Entity-Relation Diagram, Function Dependency

CS 143 Introduction to Database Systems

TA: Jin Wang and Mingda Li

02/21/2020

Announcement

- The contents of Project 2 is rescheduled to next week discussion
- Tips
  - Use Java 1.8 or lower versions
  - Repo has been updated (remove task 6 and 7 from *makefile*)
- If you did not pick up your midterm in lecture, go to Professor’s office hour

Outline

- Query Processing (cont.)
- ER Diagram
- Functional Dependencies
- Normalization

Aggregation

- SELECT COUNT(*) FROM Students;

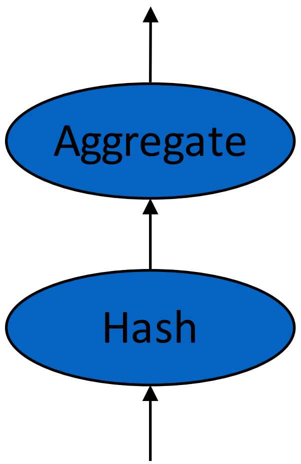
- How could we answer this query?
  - Scan…
  - Use an index?
  - System catalog

Aggregate Operations (AVG, MIN, etc.)

- Without grouping:
  - In general, requires scanning the relation.
  - Use Index Scan if possible …

- With grouping:
  - Sort on group-by attributes, then scan relation and compute aggregate for each group. (Better: combine sorting and aggregate computation.)
  - Similar approach based on hashing on group-by attributes.
  - Use Index Scan if possible …

Hash GROUP BY: Naïve Solution



- The Hash iterator permutes (re-organizes) its input so that all tuples are output in groups (sorted by group-by key).
- The Aggregate iterator keeps running info ("transition values" or "transVals") on agg functions in the SELECT list, per group
  - E.g., for COUNT, it keeps `count-so-far`
  - For SUM, it keeps `sum-so-far`
  - For AVERAGE it keeps `sum-so-far` *and* `count-so-far`
- When the Aggregate iterator sees a tuple from a new group:
  1. It produces an output for the old group based on the agg function
     E.g. for AVERAGE it returns (`sum-so-far/count-so-far`)
  2. It resets its running info.
  3. It updates the running info with the new tuple's info

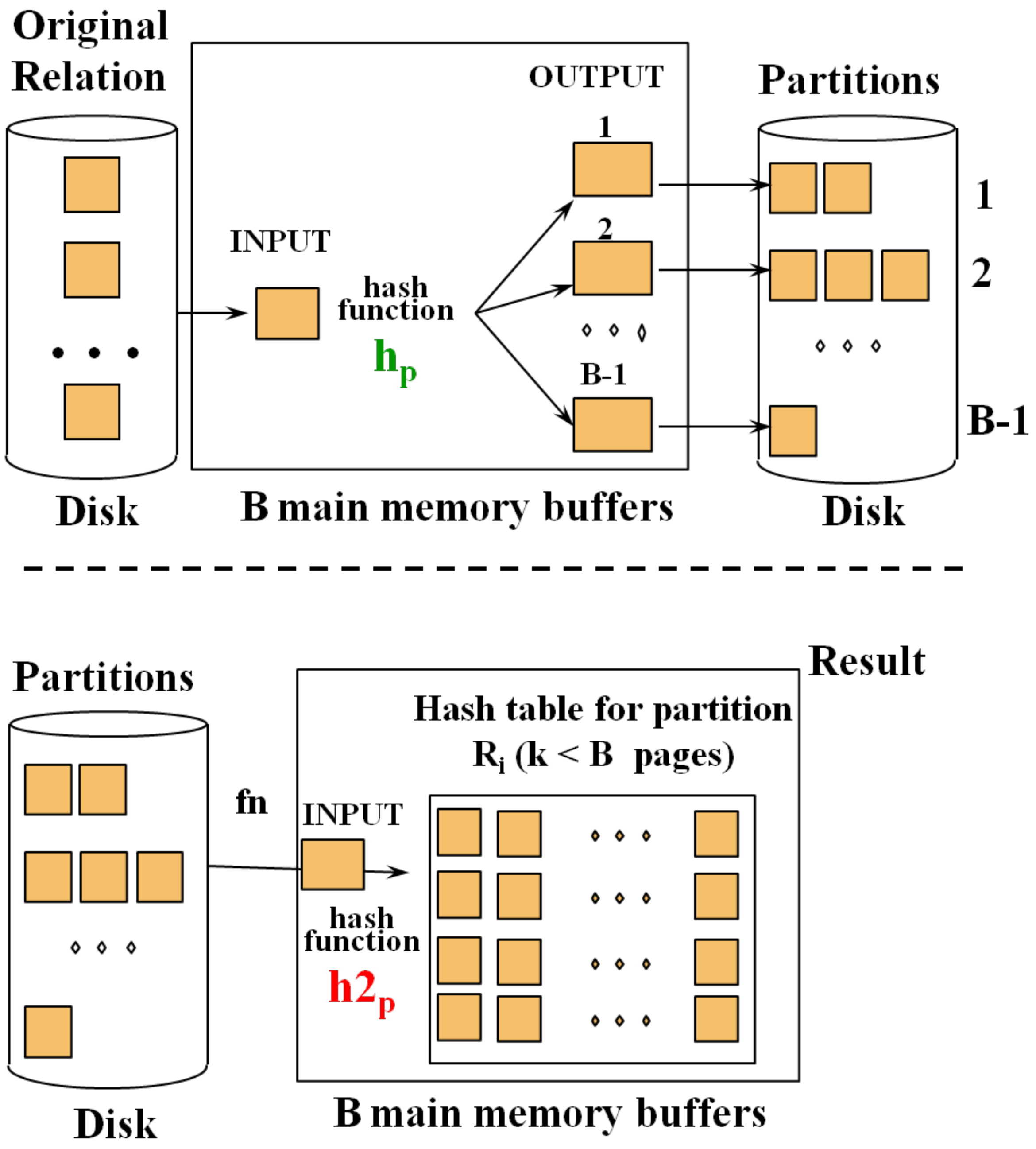
External Hashing

- **Partition:**

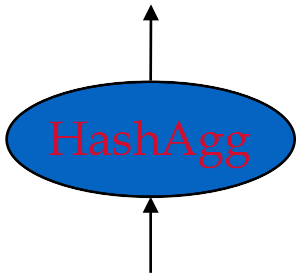
Each group will be in a single disk-based partition file. But those files have many groups inter-mixed.

- **Rehash:**

For Each Partition i:
hash i into an in-memory hash table
Return results until records exhausted then i++



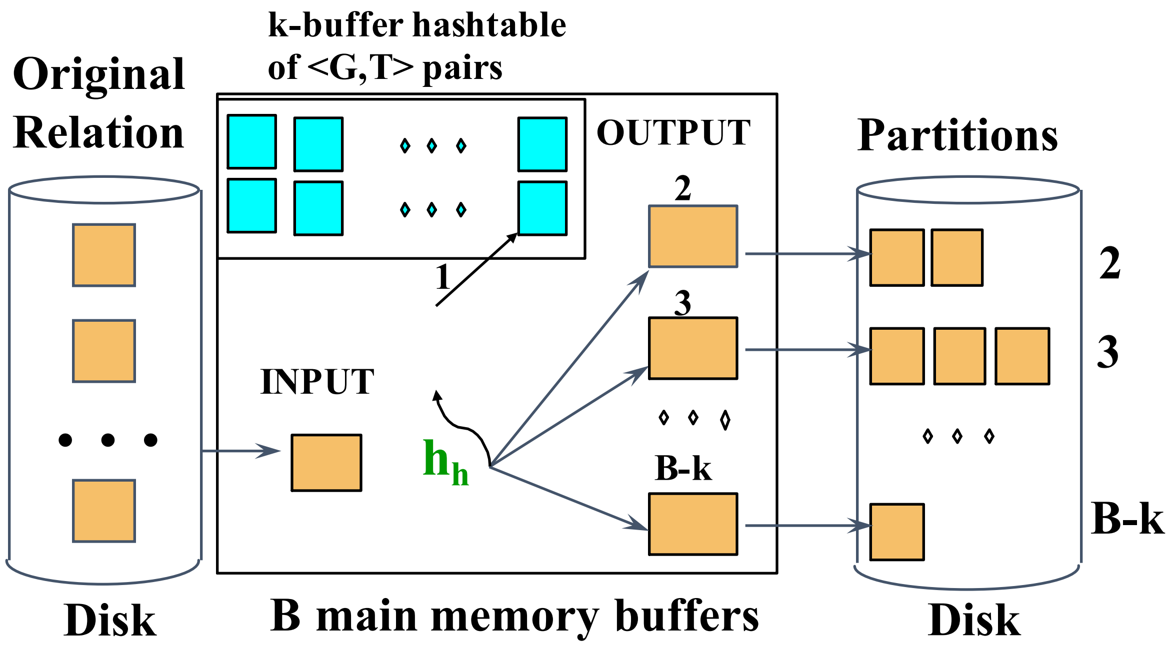
We Can Do Better!



- Put summarization into the hashing process
  - During the ReHash phase, don't store tuples, store pairs of the form <GroupVals, TransVals>
  - When we want to insert a new tuple into hash table
    - If we find a matching GroupVals, just update the TransVals appropriately
    - Else insert a new <GroupVals,TransVals> pair
- What's the benefit?
  - Q: How many pairs will we have to maintain in the rehash phase?
  - A: Number of **distinct values** of GroupVals columns
    - Not the number of tuples!!
  - Also probably "narrower" than the tuples

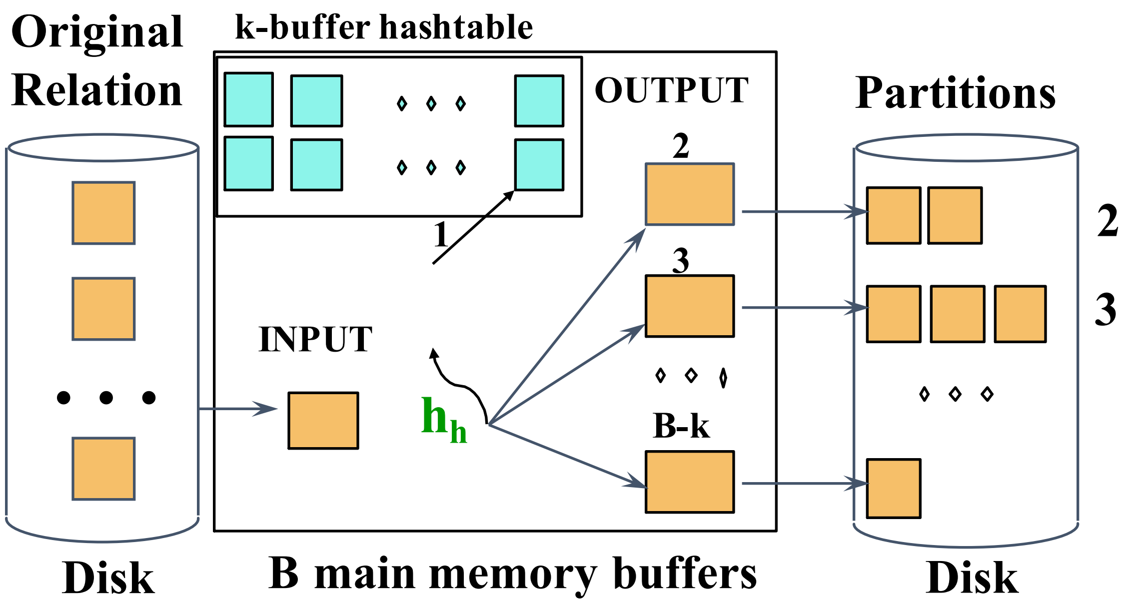
We Can Do Even Better Than That: Hybrid Hashing

- What if the set of <GroupVals, TransVals> pairs fits in memory?
  - It would be a waste to spill all the tuples to disk and read them all back back again!
  - Recall <G,T> pairs may fit even if there are *tons* of tuples!
- Idea: keep <G,T> pairs in 1st partition in memory during phase 1! (This partition occupies at most k pages)
  - Output its stuff at the end of Phase 1.
  - Q: how do we choose the number of pages (k) to allocate to this special partition?



Hash Function for Hybrid Hashing

- Assume we like the hash-partition function h_p
- Define h_h operationally as follows:
  - $h_h(x) = 1$ if x maps to a <G,T> already in the in-memory hash table
  - $h_h(x) = 1$ if in-memory hash table is not yet full (add new <G,T>)
  - $h_h(x) = h_p(x)$ otherwise
- This ensures that:
  - Bucket 1 fits in k pages of memory
  - If the entire set of distinct hash table entries is smaller than k, we do *no spilling!*



Example

- SELECT seniority, COUNT(*) FROM Students GROUP BY seniority;

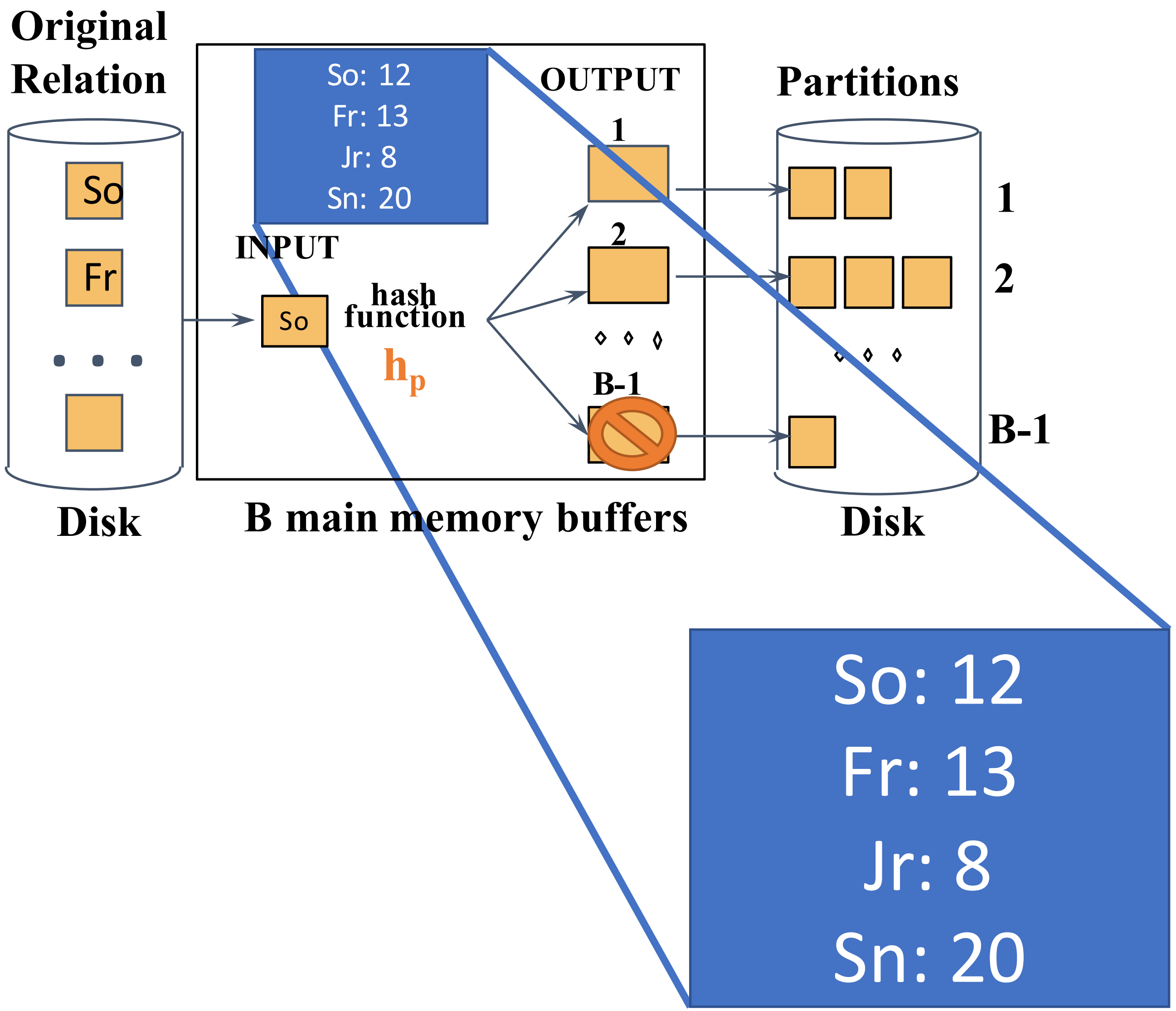
seniority	COUNT(*)
Freshman	47
Sophomore	62
Junior	85
Senior	70

- How could we answer this query?
  - Sort, then scan
  - Scan and keep 4 numbers in memory
  - Use an index on (seniority)
  - Use catalogs?

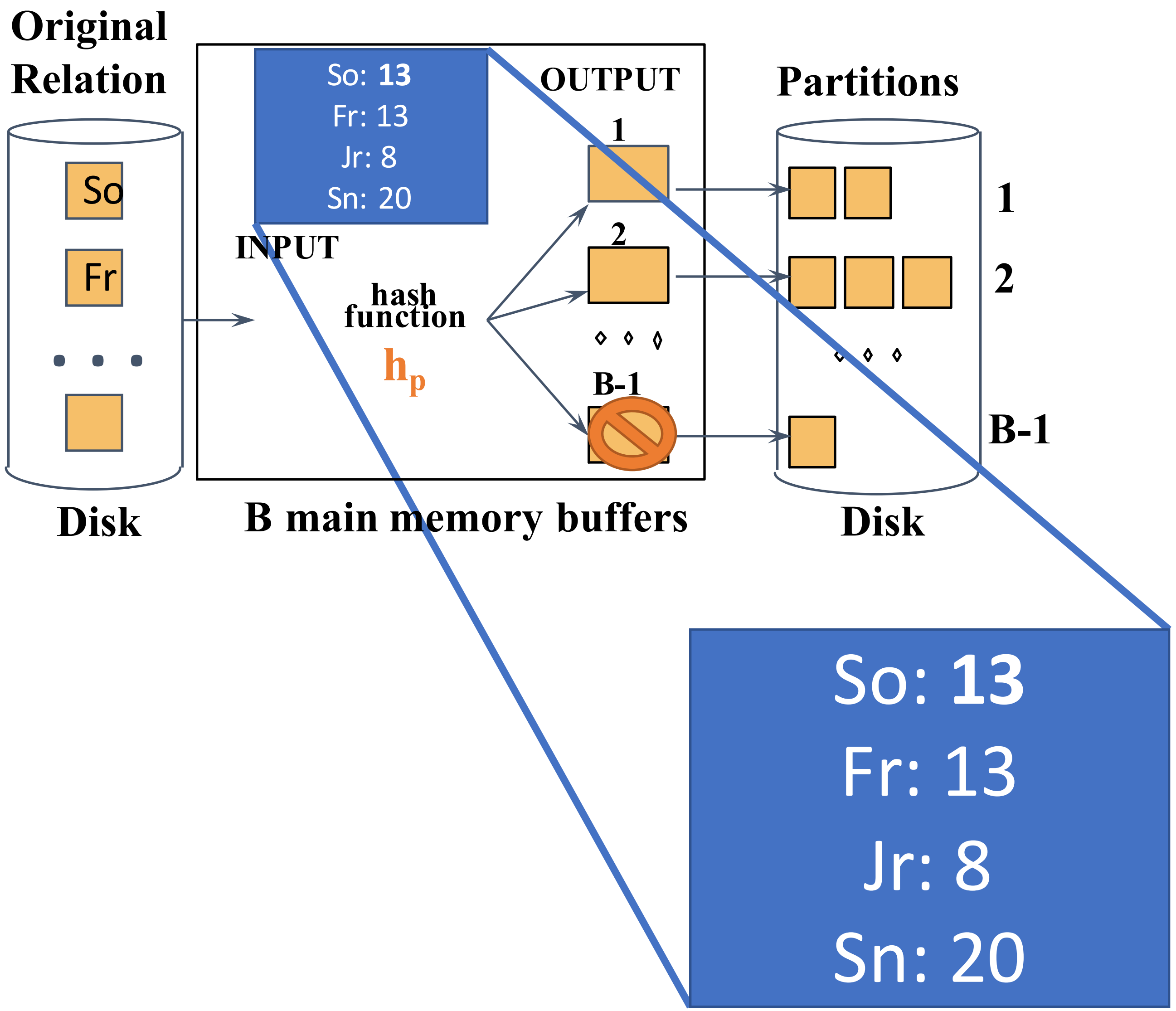
Hybrid Hashing

- Let’s take a look at the “Partition” phase.
  - Add a hash table.
  - Instead of writing data to disk, put it in the hash table if there’s room.
  - Keep running totals for each GroupVal.

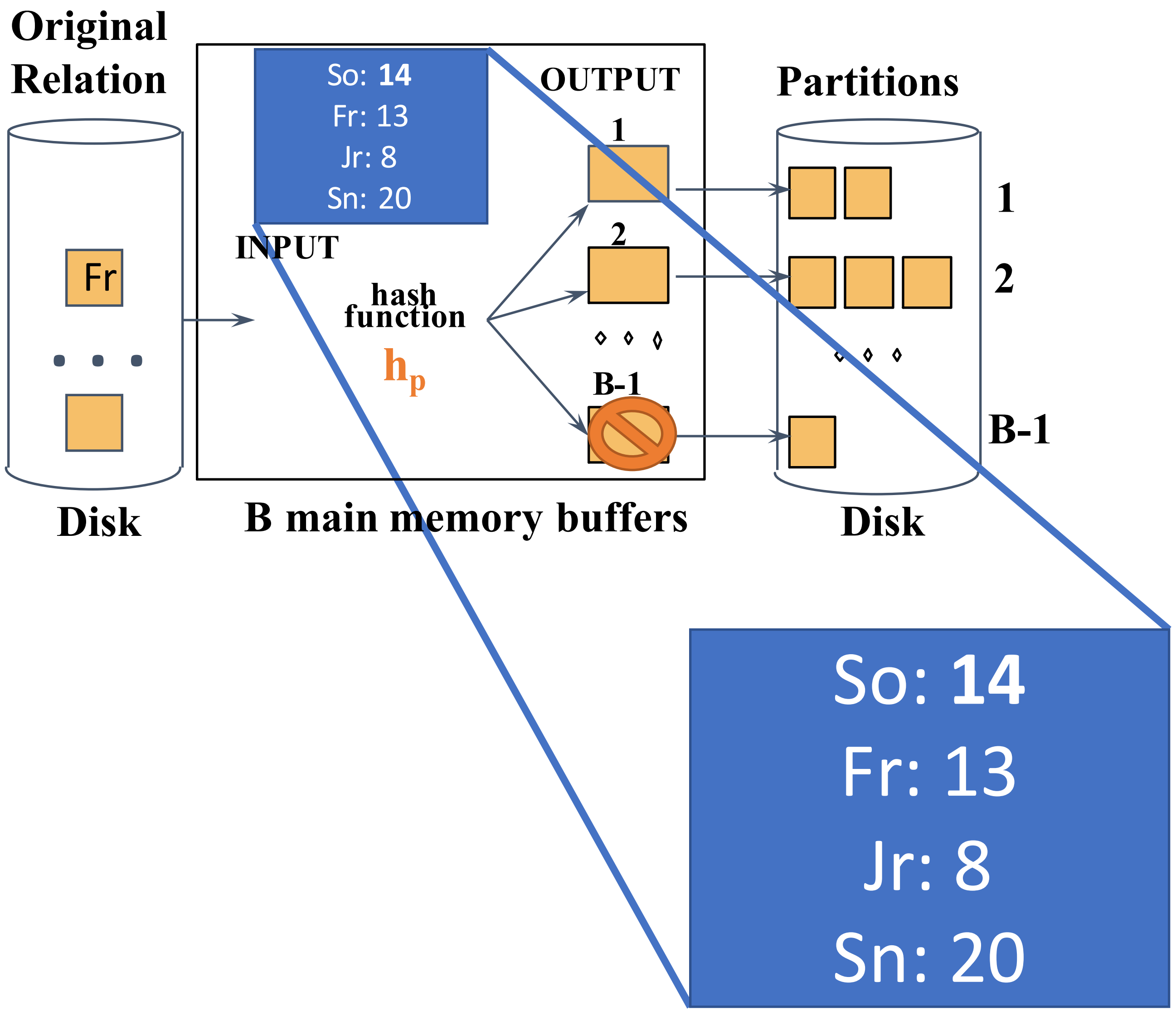
Hybrid Hashing - example



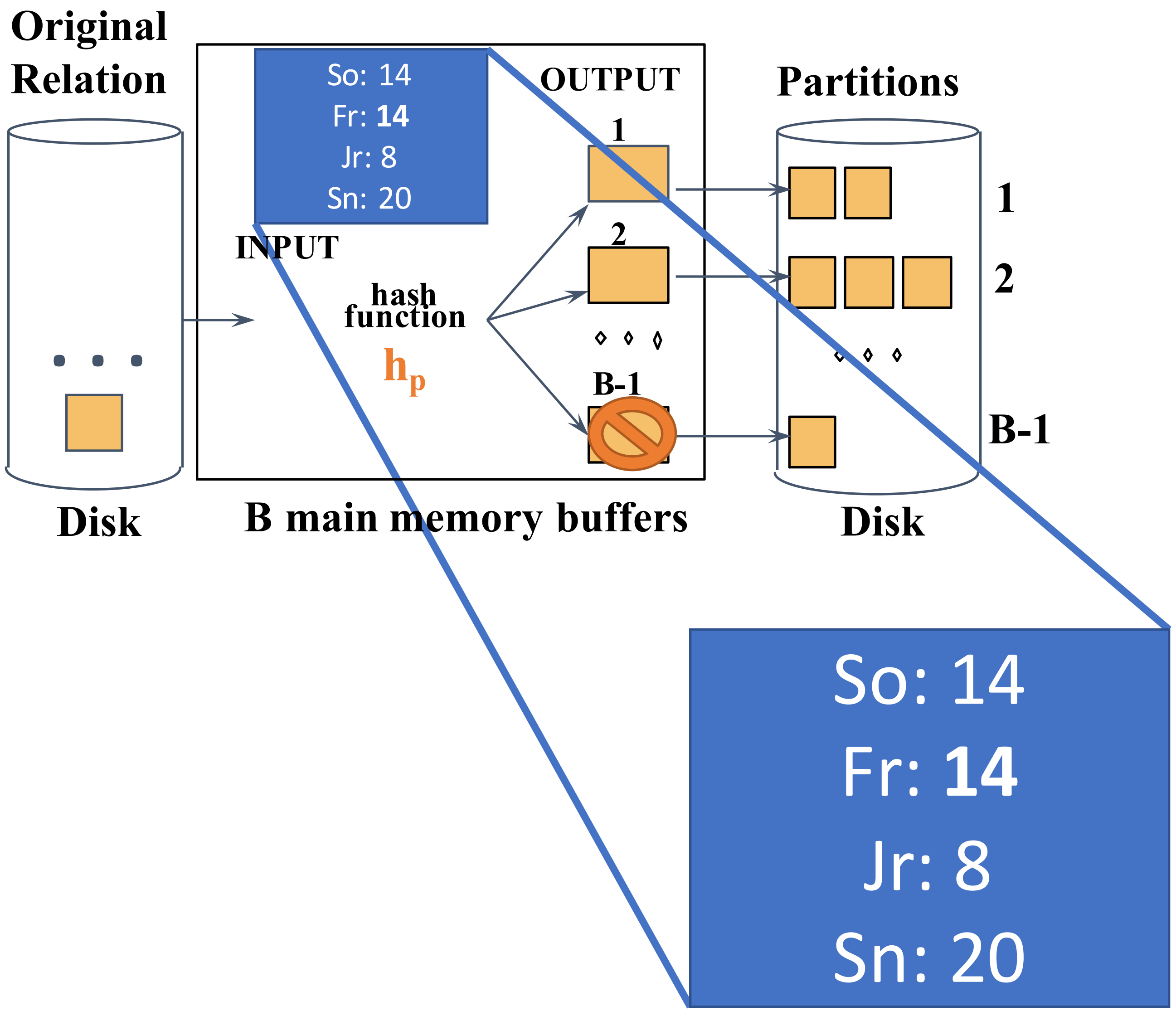
Hybrid Hashing - example



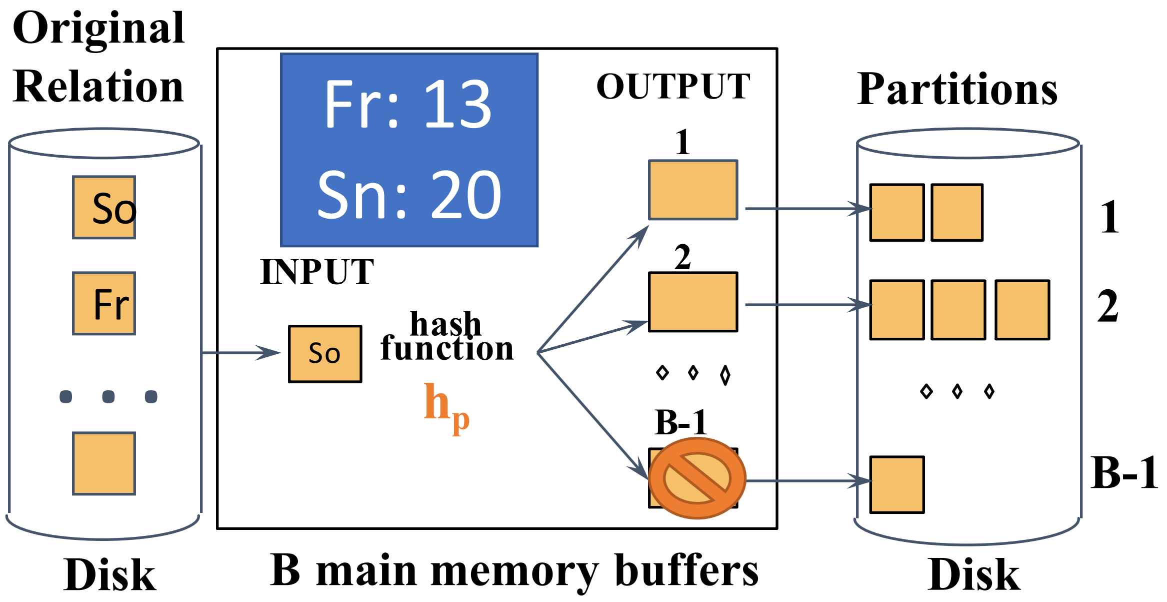
Hybrid Hashing - example



Hybrid Hashing - example



Hybrid Hashing – example 2

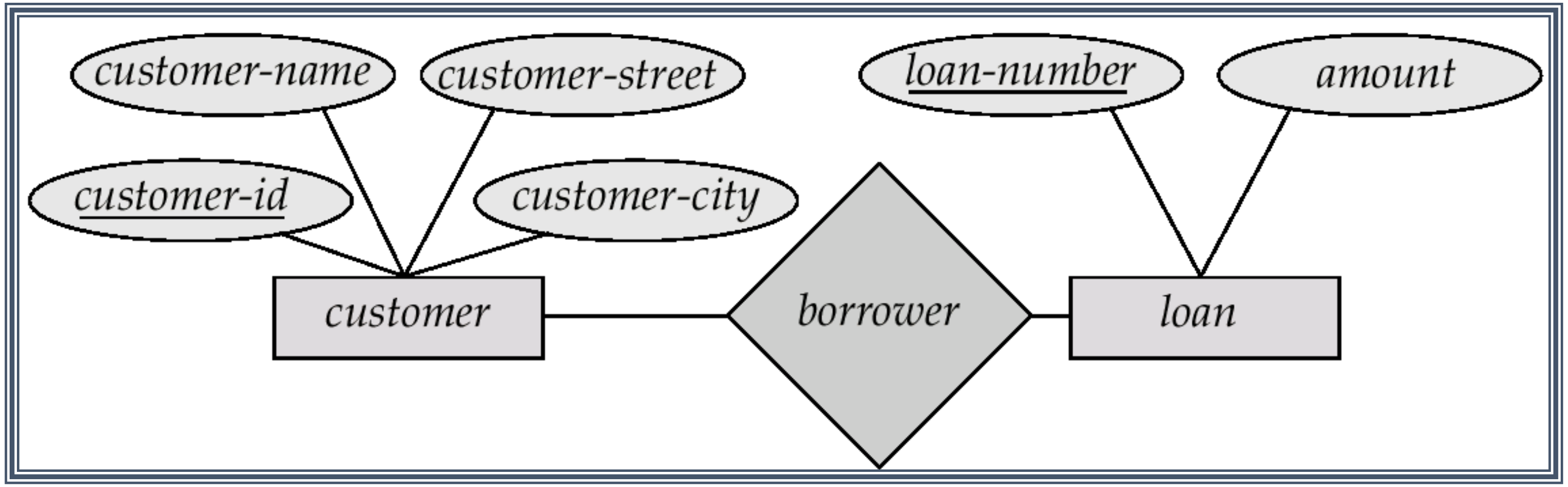


Continues Sophomores and Junior like normal hashing!

Outline

- Query Processing (cont.)
- ER Diagram
- Functional Dependencies
- Normalization

E-R Diagrams



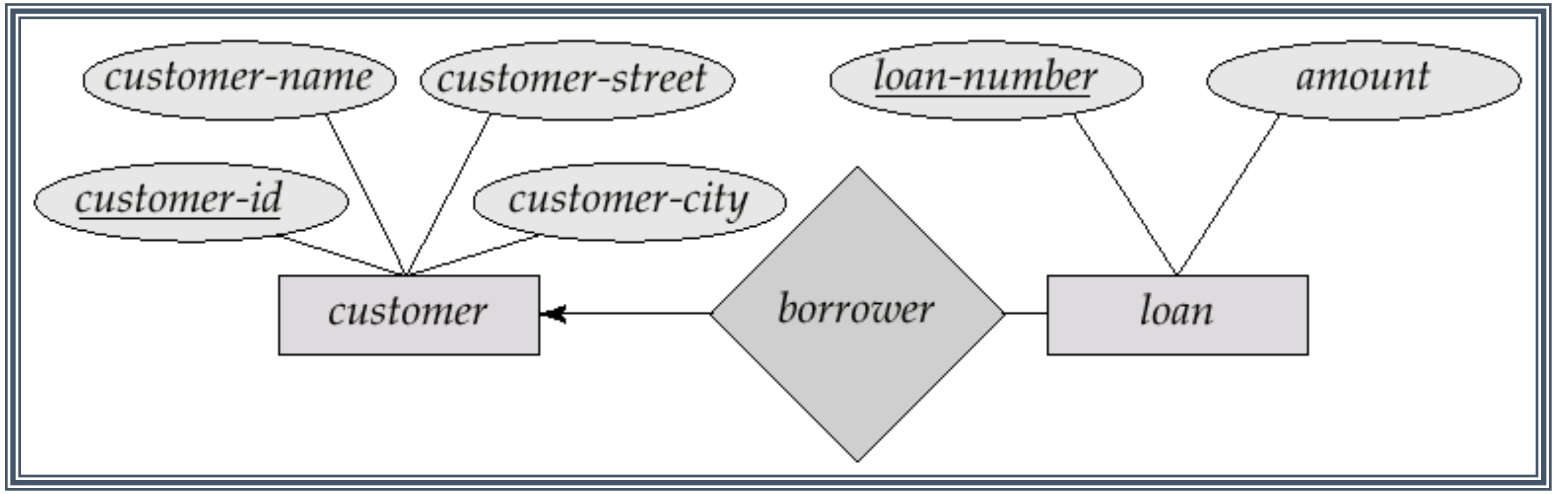
- **Rectangles** represent entity sets.
- **Diamonds** represent relationship sets.
- **Lines** link attributes to entity sets and entity sets to relationship sets.
- **Ellipses** represent attributes
  - **Double ellipses** represent multivalued attributes.
  - **Dashed ellipses** denote derived attributes.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

Mapping Cardinalities

- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

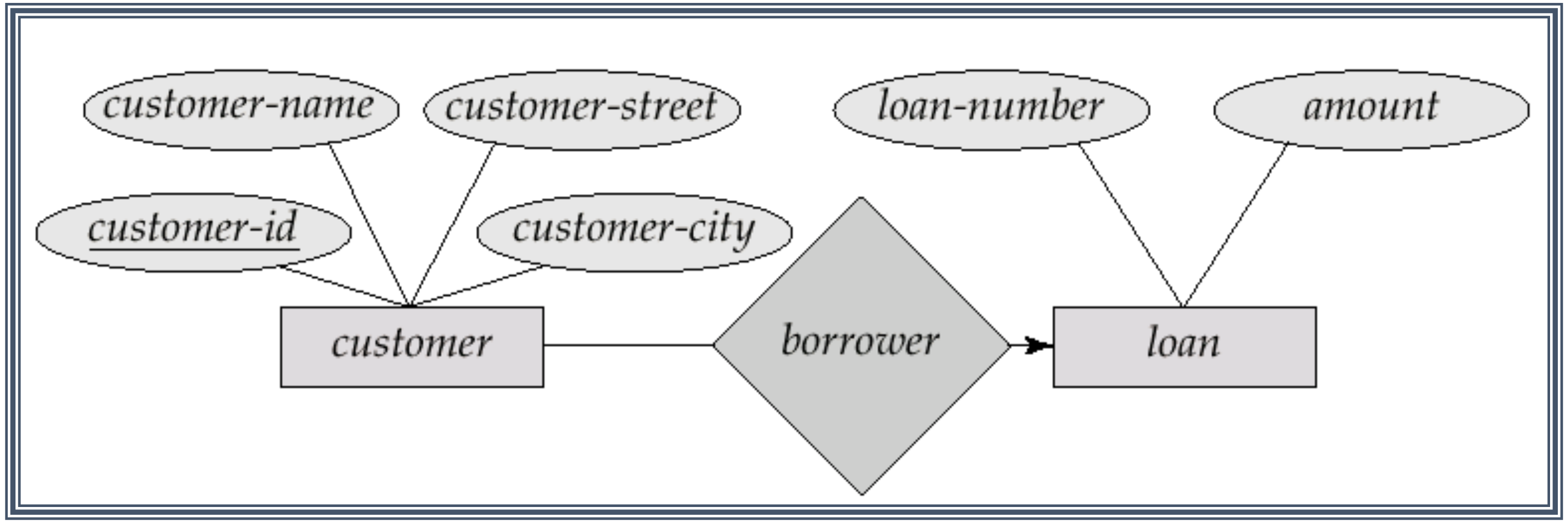
One-To-Many Relationship

- 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*



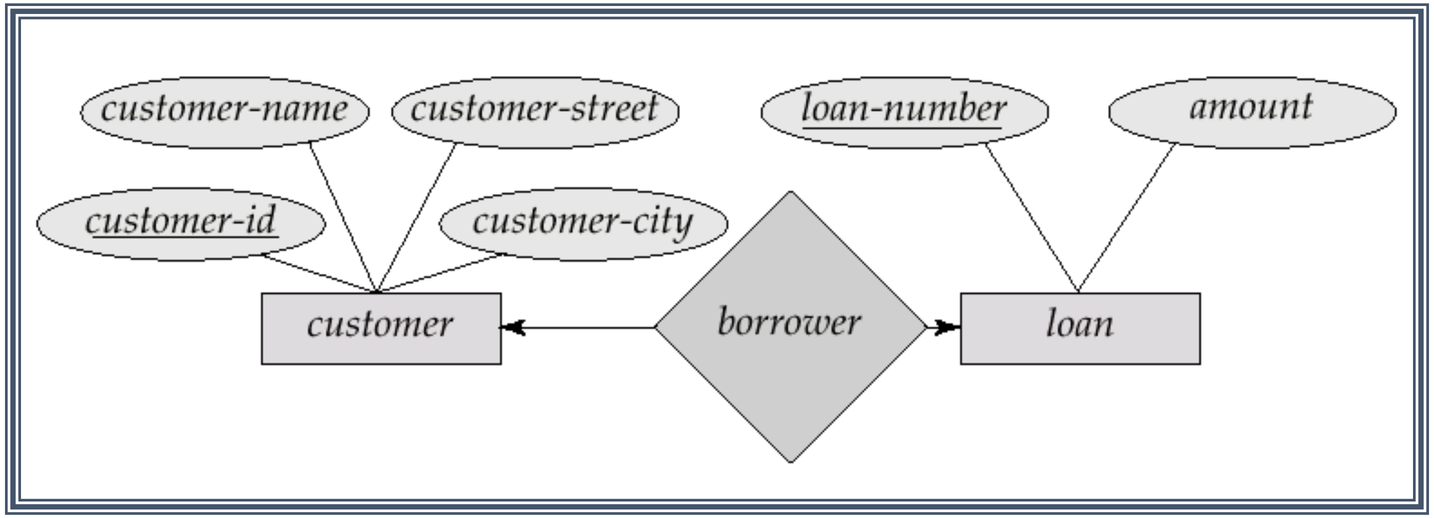
Many-To-One Relationships

- Example of many-to-one relationships: a loan is associated with several (including 0) customers via *borrower*, a customer is associated with at most one loan via *borrower*

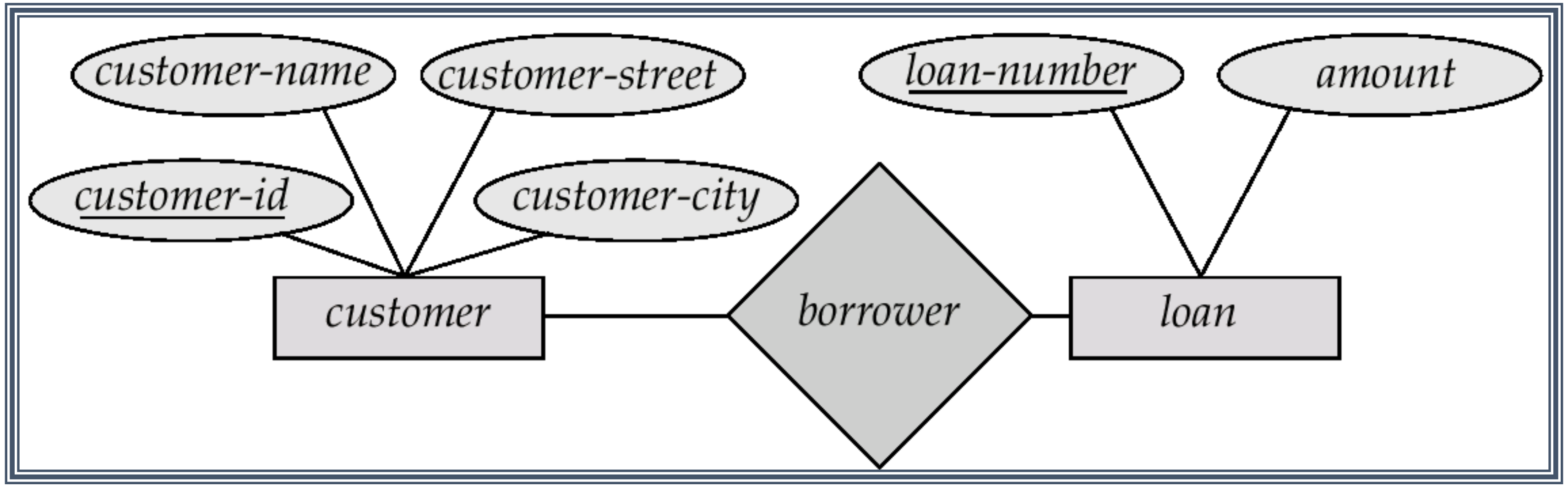


Cardinality Constraints

- 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.
- Example of 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*



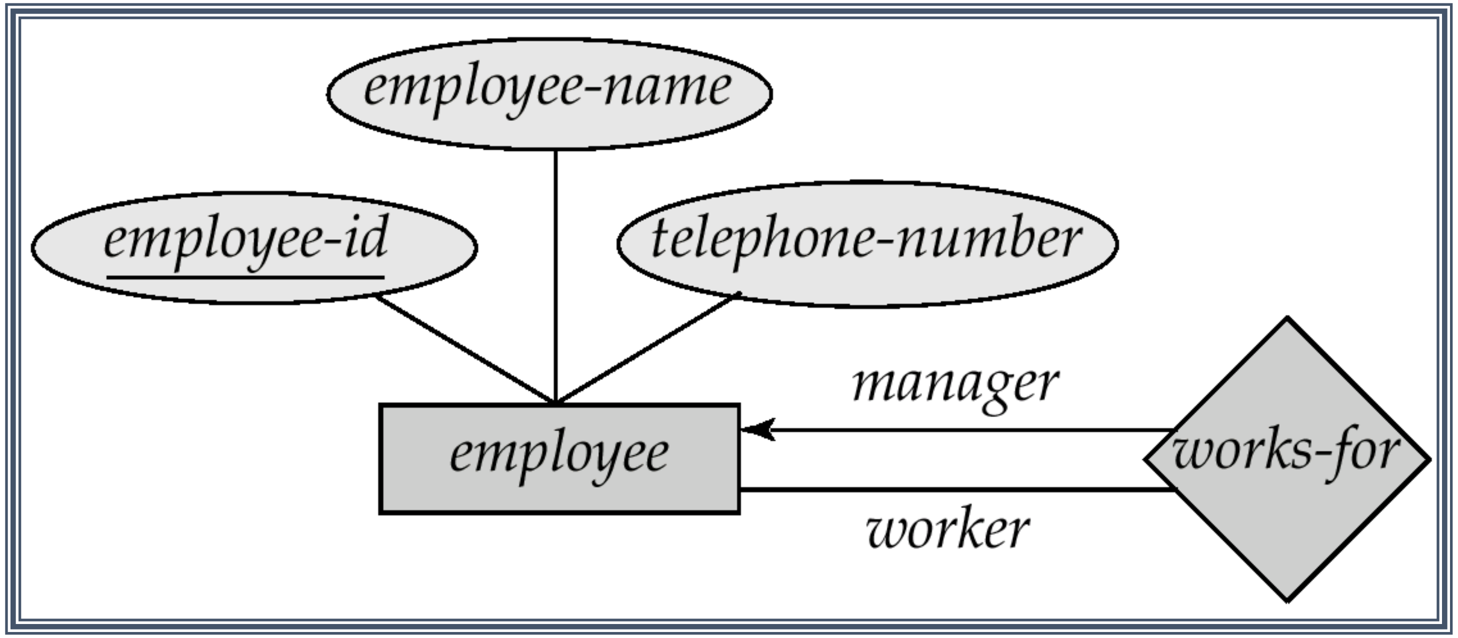
Many-To-Many Relationship



- Example of Many to Many Relationships:
  - A customer is associated with several (possibly 0) loans via borrower
  - A loan is associated with several (possibly 0) customers via borrower

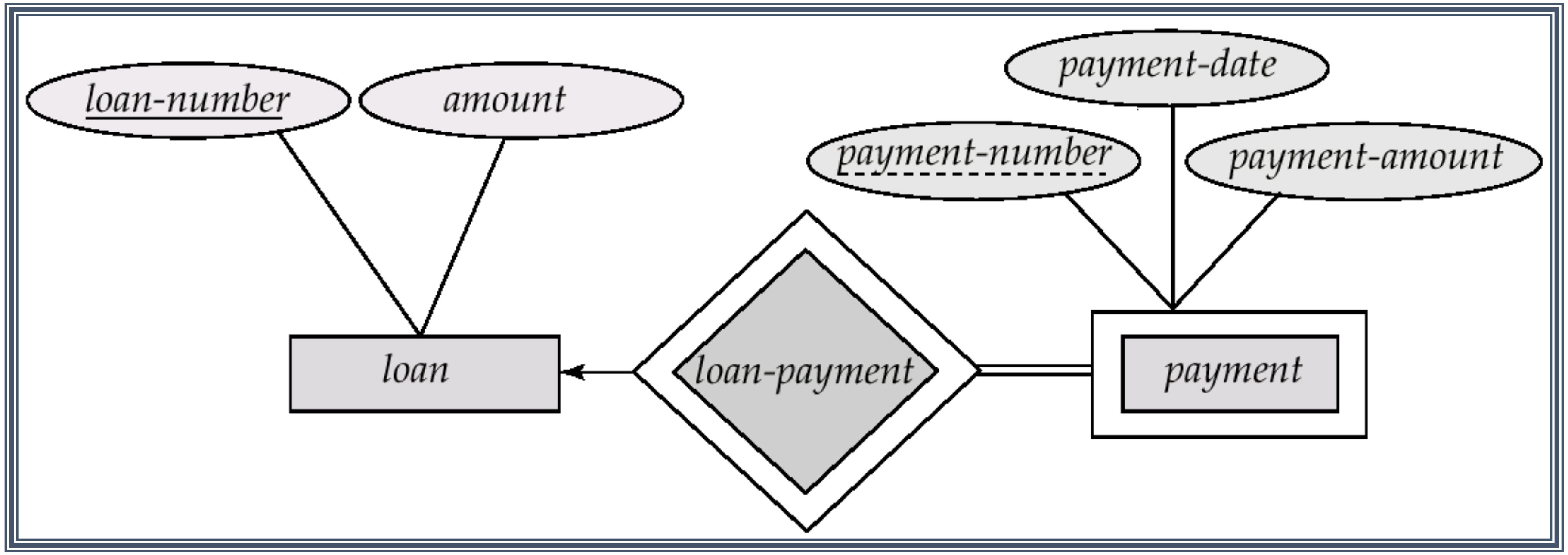
Roles

- 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. Directed line (→): Cardinality Constraints
- Role labels are optional, and are used to clarify semantics of the relationship



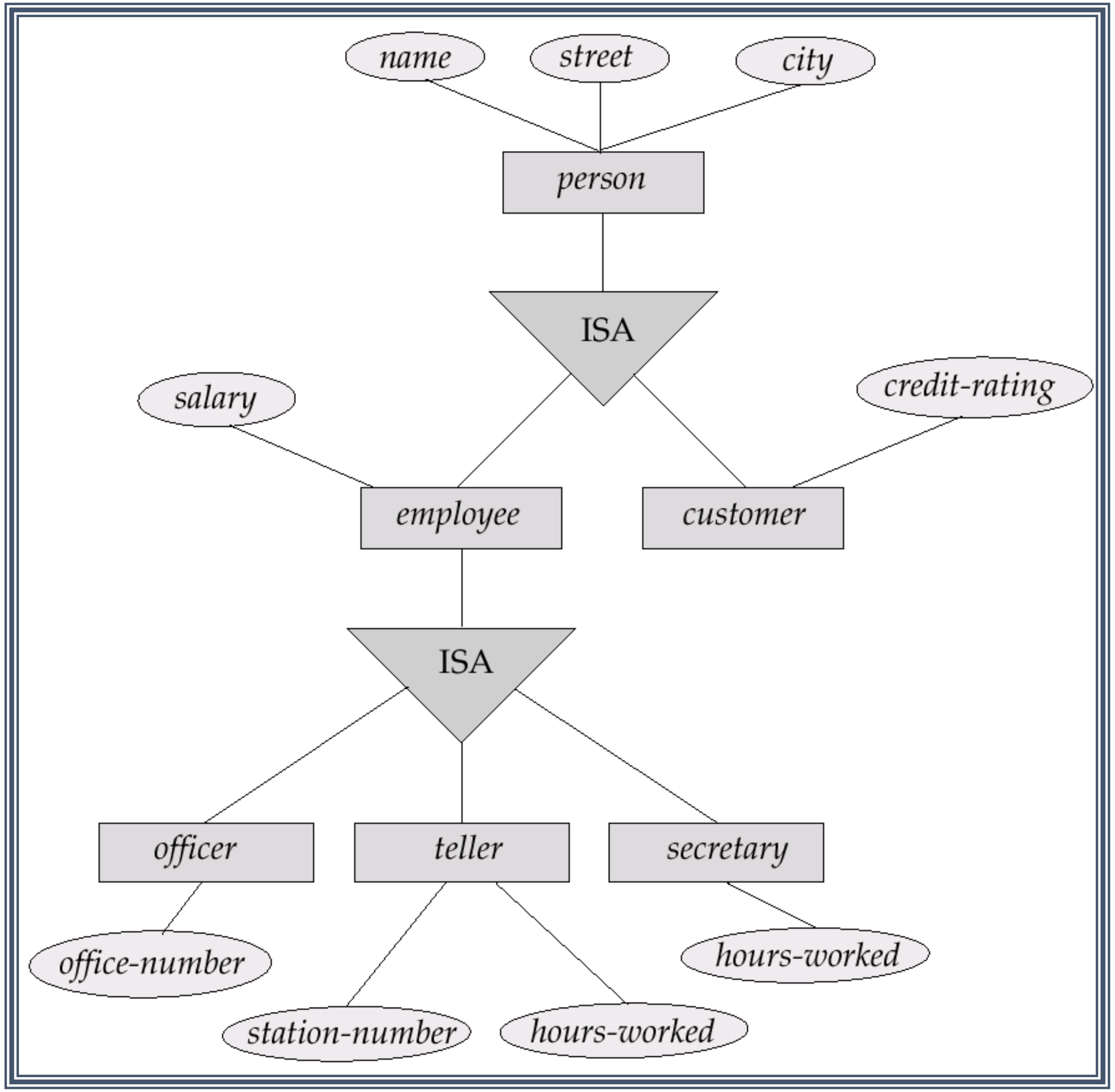
Weak Entity Sets

- We depict a weak entity set by **double rectangles**.
- We underline the **discriminator** of a weak entity set with a dashed line.
- *payment-number* – discriminator of the *payment* entity set
- Primary key for *payment* – (*loan-number, payment-number*)

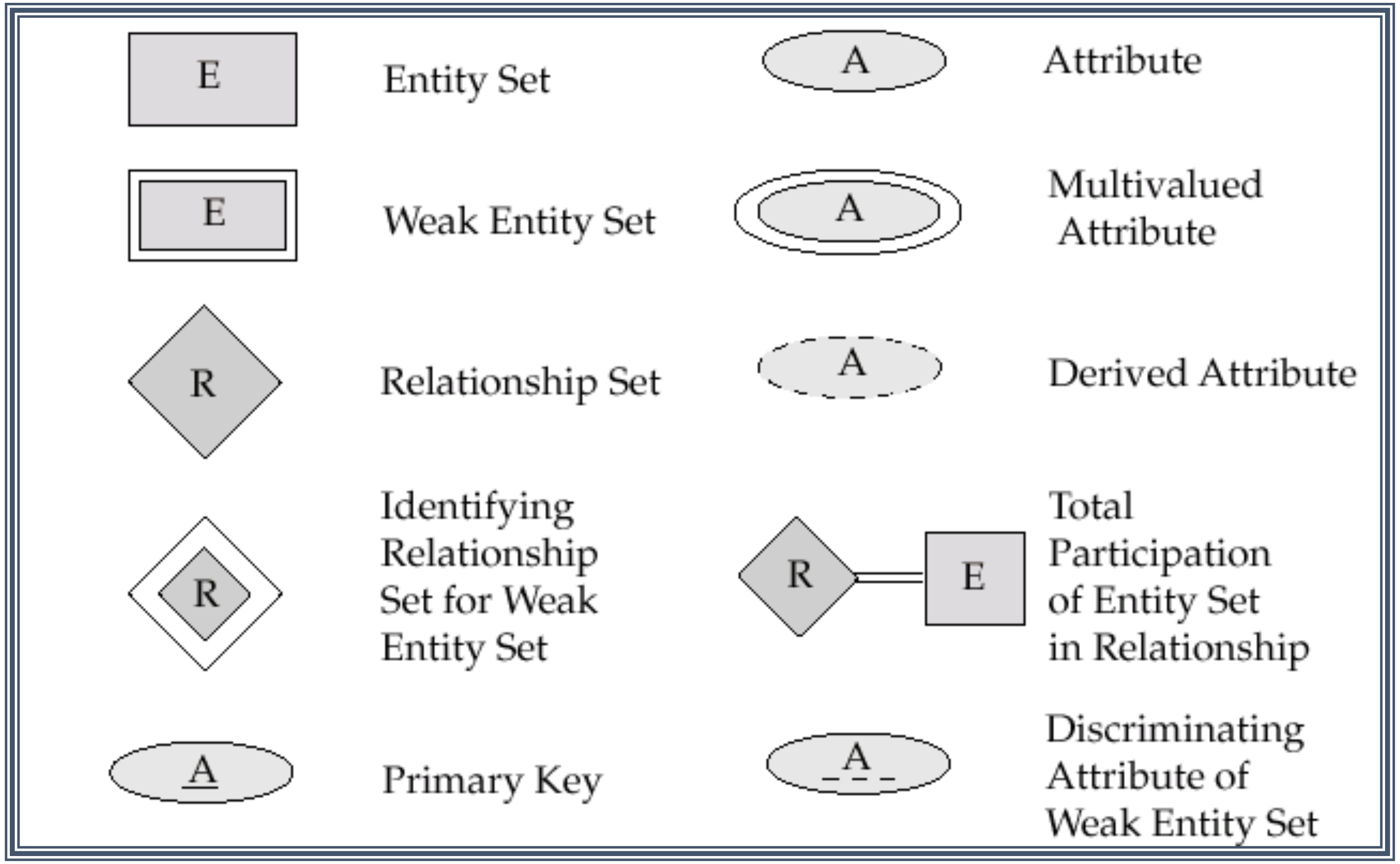


Specialization

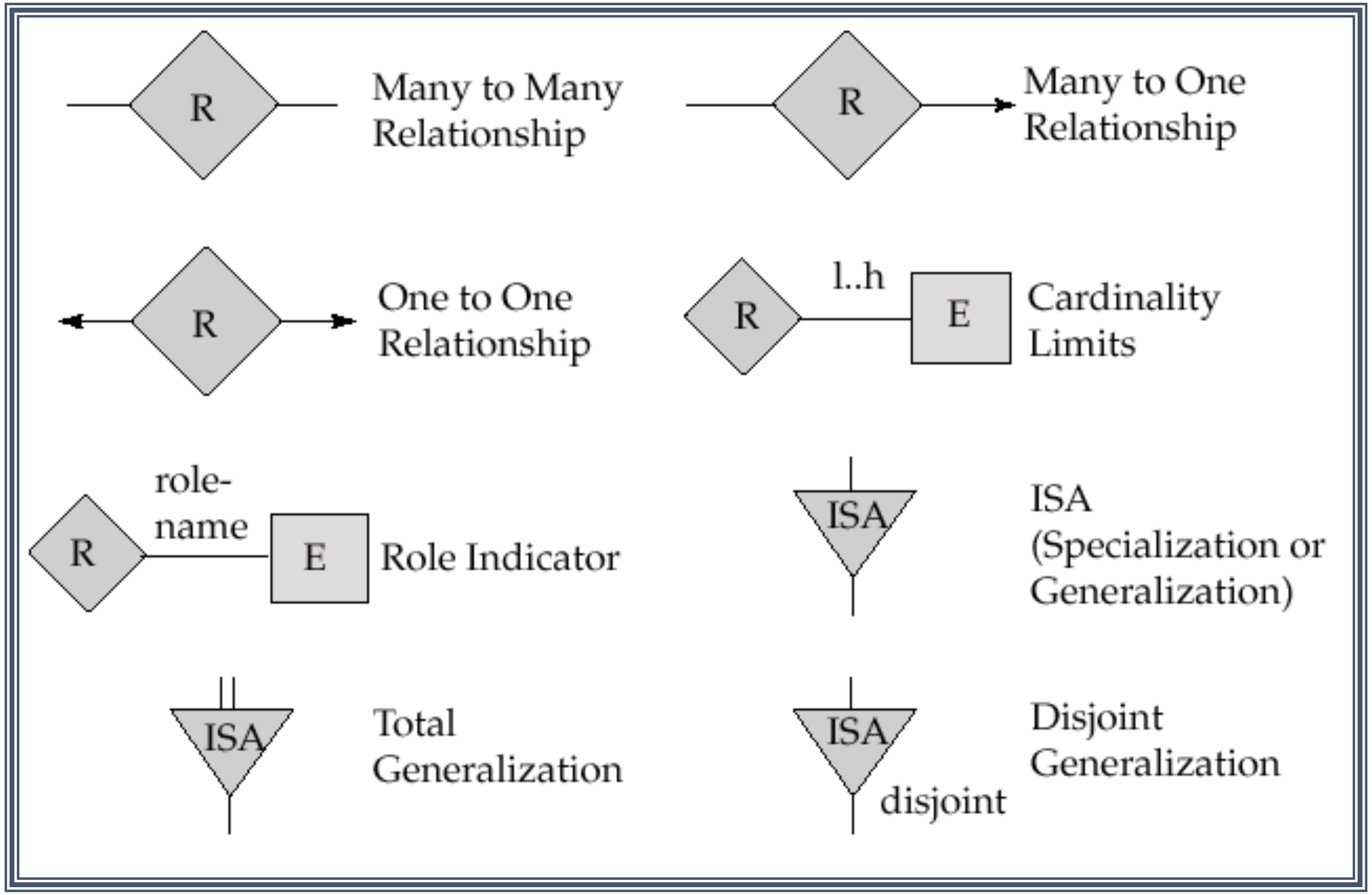
- Top-down design process; we designate *subgroupings* within an entity set that are distinctive from other entities in the set.
- These subgroupings become lower-level entity sets that have *attributes or participate in relationships* that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (E.g. *customer* "is a" *person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.


name
street
city
person
ISA
salary
credit-rating
employee
customer
ISA
officer
teller
secretary
office-number
station-number
hours-worked
hours-worked

Summary of Symbols Used in E-R Notation



Summary of Symbols (Cont.)



Outline

- Query Processing (cont.)
- ER Diagram
- Functional Dependencies
- Normalization

Functional Dependencies

Let **X= {A_1, ..., A_n}, and Y={B_1, ..., B_m}** be subsets of the attributes in **R(W)**:

We say that **Y is functionally dependent on X** and write
X -> Y when

- **For every pair of tuples u1, u2 in R,**
- **if u1[X] = u2[X], then u1[Y] = u2[Y]**
- **No two tuples in R can have the same X values but different Y values!**
- **Given R(W), let Z be a subset of W where Z -> W, then Z is said to be a *key* for R(W)**
- **Trivial FD: X -> Y where Y is a subset of X.**

FDs' Properties

Reflexivity: If Y is a nonempty subset of X then

X -> Y.

Augmentation: if X -> Y and X is a subset of Z, then Z->Y

Transitivity: X -> Y and Y->Z then X->Z

This is a sound and complete set of inference rules.

- Every other sound properties can be derived from these.

About FDs

A	B	C
a1	b1	c1
a1	b2	c2
a2	b1	c3

Does this FD hold?
AB -> C – Yes (Given only this instance)

A	B	C
a1	**b1**	**c1**
a1	b2	c2
a1	**b1**	**c3**

Here we have replaced a2 with a1
Does this FD hold?
AB -> C – No: does a1,b1→ c1 or c3?

Since the content of tables change in the DB, we are only interested in those that **hold all the time** since they result from integrity constraints that hold in the real world!

Also important: A -> BD iff A ->B and A->D

StudentClass Table

sid	name	addr	dept	cnum	title	unit
301	James	11 West	CS	143	Database	04
105	Elaine	84 East	EE	284	Signal Processing	03
301	James	11 West	ME	143	Mechanics	05
105	Elaine	84 East	CS	143	Database	04
207	Susan	12 North	EE	128	Microelectronics	03

FDs here: **sid -> name** **sid -> addr**
dept, cnum -> title, unit

FDs here are the source of redundancy & update anomalies

Only certain FD patterns are a problem. For instance consider two projections of the previous table

LeftTable

sid	name	addr
301	James	11 West
105	Elaine	84 East
207	Susan	12 North

RightTable

dept	cnum	title	unit
CS	143	**Databases**	04
EE	284	**Signal Processing**	05
ME	143	Mechanics	03
EE	128	Microeletronics	04

Same FDs here: **sid -> name** **…in LeftTable**
sid -> addr

As in the original table

dept, cnum -> title, unit **…in RightTable**

In the old table these FDs were the source of redundancy & update anomalies

But no redundancy or update anomaly here! Why?
…Because **sid** is the key for LeftTable & **dept, cnum** is the key for the right table**!**

Outline

- Query Processing (cont.)
- ER Diagram
- Functional Dependencies
- Normalization

Understanding NFs

- 1NF: flat tables: no structured fields
- 2NF: Relations are 2NF when they are 1NF and no non-key attribute is partially FD on a key (i.e. FD on a subset of key)
- 3NF: Relations are 3NF when they are 2NF and and no non-key attribute is transitively FD on a key
- BCNF: Relations are BCNF *if for every non-trivial X->A,*
  X is a key or a superset of a key
- Revisiting the definition of 3NF

Definition: ***R is 3NF iff for every non-trivial X -> A, either***

(i) X is a key or a superset of a key, or

(ii) A is an attribute of some key
. *(which will be broken if we decompose since A will go into one projection and the remaining key attributes into the other)*

Violation of the 2NF requirement

dpchair	dept	cnum	title	unit
Tom	CS	143	Databases	04
Eddy	EE	284	Signal Processing	05
Nancy	ME	143	Mechanics	03
Eddy	EE	128	Microeletronics	04

What are the FDs here?: dept -> dpchair dpchair is FD on dept
dept, cnum -> title, unit

What is the key here: dept, cnum

Thus **dpchair** is FD on only a subset of the key.
A violation of the 2nd Normal Form requirement.
Decompose into (dept, dpchair) (dept, cnum, title, unit)

Violation on 3NF: Transitive FD on Key

sid	Advisor	OfficeNo
301	James	BH 3551
105	Elaine	MH 2009
207	James	BH 3551

Sid-> Advisor
Advisor-> OfficeNo

Then
Sid -> OfficeNo

Key: Sid
Anomalies: ?

Codd's 3NF: A relation which (i) is in 2NF and (ii) no attribute is transitively dependent on a key.

3NF avoids the anomalies caused by transitive dependencies on keys.

These anomalies can be eliminated by decomposition.
For the case at hand (sid, Advisor) (Advisor, OfficeNo)

A simpler (but stronger) Normal Form

Given a relation R and its FDs:

Keys: X is a KEY of R(W) iff

1)X -> all attributes of R, i.e. X -> W (i.e. X is a superkey)

2)No subset of X satisfies 1. i.e. X is minimal

Boyce-Codd Normal Form (BCNF):

- R is in BCNF iff for every non-trivial X -> Y, X contains a key (i.e. it is a superkey)
- X->Y is trivial if Y is a subset of X: e.g. A,B ->B is trivial— i.e., it is not a real constraint since it holds in all tables

Now consider the projections of StudentClass

LeftTable

sid	name	addr
301	James	11 West
105	Elaine	84East
207	Susan	12 North

RightTable

dept	cnum	title	unit
CS	143	Databases	04
EE	284	Signal Processing	05
ME	143	Mechanics	03
EE	128	Microeletronics	04

FDs in LeftTable: sid-> name
sid-> addr

sid the key in LeftTable: **no BCNF violation**

FDs in RightTable: dept, cnum -> title
dept, cnum ->unit

(dept, cnum) is the key: **no BCNF violation**

Therefore …
(1) The original relation was not BCNF, but
(2) The two projections are in BCNF
(3) But these two projections are not enough: we need lossless decomposition.

Practice

- Consider the relation R(A, B, C, D), with functional dependency set F: {AB→C, BC→D, CD→A, AD→B, C→BD}. Which of the following is/are true?

a) C is a candidate key of R.

b) D is a candidate key of R.

c) AB is a candidate key of R.

d) R is in BCNF.

Answer: A, C, D

Lossless Decomposition

- Take a relation R(W) where W= X ∪ Y ∪ Z and replace it with:
- $R_1 = \pi_{X \cup Y} R(W)$ and $R_2 = \pi_{X \cup Z} R(W)$

Theorem: If X-> Y or X->Z then R(W) can be reconstructed as the natural join of these two projections.

By repeating this step we generate a set of relations that are BCNF and whose natural join return the original table (a lossless decomposition)

Decomposition into BCNF

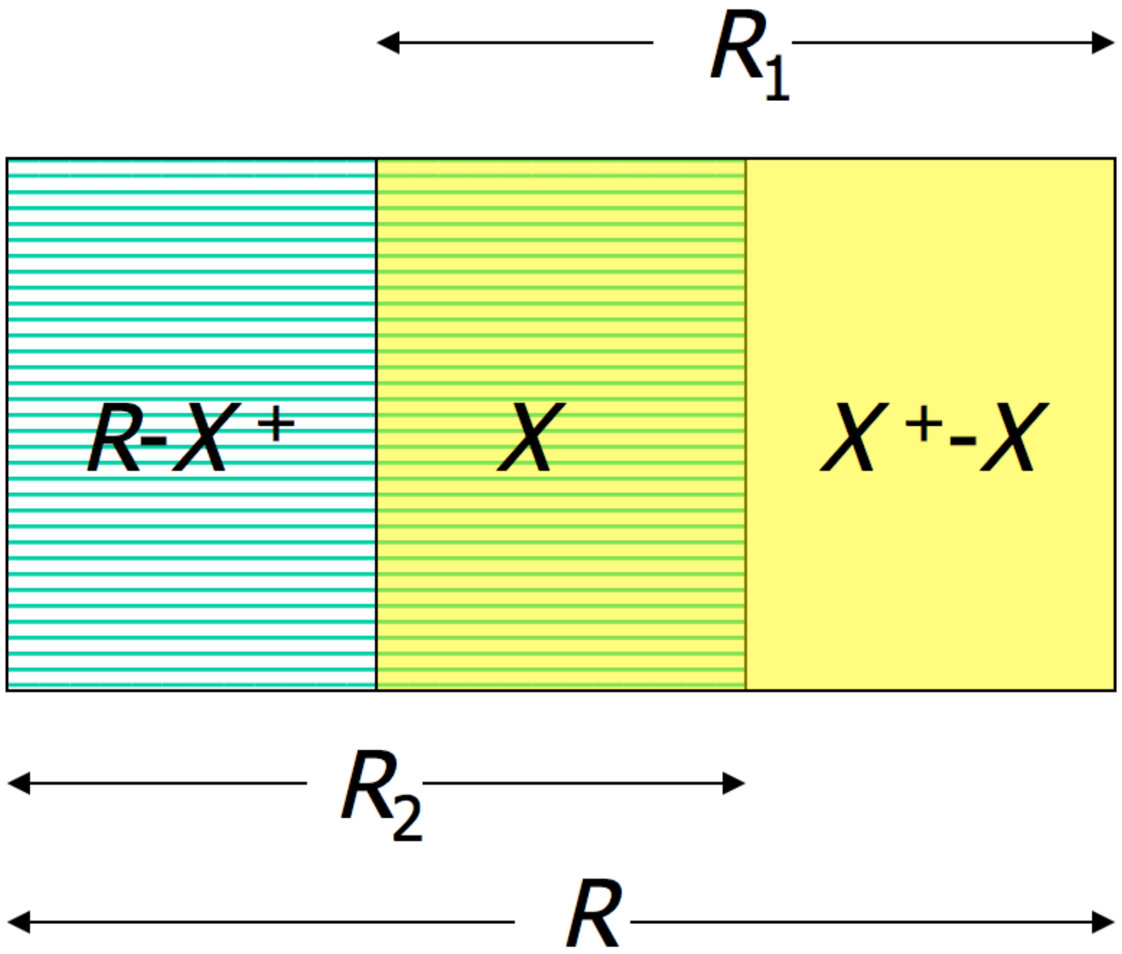
- Given a relation R, we find its FDs
- From these we find the keys, then we ask the question: is R in BCNF?
  - If the answer is yes, we smile.
  - If the answer is no: we decompose the original relation into a set of BCNF tables

- The decomposition algorithm is based on the notion of cover (closure), which is based on the formal properties of FDs.
- Three objectives: (1) Lossless decomposition, (2) dependency preservation (3) minimal relation count.

Decomposing into BCNF

- For each nontrivial FD X->Y in R(W):
  - If X is a superkey ($X^+ = W$), nothing to do.
  - Otherwise X->Y violates BCNF:
    - Decompose R to $R_1(X^+)$ and $R_2(X \cup (R - X^+))$
      - R_1 = all attributes in X->Y closure
      - R_2 = all the other attributes, plus original X
    - Recurse on both R_1 and R_2 and their corresponding FDs

BCNF Decomposition



credit: http://imada.sdu.dk/~petersk/DM505/slides/slides6.pdf

Practice Problem

- Given the attribute set R = ABCDEFGH and the functional dependency set F = {BC→GH, AD→E, A→H,E →BCF, G→H}, decompose R into BCNF by decomposing in the order of the given functional dependencies.

- Answer: ADE, BCEF, GH, BCG

Detailed Explanation

Idea: for each LH you want to identify the full closure...

BC -> {BCGH}

AD: {ADEHBCFGH} (roughly in order) -> all!

A : {AH}

E: {EBCFGH} (missing A, D)

G: {GH}

We go in order: BC->GH violates bc BC is not a key

Result: ABCDEF (removed GH), BCGH

Next we look at AD-> E: it was a key of the initial relation, so it must also be a key of the ABCDEF relation. no need to do anything

A -> H: Does not apply because no relation contains A and H. If you were keeping track of which FDs apply to which relation, you'd notice we 'lost' this one.

E->BCF: this only applies to the first relation (ABCDEF) and is unable to provide A or D (only EBCF) so it violates BCNF as it is not a super key

Result: ADE, EBCF, BCGH (unchanged)

G->H: this only applies to the last relation (BCGH) and G is not a super key, so we need to decompose:

Result: ADE (unchanged), EBCF (unchanged), GH, BCG

Final: ADE, EBCF, GH, BCG

BCNF vs 3NF (X->Y)

- **BCNF**: X must be a superkey
- **3NF**: X is superkey *or* Y is prime (member of a key)
- Takeaway:
  - 3NF provides both *lossless join* and *dependency preservation.*
  - BCNF can't always guarantee both