400	C324	A	4
400	C413	B	4
400	E412	C	4
500	C312	B	2

Remove a fact that 'the student '200' is third semester'

Introduction to normalization

- Data Anomaly
  - Insertion Anomaly
    - Insert the fact that 600 is a 2nd semester (partial insertion)
      - Must insert something in other columns (600, ?, ?, 2)

course table

student_id	course_id	grade	semester
100	C413	A	4
100	E412	A	4
200	C123	B	3
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	4
400	C324	A	4
400	C413	B	4
400	E412	C	4
500	C312	B	2

course table

student_id	course_id	grade	semester
100	C413	A	4
100	E412	A	4
200	C123	B	3
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	4
400	C324	A	4
400	C413	B	4
400	E412	C	4
500	C312	B	2
600	?	?	2

Introduction to normalization

- Data Anomaly
  - Update Anomaly
    - Update 400's semester from 4 to 3
      - Have to update whole related information

course table

student_id	course_id	grade	semester
100	C413	A	4
100	E412	A	4
200	C123	B	3
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	4
400	C324	A	4
400	C413	B	4
400	E412	C	4
500	C312	B	2

course table

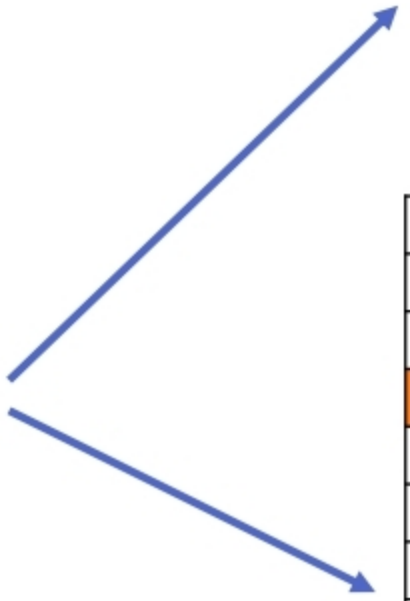
student_id	course_id	grade	semester
100	C413	A	4
100	E412	A	4
200	C123	B	3
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	3
400	C324	A	3
400	C413	B	3
400	E412	C	3
500	C312	B	2

Introduction to normalization

- Cause of Data Anomaly
  - inter-related information is stored in one table

course table

student_id	course_id	grade	semester
100	C413	A	4
100	E412	A	4
200	C123	B	3
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	4
400	C324	A	4
400	C413	B	4
400	E412	C	4
500	C312	B	2



student table

student_id	semester
100	4
200	3
300	1
400	3
500	2
600	2

course table

student_id	course_id	grade
100	C413	A
100	E412	A
~~200~~	~~C123~~	~~B~~
300	C312	A
300	C324	C
300	C413	A
400	C312	A
400	C324	A
400	C413	B
400	E412	C
500	C312	B

Introduction to normalization

- More number of relations
  - employee = (employee_id, employee_name, telephone_number, start_date)
  - →
  - employee1 = (employee_id, employee_name) employee2 = (employee_id, telephone_number, start_date)

employee_id	employee_name	telephone_number	start_date
…	…	…	…
123-45-6789	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
…	…	…	…

employee_id	employee_name
…	…
123-45-6789	Kim
987-65-4321	Kim
…	…

employee_id	telephone_number	start_date
…	…	…
123-45-6789	882-0000	1984-03-29
987-65-4321	869-9999	1981-01-16
…	…	…

employee_id	employee_name
…	…
123-45-6789	Kim
987-65-4321	Kim
…	…
991-19-8588	Jim

can avoid insertion anomaly

Introduction to normalization

Lesson:
Have to decompose a relation without **lossy decomposition** with **lossless decomposition**

- More number of relations
  - employee = (employee_id, employee_name, telephone_number, start_date)
  - →
  - employee1 = (employee_id, employee_name) employee2 = (employee_name, telephone_number, start_date)

employee_id	employee_name	telephone_number	start_date
…	…	…	…
123-45-6789	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
…	…	…	…

employee_id	employee_name
…	…
123-45-6789	Kim
987-65-4321	Kim
…	…

employee_name	telephone_number	start_date
…	…	…
Kim	882-0000	1984-03-29
Kim	869-9999	1981-01-16
…	…	…

employee_id	employee_name	telephone_number	start_date
…	…	…	…
123-45-6789	Kim	882-0000	1984-03-29
123-45-6789	Kim	869-9999	1981-01-16
987-65-4321	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
…	…	…	…

lossy decomposition

why? can you guess?

Introduction to normalization

- Resolving Data Anomaly
  - Analyzing **Functional Dependency**, data dependency between all the attributes
  - Processing **Normalization** well, the process of minimizing redundancy of relations by decomposing of attributes in order for each relation to have one dependency
    - BCNF
    - Lossless-join decomposition
    - Dependency preserving decomposition

Functional Dependency

- Legal
  - A relation is legal if the relation satisfies the characteristics below
- 1NF
  - If all the domain of R is atomic,
    - R is 1NF
  - All the legal relations are 1NF

Relational Model

- Characteristic of relational model
  - A tuple is unique in a relation

100	Jack	3
~~100~~	~~Jack~~	~~3~~
300	Brown	4

  - There is no order of each tuple

100	Jack	3
300	Brown	4

identical

300	Brown	4
100	Jack	3

  - There is no order of each attribute

Student Identifier (sid)	Name (sname)	Semester (semester)
100	Jack	3

identical

Student Identifier (sid)	Semester (semester)	Name (sname)
100	3	Jack

  - A domain is atomic

Student Identifier (sid)	Name (sname)	Semester (semester)
500	~~Gilldong, 길동~~	null

Non-atomic value is not allowable
Null value is allowed

2021-03-17 14

Functional Dependency

- Super Key
  - For each pair of $(t_1, t_2), t_1, t_2 \in r$,
    - If $t_1 \neq t_2 \rightarrow t_1[K] \neq t_2[K]$
      - K is a **super key** of R
- Hold
  - For $\alpha, \beta \; that \; \alpha \subseteq R, \beta \subseteq R$
    - For all the pairs of $(t_1, t_2), t_1, t_2 \in$ r,
      - If $t_1[\alpha] = t_2[\alpha] \rightarrow t_1[\beta] = t_2[\beta]$,
        - A **functional dependency** F: $\alpha \rightarrow \beta$ **holds** a schema R
        - β is **functionally determined** by α
        - The schema R **satisfies** F
- F: Super Key → a relation schema **holds** the schema
  - $t_1[K] = t_2[K] \rightarrow t_1[R] = t_2[R]$
  - Simply, $K \rightarrow R$
- Example
  - bor_loan = (customer_id, loan_number, amount)
  - customer_id, loan_number → customer_id, loan_number, amount
  - F: customer_id, loan_number → bor_loan
  - bor_loan **satisfies** F
  - r **satisfies** A→C
  - r does not satisfy C → A

bor_loan relation

customer_name	loan_number	amount
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-11	900
Smith	L-23	2000
Williams	L-17	1000

r relation

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

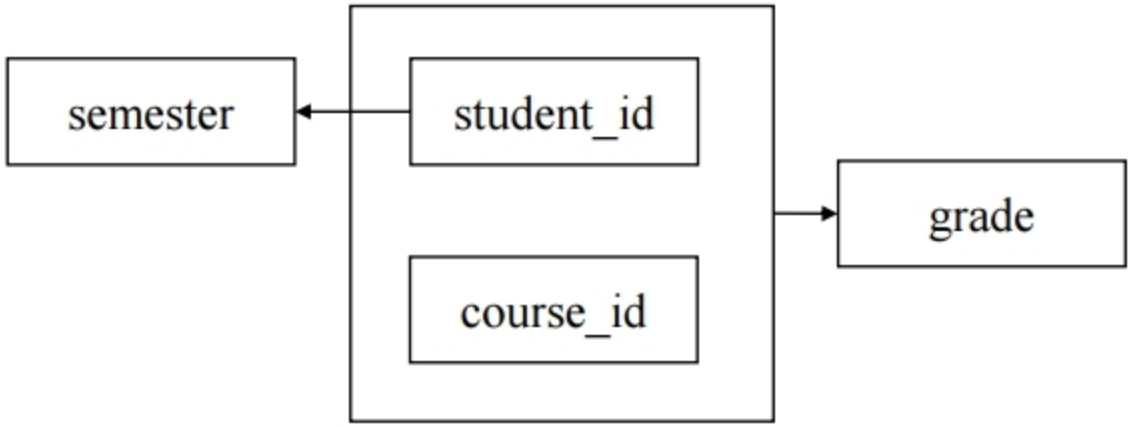
Functional Dependency

- Functional Dependency
  - $X \rightarrow Y$
    - Y is a functional dependent on X
    - every value(s) of X uniquely **determines** the value(s) of Y
    - R: relation
    - X: **determinant**, a set of attributes of R, usually primary key
    - Y: **dependent**, a set of attributes of R, usually non-key attributes
- FD Diagram
  - a diagram depicts functional dependency

course table

student_id	course_id	grade	semester
100	C413	A	4
100	E412	A	4
200	C123	B	3
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	4
400	C324	A	4
400	C413	B	4
400	E412	C	4
500	C312	B	2

course **relation**
(primary key: student_id, course_id)



- student_id, course_id → grade
- student_id → semester

Functional Dependency

- Trivial
  - F is trivial for every relation
  - F: $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$
  - e.g., A → A, AB → A if a relation has A attribute

- Closure of a functional dependency
  - R = (A,B,C)
  - If F: A →B, B→C hold R
    - We can infer A → C
  - A closure of F, F^+ includes A→C
    - F^+ indicates F ∪ all the functional dependencies inferred by F
    - A functional dependency $\beta \rightarrow \alpha \in F^+$ means $\beta \rightarrow \alpha$ **holds** a schema R

course table

student_id	course_id	grade	semester
100	C413	A	4
100	E412	A	4
200	C123	B	3
300	C312	A	1
300	C324	C	1
300	C413	A	1
400	C312	A	4
400	C324	A	4
400	C413	B	4
400	E412	C	4
500	C312	B	2

Functional Dependency

- Armstrong's axiom
  (Inference Rules of Functional Dependency)
  - Reflexivity
    - α *is an attribute set* & $\beta \subseteq \alpha$, then
      - $\alpha \rightarrow \beta$ **holds** a schema
  - Augmentation
    - $\alpha \rightarrow \beta$, then
      - $\gamma\alpha \rightarrow \gamma\beta$ or $\gamma\alpha \rightarrow \beta$ holds a schema
  - Transitive
    - $\alpha \rightarrow \beta \wedge \beta \rightarrow \gamma$, then
      - $\alpha \rightarrow \gamma$ holds a schema
  - Decomposition
    - $\alpha \rightarrow \beta\gamma$, then
      - $\alpha \rightarrow \beta$ or $\alpha \rightarrow \gamma$ holds a schema
  - Union
    - $\alpha \rightarrow \beta \wedge \alpha \rightarrow \gamma$, then
      - $\alpha \rightarrow \beta\gamma$ holds a schema
  - Pseudo-transitive
    - $\alpha \rightarrow \beta \wedge \gamma\beta \rightarrow \theta$, then
      - $\alpha\gamma \rightarrow \theta$

derived

Functional Dependency

- Armstrong's axiom
  (Inference Rules of Functional Dependency)
  - Example
    - R = (A,B,C,G,H,I)
    - F: {A→B, A→C, CG→H, CG→I,B→H}

  - F^+ includes
    - A→H from A→B and B→H (transitive)
    - CG → HI from CG → H and CG → I (union)
    - AG → I from A → C and CG → I (pseudo-transitive)

Functional Dependency

- A procedure to compute a closure of functional dependency

  - $F^+ = F$
  - repeat
    - for each f in F^+
      - apply reflexivity and augmentation rules on f
      - add the results to F^+
    - for each pair f1, f2 in F^+
      - if f1 and f2 can be combined using transitivity
        - add the results to F^+
  - until F^+ does not change any further

Functional Dependency

- Closure of an attribute set
  - $\alpha \rightarrow \beta$
    - β is functionally determined by α
  - α^+: a closure of α
    - a set of attributes determined by α under F
    - $e.g., \beta \in \alpha^+$
- A procedure to compute a closure of an attribute set
  - result:=a;
  - while(changes to result) do
    - for each $\beta \rightarrow \gamma$ in F do
      - begin
        - if $\beta \subseteq result$ then $result := result \cup \gamma$;
      - end

Functional Dependency

- A procedure to compute a closure of an attribute set
  - Example
    - R = (A,B,C,G,H,I)
    - F: {A→B, A→C, CG→H, CG→I,B→H}

- result:=a;
- while(changes to result) do
  - for each $\beta \rightarrow \gamma$ in F do
    - begin
      - if $\beta \subseteq result$ then $result := result \cup \gamma$;
    - end

- A^+:
  - A
  - AB
  - ABC
  - ABCH
- B^+:
  - B
  - BH

- $(AG)^+$:
  - AG
  - ABG
  - ABCG
  - ABCGH
  - ABCGHI

- $(CG)^+$:

Functional Dependency

- Applications of a closure of an attribute set
  1. Check super key
     - If a closure of an attribute set includes all the attributes in a relation schema, the attribute set is a super key
     - Example
       - R = (A,B,C,G,H,I)
       - F: {A→B, A→C, CG→H, CG→I,B→H}
       - $(AG)^+$:
         - AG
         - ABG
         - ABCG
         - ABCGH
         - ABCGHI

Functional Dependency

- Applications of a closure of an attribute set
  2. Check a functional dependency **holds** a relation schema
     - Remind: A functional dependency $\alpha \rightarrow \beta \in F^+$ means $\alpha \rightarrow \beta$ holds a schema R
     - If $\beta \subseteq \alpha^+$, $\alpha \rightarrow \beta$ holds a schema R
     - Example
       - R = (A,B,C,G,H,I)
       - F: {A→B, A→C, CG→H, CG→I,B→H}
       - A^+:
         - ABCH
       - B^+:
         - BH
       - $(AG)^+$:
         - ABCGHI

$A \rightarrow ABCH$ holds R?

B $\rightarrow AH$ holds R?

$AG \rightarrow BCGH$ holds R?

Functional Dependency

- Applications of a closure of an attribute set
  3. Compute F^+
     - For every $\gamma \subseteq R$, compute γ^+
     - Then, for every $S \subseteq \gamma^+$, print $\gamma \rightarrow S$ that is F^+

     - R = (A, B, C, D, E)
     - F: {A→B, B→C, C→D, D→A}

     - $A^+ = ABCD$ → A → ABCD
     - $B^+ = ABCD$ → B → ABCD
     - $C^+ = ABCD$ → C → ABCD
     - $D^+ = ABCD$ → D → ABCD
     - $E^+ = E$ → E → E
     - $(AB)^+ = ABCD$ → AB → ABCD
     - $(BC)^+ = ABCD$ → BC → ABCD
     - …

Is it candidate key?

- $AE^+ = ABCDE$ → **AE** → **ABCDE**
- $BE^+ = ABCDE$ → BE → ABCDE
- …

Learn yourself to find Candidate keys

Functional Dependency

- A canonical cover of F, F_c
  - F does not contain extraneous attribute
  - Extraneous attribute
    - If the attribute(s) is/are removed in a functional dependency, its closure does not change.

  - Example
    - R = (A,B,C)
    - F: {~~A→BC~~, B→C, A→B, ~~AB→C~~}

    - AB→C is extraneous because there is B→C (augmentation)
    - A→BC is extraneous because A→B and B→C infers A→BC

    - $F_c = \{A \rightarrow B, B \rightarrow C\}$

- An algorithm to find a canonical cover is
  - NP-Hard
  - Could have two or more covers

Functional Dependency

- BCNF
  - For every F: $\alpha \rightarrow \beta$ that $\alpha \subseteq R, \beta \subseteq R$
  - For every f $\in F^+$, f holds, at least, one of the following condition, R is BCNF
    - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
    - α is a super key of R

  - Example
    - bor_loan = (<u>customer_id, loan_number</u>, amount)
      - F: { loan_number → amount } holds R

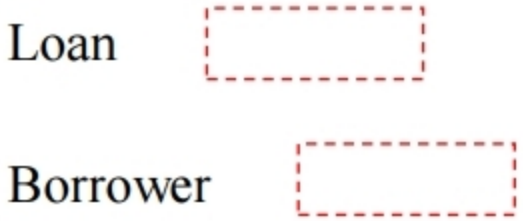
    - Check Bor_loan is not BCNF
      - because loan_number is not a super key

bor_loan relation

customer_name	loan_number	amount
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-11	900
Smith	L-23	2000
Williams	L-17	1000

Functional Dependency

- BCNF
  - Example
    - loan = (loan_number, amount)
      - F: {loan_number→amount}
    - borrower = (customer_id, loan_number)
      - F: {}



loan relation

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

borrower relation

customer_name	loan_number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

Functional Dependency

- BCNF
  - A way to convert non-BCNF schema to BCNF schema
    - If R is non-BCNF, for a non-trivial $\alpha \rightarrow \beta$ where α is not a super key
      - The schema decomposes into two relation schema
        - $(\alpha \cup \beta)$
        - $(R - (\beta - \alpha))$
      - If a decomposed schema is not BCNF, do decomposition again
  - Example
    - bor_loan = (customer_name, loan_number, amount)
    - F: {loan_number → amount}
    - loan_number → amount holds R but loan_number is not a super key
    - $(\alpha \cup \beta) = (loan\_number, amount)$
    - $\big(R - (\beta - \alpha)\big) = (customer\_name, loan\_number)$

bor_loan relation

customer_name	loan_number	amount
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-11	900
Smith	L-23	2000
Williams	L-17	1000

Functional Dependency

- BCNF and dependency preserving
  - Course = (<u>student_id, course_name</u>, prof)
  - FD
    - (student_id,course_name)→prof
    - prof→course_name
  - Course is not BCNF because prof is not super key
  - Thus, decompose to
    - Course1 = (<u>prof</u>, course_name)
    - Course2 = (<u>student_id, prof</u>)

<u>student_id</u>	<u>course_name</u>	prof
1	db	A
2	network	B
3	ai	C
4	db	A
5	db	D
6	network	B

$(\alpha \cup \beta)$
$(R - (\beta - \alpha))$

  - Problem
    - There is no (student_id,course_name)→prof in Course1 and Course2
    - Not **dependency preserving decomposition**

Functional Dependency

- Dependency preserving decomposition
  - A decomposition $D = \{R_1, R_2, R_3, \dots, R_n\}$ of R is **dependency preserving** with regard to F if
    - $(F_1 \cup F_2 \cup \cdots \cup F_n)^+ \equiv F^+$
      - Each F_i $(i = 1,2, \dots, n)$ is a **restriction** of F only using attributes in R_i

Functional Dependency

- Dependency preserving decomposition
  - Example
    - R = (A, B, C, D, E) F: {A→B, B→C, C→D, D→A}
    - to
    - R_1 = (A,B,C) and R_2 = (C,D,E)
    - Is it dependency preserving decomposition?
      - $(F_1 \cup F_2)^+ \equiv F^+$
  - How to compute F_i?
    - $R_i = (A_1, A_2, \ldots, A_n)$, then
    - $F_i = (A_1 \cup A_2 \cup \cdots \cup A_n)^+$
    - e.g., F_1 of $R_1 = (\mathrm{A} \cup B \cup C)^+$
  - Then, let's compute F_i

Functional Dependency

- Dependency preserving decomposition
  - Example
    - R = (A, B, C, D, E) F: {A→B, B→C, C→D, D→A}
    - to
    - R1 = (A,B,C) and R2 = (C,D,E)
  - F_1
    - $A^+ = \cancel{ABCD}, thus\ \{A \rightarrow BC\}$
    - $B^+ = CA, thus\ \{B \rightarrow CA\}$
    - $C^+ = AB, thus\ \{C \rightarrow AB\}$
    - $\cancel{(AB)^+ = ABCD}$, non necessary, covered by A^+ and B^+ ➔ $F_1^+ \supseteq \{A \rightarrow BC, B \rightarrow CA, C \rightarrow AB\}$
    - $\cancel{(AC)^+ = ABCD}$ $\{A \rightarrow BC, B \rightarrow CA, C \rightarrow AB\}$ holds R_1
    - $\cancel{(BC)^+ = ABCD}$
    - $\cancel{(ABC)^+ = ABCD}$
  - F_2
    - $C^+ = \cancel{ABCD}, thus\ \{C \rightarrow D\}$
    - $D^+ = \cancel{ABCD}, thus\ \{D \rightarrow C\}$ ➔ $F_2^+ \supseteq \{C \rightarrow D, D \rightarrow C\}$
    - $(CD)^+ = \cancel{ABCD}$ $\{C \rightarrow D, D \rightarrow C\}$ holds R_2
  - $F_1 \cup F_2$
    - $\{A \rightarrow BC, B \rightarrow CA, C \rightarrow AB, C \rightarrow D, D \rightarrow C\}$ ➔ $(F_1 \cup F_2)^+ \supseteq \{A \rightarrow BC, B \rightarrow CA, C \rightarrow AB, C \rightarrow D, D \rightarrow C\}$
      $\{A \rightarrow BC, B \rightarrow CA, C \rightarrow AB, C \rightarrow D, D \rightarrow C\}$ holds
  - $(F_1 \cup F_2)^+ \equiv F^+$?
    - F^+ covers $(F1 \cup F2)^+$, $((F1 \cup F2)^+ \subseteq F^+)$? definitely, yes
    - $(F1 \cup F2)^+$ $covers$ F^+, $((F1 \cup F2)^+ \supseteq F^+)$? check this

Functional Dependency

- Dependency preserving decomposition
  - Example
    - R = (A, B, C, D, E) to
    - $R_1 = (A, B, C)$ $R_2 = (C, D, E)$
  - $(F_1 \cup F_2)^+ \; covers \; F^+$, $((F_1 \cup F_2)^+ \supseteq F^+)$? YES
  - $F_1 \cup F_2$
    - F: {A→B, B→C, C→D, D→A}
    - $\{A \rightarrow BC, B \rightarrow CA, C \rightarrow AB, C \rightarrow D, D \rightarrow C\}$ covers D→ABC

Functional Dependency

- Dependency preserving decomposition
  - Example
    - R = (A, B, C)
      - (A,B)→C
      - C→B
    - R_1 = (B,C)
    - R_2 = (A,C)

A	B	C
1	db	A
2	network	B
3	ai	C
4	db	A
5	db	D
6	network	B

  - F_1
  - F_2
  - $F_1 \cup F_2$
  - F

Functional Dependency

- Dependency preserving decomposition
  - Example
    - R = (A, B, C, D)
      - {A→B,A→C,C→D}
    - $R_1 = (A, B, C)$
    - $R_2 = (C, D)$ ←*Dependency Preserving Decomposition*

  - F_1

  - F_2

  - $F_1 \cup F_2$

  - F

Functional Dependency

- Dependency preserving decomposition
  - Example
    - R = (A, B, C, D)
      - {A→B,B→C}
    - $R_1 = (A, B)$
    - $R_2 = (A, C)$ ← Not *Dependency Preserving Decomposition*
    - $R_3 = (A, D)$

$$(F_1 \cup F_2 \cup \cdots \cup F_n)^+ \equiv F^+$$

- F_1
- F_2
- F_3
- $F_1 \cup F_2 \cup F_3$
- F

Functional Dependency

- Lossless-join Decomposition
  - When R is decomposed into R1 and R2, if $\pi_{R_1}(r) \bowtie \pi_{R_2}(r) = \mathrm{r}$
    - we say the decomposition is **lossless-join decomposition**
  - else
    - we say the decomposition is **lossy decomposition**

  - One way to check **lossless-join decomposition**
    - satisfies either of the followings under F^+
      - $R_1 \cap R_2 \rightarrow R_1$ or
      - $R_1 \cap R_2 \rightarrow R_2$

  - Example
    - bor_loan = (customer_id, loan_number, amount)
    - →
    - borrower = (customer_id, loan_number)
    - loan = (loan_number, amount)

    - The attribute intersection is loan_number and it is a super key of loan relation
    - We say the decomposition is **lossless-join decomposition**
    - Is BCNF decomposition lossless? why?

FD & Normalization

- Normalization
  - Decomposition of attributes in a relation in order for each relation to have one dependency

  - Goal of normalization with regard to functional dependency
    - BCNF
    - Lossless-join decomposition
    - Dependency preserving decomposition

FD & Normalization

- Normalization
  - Types
    - 1NF, first normal form
      - Each domain of attributes in a relation is atomic
    - 2NF, second normal form
      - 1NF + eliminating partial functional dependency
    - 3NF, third normal form
      - 2NF + eliminating transitive functional dependency
    - BCNF, Boyce-Codd normal form
      - 3NF + for any dependency A $\rightarrow$ *B*, A should be a super key
    - 4NF, fourth normal form
    - 5NF, fifth normal form

FD & Normalization

- 1NF
  - A relation meets four characteristics is 1NF
    - Revisit

4. Relational Model.pptx

- Characteristic of relational model
  - A tuple is unique in a relation

100	Jack	3
~~100~~	~~Jack~~	~~3~~
300	Brown	4

  - There is no order of each tuple

100	Jack	3
300	Brown	4

identical

300	Brown	4
100	Jack	3

  - There is no order of each attribute

Student Identifier (sid)	Name (sname)	Semester (semester)
100	Jack	3

identical

Student Identifier (sid)	Semester (semester)	Name (sname)
100	3	Jack

  - A domain is atomic

Student Identifier (sid)	Name (sname)	Semester (semester)
500	~~Gilldong, 길동~~	null

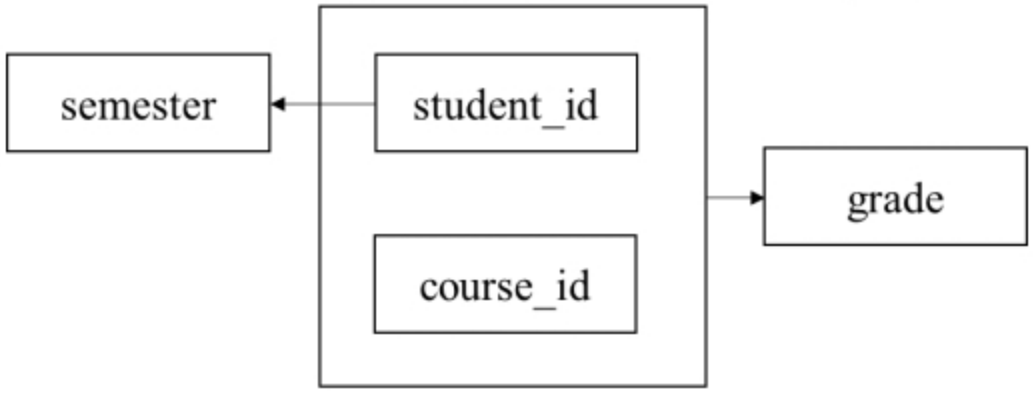
Non-atomic value is not allowable
Null value is allowed

FD & Normalization

- 2NF
  - 1NF + eliminating partial functional dependency
    - primary key: student_id, course_id
      - non key: semester, grade
        - student_id, course_id → semester
        - grade → semester

Partial functional dependency

course **relation**
(primary key: student_id, course_id)



- student_id, course_id → grade
- student_id → semester

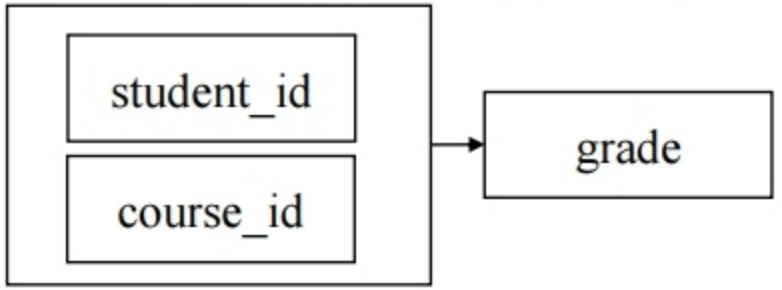
Fully functional dependency

Partial functional dependency

A subset of primary key determines determinant

student **relation**
(primary key: student_id)

course **relation**
(primary key: student_id, course_id)

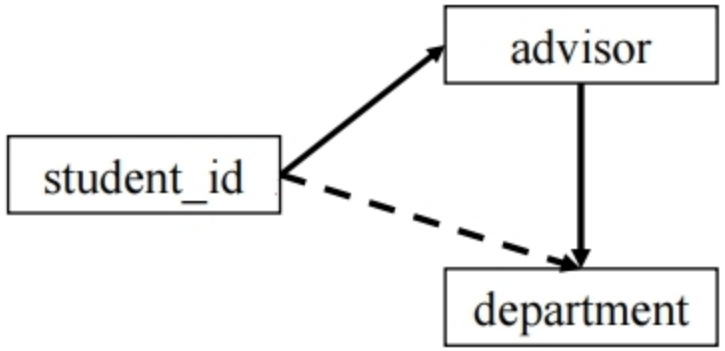


- student_id, course_id → grade
- student_id → semester

FD & Normalization

- 3NF
  - 2NF + eliminating transitive functional dependency
  - **Prime attribute**: an attribute that is a part of the candidate key
  - **Non-prime attribute**: an attribute that is not a part of the candidate key
  - transitive functional dependency exists
    - if there is a functional dependency between non-prime attributes

advisor **relation**
(primary key: student_id)



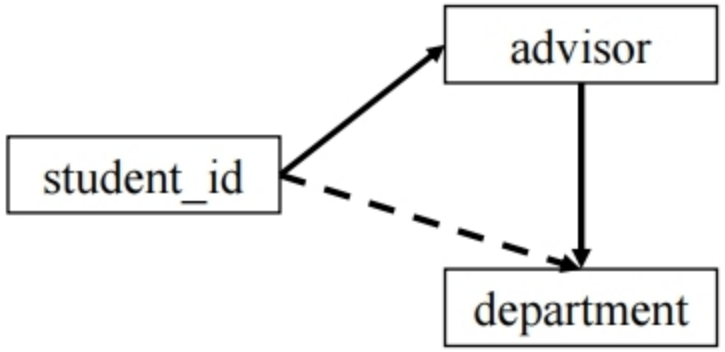
- Prime attribute
  - student_id
- Non-prime attribute
  - advisor
  - department
- there is a dependency between
  - $advisor \rightarrow department$

- *Functional Dependency*
  - $student\_id \rightarrow advisor$
  - $advisor \rightarrow department$
- Then, we say there is a transitive functional dependency, $student\_id \rightarrow department$

FD & Normalization

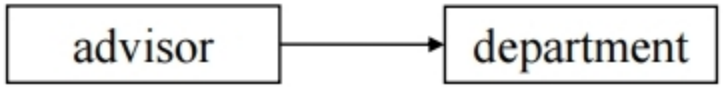
- 3NF
  - 2NF + eliminating transitive functional dependency
    - Example 2

advisor **relation**
(primary key: student_id)



- $student\_id \rightarrow advisor$
- $advisor \rightarrow department$
- $student\_id \rightarrow department$: : transitive functional dependency

advisor **relation**
(primary key: advisor)



- $advisor \rightarrow department$

HOW?

student **relation**
(primary key: student_id)

- $student\_id \rightarrow department$

FD & Normalization

- 3NF synthesis algorithm
  - let F_c is a canonical cover of F
  - i:=0;
  - For each functional dependency $\alpha \rightarrow \beta$ in F_c do
    - if non of the schemas $R_j, j = 1,2, \ldots, i$ contains $\alpha\beta$
      - then begin
        - i:=i+1;
        - $R_i := \alpha\beta$;
      - end
  - if non of the schemas $R_j, j = 1,2, \ldots, i$ contains a candidate key for R
    - then begin
      - i:=i+1;
      - $R_i :=$ any candidate key for R;
    - end
  - return $(R_1, R_2, \ldots, R_i)$

FD & Normalization

- 3NF
  - 2NF + eliminating transitive functional dependency

  - Example
    - Advisor = (<u>student_id</u>, advisor, department)
    - F: $\{student\_id \rightarrow advisor, advisor \rightarrow department\}$
    - F_c: $\{ student\_id \rightarrow advisor, advisor \rightarrow department\}$

    - In out-most for loop,
      - R1 = (<u>student_id</u>, advisor)
      - R2 = (<u>advisor</u>, department)
    - In out-most if statement,
      - R1 already contains a candidate key for R
- 3NF is
  - Lossless-join decomposition
  - Dependency preserving decomposition

- 3NF synthesis algorithm
  - let F_c is a minimal set of F
  - i:=0;
  - For each functional dependency $\alpha \rightarrow \beta$ in F_c do
    - if non of the schemas $R_j, j = 1,2, \dots, i$ contains $\alpha\beta$
      - then begin
        - i:=i+1;
        - $R_i := \alpha\beta$;
      - end
  - if non of the schemas $R_j, j = 1,2, \dots, i$ contains a candidate key for R
    - then begin
      - i:=i+1;
      - R_i := any candidate key for R;
    - end
  - return $(R_1, R_2, \dots, R_i)$

FD & Normalization

- 3NF
  - 2NF + eliminating transitive functional dependency

  - Example
    - R = (A,B,C)
    - F: {~~A→BC~~, B→C, A→B, ~~AB→C~~}
    - $F_c = \{A \rightarrow B, B \rightarrow C\}$

    - In out-most for loop,
      - R1 = (A,B)
      - R2 = (B,C)
    - In out-most if statement,
      - R1 already contains a candidate key for R

- 3NF is
  - Lossless-join decomposition
  - Dependency preserving decomposition

- 3NF synthesis algorithm
  - let F_c is a minimal set of F
  - i:=0;
  - For each functional dependency $\alpha \rightarrow \beta$ in F_c do
    - if non of the schemas $R_j, j = 1,2, \dots, i$ contains $\alpha\beta$
      - then begin
        - i:=i+1;
        - $R_i := \alpha\beta$;
      - end
  - if non of the schemas $R_j, j = 1,2, \dots, i$ contains a candidate key for R
    - then begin
      - i:=i+1;
      - R_i := any candidate key for R;
    - end
  - return $(R_1, R_2, \dots, R_i)$

FD & Normalization

- BCNF
  - 3NF + for any dependency $A \rightarrow B$, A should be a super key

  - BCNF Decomposition Algorithm
    - result := {R};
    - compute F^+;
    - while (true)
      - if (there is a schema R_i in result that is not in BCNF)
        - then begin
          - let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+, and $\alpha \cap \beta = \emptyset$;
          - result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
        - end
      - else
        - break;

  - The above algorithm ensures lossless-join decomposition
  - while the algorithm does not ensure dependency preserving decomposition

FD & Normalization

- BCNF
  - Example
    - bor_loan = (customer_id, loan_number, amount)
      - F: loan_number → amount
      - F+
        - loan_number → amount

BCNF Decomposition Algorithm

- result := {R};
- compute F^+;
- while (true)
  - if (there is a schema R_i in result that is not in BCNF)
    - then begin
      - let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+, and $\alpha \cap \beta = \emptyset$;
      - result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
    - end
  - else
    - break;

bor_loan relation

customer_name	loan_number	amount
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-11	900
Smith	L-23	2000
Williams	L-17	1000

FD & Normalization

- BCNF
  - Register = (**sid, title**, grade, dname, office)
  - F: {sid,title→grade, sid→dname, dname→office}

BCNF Decomposition Algorithm

- result := {R};
- compute F^+;
- while (true)
  - if (there is a schema R_i in result that is not in BCNF)
    - then begin
      - let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+, and $\alpha \cap \beta = \emptyset$;
      - result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
    - end
  - else
    - break;

FD & Normalization

- Anomaly and Normalization
  - Register = (**stu_id, title**, grade, dept_name, office) **1NF**
  - F: {stu_id,title→grade, stu_id→dept_name, dept_name→office}

stu_id	title	grade	dept_name	office
1292001	CS101	B+	CS -> IE	920 -> 923
1292001	DB	A+	CS -> IE	920 -> 923
1292001	OS	A	CS -> IE	920 -> 923
1292002	DS	A	CS	920
1292002	DB	B+	CS	920
1292002	ALG	C+	CS	920
1292003	DB	B	CS	920
1292003	AI	A+	CS	920
1292301	~~DS~~	~~C+~~	IE	923
1292502	?	?	EE	925

- Register is not 2NF due to , thus
  - Insertion Anomaly
    -
  - Deletion Anomaly
    -
  - Update Anomaly
    -

FD & Normalization

- Anomaly and Normalization
  - Takes = (stu_id, title, grade), Student = (stu_id, dept_name, office) **2NF**
  - F: {stu_id,title→grade, stu_id→dept_name, dept_name→office}

stu_id	title	grade	dept_name	office
1292001	CS101	B+	CS -> IE	920 -> 923
1292001	DB	A+	CS -> IE	920 -> 923
1292001	OS	A	CS -> IE	920 -> 923
1292002	DS	A	CS	920
1292002	DB	B+	CS	920
1292002	ALG	C+	CS	920
1292003	DB	B	CS	920
1292003	AI	A+	CS	920
1292301	DS	C+	IE	923
1292502	?	?	EE	925

takes **relation**

student **relation**

- Takes and Student preserve F
- Takes and Student are 2NF, thus
  - Insertion Anomaly not happen
  - Deletion Anomaly not happen
  - Update Anomaly not happen
- Student is not 3NF, thus
  - Still yield anomalies

FD & Normalization

- Anomaly and Normalization
  - Takes = (stu_id, title, grade), Student = (stu_id, dept_name, office) **2NF**
  - F: {stu_id,title→grade, stu_id→dept_name, dept_name→office}

takes **relation**

stu_id	title	grade
1292001	CS101	B+
1292001	DB	A+
1292001	OS	A
1292002	DS	A
1292002	DB	B+
1292002	ALG	C+
1292003	DB	B
1292003	AI	A+
1292301	DS	C+

student relation

stu_id	dept_name	office
1292001	CS	920->927
1292002	CS	920->927
1292003	CS	920->927
~~1292301~~	IE	923
?	EE	925

- Student is not 3NF due to [ ], thus
  - Insertion Anomaly
    - [ ]
  - Deletion Anomaly
    - [ ]
  - Update Anomaly
    - [ ]

FD & Normalization

- Anomaly and Normalization
  - Takes = (stu_id, title, grade), Student = (stu_id, dept_name), Department = (dept_name, office)
  - F: {stu_id,title→grade, stu_id→dept_name, dept_name→office}

3NF

takes **relation**

stu_id	title	grade
1292001	CS101	B+
1292001	DB	A+
1292001	OS	A
1292002	DS	A
1292002	DB	B+
1292002	ALG	C+
1292003	DB	B
1292003	AI	A+
1292301	DS	C+

student relation

stu_id	dept_name	office
1292001	CS	920->927
1292002	CS	920->927
1292003	CS	920->927
~~1292301~~	IE	923
?	EE	925

- The relations preserve F and the decomposition is not lossless
- Takes, Student, Department are 3NF, thus
  - Insertion Anomaly not happen
  - Deletion Anomaly not happen
  - Update Anomaly not happen

FD & Normalization

- Anomaly and Normalization
  - bor_loan = (customer_id, loan_number, amount)
    - F: { loan_number → amount } holds R

bor_loan relation

customer_name	loan_number	amount
Adams	L-16	1300
Curry	L-93	500
~~Hayes~~	~~L-15~~	~~1500~~
Jackson	L-14	1500
Jones	L-17	1000->1500
Smith	L-11	900
~~Smith~~	L-23	2000
Williams	L-17	1000->1500
?	L-20	350

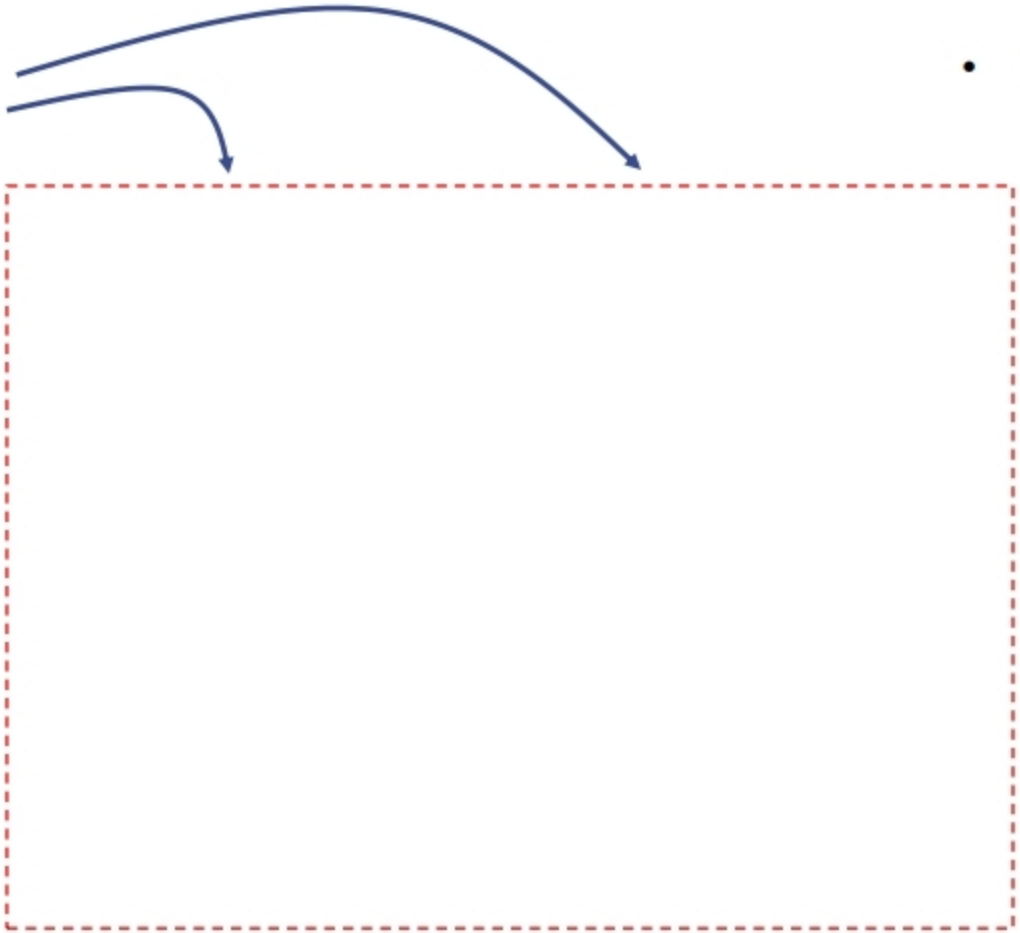
- bor_loan is not BCNF, thus
  - Insertion Anomaly
  - Deletion Anomaly
  - Update Anomaly

FD & Normalization

- Anomaly and Normalization
  - bor_loan = (customer_id, loan_number, amount)
    - F: { loan_number → amount } holds R

bor_loan relation

customer_name	loan_number	amount
Adams	L-16	1300
Curry	L-93	500
Hayes	~~L-15~~	~~1500~~
Jackson	L-14	1500
Jones	L-17	1000->1500
Smith	L-11	900
Smith	L-23	2000
Williams	L-17	1000->1500
?	L-20	350



- The relations are BCNF, thus
  - Insertion Anomaly not happen
  - Deletion Anomaly
  - Update Anomaly

FD & Normalization

- Goal of normalization with regard to functional dependency
  - BCNF
  - Lossless-join dependency
  - Dependency preserving decomposition

- BCNF vs. 3NF
  - 3NF
    - Advantage: Dependency preserving decomposition
    - Disadvantage: More Anomalies
  - BCNF
    - Advantage: Less Anomalies
    - Disadvantage: Sometimes, dependency preserving decomposition could not be achieved

- If BCNF does not preserve F, choose 3NF
- else choose BCNF

Multivalued Dependency

- Consider the following schema and functional dependencies
  - cust_loan = (loan_number, customer_id, customer_street, customer_city)
  - F: {}
  - Note: customers may have several addresses

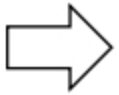
loan_number	customer_id	customer_street	customer_city
L-11	C-01	Spring	Pittsfield
L-11	C-01	Senator	Brooklyn
L-11	C-01	North	Rye
L-12	C-01	Spring	Pittsfield
L-12	C-01	Senator	Brooklyn
L-12	C-01	North	Rye
L-13	C-02	Sand Hill	Woodside

Repeatedly store the addresses for each pair of loan_number and customer_id

Multivalued Dependency

- Consider the following schema and functional dependencies
  - cust_loan = (loan_number, customer_id, customer_street, customer_city)
  - →
  - loan_cust_id = (loan_number, customer_id)
  - cust_residence = (customer_id, customer_street, customer_city)

loan_number	customer_id	customer_street	customer_city
L-11	C-01	Spring	Pittsfield
L-11	C-01	Senator	Brooklyn
L-11	C-01	North	Rye
L-12	C-01	Spring	Pittsfield
L-12	C-01	Senator	Brooklyn
L-12	C-01	North	Rye
L-13	C-02	Sand Hill	Woodside



customer_id	customer_street	customer_city
C-01	Spring	Pittsfield
C-01	Senator	Brooklyn
C-01	North	Rye
C-02	Sand Hill	Woodside

loan_number	customer_id
L-11	C-01
L-12	C-01
L-13	C-02

Much better but how?

introduce multivalued dependency and fourth normal form (4NF)

Multivalued Dependency

- Functional Dependency vs. Multivalued Dependency
  - Functional Dependency
    - Force a relation not to keep specific tuples (A → B)
    - So, it is also called, equality generating dependency
  - Multivalued Dependency
    - Does not force a relation not to keep specific tuples
    - while force a relation to keep specific tuples
    - So, it is also called, tuple generating dependency

Multivalued Dependency

- Multivalued Dependency
  - For α, β *that* $\alpha \subseteq R, \beta \subseteq R$
    - For all pairs of $(t_1, t_2), t_1, t_2 \in$ r such that $t_1[\alpha] = t_2[\alpha]$
      - There exist tuples t_3, t_4 in r such that
        - $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
        - $t_3[\beta] = t_1[\beta]$
        - $t_3[R - \beta] = t_2[R - \beta]$
        - $t_4[\beta] = t_2[\beta]$
        - $t_4[R - \beta] = t_1[R - \beta]$
  - A multivalued dependency D, $\alpha \rightarrow\rightarrow \beta$, holds a schema R
  - α multidetermines β
  - β is multidetermined by α
  - The schema R satisfies D

Tabular representation of $\alpha \rightarrow\rightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
t_2	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
t_3	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
t_4	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$

Multivalued Dependency

- Multivalued Dependency

Tabular representation of $\alpha \rightarrow\rightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
t_2	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
t_3	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
t_4	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$

$customer_{id} \rightarrow\rightarrow customer\_street, customer\_city$

	loan_number	customer_id	customer_street	customer_city
t_1	**L-11**	**C-01**	**Spring**	**Pittsfield**
	L-11	C-01	Senator	Brooklyn
	L-11	C-01	North	Rye
	L-12	C-01	Spring	Pittsfield
t_2	**L-12**	**C-01**	**Senator**	**Brooklyn**
	L-12	C-01	North	Rye
	L-13	C-02	Sand Hill	Woodside

$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
$t_3[\beta] = t_1[\beta]$
$t_3[R - \beta] = t_2[R - \beta]$
$t_4[\beta] = t_2[\beta]$
$t_4[R - \beta] = t_1[R - \beta]$

Multivalued Dependency

- Multivalued Dependency
  - A multivalued dependency D, $\alpha \rightarrow\rightarrow \beta$, holds a schema R
  - r(R) is illegal and does not satisfy D

r relation

loan_number	customer_id	customer_street	customer_city
L-23	99-123	North	Rye
L-27	99-123	Main	Manchester

  - Make it legal and satisfy D

r relation

loan_number	customer_id	customer_street	customer_city
L-23	99-123	North	Rye
L-23	99-123	Main	Manchester
L-27	99-123	North	Rye
L-27	99-123	Main	Manchester

Multivalued Dependency

- Observe the following derived rule:
  - **If $\alpha \rightarrow \beta$, then $\alpha \twoheadrightarrow \beta$**

$customer_{id} \twoheadrightarrow customer\_street, customer\_city$

	loan_number	customer_id	customer_street	customer_city
t_1	**L-11**	**C-01**	**Spring**	**Pittsfield**
	L-11	~~C-01~~	~~Senator~~	~~Brooklyn~~
	L-11	~~C-01~~	~~North~~	~~Rye~~
	L-12	~~C-01~~	~~Spring~~	~~Pittsfield~~
t_2	**L-12**	**C-01**	**Senator**	**Brooklyn**
	L-12	~~C-01~~	~~North~~	~~Rye~~
	L-13	C-02	Sand Hill	Woodside

not happen

- FD
  - For α, β *that* $\alpha \subseteq R, \beta \subseteq R$
    - For all the pairs of $(t_1, t_2), t_1, t_2 \in$ r,
      - $t_1[\alpha] = t_2[\alpha] \rightarrow t_1[\beta] = t_2[\beta]$,
- MVD
  - For α, β *that* $\alpha \subseteq R, \beta \subseteq R$
    - For all pairs of $(t_1, t_2), t_1, t_2 \in$ r such that $t_1[\alpha] = t_2[\alpha]$
      - ~~There exist tuples t_3, t_4 in r such that~~
        - ~~$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$~~
        - ~~$t_3[\beta] = t_1[\beta]$~~
        - ~~$t_3[R - \beta] = t_2[R - \beta]$~~
        - ~~$t_4[\beta] = t_2[\beta]$~~
        - ~~$t_4[R - \beta] = t_1[R - \beta]$~~

- Thus, every functional dependency is also a multivalued dependency

Multivalued Dependency

- Trivial Multivalued Dependency
  - D is trivial for every relation R
  - D: $\alpha \rightarrow\rightarrow \beta$ is trivial if $\beta \subseteq \alpha$ *or* $\alpha \cup \beta = R$ (all attributes)

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

α β

a	b	c	d
a1	b1	c1	d1
a2	b2	c2	d2

is there any case that ?
a1=a2 && b1=b2 && c1=c2 but
b1!=b2 && c1!=c2

Non-trivial MVD
other cases

α β

	a	b	c	d
t1	a	b	c1	d1
t2	a	b	c2	d2
t3	a	b	c1	d1
t4	a	b	c2	d2

suppose t1,t2 that
c1!=c2 && d1!=d2
then, t3=t1 && t4=t2

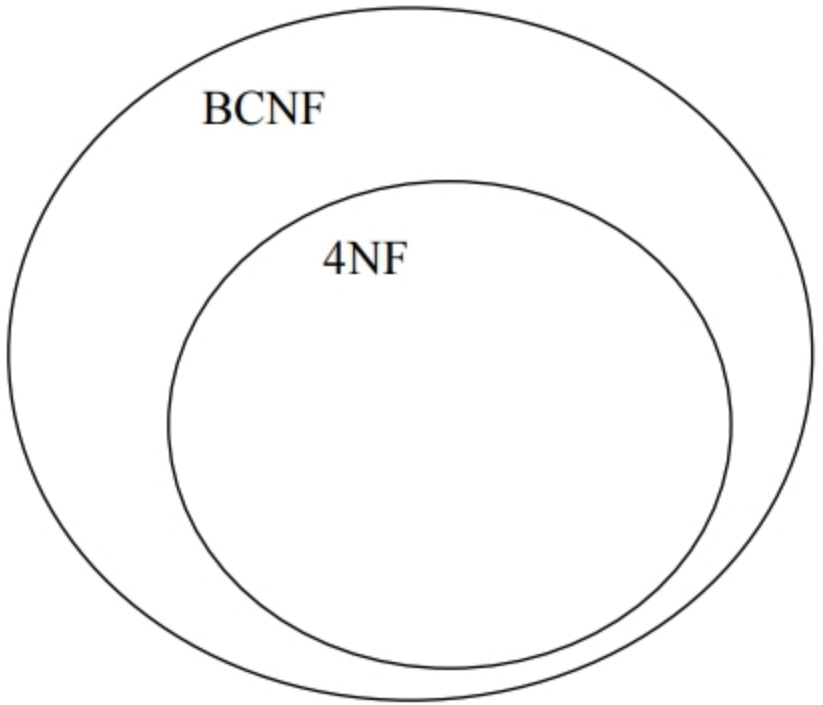
Multivalued Dependency

- Rules for Multivalued Dependencies
  - Intersection rule
    - $\alpha \twoheadrightarrow \beta \;\; \&\& \;\; \alpha \twoheadrightarrow \gamma, \;\; then \;\; \alpha \twoheadrightarrow \beta \cap \gamma$
  - Transitive rule
    - $\alpha \twoheadrightarrow \beta \;\; \&\& \;\; \beta \twoheadrightarrow \gamma, \;\; then \;\; \alpha \twoheadrightarrow \gamma - \beta$

Fourth Normal Form

- Fourth Normal Form
  - R with MVDs is in 4NF if:
    - for each non-trivial $\alpha \rightarrow\rightarrow \beta$, α is a key

- BCNF
  - For each non-trivial $\alpha \rightarrow \beta$, α is a key
  - If $\alpha \rightarrow \beta$, $then\ \alpha \rightarrow\rightarrow \beta$
  - 4NF



Fourth Normal Form

- 4NF Decomposition Algorithm
  - result:={R};
  - done:=false;
  - compute D^+; Given schema R_i, let D_i denote the restriction of D^+ to R_i
  - **while**(**not** done) do
    - **if** (there is a schema R_i, in result that is not in 4NF w.r.t. D_i)
      - **then begin**
        - let $\alpha \rightarrow\rightarrow \beta$ be a non-trivial multivalued dependency that holds on R_i such that $\alpha \rightarrow\rightarrow R_i$ is not in D_i , and $\alpha \cap \beta = \emptyset$; result:= $(\text{result} - \text{R}_\text{i}) \cup (R_i - \beta) \cup (\alpha, \beta)$;
      - end
    - **else** done:= true;

Fourth Normal Form

- 4NF Decomposition Example
  - R_2= (loan_number, customer_id, customer_street, customer_city)
  - D: $\{customer\_id \twoheadrightarrow customer\_street, customer\_city\}$

  - $customer\_id \twoheadrightarrow customer\_street, customer\_city$ is non-trivial MDVs and customer_id is not a key
  - Then decomposed to
    - cust_residence = (customer_id, customer_street, customer_city)
    - loan_cust_id = (customer_id, loan_number)

  - guarantees lossless-join decomposition
    - $R_1 \cap R_2 \twoheadrightarrow R_1$
    - $R_1 \cap R_2 \twoheadrightarrow R_2$

Wrap-up

- Conceptual Database Design
  - E-R Model
- Logical Database Design
  - Schema Mapping
  - Anomaly
  - Functional Dependency
  - Normalization
  - Multivalued Dependency