

Sc2207

= 70

Quiz  
1 ER

2 relational

1 BCNF 3rdNF

QnA  
- weak entity

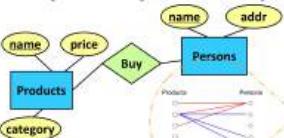
#### Example



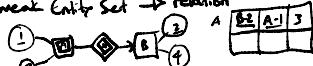
- Each student is mentored by one faculty member
- Each faculty member can mentor multiple students

Students  
Family members  
Many links coming out from each Student to many Faculty members  
Many links coming out from each Faculty member to the other ones

#### Many-to-Many Relationship



weak Entity Set → relation



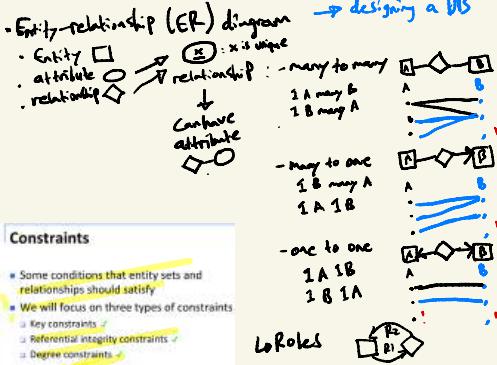
#### Checking BCNF in a Tricky Case

- We are checking  $R_2(A, C, D, E)$
- FDs that we have:  $A \rightarrow B$ ,  $BC \rightarrow D$
- Check the closures:
  - $[A]^* = [A, B]$
  - $B$  is not in  $R_2(A, C, D, E)$ , so the closure becomes:
    - $[A]^* = [A]$
    - Similarly,  $(C)^* = [C]$ ,  $(D)^* = [D]$ ,  $(E)^* = [E]$
- So far, none of these indicate a violation of BCNF (trivial FDs)
- We check further:  $[AC]^* = [ACD]$
- This indicates that  $AC \rightarrow D$  and  $AC$  does not contain key ACE of  $R_2$
- This means that  $R_2$  is not in BCNF
- So we need to decompose it

#### Exercise

- Given:  $C \rightarrow D$ ,  $AD \rightarrow E$ ,  $BC \rightarrow E$ ,  $E \rightarrow A$ ,  $D \rightarrow B$ ,  $B \rightarrow F$
- Can you prove  $D \rightarrow C$ ?
- We start with  $\{D\}$
- Since  $D \rightarrow B$ , we have  $\{D, B\}$
- Since  $B \rightarrow F$ , we have  $\{D, B, F\}$
- What else?
- No more.
- $\{D, B, F\}$  is all what can be decided by  $D$
- We refer to  $\{D, B, F\}$  as the closure of  $D$

## Week 1 - N2



### Constraints

- All conditions that entity sets and relationships should satisfy
- We will focus on three types of constraints
  - Key constraints
  - Referential integrity constraints
  - Degree constraints

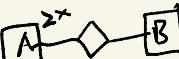
\* Constraints  
↳ key is unique represent each entity  
every entity should have a key

↳ Referential Integrity  
exactly - many-to-one  
1 B many A  
1 A 1 B

- referential  
1 B many A  
1 A 1 B  
All A must link

↳ Degree Constraint

A at least X B

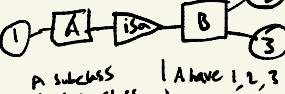


when to use:

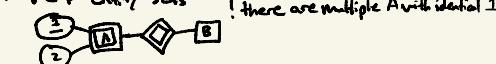
- Subclasses have attribute absent from Superclass
- Subclass has its own relationship

key of subclass (1) key of its superclass (2)

### Subclasses



### Weak Entity Sets



### ER → Relational Schema

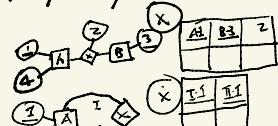


relation schema = name of table + names of its attributes  
database schema = set of relation schemas

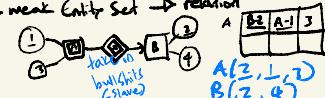
→ relation

can be decomposed

many-to-many → relation



weak Entity Set → relation



### ER Diagram → Relational Schema

- General rules:
  - Each entity becomes a relation
  - Each many-to-many relationship becomes a relation
- Special treatment needed for:
  - Weak entity sets
  - Superclasses
  - Relationships to one-to-one relationships

### Weak Entity Sets

- Products: There are two ways to implement them
  - One way: make them a weak entity set associated with the entity set they belong to.
  - The other way: implement the supporting relationship of them as a separate entity set.
- Relationships to one-to-one relationships:
  - Product: Create a separate entity set for the relationship.
  - Superclass: Create a separate entity set for the relationship.
- This is a simple composition

!! Avoid 2-way relationship!

Oracle, SQL Server, MySQL, MySQL  
↳ DB management system  
↳ store data in form of relations

### Design principle for ER

- be faithful: Capture all requirements
- Avoid Redundancy  
  
waste of space  
positive inconsistency
- keep it simple
- Don't over-use weak Entity Sets

### Example Subclass → Relation



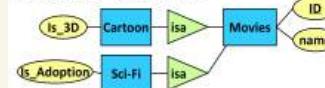
### An Object-Oriented Approach



### Comparison of Approaches

- Not too many relations
  - The old approach
  - The O/O approach uses inheritance and aggregation.
  - Object-oriented approach allows to many-to-one relations.
- Maintainance and reuse
  - The old approach
  - The O/O approach allows inheritance and aggregation.
  - The old approach is less flexible, potentially, when trying to reuse code.
  - Object-oriented approach allows inheritance and aggregation.
  - The old approach is less flexible, potentially, when trying to reuse code.
- More readable and easier readability
  - The old approach
  - The O/O approach allows inheritance and aggregation.
  - The old approach is less flexible, potentially, when trying to reuse code.
  - Object-oriented approach allows inheritance and aggregation.
  - The old approach is less flexible, potentially, when trying to reuse code.

### Subclass → Relation



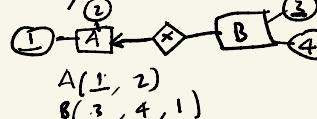
### Subclass → Relation

ER approach  
middle & oo = null  
1 record → \* relation

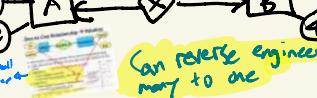
OO approach  
null = good for searching subclass combination  
Issue: may have too many rows

Null approach  
sub: only need 1 relation  
Issue: may have many null

many-to-one → relation



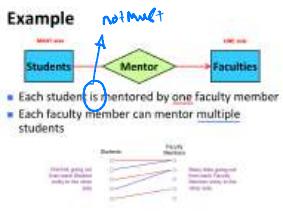
one to One Relationship → relation



is 1st  
B is 2nd server  
see A  
also

- Solution 1  
A(1..1, 1)  
B(1..1, 1)
- Solution 2  
A(1..1, 1, 1)  
B(1..1, 1)

want to  
null  
null



## Exercise

- Each player prefers only one game, but not vice versa



- Any two players are from exactly two different countries



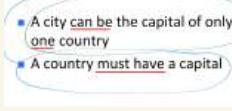
## Exercise

- No two shops sell the same product



## Exercise

- Consider two entity sets, Shops and Companies
  - Each shop sells products from at least one company
  - Each company has its product sold in at least one shop
  - A shop may be the flagship shop of at most one company
  - Each company has at least one flagship shop
  - Draw some relationships between Shops and Companies to capture the above statements

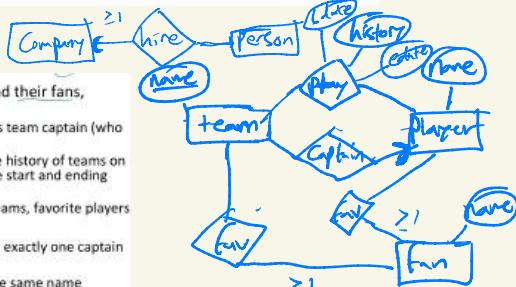


- A city can be the capital of only one country
  - A country must have a capital

- A company must hire at least one person
  - A person must be hired by exactly one company
  - To say "Each and every company must hire at least one person", need degree constraints

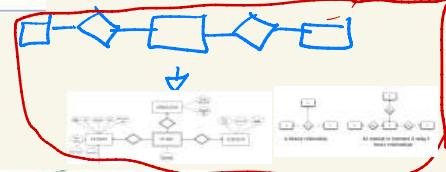


- Record info about teams, players, and their fans, including:
    - For each team, its name, its players, its team captain (who is also a player)
    - For each player, his/her name, and the history of teams on which he/she has played, including the start and ending dates for each team
    - For each fan, his/her name, favorite teams, favorite player
  - Additional information:
    - Each team has at least one player, and exactly one captain
    - Each team has a unique name
    - Two players (or two fans) may have the same name
    - Each fan has at least one favorite team and at least one favorite player



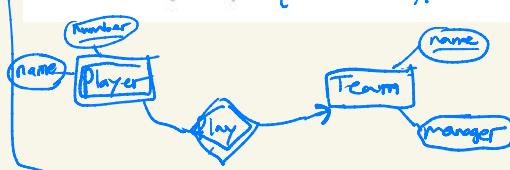
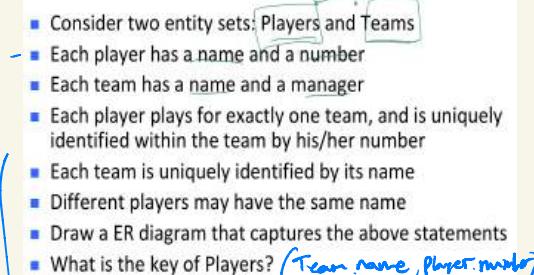
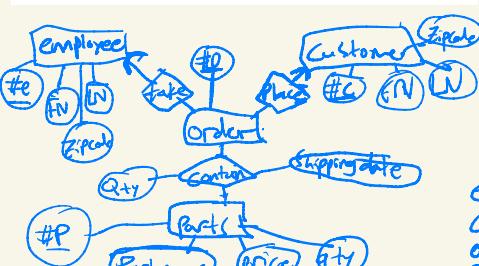
! least one  $\leq$  One = many

at most one  $\equiv$  One



## ER Diagram Design: Exercise

- Consider a mail order database in which **employees** take **orders** for parts from **customers**. The requirements are:
    - Each **employee** is identified by a unique employee number, and has a first name, a last name, and a zip code.
    - Each **customer** is identified by a unique customer number, and has a first name, last names, and a zip code.
    - Each **part** being sold is identified by a unique part number. It has a part name, a price, and a quantity in stock.
    - Each **order** placed by a customer is taken by one employee and is given a unique order number. Each order may contain certain quantities of one or more parts. The shipping data of each part is also recorded.



Employee(#e, FN, LN, ZipCode)  
Customer(#C, FN, LN, ZipCode)  
Order(#O, #E, #C)  
Parts(#P, archive, Qty, Atv)

Contain (#O, #P, Qty, Shipping date)

## Week 3

Problem from not designing good ER diagram

↳ Data anomalies  $\xrightarrow{\text{use normalization}}$

↳ redundancy - value duplicated, attribute appear multiple times

↳ update anomalies - if update wrongly if value duplicated

↳ deletion anomalies

↳ insertion anomalies

$$SA \rightarrow SB$$

Functional Dependencies (FD)  $A_1, A_2, \dots, A_m \rightarrow B_1, B_2, \dots, B_n$

if  $A \rightarrow B, B \rightarrow C$

↳ decide if table is bad

then  $A \rightarrow C$

### Armstrong's Axioms

- Axiom of Reflexivity (Set of attributes  $\rightarrow$  subset of the attributes)

$$A, B, C, D \rightarrow A, C$$

$\square$  If NRIC  $\rightarrow$  Name

$\square$  Then NRIC, Age  $\rightarrow$  Name, Age

$\square$  and NRIC, Salary, Weight  $\rightarrow$  Name, Salary, Weight

$\square$  and NRIC, Addr, Postal  $\rightarrow$  Name, Addr, Postal

- Axiom of Augmentation

given  $A \rightarrow B$  we always have  $AC \rightarrow BC$  for any C

- Axiom of Transitivity

given  $A \rightarrow B$  and  $B \rightarrow C$  we always have  $A \rightarrow C$

$\square$  Example

$\square$  If NRIC  $\rightarrow$  Addr, and Addr  $\rightarrow$  Postal

$\square$  Then NRIC  $\rightarrow$  Postal

- Superkeys of a table = a set of attributes in a table that determines all other attributes

↳ # keys in each set

↳ minimal if A is superkey

$$AB \rightarrow \text{superkey}$$

$$AB \rightarrow \text{superkey} A \rightarrow \text{key}$$

Not key

↳ Candidate Keys

↳ each key  $\rightarrow$  candidate key

↳ Primary key n Secondary keys

↳ we choose 1 primary key and the rest is secondary keys

intuitive solution  $\rightarrow$  draw diagram & find directive path

↳ check if  $R \rightarrow B$  valid

If unique mean either key/superkey

### Where Do FDs Come From?

Example

$\square$  Purchase( CustomerID, ProductID, ShopID, Price, Date )

$\square$  Requirement: No two customers buy the same product

$\square$  FD implied: ProductID  $\rightarrow$  CustomerID



C1  
C2  
P1  
P2

C1  
C1  
P1

- Four attributes: A, B, C, D
- Given:  $B \rightarrow D$ ,  $DB \rightarrow A$ ,  $AD \rightarrow C$
- Can you prove  $B \rightarrow C$ ?
- Doable with Armstrong's axioms, but troublesome
- We will discuss a more convenient approach

Parts( p#, name, price, qnty )

Order( o#, e#, c# ) / from 2 many-to-one relationships

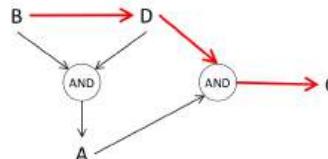
Contain( p#, o#, qty, shipdate )

employee( e#, fn, ln, zipcode )

customer( c#, (fn, ln, zipcode) )

## An Intuitive Solution

- Four attributes: A, B, C, D
- Given:  $B \rightarrow D$ ,  $DB \rightarrow A$ ,  $AD \rightarrow C$
- Can you prove  $B \rightarrow C$ ?

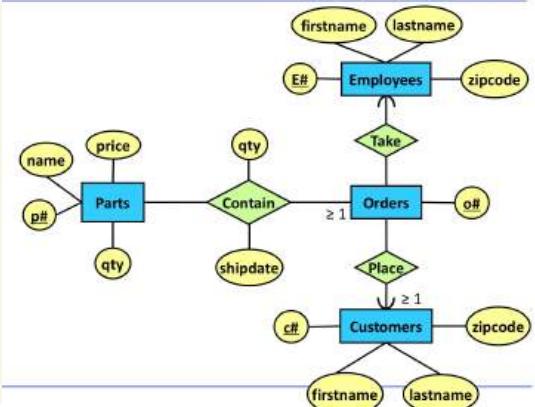


- First, activate B
  - Activated set = { B }
- Second, activate whatever B can activate
  - Activated set = { B, D }, since  $B \rightarrow D$
- Third, use all activated elements to activate more
  - Activated set = { B, D, A }, since  $DB \rightarrow A$
- Repeat the third step, until no more activation is possible
  - Activated set = { B, D, A, C }, since  $AD \rightarrow C$ ; done

## ER Diagram Design: Exercise

- Consider a mail order database in which **employees** take **orders** for parts from **customers**. The requirements are:
- Each **employee** is identified by a unique employee number, and has a first name, a last name, and a zip code.
- Each **customer** is identified by a unique customer number, and has a first name, last names, and a zip code.
- Each **part** being sold is identified by a unique part number. It has a part name, a price, and a quantity in stock.
- Each **order** placed by a customer is taken by one employee and is given a unique order number. Each order may contain certain quantities of one or more parts. The shipping date of each part is also recorded.

- Parts( p#, name, price, stock )
- Employees( e#, fname, lname, ZIP )
- Customers( c#, fname, lname, ZIP )
- Orders( o#, e#, c# ) – from two many-to-one relationships
- Contain( o#, p#, qty, shipdate ) – from many-to-many relationships



Employees  
E#, FN, LN, ZIP

order  
O#, E#, C#

Customer  
C#, ZIP, FN, LN

parts  
P#, name, price, qty

contain  
P#, O#, qty, shipdate

## Closure

- Let  $S = \{A_1, A_2, \dots, A_n\}$  be a set of  $n$  attributes
- Closure of  $S$  is the set of attributes that can be determined by  $A_1, A_2, \dots, A_n$

Notation:  $\{A_1, A_2, \dots, A_n\}^+$

### Example

- Given  $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$
- $\{A\}^+ = \{A, B, C, D, E\}$
- $\{B\}^+ = \{B, C, D, E\}$
- $\{D\}^+ = \{D, E\}$
- $\{E\}^+ = \{E\}$

We discuss the usage of closure for finding the keys in a table

- This is a necessary step for checking whether a table is good or not
- Concepts
- Superkeys
- Keys
- Candidate keys
- Primary key / Secondary keys

Notion of keys in a table  
keys in an entity set

## Superkeys of a Table

Name	NRIC	Postal	Address
Alice	1234	939450	Jurong East
Bob	5678	234122	Pasir Ris
Cathy	3576	420923	Yishun

- Definition: A set of attributes in a table that determines all other attributes

### Example:

- $\{NRIC\}$  is a superkey (happens to be minimal)
- Since  $NRIC \rightarrow Name, Postal, Address$
- $\{NRIC, Name\}$  is a superkey (not minimal)
- Since  $\{NRIC, Name\} \rightarrow Postal, Address$

## Keys of a Table

Name	NRIC	Postal	Address
Alice	1234	939450	Jurong East
Bob	5678	234122	Pasir Ris
Cathy	3576	420923	Yishun

- Definition: A superkey that is minimal

i.e., if we remove any attribute from the superkey, it will not be a superkey anymore

### Example:

- $\{NRIC\}$  is a superkey (happens to be minimal)
- Since  $NRIC \rightarrow Name, Postal, Address$
- $\{NRIC, Name\}$  is a superkey
- Since  $\{NRIC, Name\} \rightarrow Postal, Address$
- $NRIC$  is a key, but  $\{NRIC, Name\}$  is not a key, but a superkey

- Note: Not to be confused with the keys of entity sets because keys in entity sets need not be minimal.

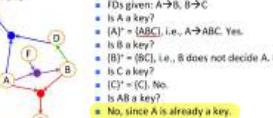
## Exercise

- A table with six attributes  $A, B, C, D, E, F$

$AB \rightarrow C, AD \rightarrow E, B \rightarrow D, AF \rightarrow B$

Compute the following closures

- $\{B, C\}^+ = \{B, C, D\}$
- $\{A, B\}^+ = \{A, B, C, D, E\}$
- $\{A, F\}^+ = \{A, B, C, D, E, F\}$



### Example

- Definition of a Key: A minimal set of attributes that determines all other attributes
- A table  $R(A, B, C)$
- FDs given:  $A \rightarrow B, B \rightarrow C$
- Is  $A$  a key?
- $\{A\}^+ = \{ABC\}$ , i.e.,  $A \rightarrow ABC$ . Yes.
- $\{B\}^+ = \{BC\}$ , i.e.,  $B$  does not decide  $A$ . No.
- $\{C\}^+ = \{C\}$ . No.
- Is  $B$  a key?
- Is  $C$  a key?
- Is  $A$  a key?
- No, since  $A$  is already a key.
- What about  $BC, AC, ABC$ ?

### Example

- Definition of a Key: A minimal set of attributes that determines all other attributes
- A table  $R(A, B, C)$
- FDs given:  $A \rightarrow B$
- Is  $A$  a key?
- $\{A\}^+ = \{A, B\}$ . No.
- Is  $B$  or  $C$  a key?
- $\{B\}^+ = \{B\}$ ,  $\{C\}^+ = \{C\}$ . No.
- Is  $AB$  or  $BC$  a key?
- Is  $AC$  a key?
- $\{AC\}^+ = \{ABC\}$ . Yes.
- Is  $ABC$  a key?

## Example

- A table  $R(A, B, C, D)$

$AB \rightarrow C, AD \rightarrow B, B \rightarrow D$

First, enumerate all attribute combinations:

- $\{A\}, \{B\}, \{C\}, \{D\}$

$\{AB\}, \{AC\}, \{AD\}, \{BC\}, \{BD\}, \{CD\}$

$\{ABC\}, \{ABD\}, \{ACD\}, \{BCD\}$

$\{ABCD\}$

$\{A\}^+ = \{A\}$

$\{B\}^+ = \{B\}$

$\{C\}^+ = \{C\}$

$\{D\}^+ = \{D\}$

$\{AB\}^+ = \{AB\}$

$\{AC\}^+ = \{AC\}$

$\{AD\}^+ = \{AD\}$

$\{BC\}^+ = \{BC\}$

$\{BD\}^+ = \{BD\}$

$\{CD\}^+ = \{CD\}$

$\{ABD\}^+ = \{ABD\}$

$\{ACD\}^+ = \{ACD\}$

$\{BCD\}^+ = \{BCD\}$

$\{ABCD\}^+ = \{ABCD\}$

## Closure & FD

- To prove that  $X \rightarrow Y$  holds, we only need to show that  $\{X\}^+$  contains  $Y$
- $AB \rightarrow C, AD \rightarrow E, B \rightarrow D, AF \rightarrow B$
- Prove that  $AF \rightarrow D$
- $\{AF\}^+ = \{AFBCDE\}$ , which contains  $D$
- Therefore,  $AF \rightarrow D$  holds
- To prove that  $X \rightarrow Y$  does not hold, we only need to show that  $\{X\}^+$  does not contain  $Y$
- $AB \rightarrow C, AD \rightarrow E, B \rightarrow D, AF \rightarrow B$
- Prove that  $AD \rightarrow F$  does not hold
- $\{AD\}^+ = \{ADE\}$ , which does not contain  $F$
- Therefore,  $AD \rightarrow F$  does not hold

## Finding the Keys: Algorithm

- Check all possible combinations of attributes in the table
- Example:  $A, B, C, AB, BC, AC, ABC$
- For each combination, compute its closure
- Example:  $\{A\}^+ = \dots, \{B\}^+ = \dots, \{C\}^+ = \dots, \{ABC\}^+ = \dots$
- If a closure contains ALL attributes, then the combination might be a key (or superkey)
- Example:  $\{A\}^+ = \{ABC\}$
- Make sure that you select only keys
- Example:  $\{A\}^+ = \{ABC\}, \{AB\}^+ = \{ABC\}$ , don't select  $AB$

## Candidate Keys

Name	NRIC	StudentID	Postal	Address
Alice	1234	1	939450	Jurong East
Bob	5678	2	234122	Pasir Ris
Cathy	3576	3	420923	Yishun

- A table may have multiple keys
- In that case, each key is referred to as a candidate key

### Example:

- $\{NRIC\}$  is a key
- Since  $NRIC \rightarrow Name, StudentID, Postal, Address$
- $\{StudentID\}$  is a key
- Since  $StudentID \rightarrow Name, NRIC, Postal, Address$
- Both  $\{NRIC\}$  and  $\{StudentID\}$  are candidate keys

## Primary and Secondary Keys

Name	NRIC	StudentID	Postal	Address
Alice	1234	1	939450	Jurong East
Bob	5678	2	234122	Pasir Ris
Cathy	3576	3	420923	Yishun

- When a table have multiple keys ...

We choose one of them as the primary key

The others are referred to as secondary keys

### Example:

- $\{NRIC\}$  is a key
- $\{StudentID\}$  is a key
- If we choose  $\{NRIC\}$  as the primary key
- Then  $\{StudentID\}$  is the secondary key

### Example

- Definition of a Key: A minimal set of attributes that determines all other attributes
- A table  $R(A, B, C)$
- FDs given:  $A \rightarrow B$
- Is  $A$  a key?
- $\{A\}^+ = \{A, B\}$ . No.
- Is  $B$  or  $C$  a key?
- $\{B\}^+ = \{B\}$ ,  $\{C\}^+ = \{C\}$ . No.
- Is  $AB$  or  $BC$  a key?
- Is  $AC$  a key?
- $\{AC\}^+ = \{ABC\}$ . Yes.
- Is  $ABC$  a key?

## A Small Trick

- Always check small combinations first
- A table  $R(A, B, C, D)$
- $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A$
- Compute the closures:
  - $\{A\}^+ = \{ABCD\}$ ,  $\{B\}^+ = \{ABCD\}$ ,  $\{C\}^+ = \{ABCD\}$ ,  $\{D\}^+ = \{ABCD\}$
  - No need to check others
  - The others are all superkeys but not keys
- Keys:  $\{A\}, \{B\}, \{C\}, \{D\}$ 
  - A table  $R(A, B, C, D)$
  - $AB \rightarrow C, AD \rightarrow B, B \rightarrow D$
  - Notice that  $A$  does not appear at the right hand side of any functional dependencies
  - In that case,  $A$  must be in every key
  - Keys of  $R$ :  $AB, AD$  (From the previous slide)
  - In general, if an attribute that does not appear in the right hand side of any FD, then it must be in every key

## Summary

- Superkeys
  - A set of attributes that determines all other attributes in a table
- Keys
  - A minimal set of attributes that determines all other attributes in a table
- Example
  - $R(A, B, C, D)$
  - Given:  $A \rightarrow BCD$ ,  $BC \rightarrow A$
  - $A$  is a key and a superkey
  - $BC$  is also a key and a superkey
  - $AB$  is not a key; it is only a superkey

## Exercise (Find the Keys)

- A table  $R(A, B, C, D)$
- $A \rightarrow B, A \rightarrow C, C \rightarrow D$
- A must be in every key
- Compute the closures:
  - $\{A\}^+ = \{ABCD\}$
  - No need to check others
- Keys:  $\{A\}$

A

## Exercise (Find the Keys)

- A table  $R(A, B, C, D, E)$
- $AB \rightarrow C, C \rightarrow B, CB \rightarrow D, D \rightarrow E$
- A must be in every key
- Compute the closures:
  - $\{A\}^+ = \{A\}$
  - $\{AB\}^+ = \{ABCDE\}$
  - $\{AC\}^+ = \{ACBDE\}$
  - $\{AD\}^+ = \{AD\}$ ,  $\{AE\}^+ = \{AE\}$
  - $\{ADE\}^+ = \{ADE\}$

A

## Exercise (Find the Keys)

- A table  $R(A, B, C, D, E, F)$
- $AB \rightarrow C, C \rightarrow B, CB \rightarrow D, D \rightarrow E, E \rightarrow F$
- A must be in every key
- Compute the closures:
  - $\{A\}^+ = \{A\}$
  - $\{AB\}^+ = \{ABCDEF\}$
  - $\{AC\}^+ = \{ACBDEF\}$
  - $\{AD\}^+ = \{AD\}$ ,  $\{AE\}^+ = \{AE\}$
  - $\{ABC\}^+ = \{ABC\}$
  - $\{ABD\}^+ = \{ABD\}$ ,  $\{ACD\}^+ = \{ACD\}$
  - $\{ACE\}^+ = \{ACE\}$
  - $\{ABCDEF\}^+ = \{ABCDEF\}$
  - $\{ADE\}^+ = \{ADE\}$

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A

B

C

D

E

F

A



$\text{key } BC\bar{C}^+ = \{B, C, D, E, F\}$   
 Violation  $B \rightarrow D$   $\{BC\}^+ = \{B, D\}$   
 $R(B, D) \checkmark$   
 $R(B, C, D, E, F)$   $\text{key } BCF \rightarrow EX$   
 $EX^+ = \{E, C, EF\}$   
 $R(C, E) \checkmark$   
 $R(A, B, C) \text{ key } BCF$   
 $BC^+ = \{B\}$   $EF^+ = \{E, F\}$   
 $EBC^+ = \{E\}$   $EX^+ = \{E, C, EF\}$   
 $EC\bar{C}^+ = EC^+ = \{E, C\}$   
 $EF\bar{F}^+ = \{F\}$   $(EBC\bar{C}^+ = EBC) \rightarrow A$   
 $EX\bar{C}^+ = \{E, B, C, AF\}$   
 $R(A, B, C) \checkmark$   
 $R(E, B, C, F) \checkmark$

## Exercise

- $R(A, B, C, D, E, F)$
- Given FDs:  $B \rightarrow D$ ,  $C \rightarrow E$ ,  $DE \rightarrow A$
- Keys:  $BCF$
- $B \rightarrow D$  violates BCNF
- Decompose  $R$ :
  - $R_1(B, D)$ ,  $R_2(A, B, C, E, F)$
  - $R_1$  is in BCNF
- What about  $R_2$ ? Tricky case. We could address the tricky case, but ...
- we also notice  $C \rightarrow E$  is violating. We can just decompose rightaway:
- Decompose  $R_2$ ,  $[C] = \{C, E\}$ 
  - $R_3(C, E)$ ,  $R_4(A, B, C, F)$
- $R_3(C, E)$  is in BCNF
- What about  $R_4(A, B, C, F)$ ?
- Tricky case
- Check closures
  - $(A)^+ = \{A\}$ ,  $(B)^+ = \{BD\}$ ,  $(C)^+ = \{CE\}$ ,  $(F)^+ = \{F\}$
  - $(AB)^+ = \{ABD\}$ ,  $(AC)^+ = \{ACE\}$ ,  $(AF)^+ = \{AF\}$ ,  $(BC)^+ = \{BCDF\}$
- So there is a non-trivial FD:  $BC \rightarrow A$
- Keys of  $R_4$ :  $BCF$
- There is a non-trivial FD:  $BC \rightarrow A$
- It violates BCNF
- Decompose  $R_4$ 
  - $R_5(B, C, A)$ ,  $R_6(B, C, F)$
- Final decomposition:
  - $R_1(B, D)$ ,  $R_3(C, E)$ ,  $R_5(B, C, A)$ ,  $R_6(B, C, F)$

## Properties of BCNF Decomposition

- Good properties
  - No update or deletion anomalies
  - Very small redundancy
  - The original table can always be reconstructed from the decomposed tables if functional dependencies are preserved (this is called the **lossless join** property)
  - Reconstruction is at the schema level only if some FDs not preserved
- Bad property
  - It may not preserve all functional dependencies

## Lossless Join Property

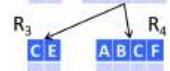
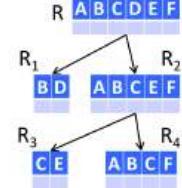
Name	NRIC	Phone	Address
Alice	1234	67899876	Jurong East
Alice	1234	83848384	Jurong East
Bob	5678	98765432	Pasir Ris

- The table above can be perfectly reconstructed using the decomposed tables below as all FDs preserved:
- $\{NRIC, Phone\} \rightarrow Name, Address$ ,  $NRIC \rightarrow Name, Address$

Name	NRIC	Address	NRIC	Phone
Alice	1234	Jurong East	1234	67899876
Bob	5678	Pasir Ris	1234	83848384

## Why BCNF guarantees lossless join?

- Say we decompose a table  $R$  into two tables  $R_1$  and  $R_2$
- The decomposition guarantees lossless join, whenever the common attributes in  $R_1$  and  $R_2$  constitute a **superkey** of  $R_1$  or  $R_2$
- Example
  - $R(A, B, C)$  decomposed into  $R_1(A, B)$  and  $R_2(B, C)$ , with  $B$  being the key of  $R_2$
  - $R(A, B, C, D)$  decomposed into  $R_1(A, B, C)$  and  $R_2(B, C, D)$ , with  $BC$  being the key of  $R_1$
- The decomposition of  $R$  guarantees lossless join, whenever the common attributes in  $R_1$  and  $R_2$  constitute a superkey of  $R_1$  or  $R_2$
- BCNF Decomposition of  $R$ 
  - Find a BCNF violation  $X \rightarrow Y$
  - Compute  $\{X\}^+$
  - $R_1$  contains all attributes in  $\{X\}^+$
  - $R_2$  contains  $X$  and all attributes NOT in  $\{X\}^+$
  - $X$  is both in  $R_1$  and  $R_2$
  - And  $X$  is a superkey of  $R_1$
  - Therefore,  $R_1$  and  $R_2$  is a lossless decomposition of  $R$



## Dependency Preservation

- Given: Table R(A, B, C)

with  $AB \rightarrow C, C \rightarrow B$

Keys: AB, AC

BCNF Decomposition

$R_1(B, C)$

$R_2(A, C)$

Non-trivial FDs on  $R_1$ :  $C \rightarrow B$

Non-trivial FDs on  $R_2$ : none

The other FD,  $AB \rightarrow C$ , should hold on any individual table, but it is "lost"

We say that a BCNF decomposition does not always preserve all FDs

Why do we want to preserve FDs?

Because we want to make it easier to avoid "inappropriate" updates

Previous example

We have two tables  $R_1(B, C)$ ,  $R_2(A, C)$

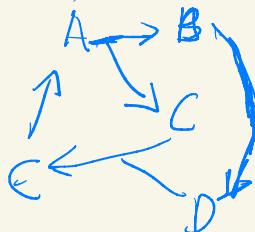
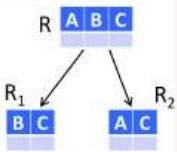
We have  $C \rightarrow B$  and  $AB \rightarrow C$

Due to  $AB \rightarrow C$ , we are not suppose to have two tuples  $(a1, b1, c1)$  and  $(a1, b1, c2)$

But as we store A and C separately in  $R_1$  and  $R_2$ , it is not easy to check whether such two tuples exist at the same time

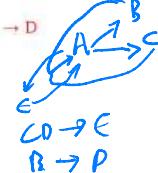
That is, if someone wants to insert  $(a1, b1, c2)$ , it is not easy for us to check whether  $(a1, b1, c1)$  already exists

This is less than ideal



## Question 4(a)

- Decomposition:  $R1(A, B, C)$  and  $R2(A, D, E)$ ; FDs:  $A \rightarrow BC$ ,  $E \rightarrow A$ ,  $CD \rightarrow E$ ,  $B \rightarrow D$ ; Keys: A, E, CD, BC
- Decomposition  $R1(A, B, C)$  and  $R2(A, D, E)$  is lossless because
  - $R1$  and  $R2$  have a common attribute A, and
  - A is a superkey for  $R1(A, B, C)$
- FDs that hold on  $R1(A, B, C)$ 
  - $A \rightarrow BC$ ,  $BC \rightarrow A$  since  $R1$  contains A, B, and C
- FDs that hold on  $R2(A, D, E)$ 
  - $E \rightarrow A$ ,  $A \rightarrow E$  since  $R2$  contains A and E
- Two other FDs need to be checked:  $CD \rightarrow E$ ,  $B \rightarrow D$ 
  - From  $A \rightarrow BC$ ,  $BC \rightarrow A$ ,  $B \rightarrow A$ ,  $A \rightarrow E$ , we have:
  - $\{B\}^+ = \{B\}$ , so  $B \rightarrow D$  is not preserved
  - $\{CD\}^+ = \{CD\}$ , so  $CD \rightarrow E$  is not preserved
- Decomposition is NOT dependency-preserving



## Question 4(b)

- Decomposition:  $R3(A, B, C, D)$  and  $R4(C, D, E)$ ; FDs:  $A \rightarrow BC$ ,  $E \rightarrow A$ ,  $CD \rightarrow E$ ,  $B \rightarrow D$ ; Keys: A, E, CD, BC
- Decomposition  $R3(A, B, C, D)$  and  $R4(C, D, E)$  is lossless because
  - $R3$  and  $R4$  have common attributes CD, and
  - CD is a superkey for  $R4(C, D, E)$
- FDs that hold on  $R3(A, B, C, D)$ 
  - $A \rightarrow BC$ ,  $BC \rightarrow A$ ,  $CD \rightarrow E$ ,  $CD \rightarrow A$ , and  $B \rightarrow D$  since  $R1$  contains A, B, C, and D
- FDs that hold on  $R4(C, D, E)$ 
  - $CD \rightarrow E$ ,  $E \rightarrow CD$
- One other FD needs to be checked:  $E \rightarrow A$ 
  - From  $A \rightarrow BC$ ,  $BC \rightarrow A$ ,  $CD \rightarrow A$ ,  $B \rightarrow D$ ,  $CD \rightarrow E$ ,  $E \rightarrow CD$  and, we have:
  - $\{E\}^+ = \{A, B, C, D, E\}$ ,  $E \rightarrow CD$  holds in  $R4$ ,  $CD \rightarrow A$  holds in  $R3$ ,  $E \rightarrow A$  is preserved
- Decomposition is dependency-preserving

# WKS 4 Third normal form (3NF)

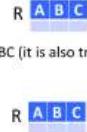
## 1NF, 2NF

- Key-attribute:** An attribute in a multi-attribute key
- Key-attribute(s) = Partial key or part of a key**
- 1NF:** All attributes have atomic values
- 2NF:** Every non-key attribute is dependent on the whole of **EVERY** candidate key
  - Even so, may still have additional dependencies, such as (non-key-attribute X)  $\rightarrow$  (non-key-attribute Y) in relation

## Third Normal Form (3NF)

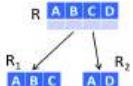
all attr important

- A relaxation of BCNF that
  - Is less strict
  - Allows decompositions that **always** preserve functional dependencies
- Definition: A table satisfies 3NF, if and only if for every non-trivial  $X \rightarrow Y$ 
  - Either X contains a key
  - Or each attribute in Y is contained in a key
- Example:
  - Given FDs:  $C \rightarrow B$ ,  $AB \rightarrow C$ ,  $BC \rightarrow C$
  - Keys:  $\{AB\}$ ,  $\{AC\}$
  - $AB \rightarrow C$  is OK, since  $AB$  is a key of R
  - $C \rightarrow B$  is OK, since  $B$  is in a key of R
  - $BC \rightarrow C$  is OK, since  $C$  is in AC and in BC (it is also trivial)
  - So R is in 3NF
- Another Example:
  - Given FDs:  $A \rightarrow B$ ,  $B \rightarrow C$
  - Keys:  $\{A\}$
  - $A \rightarrow B$  is OK, since  $A$  is a key of R
  - $B \rightarrow C$  is NOT OK, since  $C$  is NOT in a key of R, and it is NOT in the left hand side
  - So R is NOT in 3NF



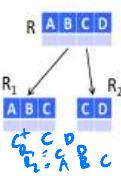
## 3NF Decomposition

- Given: A table NOT in 3NF
- Objective: Decompose it into smaller tables that are in 3NF
- Example
  - Given:  $R(A, B, C, D)$
  - FDs:  $AB \rightarrow C$ ,  $C \rightarrow B$ ,  $A \rightarrow D$
  - Keys:  $\{AB\}$ ,  $\{AC\}$
  - $R$  is not in 3NF, due to  $A \rightarrow D$
  - 3NF decomposition of R:  
 $R_1(A, B, C)$ ,  $R_2(A, D)$



## 3NF Decomposition Algorithm

- Given: A table R, and a set S of FDs
  - e.g.  $R(A, B, C, D)$
  - $S = \{A \rightarrow BD, AB \rightarrow C, B \rightarrow A, C \rightarrow D\}$
- Step 1: Derive a minimal basis of S
  - e.g., a minimal basis of 5 is  $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$
- Step 2: In the minimal basis, combine the FDs whose left hand sides are the same
  - e.g., after combining  $A \rightarrow B$  and  $A \rightarrow C$ , we have  $A \rightarrow BC$ ,  $C \rightarrow D$
- Step 3: Create a table for each FD remained
  - e.g.,  $R_1(A, B, C)$ ,  $R_2(C, D)$
- Step 4: If none of the tables contain a key of the original table R, create a table that contains a key of R
- Step 5: Remove redundant tables (schema is a subset of another)



### Minimal Basis is not always unique

- For given set of FDs, its minimal basis may not be unique
- Example:
  - Given  $R(A, B, C)$  and  $\{A \rightarrow B, A \rightarrow C, B \rightarrow C, C \rightarrow A\}$
  - Minimal basis 1:  $\{A \rightarrow B, C \rightarrow A\}$
  - Minimal basis 2:  $\{A \rightarrow C, B \rightarrow C, C \rightarrow A\}$
  - Different minimal basis may lead to different 3NF decompositions



## 3NF, BCNF

- 3NF:** 2NF + all key-attributes determined **ONLY** by candidate keys (in whole or in part)
  - (non-key-attribute X)  $\rightarrow$  (non-key-attribute Y) cannot exist anymore, RHS must be key attribute in 3NF
  - But candidate keys may have **overlapping** attributes
  - May result in key-attribute(s) of one key depends on key-attribute(s) of another key (this dependency is eliminated in BCNF)
- BCNF:** In all dependencies (FDs), LHS must contain key (cannot depend on partial key)

## BCNF vs. 3NF

$BCNF > 3NF$

- BCNF:** For any non-trivial FD
  - its left hand side (lhs) is a superkey
- 3NF:** For any non-trivial FD
  - Either its lhs is a superkey
  - Or each attribute on its right hand side either appear in the lhs or in a key
- Observation:** BCNF is stricter than 3NF
- Therefore**
  - A table that satisfies BCNF must satisfy 3NF, but not vice versa
  - A table that violates 3NF must violate BCNF, but not vice versa
- BCNF Decomposition:**
  - Avoids insertion, deletion, and update anomalies
  - Eliminates most redundancies
  - But does not always preserve all FDs
- 3NF Decomposition:**
  - Avoids insertion, deletion, and update anomalies
  - May lead to a bit more redundancy than BCNF
  - Always preserve all FDs
- So which one to use?**
  - A logical approach
    - Go for a BCNF decomposition first
    - If it preserves all FDs, then we are done
    - If not, then go for a 3NF decomposition instead

## Minimal Basis

- Given a set S of FDs, the **minimal basis** of S is a **simplified** version of S
- Previous example:
  - $S = \{A \rightarrow BD, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
  - A minimal basis:  $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$
- How simplified?
- Three conditions.
  - Condition 1:** For any FD in the minimal basis, its right hand side has only one attribute.
  - Example in S:  $A \rightarrow BD$  does not satisfy this condition
  - That is why  $A \rightarrow BD$  is not in the minimal basis
- Previous example:
  - $S = \{A \rightarrow BD, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
  - A minimal basis:  $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$
- Condition 2:** No FD in the minimal basis is redundant.
  - That is, no FD in the minimal basis can be derived from the other FDs.
  - Example in S:  $BC \rightarrow D$  can be derived from  $C \rightarrow D$
  - That is why  $BC \rightarrow D$  is not in the minimal basis
- Condition 3:** For each FD in the minimal basis, none of the attributes on the left hand side is redundant
  - That is, for any FD in the minimal basis, if we remove an attribute from the left hand side, then the resulting FD is a new FD that cannot be derived from the original set of FDs
  - Example:
    - S contains  $AB \rightarrow C$
    - If we remove B from the left hand side, we have  $A \rightarrow C$
    - $A \rightarrow C$  can be derived from S, as  $\{A\}^* = \{ABDC\}$
    - This indicates that  $A \rightarrow C$  is "hidden" in S
    - Hence, we can replace  $AB \rightarrow C$  with  $A \rightarrow C$ , as  $A \rightarrow C$  is "simpler"
    - This is why  $AB \rightarrow C$  is not in the minimal basis

## Algorithm for Minimal Basis

- Given: a set S of FDs
- Example:  $S = \{A \rightarrow BD, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Step 1: Transform the FDs, so that each right hand side contains only one attribute
- Result:  $S = \{A \rightarrow B, A \rightarrow D, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Reason:
  - Condition 1 for minimal basis: The right hand side of each FD contains only one attribute
- Result of Step 1:
  - $S = \{A \rightarrow B, A \rightarrow D, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
  - Step 2: Remove redundant FDs
  - Is  $A \rightarrow B$  redundant?
    - i.e., Is  $A \rightarrow B$  implied by other FDs in S?
    - Let's check
    - Without  $A \rightarrow B$ , we have  $\{A \rightarrow D, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
    - Given those FDs, we have  $\{A\}^* = \{AD\}$ , which does not contain B
    - Therefore,  $A \rightarrow B$  is not implied by the other FDs
  - Continue Step 2: Remove redundant FDs
  - Is  $A \rightarrow D$  redundant?
    - i.e., Is  $A \rightarrow D$  implied by other FDs in S?
    - Let's check
    - Without  $A \rightarrow D$ , we have  $\{A \rightarrow B, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
    - Given those FDs, we have  $\{A\}^* = \{ABCD\}$ , which contains D
    - Therefore,  $A \rightarrow D$  is implied by the other FDs
    - Hence,  $A \rightarrow D$  is redundant and should be removed
    - Result:  $S = \{A \rightarrow B, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
    - Is  $AB \rightarrow C$  redundant?
      - i.e., Is  $AB \rightarrow C$  implied by other FDs in S?
      - Let's check
      - Without  $AB \rightarrow C$ , we have  $\{A \rightarrow B, C \rightarrow D, BC \rightarrow D\}$
      - Given those FDs, we have  $\{AB\}^* = \{AB\}$ , which does not contain C
      - Therefore,  $AB \rightarrow C$  is NOT implied by the other FDs
      - Hence,  $AB \rightarrow C$  is not redundant and should not be removed
      - Is  $C \rightarrow D$  redundant?
        - i.e., Is  $C \rightarrow D$  implied by other FDs in S?
        - Let's check
        - Without  $C \rightarrow D$ , we have  $\{A \rightarrow B, AB \rightarrow C, BC \rightarrow D\}$
        - Given those FDs, we have  $\{C\}^* = \{C\}$ , which does not contain D
        - Therefore,  $C \rightarrow D$  is NOT implied by the other FDs and should not be removed
        - Is  $BC \rightarrow D$  redundant?
          - i.e., Is  $BC \rightarrow D$  implied by other FDs in S?
          - Let's check
          - Without  $BC \rightarrow D$ , we have  $\{A \rightarrow B, AB \rightarrow C, C \rightarrow D\}$
          - Given those FDs, we have  $\{BC\}^* = \{BCD\}$ , which contains D
          - Therefore,  $BC \rightarrow D$  is implied by the other FDs
          - Hence,  $BC \rightarrow D$  is redundant and should be removed
          - Result:  $S = \{A \rightarrow B, AB \rightarrow C, C \rightarrow D\}$
          - Result of Step 2:
            - $S = \{A \rightarrow B, AB \rightarrow C, C \rightarrow D\}$
            - Step 3: Remove redundant attributes on the left hand side (lhs) of each FD
              - Only  $AB \rightarrow C$  has more than one attribute on the lhs
              - Let's check
              - Is  $A$  redundant?
                - If we remove A, then  $AB \rightarrow C$  becomes  $B \rightarrow C$
                - Whether this removal is OK depends on whether  $B \rightarrow C$  is "hidden" in S already
                - If  $B \rightarrow C$  is "hidden" in S, then the removal of A is OK, (since the removal does not add extra information into S)
                - Is  $B \rightarrow C$  "hidden" in S?
                - Check: Given S, we have  $\{B\}^* = \{B\}$ , which does NOT contain C
                - Therefore,  $B \rightarrow C$  is not "hidden" in S, and hence, A is NOT redundant
              - Step 3: Remove redundant attributes on the left hand side (lhs) of each FD
                - Only  $AB \rightarrow C$  has more than one attribute on the lhs
                - Let's check
                - Is B redundant?
                  - If we remove B, then  $AB \rightarrow C$  becomes  $A \rightarrow C$
                  - Whether this is OK depends on whether  $A \rightarrow C$  is "hidden" in S
                  - Is  $A \rightarrow C$  "hidden" in S?
                  - Check: Given S, we have  $\{A\}^* = \{ABCD\}$ , which contains C
                  - Therefore,  $A \rightarrow C$  is "hidden" in S
                  - Hence, we can simplify  $AB \rightarrow C$  to  $A \rightarrow C$
                  - Final minimal basis:  $S = \{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$



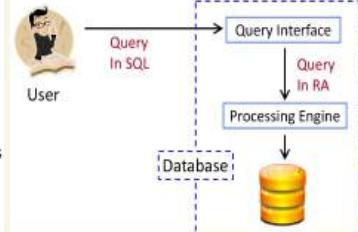
## This Lecture

- Motivation for relational algebra
- Relational algebraic operators
  - Selection:  $\sigma_{A > 100} R_1$
  - Projection:  $\Pi_{A, B} R_1$
  - Union:  $R_1 \cup R_2$
  - Intersection:  $R_1 \cap R_2$
  - Difference:  $R_1 - R_2$
  - Natural Join:  $R_1 \bowtie R_2$
  - Theta Join:  $R_1 \bowtie_{R1.A=R2.B \text{ AND } R1.B=R2.B} R_2$

## Relational Algebra: Motivation

- We have **specification** of an DB application
- We use **ER-diagram** for a **conceptual design** of database
- We transform ER-diagram into **database schema** (i.e., the schemas of a set of tables)
- We **normalize** the schema, and then insert some tuples into the tables
- Now what?
- How do we perform queries on those tables?
  - Database side: Relational Algebra (RA)
  - User side: Structured Query Language (SQL)

## Relational Algebra: Motivation



## Relational Algebra

- A mathematical way to formulate queries on relations (i.e., tables)
- Has numerous **operators** for query formulation
- Example
  - Given: Two relations  $R_1(A, B, C)$ ,  $R_2(A, B, C)$
  - Selection:  $\sigma_{A > 100} R_1$
  - Projection:  $\Pi_{A, B} R_1$
  - Union:  $R_1 \cup R_2$
  - Intersection:  $R_1 \cap R_2$
  - And a few others...

## Selection $\sigma$ (row-wise operation)

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

No agg  
(sum avg)

- Query: "Find me the student named Alice"

$\sigma_{Name = 'Alice'}$  Students  $\rightarrow$  1 tuple

| Results | ID   | Name  | Age | School |
|---------|------|-------|-----|--------|
|         | 1234 | Alice | 20  | SCSE   |

- Query: "Find the students in SCSE"

$\sigma_{School = 'SCSE'}$  Students

| Results | ID   | Name  | Age | School |
|---------|------|-------|-----|--------|
|         | 1234 | Alice | 20  | SCSE   |
|         | 3742 | Cathy | 22  | SCSE   |

- Query: "Find the SCSE students under 21"

$\sigma_{School = 'SCSE' \text{ AND } Age < 21}$  Students

| Results | ID   | Name  | Age | School |
|---------|------|-------|-----|--------|
|         | 1234 | Alice | 20  | SCSE   |

- Query: "Find the students who are either in SCSE or under 21"

$\sigma_{School = 'SCSE' \text{ OR } Age < 21}$  Students

- Query: "Find the persons who are either students or volunteers"

$Students \cup Volunteer$

Note 1: Duplicate tuples are automatically removed

| Students | Volunteer | Results |     |
|----------|-----------|---------|-----|
| Name     | Age       | Name    | Age |
| Alice    | 20        | Alice   | 20  |
| Bob      | 21        | Bob     | 21  |
| Cathy    | 22        | Cathy   | 22  |
| David    | 21        | David   | 21  |
|          |           | Cathy   | 22  |
|          |           | David   | 21  |
|          |           | Eddie   | 43  |
|          |           | Fred    | 35  |

- Query: "Find the persons who are either students or volunteers"

$Students \cup Volunteer$

Note 1: Duplicate tuples are automatically removed

| Students | Volunteer | Results |     |
|----------|-----------|---------|-----|
| Name     | Age       | Name    | Age |
| Alice    | 20        | Alice   | 20  |
| Bob      | 21        | Bob     | 21  |
| Cathy    | 22        | Cathy   | 22  |
| David    | 21        | David   | 21  |
|          |           | Eddie   | 43  |
|          |           | Fred    | 35  |

- Query: "Find the names of the persons who are either students or volunteers"

$\Pi_{Name} (Students \cup Volunteer)$

$(\Pi_{Name} Students) \cup (\Pi_{Name} Volunteer)$

| Students | Volunteer | Results |  |
|----------|-----------|---------|--|
| Name     |           | Name    |  |
| Alice    |           | Alice   |  |
| Bob      |           | Bob     |  |
| Cathy    |           | Cathy   |  |
| David    |           | David   |  |
|          |           | Eddie   |  |
|          |           | Fred    |  |

- Query: "Find the persons who are either students or volunteers"

$Students \cup Volunteer$

Wrong

Note 2: The two sides of a union must have the same schema (i.e., the same set of attributes)

Correct solution:  $(\Pi_{Name} Students) \cup Volunteer$

## Projection $\Pi$ (column-wise)

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

- Query: "Find the IDs and Names of all students"

$\Pi_{ID, Name} Students$

| Results | ID   | Name  |
|---------|------|-------|
|         | 1234 | Alice |
|         | 5678 | Bob   |
|         | 3742 | Cathy |
|         | 9413 | David |

## Combining Operators

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

- Query: "Find the IDs and Names of all students in SCSE"

$\Pi_{ID, Name} (\sigma_{School = 'SCSE'} Students)$

| Results | ID   | Name  |
|---------|------|-------|
|         | 1234 | Alice |
|         | 3742 | Cathy |

- Query: "Find the IDs and Names of all students in SCSE"

How about  $\sigma_{School = 'SCSE'} (\Pi_{ID, Name} Students)$ ?

Wrong

The projection goes before the selection here

Since the projection eliminates "School", the selection cannot be performed

## Intersection $\cap$

### Students

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |

### Volunteer

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

| Name  | Age |
|-------|-----|
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

- Query: "Find the persons who are both students and volunteers"

$(\Pi_{Name} Students) \cap Volunteer$

- Note 1: Duplicate tuples are automatically removed

## Intersection $\cap$

### Students

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

### Volunteer

| Name  | Age |
|-------|-----|
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

### Results

| Name  |
|-------|
| Cathy |
| David |

- Query: "Find the persons who are both students and volunteers"

$(\Pi_{Name} Students) \cap Volunteer$

- Note 2: The two sides of an intersection must have the same schema (i.e., the same set of attributes)

## Difference - *but not / only*

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

| Volunteer | ID   | Name  | Age |
|-----------|------|-------|-----|
|           | 1234 | Alice | 20  |
|           | 5678 | Bob   | 21  |
|           | 3742 | Cathy | 22  |
|           | 9413 | David | 21  |
|           |      | Eddie | 43  |
|           |      | Fred  | 35  |

| Results | Name  | Age |
|---------|-------|-----|
|         | Alice | 20  |
|         | Bob   | 21  |

- Query: "Find the persons who are volunteers but not students"

$Volunteer - Students$

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

| Volunteer | ID   | Name  | Age |
|-----------|------|-------|-----|
|           | 1234 | Alice | 20  |
|           | 5678 | Bob   | 21  |
|           | 3742 | Cathy | 22  |
|           | 9413 | David | 21  |
|           |      | Eddie | 43  |
|           |      | Fred  | 35  |

| Results | Name  | Age |
|---------|-------|-----|
|         | Cathy | 22  |
|         | David | 21  |

- Query: "Find the persons who are students but not volunteers"

$Students - Volunteer$

- Note 2: The two sides of a difference must have the same schema (i.e., the same set of attributes)

## Natural Join

*Exam*

### Students

| NRIC | Name  |
|------|-------|
| 11   | Alice |
| 2    | Bob   |
| 33   | Cathy |
| 4    | David |

### Phones

| NRIC | Number  |
|------|---------|
| 11   | 9123234 |
| 11   | 8635168 |
| 33   | 8213654 |
| 5    | 9653154 |

### Results

| NRIC | Name  | Number  |
|------|-------|---------|
| 11   | Alice | 9123234 |
| 11   | Alice | 8635168 |
| 33   | Alice | 8213654 |
| 33   | Cathy | 8213654 |
| 5    | David | 9653154 |

### Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

### Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

### Results

| Name  | School | Amount |
|-------|--------|--------|
| Cathy | CEE    | 100    |
| David | SCSE   | 200    |
| Eddie | EEE    | 300    |
| Fred  | SCSE   | 400    |

- Query: "Find the NRIC, Name, and Phone of each student, omitting those without a phone" (one without phone will not appear in table)

### Students × Phones

- Note 1: The join is performed based on the common attributes of the two relations

- Note 2: Each common attribute appears only once in the result

### Students × Donations

- Meaning: "For those students who have made donation, find their names, schools, and amounts of their donations"

### ( $\sigma_{School = 'SCSE'}$ Students) × Donations

- Meaning: "For those SCSE students who have made a donation, find their names, schools, and amounts of their donations"

## Theta Join $\bowtie$ condition

| Students |        | Donations |        | Results |             |
|----------|--------|-----------|--------|---------|-------------|
| Name     | School | Name      | Amount | SName   | Name School |
| Alice    | SCSE   | Cathy     | 100    | Cathy   | Cathy CEE   |
| Bob      | EEE    | David     | 200    | David   | David SCSE  |
| Cathy    | CEE    | Eddie     | 300    |         |             |
| David    | SCSE   | Fred      | 400    |         |             |

- Query: "For those students who have made donations, find their names, schools, and amounts of their donations"
- Students  $\bowtie_{Sname = Name}$  Donations
- Difference from natural join: Duplicate attributes will NOT be removed from the results
- In general, the join condition in a theta join can also be omitted

### Exercise

| Grades |        |       |
|--------|--------|-------|
| Name   | Course | Grade |
| Alice  | DB     | A     |
| Bob    | DB     | B     |
| Bob    | AI     | B     |
| Cathy  | CG     | A     |
| David  | NN     | C     |

- Query: "Find the students who have taken DB and DM, but not AI or CG"
- $(\sigma_{Course = 'DB'} Grades) \cap (\sigma_{Course = 'DM'} Grades) - (\sigma_{Course = 'AI'} Grades) \cup (\sigma_{Course = 'CG'} Grades)$
- Result is empty set
- Wrong

### Exercise

| Grades |        |       |
|--------|--------|-------|
| Name   | Course | Grade |
| Alice  | DB     | A     |
| Bob    | DB     | B     |
| Bob    | AI     | B     |
| Cathy  | CG     | A     |
| David  | NN     | C     |

- Query: "Find the students who have never taken DM"
- $\sigma_{Course \neq 'DM'} Grades$

Alice has taken DM but still appear in the result

- Wrong

### Exercise

| Grades |        |       |
|--------|--------|-------|
| Name   | Course | Grade |
| Alice  | DB     | A     |
| Alice  | DM     | C     |
| Bob    | DB     | B     |
| Bob    | AI     | B     |
| Cathy  | CG     | A     |
| David  | NN     | C     |

- Query: "Find the students who have never taken DM"
- Grades –  $(\sigma_{Course = 'DM'} Grades)$

Alice has taken DM but still appear in the result

- Wrong

### Exercise

| Grades |        |       | CrsSch |        |
|--------|--------|-------|--------|--------|
| Name   | Course | Grade | Course | School |
| Alice  | DB     | A     | DB     | SCSE   |
| Alice  | DM     | C     | DM     | SCSE   |
| Bob    | DB     | B     | AI     | SCSE   |
| Bob    | NN     | B     | NN     | EEE    |
| Cathy  | SP     | B     | SP     | EEE    |
| Cathy  | NN     | A     | NN     | EEE    |

- Query: "Find the students who have taken SCSE courses but not EEE courses"

$$(\Pi_{Name} Grades \bowtie (\sigma_{School = 'SCSE'} CrsSch)) - (\Pi_{Name} Grades \bowtie (\sigma_{School = 'EEE'} CrsSch))$$

### Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

### Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

### Results

| Name  | School | Amount |
|-------|--------|--------|
| Cathy | CEE    | 100    |
| David | SCSE   | 200    |
| Eddie | EEE    | 300    |
| Fred  | SCSE   | 400    |

### Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

### Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

### Results

| Name  | School | Amount |
|-------|--------|--------|
| David | SCSE   | 200    |
| Eddie | EEE    | 300    |
| Fred  | SCSE   | 400    |

### Students × Donations

- Meaning: "For those students who have made donation, find their names, schools, and amounts of their donations"

### ( $\sigma_{School = 'SCSE'}$ Students) × Donations

- Meaning: "For those SCSE students who have made a donation, find their names, schools, and amounts of their donations"

## Cartesian Product $\times$

### Students

| Name  | Age |
|-------|-----|
| Alice | 19  |
| Bob   | 22  |
| Cathy | 22  |
| David | 22  |

### Courses

| ID | Name |
|----|------|
| C1 | DB   |
| C2 | Algo |
| C3 | DB   |
| C4 | Algo |

### Results

| Name  | Age | ID | Name |
|-------|-----|----|------|
| Alice | 19  | C1 | DB   |
| Alice | 19  | C2 | Algo |
| Bob   | 22  | C1 | DB   |
| Bob   | 22  | C2 | Algo |
| Cathy | 22  | C1 | DB   |
| Cathy | 22  | C2 | Algo |
| David | 22  | C1 | DB   |
| David | 22  | C2 | Algo |

### Effect: Theta join without a condition

- Effect: Theta join without a condition
- Query: "Create a table that provides all possible student-course combinations"
- Students × Donations

### Exercise

| Grades |        |       |
|--------|--------|-------|
| Name   | Course | Grade |
| Alice  | DB     | A     |
| Alice  | DM     | C     |
| Bob    | DB     | B     |
| Bob    | AI     | B     |
| Cathy  | CG     | A     |
| David  | NN     | C     |

- Query: "Find the students who have never taken DM"

$$(\Pi_{Name} Grades) - (\Pi_{Name} \sigma_{Course = 'DM'} Grades)$$

### Exercise

| Grades |        |       |
|--------|--------|-------|
| Name   | Course | Grade |
| Alice  | DB     | A     |
| Alice  | DM     | C     |
| Bob    | DB     | B     |
| Bob    | AI     | B     |
| Cathy  | CG     | A     |
| David  | NN     | C     |

- Query: "Find the students who have never taken DM"

$$(\Pi_{Name} Grades) - (\Pi_{Name} \sigma_{Course = 'DM'} Grades)$$

### Exercise

| Grades |        |       |
|--------|--------|-------|
| Name   | Course | Grade |
| Alice  | DB     | A     |
| Alice  | DM     | C     |
| Bob    | DB     | B     |
| Bob    | AI     | B     |
| Cathy  | CG     | A     |
| David  | NN     | C     |

- Query: "Find the students who have only taken EEE courses"

$$\begin{aligned} & \text{Query: "Find the students who have only taken EEE courses"} \\ & \Pi_{Name} Grades - (\Pi_{Name} \sigma_{Course \bowtie 'EEE' CrsSch}) \end{aligned}$$

$$\begin{aligned} & \text{Query: "Find the students who have only taken EEE courses"} \\ & \Pi_{Name} Grades - (\Pi_{Name} \sigma_{Course \bowtie 'EEE' CrsSch}) \end{aligned}$$

$$\begin{aligned} & \text{Query: "Find the students who have only taken EEE courses"} \\ & \Pi_{Name} Grades - (\Pi_{Name} \sigma_{Course \bowtie 'EEE' CrsSch}) \end{aligned}$$

$$\begin{aligned} & \text{Query: "Find the students who have only taken EEE courses"} \\ & \Pi_{Name} Grades - (\Pi_{Name} \sigma_{Course \bowtie 'EEE' CrsSch}) \end{aligned}$$

$$\begin{aligned} & \text{Query: "Find the students who have only taken EEE courses"} \\ & \Pi_{Name} Grades - (\Pi_{Name} \sigma_{Course \bowtie 'EEE' CrsSch}) \end{aligned}$$

# This Lecture

- Assignment:  $T_1 := \sigma_{A > 100} R_1$
- Rename:  $\rho_{\text{test}(A, B, C)} R_1$
- Duplicate Elimination  $\delta$
- Extended Projection  $\Pi$
- Grouping and Aggregation  $\gamma$

## Duplicate Elimination $\delta$

| Purchase |         |            |
|----------|---------|------------|
| Name     | Product | Date       |
| Alice    | iPhone  | 2017.01.01 |
| Bob      | Xbox    | 2017.01.01 |
| Cathy    | iPhone  | 2017.01.01 |
| David    | Xbox    | 2017.02.17 |

| R1    |         |            |
|-------|---------|------------|
| Name  | Product | Date       |
| Alice | iPhone  | 2017.01.01 |
| Bob   | Xbox    | 2017.01.01 |
| Cathy | iPhone  | 2017.01.01 |
| David | Xbox    | 2017.02.17 |

- Effect: Eliminate duplicate tuples
- Query: Find the list of products sold on 2017.01.01
- $R1 := \Pi_{\text{Product}} (\sigma_{\text{Date} = 2017.01.01} \text{Purchase})$
- $R2 := \delta(R1)$

## Assignment :=

| Quiz1 |       | Evaluation1 |       |
|-------|-------|-------------|-------|
| Name  | Score | Name        | Score |
| Alice | 70    | Alice       | 70    |
| Bob   | 90    | Bob         | 90    |
| Cathy | 80    | Cathy       | 80    |
| David | 100   | David       | 100   |

## Evaluation1

| Evaluation1 |       |
|-------------|-------|
| Name        | Score |
| Alice       | 70    |
| Bob         | 90    |
| Cathy       | 80    |
| David       | 100   |

## Over85

| Over85 |       |
|--------|-------|
| Name   | Score |
| Alice  | 70    |
| Bob    | 90    |
| Cathy  | 80    |
| David  | 100   |

- Conceptually: Make another copy of the table and give it a new name
- Example
  - $\text{Evaluation1} := \text{Quiz1}$
  - $\text{Over85} := \rho_{\text{Score} > 85} \text{ Quiz1}$
- Note: All attribute names are retained

## Assignment :=

- Useful to break down steps
- Example:
  - $(\Pi_{\text{Name}} \text{ Students}) \cup (\Pi_{\text{Name}} \text{ Volunteer})$
- Equivalent Representation
  - $R1 := \Pi_{\text{Name}} \text{ Students}$
  - $R2 := \Pi_{\text{Name}} \text{ Volunteer}$
  - $R1 \cup R2$
- This makes your solution easier to write and easier for others to understand

## Rename p

### Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

### Evaluation1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

- Similar to assignment, but allows change of attribute names

### Example

- $\rho_{\text{Evaluation1}} \text{ Quiz1}$
- $\rho_{\text{Eval1}}(\$Name, QScore) \text{ Quiz1}$

## Exercise

### Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

### R2

| Name1 | Score1 | Name2 | Score2 |
|-------|--------|-------|--------|
| Bob   | 90     | Cathy | 80     |
| David | 100    | Cathy | 80     |

### R1

| Name  | Score |
|-------|-------|
| Cathy | 80    |

### R3

| Name  | Score |
|-------|-------|
| Bob   | 90    |
| David | 100   |

### R4

| Name |
|------|
| Bob  |

- Find the students who score higher than Cathy in Quiz1

$$R1 := \sigma_{\text{Name} = \text{Cathy}} \text{ Quiz1}$$

$$R2 := \rho_{\text{Name1} < \text{Score1}, \text{Name2} < \text{Score2}} \text{ Quiz1} \bowtie \text{Quiz1.Score} > \text{R1.Score} \text{ R1}$$

$$R3 := \Pi_{\text{Name1}, \text{Score1}} \text{ R2}$$

$$R4 := \Pi_{\text{Name1}} \text{ Quiz1} - R3$$

## Exercise

### Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

### R2

| Name1 | Score1 | Name2 | Score2 |
|-------|--------|-------|--------|
| Alice | 70     | Bob   | 90     |
| Alice | 70     | Cathy | 80     |

### R1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |

### R3

| Name  | Score |
|-------|-------|
| Bob   | 90    |
| David | 100   |

### R4

| Name |
|------|
| Bob  |

- Find the students who score the highest in Quiz1

$$R1 := \text{Quiz1}$$

$$R2 := \text{Quiz1} \bowtie \text{Quiz1.Name} < \text{Quiz1.Score} \text{ Quiz1}$$

$$R3 := \Pi_{\text{Quiz1.Name}} \text{ R2}$$

$$R4 := \Pi_{\text{Quiz1.Name}} \text{ Quiz1} - R3$$

## Exercise

### Quiz1

| Name  | Score |
|-------|-------|
| Alice | 80    |
| Bob   | 90    |
| Cathy | 90    |
| David | 70    |

- Query: "Find the students whose scores in Quizzes 1, 2, and 3 keep increasing"

$$(Quiz1 \bowtie \text{Quiz1.Name} = \text{Quiz2.Name} \text{ AND } \text{Quiz1.Score} < \text{Quiz2.Score}) \text{ Quiz2}$$

$$\bowtie \text{Quiz2.Name} = \text{Quiz3.Name} \text{ AND } \text{Quiz2.Score} < \text{Quiz3.Score} \text{ Quiz3}$$

## Extended Projection $\Pi$

### Scores

| Name  | Quiz1 | Quiz2 |
|-------|-------|-------|
| Alice | 70    | 90    |
| Bob   | 90    | 80    |
| Cathy | 80    | 100   |
| David | 100   | 90    |

### Results

| Name  | Total |
|-------|-------|
| Alice | 160   |
| Bob   | 170   |
| Cathy | 180   |
| David | 190   |

- Similar to ordinary projection, but allows the creation of new attributes via arithmetic
- Query: "For each student, find his/her total score in Quiz 1 and 2"
- $\Pi_{\text{Name}, \text{Quiz1} + \text{Quiz2} \rightarrow \text{Total}}$  Scores
- The left hand side of " $\rightarrow$ " gives the arithmetic performed
- The right hand side gives an attribute name to the result

## Grouping and Aggregation $\gamma$

### Quiz1

| Name  | School | Score |
|-------|--------|-------|
| Alice | SCSE   | 90    |
| Bob   | EEE    | 80    |
| Cathy | EEE    | 100   |
| David | SCSE   | 90    |

### Results

| MaxScore |
|----------|
| 100      |

- Query: "Find the highest score in Quiz1"
- $\gamma_{\text{MAX}(\text{Score})} \rightarrow \text{MaxScore}$  Quiz1
- The attribute name on right hand side of " $\rightarrow$ " can be arbitrary

### Quiz1

| Name  | School | Score |
|-------|--------|-------|
| Alice | SCSE   | 90    |
| Bob   | EEE    | 80    |
| Cathy | EEE    | 100   |
| David | SCSE   | 90    |

### Results

| AvgScore |
|----------|
| 90       |

- Query: "Find the average score in Quiz1"
- $\gamma_{\text{AVG}(\text{Score})} \rightarrow \text{AvgScore}$  Quiz1

### Quiz1

| Name  | School | Score |
|-------|--------|-------|
| Alice | SCSE   | 90    |
| Bob   | EEE    | 80    |
| Cathy | EEE    | 100   |
| David | SCSE   | 90    |

### Results

| SumScore |
|----------|
|----------|

360

- Query: "Find the sum of scores in Quiz1"
- $\gamma_{\text{SUM}(\text{Score})} \rightarrow \text{SumScore}$  Quiz1

### Quiz1

| Name  | School | Score |
|-------|--------|-------|
| Alice | SCSE   | 90    |
| Bob   | EEE    | 80    |
| Cathy | EEE    | 100   |
| David | SCSE   | 90    |

### Results

| NumStu |
|--------|
|--------|

4

- Query: "Find the number of students in Quiz1"
- $\gamma_{\text{COUNT}(\text{Name})} \rightarrow \text{NumStu}$  Quiz1
- $\gamma_{\text{COUNT}(\text{School})} \rightarrow \text{NumStu}$  Quiz1
- $\gamma_{\text{COUNT}(\text{Score})} \rightarrow \text{NumStu}$  Quiz1
- All three queries above give the number of tuples Quiz1

## Quiz1

| Name  | School | GPA |
|-------|--------|-----|
| Alice | SCSE   | 4   |
| Bob   | EEE    | 3   |
| Cathy | EEE    | 3.4 |
| David | SCSE   | 3.6 |

## Results

### School

SCSE    3.8

### School

EEE    3.2

## Aggregate Functions

- $\sigma_{Score}$  = MAX(Score)
- $\sigma_{Score}$  = MIN(Score)
- $\sigma_{Score}$  = AVG(Score)
- $\sigma_{Score}$  = SUM(Score)
- $\sigma_{Score}$  = COUNT(Score)

## Exercise

### R3

Student    Score    School    MaxScore

Alice    90    EEE    90

David    200    SCSE    100

## Quiz1

### Name

Alice    70

Bob    90

Cathy    80

David    100

## Students

### Name

Alice    SCSE

Bob    EEE

Cathy    EEE

David    SCSE

- Query: "Find the average GPA in each school"

$\gamma_{School, Avg(GPA)} \rightarrow AvgGPA$  Quiz1

- Effect: Divide tuples into separate groups based on their "School" value, and then compute the average GPA in each group

## Quiz1

| Name  | School | GPA |
|-------|--------|-----|
| Alice | SCSE   | 4   |
| Bob   | EEE    | 3   |
| Cathy | EEE    | 3.4 |
| David | SCSE   | 3.6 |

## Results

### School

SCSE    3.8

### School

EEE    3.2

### School

EEE    3.4

- Query: "Find the average GPA and highest GPA in each school"

$\gamma_{School, Avg(GPA), MAX(GPA)} \rightarrow AvgGPA, MaxGPA$  Quiz1

## Quiz1

| Name  | School | Year | GPA |
|-------|--------|------|-----|
| Alice | SCSE   | 3    | 4   |
| Bob   | EEE    | 1    | 3   |
| Cathy | EEE    | 2    | 3.4 |
| David | SCSE   | 3    | 3.6 |

## Results

### School

SCSE    3    3.8

### School

EEE    1    3

### School

EEE    2    3.4

$\gamma_{School, Year, Avg(GPA)} \rightarrow AvgGPA$  Quiz1

- Effect: Divide tuples into separate groups based on their "School, year" value combination, and then compute the average GPA in each group

## Example

## Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

## R1

### Score

100

## R2

### Score

MaxScore

David    100    100

## R3

### Name

David

- Query: "Find the student that scores the highest in Quiz1"

$\sigma_{Score} = MAX(Score)$  Quiz1 ?

■ Wrong: Aggregate functions can only be used with the aggregation operation  $\gamma$

■ R1 :=  $\gamma_{MAX(Score)} \rightarrow MaxScore$  (Quiz1)

■ R2 := Quiz1  $\bowtie_{Score = MaxScore}$  R1

■ R3 :=  $\Pi_{Name}$  (R2)

- Query: "Find the student that scores the second highest in Quiz1"

■ R1 :=  $\gamma_{MAX(Score)} \rightarrow MaxScore$  (Quiz1)

■ R2 := Quiz1  $\bowtie_{Score = MaxScore}$  R1

■ R3 :=  $\Pi_{Name, Score}$  (R2)

■ R4 := Quiz1 - R3

■ R5 :=  $\gamma_{MAX(Score)} \rightarrow 2ndMaxScore$  (R4)

■ R6 := R4  $\bowtie_{Score = 2ndMaxScore}$  R5

## Exercise

### R3

Student    Score    School    MaxScore

Alice    90    EEE    90

David    200    SCSE    100

## Quiz1

### Name

Alice    70

Bob    90

Cathy    80

David    100

## Students

### Name

Alice    SCSE

Bob    EEE

Cathy    EEE

David    SCSE

## Exercise

## Grades

### Name

### Course

### Grade

Alice    DB    A

Alice    DM    C

Bob    DB    B

Bob    NN    B

Cathy    SP    B

Cathy    NN    A

## CrsSch

### Course

### School

DB    SCSE

DM    SCSE

NN    EEE

SP    EEE

- Query: "Find the students who have taken all courses from SCSE"

■ R1 :=  $\sigma_{School = "SCSE"} CrsSch$

■ R2 := Grades  $\bowtie$  R1

■ R3 :=  $\gamma_{Name, COUNT(Course)} \rightarrow CrsCNT$  (R2)

■ R4 :=  $\gamma_{COUNT(Course)} \rightarrow ScseCNT$  (R1)

■ R5 := R3  $\bowtie_{CrsCNT = ScseCNT}$  R4

■ R6 :=  $\Pi_{Name}$  (R5)

- Query: "For each school, find the students who have taken all courses in the school"

■ R1 := Grades  $\bowtie$  CrsSch

■ R2 :=  $\gamma_{Name, School, COUNT(Course)} \rightarrow CrsCNT$  (R1)

■ R3 :=  $\gamma_{School, COUNT(Course)} \rightarrow CrsCNT$  (CrsSch)

■ R4 := R2  $\bowtie_{School = R3.School}$  AND R2.CrsCNT = R3.CrsCNT R3

# This Lecture

- Division:  $\bowtie$
- Left Outerjoin:  $\bowtie_L$  condition
- Right Outerjoin:  $\bowtie_R$  condition
- Full Outerjoin:  $\bowtie$

## Division $\bowtie$

### Owns

| Name  | Product |
|-------|---------|
| Alice | iPad    |
| Alice | iPhone  |
| Bob   | iPhone  |
| Cathy | iPad    |

### AppleP

| Product | Price |
|---------|-------|
| iPhone  | 999   |
| iPad    | 699   |

### Results

| Name  | Product |
|-------|---------|
| Alice | iPhone  |

## Exercise

### Grades

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | NN     | B     |
| Cathy | SP     | B     |
| Cathy | NN     | A     |

### CrsSch

| Course | School |
|--------|--------|
| DB     | SCSE   |
| DM     | SCSE   |
| NN     | EEE    |
| SP     | EEE    |

## Full Outerjoin $\bowtie$ condition

### Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

### Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

### Results

| Name  | School | Amount |
|-------|--------|--------|
| Alice | SCSE   | NULL   |
| Bob   | EEE    | NULL   |
| Cathy | CEE    | 100    |
| David | SCSE   | 200    |
| Eddie | NULL   | 300    |
| Fred  | NULL   | 400    |

- The combination of left and right outerjoins

- Students  $\bowtie$  Donations

### Students

| SName | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

### Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

### Results

| Name  | Name  | School | Amount |
|-------|-------|--------|--------|
| Alice | NULL  | SCSE   | NULL   |
| Bob   | NULL  | EEE    | NULL   |
| Cathy | Cathy | CEE    | 100    |
| David | David | SCSE   | 200    |
| Eddie | NULL  | NULL   | 300    |
| Fred  | NULL  | NULL   | 400    |

## Left Outerjoin $\bowtie_L$ condition

### Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

### Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

### Results

| SName | School | Name | Amount |
|-------|--------|------|--------|
| Alice | NULL   | SCSE | NULL   |
| Bob   | NULL   | EEE  | NULL   |
| Cathy | Cathy  | CEE  | 100    |
| David | David  | SCSE | 200    |

- Query: "For each student, find the amount of his/her donation"

- Students  $\bowtie_L$  Donations

- All tuples in Students are retained in the results

- For each student who has not made a donation, a "NULL" value is given as his/her Amount

### Students

| SName | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

### Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

### Results

| SName | Name  | School | Amount |
|-------|-------|--------|--------|
| Alice | NULL  | SCSE   | NULL   |
| Bob   | NULL  | EEE    | NULL   |
| Cathy | Cathy | CEE    | 100    |
| David | David | SCSE   | 200    |

## Right Outerjoin $\bowtie_R$ condition

### Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

### Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

### Results

| SName | Name  | School | Amount |
|-------|-------|--------|--------|
| Alice | NULL  | SCSE   | NULL   |
| Bob   | NULL  | EEE    | NULL   |
| Cathy | Cathy | CEE    | 100    |
| David | NULL  | SCSE   | 200    |

- Query: "For each donor, find the school he/she is in"

- Students  $\bowtie_R$  Name = Name, Donations

- Similar to theta joins in that all attributes are retained

### Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

### Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

### Results

| SName | Name  | School | Amount |
|-------|-------|--------|--------|
| Alice | NULL  | SCSE   | NULL   |
| Bob   | NULL  | EEE    | NULL   |
| Cathy | Cathy | CEE    | 100    |
| David | NULL  | SCSE   | 200    |

### Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

### Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

### Results

| SName | Name  | School | Amount |
|-------|-------|--------|--------|
| Alice | NULL  | SCSE   | NULL   |
| Bob   | NULL  | EEE    | NULL   |
| Cathy | Cathy | CEE    | 100    |
| David | NULL  | SCSE   | 200    |

|                            |   |   |
|----------------------------|---|---|
|                            |   | agg   |
| Selection                  | $\Pi_{\text{all}} A$  |   |
| Projection                 | $\Pi_{\text{A}, \text{B}}$  |   |
| Union                      | $A \cup B$  |   |
| Intersection               | $A \cap B$  | $\left\{ \begin{array}{l} \text{Max} \\ \text{Min} \\ \text{AVG} \\ \text{Sum} \\ \text{Count} \end{array} \right.$ |
|                            |   | $\text{Count null} - \text{Count}$  |
| Difference                 | $A - B$ (look for differences of $A_1$ )                            |   |
| Natural Join               | $A \bowtie B$   |   |
| Theta Join                 | $A \bowtie_{\theta} B$  |   |
| Cartesian Product          | $A \times B$  |   |
| Rename                     | $P_{B(b_1, b_2)} A$   |   |
| Assign                     | $B := A$ (make copy of the table)                                   |   |
| Duplicate Elimination      | $\delta$  | new<br>↑  |
| Extention projection $\Pi$ | $\Pi_{\text{a}_1, \text{a}_2, \text{a}_3 \rightarrow \text{a}_3} A$ |   |
| grouping n aggregation     | $\Pi_{\text{a}_1, \text{agg}(\text{a}_2)} A$                        |   |
| Division                   | $A \div B$  |   |
| Left Outerjoin             | $A \bowtie_{L \text{ a}_1, \text{b}_1} B$                           |   |
| Right Outerjoin            | $A \bowtie_{R \text{ a}_1, \text{b}_1} B$                           |   |
| Full outerjoin             | $A \bowtie_{a_1:b_1} B$   |   |

## Part 2

**Querying Relational Databases using SQL**

**Relational Database Management System**

**What is SQL?**

- Structured Query Language (SQL) – standard query language for relational databases. Procedural, SQL is “declarative”
- A brief history
  - IBM DB2 (1970)
  - Oracle (1974)
  - PostgreSQL (1983)
  - Microsoft SQL Server (1988)
  - MySQL (1995)
  - Oracle (1995)
  - PostgreSQL (1996)
  - Microsoft Access (1996)
  - MySQL (1997)
  - Oracle (1997)
  - Microsoft SQL Server 2000 (1999)
  - PostgreSQL (2000)
  - Microsoft SQL Server 2005 (2005)
  - MySQL (2005)
  - Microsoft SQL Server 2008 (2008)
  - PostgreSQL (2008)
  - Microsoft SQL Server 2012 (2012)
  - PostgreSQL (2012)
  - Microsoft SQL Server 2014 (2014)
  - PostgreSQL (2014)
  - Microsoft SQL Server 2016 (2016)
  - PostgreSQL (2016)

**Present Days: Big Data**

**What SQL we shall study?**

- All major databases (Oracle, MySQL, Microsoft, Sybase, Informix, etc.) support SQL.
- Although database companies have added “proprietary” extensions, the core SQL standard is the same.
- Commercial software packages are not part of the standard – incompatible system issues (e.g., Ingres vs SQL 1999).
- We can concentrate more on the principles (concepts). We will study SQL 87 – a basic subset.

**Good Practice for learning SQL**

- Install a DBMS in your machine, such as MySQL, PostgreSQL, etc.
- Set up a database and tables with example data.
- Run SQL, debug yourself.
- Get help, search in school lab.
- Database environment (no overflow).
- Consult textbooks.
- Other sources:
  - Are today’s sites available?
  - Google “SQL tutorial”
  - Get it run SQL using the simple database there.
  - Description of different SQL dialects – try Oracle’s SQL Reference Guide (http://www.oracle.com/technetwork/learnsql/index.html).

**What we want to do with SQL? \***

- Manage and query the database to set of relations / tables
- Retrieve
  - Select
  - Insert
  - Update
  - Delete

**More about SQL**

**Declarative Languages**

- SQL is a declarative language (non-procedural)
- It specifies what to extract from the database, not how to do it.
- C, C++, Java
- SQL is not a complete programming language
- It does not have control or iterative commands

**Stuffs supported by SQL**

**Data Manipulation Language (DML)**

- Creates, updates, deletes data
- Creates, updates, deletes tables
- Searches, connects

**Data Definition Language (DDL)**

- Creates, drops, alters tables
- Creates, drops, alters columns

**Tables in SQL**

A relation or table is a set of tuples (rows) having attributes (columns). It is the schema of the table.

Schema of a relation is defined by its attributes.

Attributes are used to identify rows in a table.

Example: Product

| ProductID | Name       | Price | Category | Manufacture |
|-----------|------------|-------|----------|-------------|
| 1         | Phone X    | 888   | Phone    | Apple       |
| 2         | iPad       | 668   | Tablet   | Apple       |
| 3         | MacBook 10 | 798   | Phone    | Huawei      |
| 4         | EOS 550D   | 1199  | Camera   | Canon       |

**Data Types in SQL**

- String: for string values
- Integer: for integer values
- Real: for real numbers
- Date: for dates
- Boolean: true/false

**Product**

| ProductID | Name       | Price | Category | Manufacture |
|-----------|------------|-------|----------|-------------|
| 1         | Phone X    | 888   | Phone    | Apple       |
| 2         | iPad       | 668   | Tablet   | Apple       |
| 3         | MacBook 10 | 798   | Phone    | Huawei      |
| 4         | EOS 550D   | 1199  | Camera   | Canon       |

**Patterns for Strings**

Product

| ProductID | Name       | Price | Category | Manufacture |
|-----------|------------|-------|----------|-------------|
| 1         | Phone X    | 888   | Phone    | Apple       |
| 2         | iPad       | 668   | Tablet   | Apple       |
| 3         | MacBook 10 | 798   | Phone    | Huawei      |
| 4         | EOS 550D   | 1199  | Camera   | Canon       |

**Principle Form of SQL**

**SQL Syntax**

There is a set of reserved words that cannot be used as identifiers in the database. Never use them as examples: SELECT, FROM, WHERE, etc.

Use single quotes for constants.

- “Not” – !
- “Not =” – !=
- SQL is generally case-insensitive.
- White space is ignored.
- All statements end with a semicolon (;).

**Moving to Relational Algebra**

$$\Pi_{A_1, A_2, \dots, A_n} (R_1 \times R_2 \times \dots \times R_n) = \{ \text{row} | \text{row} \in R_1 \wedge \text{row} \in R_2 \wedge \dots \wedge \text{row} \in R_n \}$$

**Simple SQL Query**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
WHERE Price > 1000**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
DESC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |
| EOS 550D   | 1199  | Camera   | Canon        |

**SELECT DISTINCT Category  
FROM Product  
ORDER BY Price  
ASC**

**Exercise**

**Product**

| PName      | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| Phone X    | 888   | Phone    | Apple        |
| iPad       | 668   | Tablet   | Apple        |
| MacBook 10 | 798   | Phone    | Huawei       |



## NULL in SQL

- In SQL, whenever we want to leave a value blank, we set it as **NULL**.

- The DBMS regards **NULL** as "an unknown value".

- This makes sense but it leads to a lot of complications...

## Issues with NULL (cont.)

- What about joins?

| Phone | PName     | Price | Tablet | PName  | Price |
|-------|-----------|-------|--------|--------|-------|
|       | iPhone 4  | 888   |        |        |       |
|       | iPhone xx | NULL  |        | iPad 2 | 668   |
|       | IdeaPod   | NULL  |        |        |       |

SELECT P.PName, T.PName  
FROM Phone P, Tablet T  
WHERE P.Price > T.Price

SELECT P.PName, T.PName  
FROM Phone P, Tablet T  
WHERE P.Price = T.Price

Join

| Product | PName     | Price | Sold | PName | Shop | Product | PName    | Price |
|---------|-----------|-------|------|-------|------|---------|----------|-------|
|         | iPhone 4  | 888   |      |       |      |         | iPhone 4 | 888   |
|         | iPhone xx | NULL  |      |       |      |         | iPad 2   | 668   |

SELECT P.PName, Price, Shop  
FROM Product AS P, Sold AS S  
WHERE P.PName = S.PName

SELECT P.PName, Price, Shop  
FROM Product AS P **JOIN** Sold AS S  
ON P.PName = S.PName;

How? Include the left tuples even when there is no match

SELECT P.PName, Price, Shop  
FROM Product AS P **LEFT OUTER JOIN** Sold AS S  
ON P.PName = S.PName;

Include the right tuples even when there is no match

SELECT P.PName, Price, Shop  
FROM Product AS P **RIGHT OUTER JOIN** Sold AS S  
ON P.PName = S.PName;

Include both left and right tuples even if there is no match

SELECT P.PName, Price, Shop  
FROM Product AS P **FULL OUTER JOIN** Sold AS S  
ON P.PName = S.PName;

Tuple Insertion via Subqueries

The 'Sold' table is initially empty.

INSERT INTO Sold  
SELECT PName, 'Suntec'  
FROM Product

Assume that a new shop at the ION sells all products sold at the Suntec shop

INSERT INTO Sold  
SELECT PName, ION  
FROM Product

DELETE FROM Sold  
WHERE PName = 'iPad 2'

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668

Sold

PName | Shop  
iPhone 4 | Suntec  
iPod 2 | Suntec

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668

Sold

PName | Shop  
iPhone 4 | Suntec  
iPod 2 | Suntec  
iPhone 4 | ION

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Tuple Deletion

DELETE FROM Sold  
WHERE PName = 'iPad 2'

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668

Sold

PName | Shop  
iPhone 4 | Suntec  
iPod 2 | Suntec

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668

Sold

PName | Shop  
iPhone 4 | Suntec  
iPod 2 | Suntec

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668

Sold

PName | Shop  
iPhone 4 | Suntec  
iPod 2 | Suntec

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Tuple Update

Assume that the price of iPhone 4 should be reduced to 777

UPDATE Product  
SET Price = 777  
WHERE PName = 'iPhone 4'

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668

Sold

PName | Shop  
iPhone 4 | Suntec  
iPod 2 | Suntec

Product

PName | Price  
iPhone 4 | 777  
iPod 2 | 668

Sold

PName | Shop  
iPhone 4 | Suntec  
iPod 2 | Suntec

Product

PName | Price  
iPhone 4 | 777  
iPod 2 | 668  
Milestone | 798

Tuple Update (cont.)

Reduce the price of all Apple products by 10%

UPDATE Product  
SET Price = Price \* 0.9  
WHERE PName IN  
(SELECT Maker.PName FROM Maker  
WHERE Company = 'Apple')

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Sold

PName | Shop  
iPhone 4 | Suntec  
iPod 2 | Suntec

Product

PName | Price  
iPhone 4 | 799  
iPod 2 | 604  
Milestone | 718

Sold

PName | Shop  
iPhone 4 | Suntec  
iPod 2 | Suntec

Product

PName | Price  
iPhone 4 | 799  
iPod 2 | 604  
Milestone | 718

Tuple Insertion

Any comparison involving NULL results in FALSE

Use IS NULL to check whether a value is NULL

SELECT \*  
FROM Product  
WHERE Price > 1000

SELECT \*  
FROM Product  
WHERE Price < 1000

SELECT \*  
FROM Product  
WHERE Price = 1000

SELECT \*  
FROM Product  
WHERE Price IS NULL

SELECT \*  
FROM Product  
WHERE Price IS NOT NULL

What about GROUP BY?  
NULL values are taken into account in group function

SELECT COUNT(\*) AS Cnt  
FROM Product  
GROUP BY Price

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

Product

PName | Price  
iPhone 4 | 888  
iPod 2 | 668  
Milestone | 798

| Table Creation  | Product              | Column Types   | PRIMARY KEY  | Product              | UNIQUE   |
|---|----------------------|--|--|----------------------|--|
| <ul style="list-style-type: none"> <li><b>CREATE TABLE</b> Product(<br/>PName VARCHAR(30),<br/>Price INT);</li> <li>In general:<br/><b>CREATE TABLE</b> &lt;table name&gt; (<br/>&lt;column name 1&gt; &lt;column type 1&gt;,<br/>&lt;column name 2&gt; &lt;column type 2&gt;, ...);</li> <li><b>CREATE TABLE</b> Product(<br/>PName VARCHAR(30), Price INT);</li> <li>What if we want to make sure that all products have distinct names?</li> <li>Solution: declare PName as <b>PRIMARY KEY</b> or <b>UNIQUE</b></li> </ul> | PName   Price        | <ul style="list-style-type: none"> <li>INT or INTEGER (synonyms)</li> <li>REAL or FLOAT (synonyms)</li> <li>CHAR(n): fixed-length string of n characters</li> <li>VARCHAR(n): variable-length string of up to n characters</li> <li>DATE: In 'yyyy-mm-dd' format</li> <li>...</li> </ul> | <p><b>PRIMARY KEY</b></p> <ul style="list-style-type: none"> <li><b>CREATE TABLE</b> Product(<br/>PName VARCHAR(30), Price INT);</li> <li>What if we want to make sure that all products have distinct names?</li> </ul> | PName   Price        | <p><b>CREATE TABLE</b> Product(<br/>PName VARCHAR(30), Price INT);</p> <ul style="list-style-type: none"> <li>What if we want to make sure that all products have distinct names?</li> </ul>                   |
|   |                      |  | <p><b>PRIMARY KEY (cont.)</b></p> <ul style="list-style-type: none"> <li><b>CREATE TABLE</b> Product(<br/>PName VARCHAR(30), Price INT,<br/><b>PRIMARY KEY</b> (PName));</li> </ul>                                      | Catalog              | <p><b>UNIQUE (cont.)</b></p> <ul style="list-style-type: none"> <li><b>CREATE TABLE</b> Catalog(<br/>PName VARCHAR(30),<br/>Shop VARCHAR(30),<br/>Price INT,<br/><b>PRIMARY KEY</b> (PName, Shop));</li> </ul> |
|   |                      |  |  | PName   Shop   Price | <p><b>UNIQUE (cont.)</b></p> <ul style="list-style-type: none"> <li>We want to make sure that there is no duplicate PName with the same Shop</li> </ul>  |
|   |                      |  |  |                      | <p><b>CREATE TABLE</b> Catalog(<br/>PName VARCHAR(30),<br/>Shop VARCHAR(30),<br/>Price INT,<br/><b>UNIQUE</b> (PName, Shop));</p>  |
| PRIMARY KEY vs. UNIQUE  | Catalog              | NOT NULL   | NOT NULL (cont.)   | Catalog              |  |
| <ul style="list-style-type: none"> <li><b>Difference 1</b> <ul style="list-style-type: none"> <li>Only ONE set of attributes in a table can be declared as <b>PRIMARY KEY</b></li> <li>But we can declare multiple sets of attributes as <b>UNIQUE</b></li> </ul> </li> <li><b>CREATE TABLE</b> Catalog(PName VARCHAR(30),<br/>Shop VARCHAR(30), Price INT,<br/><b>UNIQUE</b> (PName, Shop), <b>UNIQUE</b> (Shop, Price));</li> </ul>   | PName   Shop   Price | <p><b>NOT NULL</b></p> <ul style="list-style-type: none"> <li>We want to make sure that the price of each product is not NULL</li> </ul>   | <p><b>NOT NULL (cont.)</b></p> <ul style="list-style-type: none"> <li><b>CREATE TABLE</b> Catalog(<br/>PName VARCHAR(30),<br/>Shop VARCHAR(30),<br/>Price <b>INT NOT NULL</b>);</li> </ul>                               | PName   Shop   Price |  |
| <ul style="list-style-type: none"> <li><b>Difference 2</b> <ul style="list-style-type: none"> <li>If set of attributes are declared as <b>PRIMARY KEY</b>, then none of these attributes can be NULL</li> <li>UNIQUE attributes still allow NULLS</li> </ul> </li> <li><b>CREATE TABLE</b> Catalog(<br/>PName VARCHAR(30),<br/>Shop VARCHAR(30),<br/>Price INT,<br/><b>UNIQUE</b> (PName, Shop));</li> </ul>  | PName   Shop   Price | <p><b>NOT NULL (cont.)</b></p> <ul style="list-style-type: none"> <li>We want to make sure that the price as well as PName of each product is not NULL</li> </ul>  | <ul style="list-style-type: none"> <li><b>NOT NULL may prevent partial insertions</b></li> <li><b>INSERT INTO</b> Product(PName)<br/>Values('iPhone 5') <span style="color:red">Error!</span></li> </ul>                 | PName   Shop   Price |  |
| DEFAULT   | Catalog              | Combination  | Table Deletion   | Catalog              |  |
| <ul style="list-style-type: none"> <li>We want to specify that, by default, the shop and price of a product is 'Suntec' and 1, respectively</li> <li><b>CREATE TABLE</b> Catalog(<br/>PName VARCHAR(30) <b>DEFAULT</b> 'Suntec',<br/>Shop VARCHAR(30),<br/>Price INT <b>DEFAULT</b> 1);</li> </ul>  | PName   Shop   Price | <p><b>Combination</b></p> <ul style="list-style-type: none"> <li>We want to specify that, by default, the shop and price of a product is 'Suntec' and 1, respectively. In addition, the shop should not be NULL</li> </ul>   | <p><b>Table Deletion</b></p> <ul style="list-style-type: none"> <li><b>DROP TABLE</b> Catalog</li> </ul>   | PName   Shop   Price |  |
|   |                      | <p><b>CREATE TABLE</b> Catalog(<br/>PName VARCHAR(30) <b>NOT NULL</b><br/><b>DEFAULT</b> 'Suntec',<br/>Shop VARCHAR(30),<br/>Price INT <b>DEFAULT</b> 1);</p>  |  |                      |  |
| Table Modification  | Catalog              |  |  | Catalog              |  |
| <ul style="list-style-type: none"> <li>Adding a new attribute</li> <li><b>ALTER TABLE</b> Catalog<br/><b>ADD</b> Price INT</li> <li>We can also add some declarations</li> <li><b>ALTER TABLE</b> Catalog<br/><b>ADD</b> Price INT <b>NOT NULL</b><br/><b>DEFAULT</b> 1</li> <li>Deleting an attribute</li> <li><b>ALTER TABLE</b> Catalog<br/><b>DROP</b> Price</li> </ul>   | PName   Shop         |  |  | PName   Shop   Price |  |

## Exercise

- Remove (i) any product from Suntec that is also sold at ION  
and (ii) any product from ION that is also sold at Suntec
- DELETE FROM Sold WHERE (Sold.Shop = 'Suntec' AND Sold.PName IN (Select S1.PName FROM Sold AS S1 WHERE S1.Shop = 'ION')) OR (Sold.Shop = 'ION' AND Sold.PName IN (Select S2.PName FROM Sold AS S2 WHERE S2.Shop = 'Suntec'))

Sold

| PName    | Shop   |
|----------|--------|
| iPhone 4 | Suntec |
| iPad 2   | Suntec |
| iPhone 4 | ION    |

## Exercise

| Beer      |                |       |
|-----------|----------------|-------|
| Name      | Maker          | Price |
| Budweiser | Anheuser-Busch | 10    |

| Wine       |                   |       |
|------------|-------------------|-------|
| Name       | Maker             | Price |
| Chardonnay | Vintner's Reserve | 15    |

- Update the price of every beer to the average price of the wine by the same maker
- UPDATE Beer  
SET Beer.Price =  
(SELECT AVG(Wine.Price)  
FROM Wine  
WHERE Wine.Maker = Beer.Maker)
- Delete any beer by a maker that does not produce any wine
- DELETE FROM Beer  
WHERE NOT EXISTS  
(SELECT \*  
FROM Wine  
WHERE Wine.Maker = Beer.Maker)

- For each beer, if there does not exist a wine with the same name, then create a wine with the same name and maker, but twice the price
- INSERT INTO Wine  
SELECT B.Name, B.Maker, B.Price \* 2  
FROM Beer AS B  
WHERE NOT EXISTS  
(SELECT \* FROM Wine  
WHERE Wine.Name = B.Name)

Delete from Sold

where  
Shop IN ('iphone')  
AND  
Pname IN  
(select PName  
from Shop as S1  
where S1.shop = 'ion')

Update beer

Set Price =  
(Select Avg(Price)  
From Wine AS W  
Where  
W.Maker = Beer.Maker  
Group By WineName)

Delete from Beer

where  
NOT EXIST  
(Select \* From  
Wine  
Where  
W.Maker = Beer.Maker)

INSERT INTO WINE

Select  
B.name,  
B.maker,  
(B.Price \* 2) as Price  
From Beer as B  
Where  
Not Exist  
(Select \* From  
WINE AS W1  
Where  
W1.Wine = B.Beer)



**Default**

| Sells |          | Cascade |      |       | Sells    |       |       |    |
|-------|----------|---------|------|-------|----------|-------|-------|----|
| Name  | Brand    | Bar     | Beer | Price | Bar      | Beer  | Price |    |
| Beers | B1 Tiger | Lotus   | B1   | 10    | B1 Tiger | Lotus | B1    | 10 |

- Sells(Beer) is a foreign key referencing Beer(Name)
- DELETE FROM Beers WHERE Name = 'B1' **Reject!**
- UPDATE Beers SET Name = 'B2' Where Name = 'B1' **Reject!**

**attribute-based check**

We want to make sure that any beer does not sell over 100 dollars. How?

```
CREATE TABLE Sells (
    bar CHAR(20),
    beer CHAR(30),
    price REAL
    CHECK (price <= 100),
);
```

- CHECK (<condition>) must be added to the declaration for the attribute
- An attribute-based check is checked only when a value for that attribute is inserted or updated (but not deleted)

**Example**

**Constraint**

In Sells(Bar, Beer, Price) no bar may charge an average of more than \$10.

```
CREATE ASSERTION NOT EXISTS (
    SELECT bar
    FROM Sells
    GROUP BY bar
    HAVING 10.00 < AVG(price)
);
```

**Trigger (AFTER DELETE)**

| Sells |          |       |      |       |
|-------|----------|-------|------|-------|
| Name  | Brand    | Bar   | Beer | Price |
| Beers | B1 Tiger | Lotus | B1   | 10    |

- Every time the Lotus Bar stops selling a beer, the Cheetah Bar would sell it at its last price
- CREATE TRIGGER BeerTrig AFTER DELETE ON Sells REFERENCING OLD ROW AS OldTuple FOR EACH ROW WHEN ((OldTuple.Bar = 'Lotus')) INSERT INTO Sells VALUES ('Cheetah', OldTuple.Beer, OldTuple.Price);

**Exercise (1)**

| Sells |          |       |      |       |
|-------|----------|-------|------|-------|
| Name  | Brand    | Bar   | Beer | Price |
| Beers | B1 Tiger | Lotus | B1   | 10    |

- Every time the Lotus Bar sells a new beer, the Cheetah Bar would sell it with 1 dollar less
- CREATE TRIGGER BeerTrig AFTER INSERT ON Sells REFERENCING NEW ROW AS NewTuple FOR EACH ROW WHEN ((NewTuple.Bar = 'Lotus')) BEGIN DELETE FROM Sells WHERE Bar = 'Cheetah' AND Beer = NewTuple.Beer; INSERT INTO Sells VALUES ('Cheetah', NewTuple.Beer, NewTuple.Price - 1); END

**Exercise (2)**

| Sells |          |       |      |       |
|-------|----------|-------|------|-------|
| Name  | Brand    | Bar   | Beer | Price |
| Beers | B1 Tiger | Lotus | B1   | 10    |

- Every time the Lotus Bar cut the price of a beer by x dollars, the Cheetah Bar would cut the price by  $2^x$  dollars
- CREATE TRIGGER BeerTrig AFTER UPDATE ON Sells REFERENCING OLD ROW AS OldTuple NEW ROW AS NewTuple FOR EACH ROW WHEN ((OldTuple.Price > NewTuple.Price)) BEGIN UPDATE Sells SET Price = Price - 2 \* (OldTuple.Price - NewTuple.Price) WHERE Bar = 'Cheetah' AND Beer = OldTuple.Beer; END

**Exercise (3)**

| Sells |          |       |      |       |
|-------|----------|-------|------|-------|
| Name  | Brand    | Bar   | Beer | Price |
| Beers | B1 Tiger | Lotus | B1   | 10    |

- Every time the Lotus Bar stops selling a beer, if the Cheetah bar sells the beer, it would set the price of the beer to the average price of the beers sold at the Lotus Bar
- CREATE TRIGGER BeerTrig AFTER DELETE ON Sells REFERENCING OLD ROW AS OldTuple FOR EACH ROW WHEN ((OldTuple.Bar = 'Lotus')) BEGIN UPDATE Sells SET Price = (SELECT AVG(Price) FROM Sells WHERE Bar = 'Lotus') AND Brand = 'Tiger' AND name = beer WHERE Bar = 'Cheetah' AND Beer = OldTuple.Beer; END

**Choosing a Policy**

| Sells |          |       |      |       |
|-------|----------|-------|------|-------|
| Name  | Brand    | Bar   | Beer | Price |
| Beers | B1 Tiger | Lotus | B1   | 10    |

- We can specify whether to default, cascade, or set null when we declare a foreign key
- Example:  
CREATE TABLE Sells  
Bar VARCHAR(30),  
Beer VARCHAR(30),  
Price FLOAT,  
FOREIGN KEY (Beer) REFERENCES Beers(Name)  
ON DELETE SET NULL  
ON UPDATE CASCADE
- Default is taken when SET NULL and CASCADE are absent

**Assertions**

| Sells |          |       |      |       |
|-------|----------|-------|------|-------|
| Name  | Brand    | Bar   | Beer | Price |
| Beers | B1 Tiger | Lotus | B1   | 10    |

- A more flexible type of checks
- But not available with most of DBMS
- Example: There should not be more bars than beers
- CREATE ASSERTION MoreBars CHECK (
 (SELECT COUNT(\*) FROM Beers) >=
 (SELECT COUNT(DISTINCT Bar) FROM Sells)
)

**Trigger (AFTER INSERT)**

| Sells |          |       |      |       |
|-------|----------|-------|------|-------|
| Name  | Brand    | Bar   | Beer | Price |
| Beers | B1 Tiger | Lotus | B1   | 10    |

- Every time there is a new beer from Tiger, the Lotus Bar would sell it at 10 dollars.
- How to capture this?  
we can use triggers  
Triggering event  
Action

**Trigger (cont.)**

| Sells |          |       |      |       |
|-------|----------|-------|------|-------|
| Name  | Brand    | Bar   | Beer | Price |
| Beers | B1 Tiger | Lotus | B1   | 10    |

- Every time the same bar raises the price of a beer by over 10 dollars, the Cheetah Bar would sell it at its old price
- CREATE TRIGGER BeerTrig AFTER UPDATE ON Sells REFERENCING OLD ROW AS OldTuple NEW ROW AS NewTuple FOR EACH ROW WHEN ((OldTuple.Price + 10 < NewTuple.Price)) INSERT INTO Sells VALUES ('Cheetah', OldTuple.Beer, OldTuple.Price);
- duplicate beers at the Cheetah Bar. How to avoid?

**Types of actions**

- An SQL query, a DELETE, INSERT, UPDATE, ROLLBACK, SQL/PSM
- There can be more than one SQL statements
- Queries make no sense in an action  
So only DB modification

**Question**

- All customers must have an account balance of at least \$1000 in their account.  
Do you use Check or Trigger?
- All new customers opening an account must have a balance of \$1000  
Do you use Check or Trigger?

## BEFORE - Trigger Execution (Example)

### Employee

```
CREATE TRIGGER Last_Name_Upper
BEFORE INSERT ON Employee
REFERENCING NEW ROW AS N
FOR EACH ROW
    N.Last_Name = Upper(N.Last_Name);
```

- No When Clause: The trigger action is executed.
- Then the triggering event (Insert N into Employee) is executed.

## Trigger - Syntax

```
CREATE TRIGGER triggerName
[BEFORE | AFTER] INSTEAD OF
    [INSERT | DELETE | UPDATE] [OF column-name-is]
ON table-name
    [REFERENCING [OLD ROW AS var-to-refer-to-old-tuple]
        [NEW ROW AS var-to-refer-to-new-tuple]
        [OLD TABLE AS name-to-refer-to-new-table]
        [NEW TABLE AS name-to-refer-to-old-table]
    ] [FOR EACH ROW | STATEMENT]
    [WHEN (condition)]
[BEGIN]
statement-list
[END];
```

Granularity

- WHEN condition is tested before the triggering event
  - If the condition is true then the action of the trigger is executed
  - Next, the triggering event is executed, regardless of whether the condition is true
- # Action:
- update or validate record values (the same record in the triggering event) before they are saved to the database
  - Not allowed to modify the database

## Trigger Referencing a table

```
CREATE TRIGGER flights_delete
AFTER DELETE ON flights
REFERENCING OLD TABLE AS deleted_flights
FOR EACH STATEMENT
BEGIN
    DELETE FROM flight_availability
    WHERE flight_id IN
        (SELECT flight_id
        FROM deleted_flights);
END;
```

- The OLD TABLE maps those rows affected by the triggering event (i.e., only deleted rows of FLIGHT due to the execution of the triggering SQL statement).

NOT the old version of FLIGHT table before deletion

## INSTEAD OF - Trigger Execution (Example)

### Likes

| Drinker | Beer | Bar  | Beer | Price |
|---------|------|------|------|-------|
| John    | A1   | John | B2   | Lotus |

### Sells

| Likes         | Sells             | Frequents |
|---------------|-------------------|-----------|
| Likes:drinker | Likes:beer        | Sells:bar |
| Likes:drinker | Frequents:drinker | AND       |
| Sells:beer    | Frequents:bar     |           |

view Synergy has (drinker, beer, bar) triples: the bar serves the beer and the drinker frequents the bar and likes the beer

Use a INSTEAD OF trigger to turn a (drinker, beer, bar) triple into three insertions of projected parts

## INSTEAD OF - Trigger Execution (Example)

CREATE TRIGGER ViewTrig
INSTEAD OF INSERT ON Synergy
REFERENCING

NEW ROW AS n

FOR EACH ROW

BEGIN

```
    INSERT INTO Likes VALUES(n,drinker,n,beer);
    INSERT INTO Sells VALUES(n,bar,n,beer);
    INSERT INTO Frequents VALUES(n,drinker,n,bar);
```

END;

- Semantic: Trigger defined on view instead of triggering event, we implement action
- Generally it is impossible to modify views because it doesn't exist physically.

## Trigger Execution Granularity (Example)

```
CREATE TRIGGER triggerName
AFTER INSERT ON
REFERENCING NEW ROW AS newRow
FOR EACH ROW
WHEN (newRow.beer)
NOT IN
    (SELECT name FROM Beers)
    (Beers.name)
INSERT INTO
VALUES (newRow.beer);
```

```
CREATE TRIGGER triggerName
AFTER UPDATE OF
FOR EACH STATEMENT
INSERT INTO
VALUES (Current_Date, SELECT AVG(Salary) FROM Employee);
```

## Trigger Execution Granularity

### Row level

- FOR EACH ROW indicates row-level
- Row-level triggers are executed once for each modified (inserted, updated, or deleted) tuple/row

### Statement level

- Executed once for an SQL statement, regardless of the number of tuples modified
  - even if 0 tuple is modified: An UPDATE statement that makes no changes (condition in the WHERE clause does not affect any tuples)
- If neither is specified, default is "Statement level"

## Trigger - Syntax

```
CREATE TRIGGER triggerName
[BEFORE | AFTER] INSTEAD OF
    [INSERT | DELETE | UPDATE] [OF column-name-is]
ON table-name
    [REFERENCING [OLD ROW AS var-to-refer-to-old-tuple]
        [NEW ROW AS var-to-refer-to-new-tuple]
        [OLD TABLE AS name-to-refer-to-new-table]
        [NEW TABLE AS name-to-refer-to-old-table]
    ] [FOR EACH ROW | STATEMENT]
    [WHEN (condition)]
[BEGIN]
statement-list
[END];
```

# Querying Relational Databases using SQL

## Part-5



|  |  |
|--|--|
| average typical row size                         | 11,000,000,000 bytes = 1               |
| total rows in table                              | 8.5 million                            |
| total bytes in table                             | 75 million                             |
| total disk space required                        | 75 million                             |
| total disk space required from memory            | 200,000,000 bytes = 200 MB (0.25 mins) |
| total disk space required from memory (approx)   | 1,000,000 bytes = 1 MB (10 mins)       |
| total disk space required from memory (exact)    | 20,000,000 bytes = 20 KB (20 mins)     |
| total disk space required from memory + overhead | 20,000,000 bytes = 20 KB (20 mins)     |

## Example: Search for books

| Billion_Books |                      |             |           |
|---------------|----------------------|-------------|-----------|
| ID            | Title                | Author      | Published |
| 1001          | War and Peace        | Tolstoy     | 1869      |
| 1002          | Crime and Punishment | Dostoyevsky | 1866      |
| 1003          | Anna Karenina        | Tolstoy     | 1877      |

All books written by Rowling?

```
SELECT * FROM Billion_Books WHERE Author like 'Rowling'
```

## Example: Search for books

| Billion_Books |                      |             |           |
|---------------|----------------------|-------------|-----------|
| ID            | Title                | Author      | Published |
| 1001          | War and Peace        | Tolstoy     | 1869      |
| 1002          | Crime and Punishment | Dostoyevsky | 1866      |
| 1003          | Anna Karenina        | Tolstoy     | 1877      |

Design Choices:

- Data in RAM:
  - Scan RAM sequentially & filter: Scan 1000-10000 - 1.25 million rows = 200 GB (1000-10000 rows) (1 TB) = 3000 GB or 1 TB
  - Seek each record on disk & filter: Scan 1000-10000 - 1.25 million rows = 1.25 TB (1 TB) = 1.25 TB (1.25 mins)
  - Seek data sequentially organized:
    - Seek to index, and sequentially scan records on disk & filter: Scan 1000-10000 - 1.25 million rows = 1.25 TB (1.25 mins)
    - Cost 0.01TB of disk = 1.25 mins

Input Data size: 1 Billion books. Each record = 1000 bytes. Total = 1000,000,000,000 bytes (1.25 TB) = 1.25 TB (1.25 mins)

Indexes in RAM:
 

- Author index: 1000 entries (1000 rows)
- Dept index: 1000 entries (1000 rows)
- Salary index: 1000 entries (1000 rows)

Index Maintenance overhead:
 

- Author index: 1000 rows
- Dept index: 1000 rows
- Salary index: 1000 rows

Notes:
 

- Index needs to fit in memory for fast lookups
- Index needs to be small
- Index needs to be updated frequently

## Example: Search for books

| Billion_Books |                      |             |           |
|---------------|----------------------|-------------|-----------|
| ID            | Title                | Author      | Published |
| 1001          | War and Peace        | Tolstoy     | 1869      |
| 1002          | Crime and Punishment | Dostoyevsky | 1866      |
| 1003          | Anna Karenina        | Tolstoy     | 1877      |

Example Record:

```
Name=Joe DEPT=Sales SAL=15k
```

Index Maintenance overhead:
 

- Author index: 1000 rows
- Dept index: 1000 rows
- Salary index: 1000 rows

Notes:
 

- Index needs to fit in memory for fast lookups
- Index needs to be small
- Index needs to be updated frequently

## Indexes on a table

- An index speeds up selections on search key(s)
  - Any subset of fields
- Example

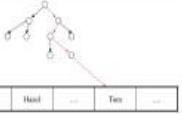
Books(BID, name, author, price, year, text)

On which attributes would you build indexes?

## Creating Indexes in Databases

### Indexes in databases

- Tree-structured (think of binary search tree)
- Hash-based



## Functionality

- Used by query processor to speed up data access

Index on T(A, B)

- T.A = 'cat' and T.B = 2
- T.A < 'd' and T.B < 4
- 3 <= T.B < 5

| T   | A   | B   | C   |
|-----|-----|-----|-----|
| 1   | cat | 2   | ... |
| 2   | dog | 3   | ... |
| 3   | one | 1   | ... |
| 4   | dog | 9   | ... |
| 5   | cat | 2   | ... |
| 6   | cat | 8   | ... |
| 7   | cow | 6   | ... |
| ... | ... | ... | ... |

## Covering Indexes

An index covers for a specific query if the index contains all the needed attributes, meaning the query can be answered using the index alone!

The "needed" attribute is the union of those in the SELECT and WHERE clauses.

Example: SELECT Published, BID FROM Billion\_Books WHERE Published > 1867

## Answering Queries using Indexes

Select sName, cName  
From Student, Apply  
Where Student.sID = Apply.sID

- Scan Student, use an Index on Apply
- Scan Apply, use an Index on Student
- Use Indexes on both Apply and Student

## Storing a Relation

### Recall

- Tuples are unordered
- Focus (in SQL) is on tuples individually

| Relation table 1 |        | Relation table 2 |        | Tuples |      |
|------------------|--------|------------------|--------|--------|------|
| E1               | Salary | E2               | Salary | 1      | 1000 |
| 1                | 1000   | 1                | 1000   | 2      | 1000 |
| 1                | 1500   | 2                | 1000   | 3      | 1500 |
| 2                | 1000   | 2                | 1500   | 4      | 1000 |
| 3                | 1400   | 3                | 1500   | 5      | 1400 |
| 4                | 1500   | 3                | 2000   | 6      | 1500 |
| 5                | 1000   | 4                | 1500   | 7      | 1000 |
| 6                | 1500   | 4                | 2000   | 8      | 1500 |
| 7                | 2000   | 5                | 1500   | 9      | 2000 |
| 8                | 1500   | 5                | 2000   | 10     | 1500 |

## Motivation

### Strategy 1: index on single attribute

- Use one index on Dept: Get all Dept = "Art" records and check their salary
- Use one index on Salary: Get all Salary > 50k records and check their Dept

### Strategy 3 Composite Index:

- Create index DeptSalaryIndex on EMP (Dept, Salary)
- See next slide
- Create index SalaryDeptIndex on EMP (Salary, Dept)

## Build index on attribute list

You can build an index on multiple attributes, also called **Composite Index**

Q Syntax: CREATE INDEX foo ON R(attr1, attr2)

Q Example 1:

- CREATE INDEX PnameIndex ON FacebookUser(firstname, lastname)

Q Why?

Motivation: Find records where

DEPT = "Art" AND SAL > 50k

## Example: Search for books

SELECT \* FROM Billion\_Books WHERE Author like 'Rowling'

| ID   | Title                | Author      | Published |
|------|----------------------|-------------|-----------|
| 1001 | War and Peace        | Tolstoy     | 1869      |
| 1002 | Crime and Punishment | Dostoyevsky | 1866      |
| 1003 | Anna Karenina        | Tolstoy     | 1877      |

Import Data size: 1 Billion books. Each record = 1000 bytes. Total = 1000,000,000,000 bytes (1.25 TB) = 1.25 TB (1.25 mins)

Design Choices:

- Data in RAM:
  - Scan RAM sequentially & filter: Scan 1000-10000 - 1.25 million rows = 200 GB (1000-10000 rows) (1 TB) = 3000 GB or 1 TB
  - Seek each record on disk & filter: Scan 1000-10000 - 1.25 million rows = 1.25 TB (1 TB) = 1.25 TB (1.25 mins)
  - Seek data sequentially organized:
    - Seek to index, and sequentially scan records on disk & filter: Scan 1000-10000 - 1.25 million rows = 1.25 TB (1.25 mins)
    - Cost 0.01TB of disk = 1.25 mins

Index Maintenance overhead:
 

- Author index: 1000 rows
- Dept index: 1000 rows
- Salary index: 1000 rows

Notes:
 

- Index needs to fit in memory for fast lookups
- Index needs to be small
- Index needs to be updated frequently

## Example

By\_Yr\_Index

Published BID

1866 1002

1869 1001

1877 1003

...

Why might just keeping the table sorted by year not be good enough?

Maintain an index for this, and search over that!

By\_Author\_Title\_Index

Author Title

Tolstoy Date and Payment

Tolstoy Anna Karenina 1003

Tolstoy War and Peace 1001

Tolstoy Crime and Punishment 1002

Dostoyevsky Crime and Punishment 1002

Dostoyevsky War and Peace 1001

Dostoyevsky Anna Karenina 1003

Can have multiple indexes to support multiple search keys

## Example

By\_Yr\_Index

Published BID

1866 1002

1869 1001

1877 1003

...

Why might just keeping the table sorted by year not be good enough?

Maintain an index for this, and search over that!

By\_Author\_Title\_Index

Author Title

Tolstoy Date and Payment

Tolstoy Anna Karenina 1003

Tolstoy Crime and Punishment 1002

Tolstoy War and Peace 1001

Dostoyevsky Crime and Punishment 1002

Dostoyevsky War and Peace 1001

Dostoyevsky Anna Karenina 1003

Can have multiple indexes to support multiple search keys

## Functionality

- Used by query processor to speed up data access

| A   | B   | C   |
|-----|-----|-----|
| 1   | cat | 2   |
| 2   | dog | 5   |
| 3   | cow | 1   |
| 4   | cat | 8   |
| 5   | cat | 2   |
| 6   | cat | 8   |
| 7   | cow | 6   |
| ... | ... | ... |

Index on T.A

- T.A = 'cow'

- T.A = 'cat'

| A   | B   | C   |
|-----|-----|-----|
| 1   | cat | 2   |
| 2   | dog | 5   |
| 3   | cow | 1   |
| 4   | cat | 8   |
| 5   | cat | 2   |
| 6   | cat | 8   |
| 7   | cow | 6   |
| ... | ... | ... |

Index on T.B

- T.B = 2

- T.B < 4

- 3 <= T.B < 5

## Functionality

- Search: Quickly find all records which meet some condition on the search key attributes
- (Advanced: across rows, across tables)

- Insert / Remove entries
- Bulk Load / Delete. Why?

Indexing is one of the most important features provided by a database for performance

## Indexing Definition in SQL

### Syntax

CREATE INDEX name ON rel (attr)

CREATE UNIQUE INDEX name ON rel (attr)

Duplicate values are not allowed

DROP INDEX name;

Note: The syntax for creating indexes varies amongst different databases.

### In practice

- PRIMARY KEY declaration: Automatically creates a primary/clustered index

- UNIQUE declaration: Automatically creates a secondary/nonclustered index

## Example

| Art | Sales | Toys |
|-----|-------|------|
| 10k | 15k   | 21k  |
| 15k | 15k   | 19k  |
| 17k | 15k   | 19k  |
| 21k | 15k   | 19k  |

Dept Index

Salary Index

Example Record

Name=Joe

DEPT=Sales

SAL=15k

## Semi-Structured Data

### The More Data, The Merrier

#### Power of Data

- the more data the merrier (GB > TB > PB)
- data comes from everywhere in all shapes
- value of data often discovered later

#### Services turn data into \$SS

- the more services the merrier (10 > 1000 > 3M > 1B)
- need to adapt quickly

#### Goal: Platforms for data and services

- any data, any service, anywhere and anytime

### A Little Bit of History ...

#### Database world

- 1970 relational databases
- 1980 needed relational model and object oriented databases
- 1995 semi-structured databases

Data - documents - information  
1996 XML (Extended Markup Language)  
URI (Universal Resource Identifier)

#### XML Terminology

##### Tag names, etc., ...

##### Start tags, end tags, ...

##### Attributes, ...

##### Elements can have ...

##### Empty elements, ...

##### Car or abbreviations, ...

##### Elements can also have ...

##### Ordering generally matters, except for attributes

#### Documents world

- 1974 SGML (Structured Generalized Markup Language)
- 1990 HTML (HyperText Markup Language)
- 1992 URL (Universal Resource Locator)

Data - documents - information  
1996 XML (Extended Markup Language)

#### URI (Universal Resource Identifier)

#### Bibliography

##### book

##### bookid

##### author

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

##### language

##### title

##### year

##### subject

##### format

</div

**Entity**

- Relationships
  - Customer
  - Order
  - Product
  - Supplier
  - Employee
  - Address
  - Region

Note: An entity can have many "order" values.

**JSON Example**

```

  <!-- External Entity Declaration -->
  <entity id="Customer" entity-name="Customer" />
  <!-- Example -->
  <customer>
    <id>1</id>
    <name>John Doe</name>
    <address>
      <street>123 Main Street</street>
      <city>Anytown, USA</city>
    </address>
    <orders>
      <order id="1" customer-id="1" date="2014-01-01" />
      <order id="2" customer-id="1" date="2014-01-02" />
    </orders>
  </customer>
  
```

**Difficulties with XML**

- "One and a half" - A node can have both attributes and children.
- "No child elements" - The element reference is e.g. `<A>A</A>`. Elements that don't have any children are omitted.
- "No type enforcement" - Many tools do not do type checking, so it's possible to add a "string" value under the `age` element.
- "Complex Rule Enforcement" - Its semantics, documents etc.
- "No inheritance" - It's not possible to reuse code.
- "String" - As a file technology, XML values must be strings.

**Other Semi-Structured Data**

- JSON
  - Relaxed Object notation
  - Lightweight, nested structure (trees)
  - Does not require predefined schema ("self-describing")
  - Text representation: good for exchange, bad for performance
  - Most common uses: Language API

**Entity**

- Relationships
  - Customer
  - Order
  - Product
  - Supplier
  - Employee
  - Address
  - Region

Note: An entity can have many "order" values.

**JSON - Syntax**

```

  {
    "id": 1,
    "name": "John Doe",
    "address": {
      "street": "123 Main Street",
      "city": "Anytown, USA"
    },
    "orders": [
      {"id": 1, "customer_id": 1, "date": "2014-01-01"},  

      {"id": 2, "customer_id": 1, "date": "2014-01-02"}
    ]
  }
  
```

**JSON - Terminology**

- Curly braces
  - Hold objects
  - Each object is a list of name/value pairs separated by commas
  - Each pair is a name followed by a colon, followed by the value
- Square brackets
  - Hold arrays and values are separated by commas

**JSON - Data Structure**

**Summary**

**Data Exchange Format**

- Well suited for exchanging data between applications
- XML, JSON

**Data Models**

- NoSQL - use them as data models
  - SQL Server - relational DBs, relational
  - CouchDB, MongoDB - JSON as data model

**Query Languages**

- Xpath, Xquery
- Cassandra - NTQL
- SQL-like

**XML vs. JSON**

**XML**

- Relational
- Hierarchical
- Extensible
- Standardized
- Human readable
- Machine readable
- Complex
- Large
- Slow
- Large memory footprint
- Large storage footprint
- Large bandwidth usage
- Large processing overhead

**JSON**

- Relational
- Hierarchical
- Extensible
- Standardized
- Human readable
- Machine readable
- Simple
- Small
- Fast
- Small memory footprint
- Small storage footprint
- Small bandwidth usage
- Small processing overhead

**Difficulties with XML**

- "One and a half" - A node can have both attributes and children.
- "No child elements" - The element reference is e.g. `<A>A</A>`. Elements that don't have any children are omitted.
- "No type enforcement" - Many tools do not do type checking, so it's possible to add a "string" value under the `age` element.
- "Complex Rule Enforcement" - Its semantics, documents etc.
- "No inheritance" - It's not possible to reuse code.
- "String" - As a file technology, XML values must be strings.

**Tree View of JSON Data**

**Self-describing**

**Mapping Relational Data to JSON**

**JSON**

```

  {
    "id": 1,
    "name": "John Doe",
    "age": 30,
    "sex": "M"
  }
  
```

**Persons**

| Name | Age | Sex |
|------|-----|-----|
| John | 30  | M   |
| Jane | 32  | F   |

**JSON**

```

  [
    {
      "id": 1,
      "name": "John Doe",
      "age": 30,
      "sex": "M"
    },
    {
      "id": 2,
      "name": "Jane Doe",
      "age": 32,
      "sex": "F"
    }
  ]
  
```

**Persons**

| Name | Age | Sex |
|------|-----|-----|
| John | 30  | M   |
| Jane | 32  | F   |

**JSON**

```

  [
    {
      "id": 1,
      "name": "John Doe",
      "age": 30,
      "sex": "M"
    },
    {
      "id": 2,
      "name": "Jane Doe",
      "age": 32,
      "sex": "F"
    }
  ]
  
```

**Persons**

| Name | Age | Sex |
|------|-----|-----|
| John | 30  | M   |
| Jane | 32  | F   |

# **SC2207/CZ2007 Introduction to Database Systems (Week 1)**

---

**Topic 1: Entity Relationship Diagram (1)**



# This Lecture

- Database and DBMS 
- ER diagram *modeling diagrams/syntax used*
- Types of relationships
- Roles

# Database and DBMS

- What is a database?
  - A collection of data specially **organized** for efficient retrieval by a computer
- What is a database system?
  - A piece of **software** that helps us **efficiently** manage/retrieve information from databases
- More formal name: Database Management System (DBMS)

# DBMS in Practice

- Large web sites rely heavily on DBMS
  - Facebook
  - Twitter
- Many non-web companies, too
  - Banks, hospitals, etc
- Even small pieces of software on your computer
  - Google Chrome

# Relational Model

- Numerous DBMS exist on the market
  - Oracle, SQL Server, MongoDB, ...*DB2, mysql*
- Most of them follow the relational model
- What does it mean?
- Answer: They store all data in the form of relations.

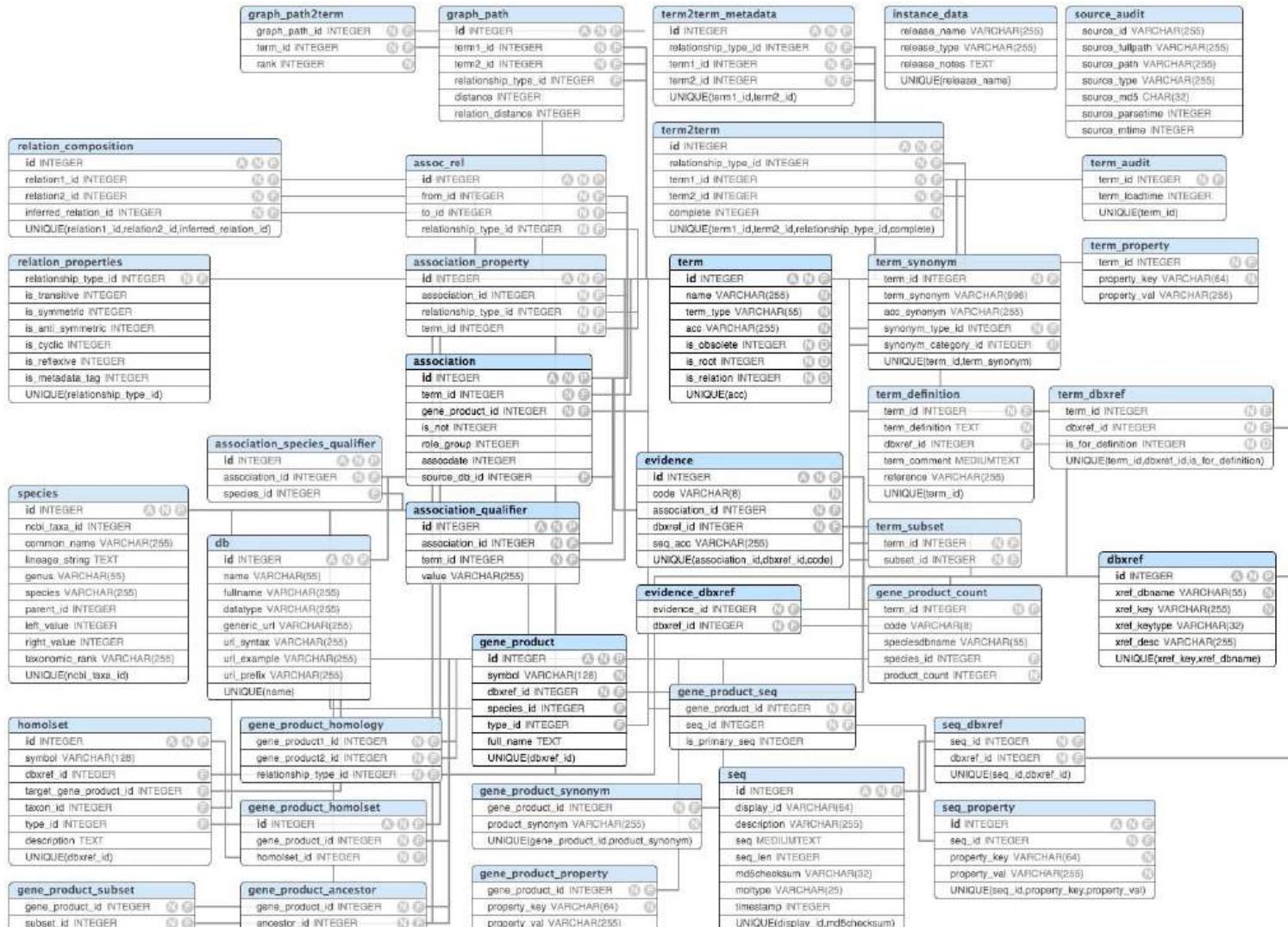
# Relation

name → Product

| Name       | Price | Category | Manufacturer |
|------------|-------|----------|--------------|
| iPhone 6   | 888   | Phone    | Apple        |
| iPad Air 2 | 668   | Tablet   | Apple        |
| Galaxy     | 798   | Phone    | Samsung      |
| EOS-1D X   | 1199  | Camera   | Canon        |

- Some jargons:
  - A relation is often referred to as a **table**
  - A row in a table is also called a **tuple** or a **record**
  - A column in a table is also called an **attribute** of the table

A real database may have a large number of tables ...



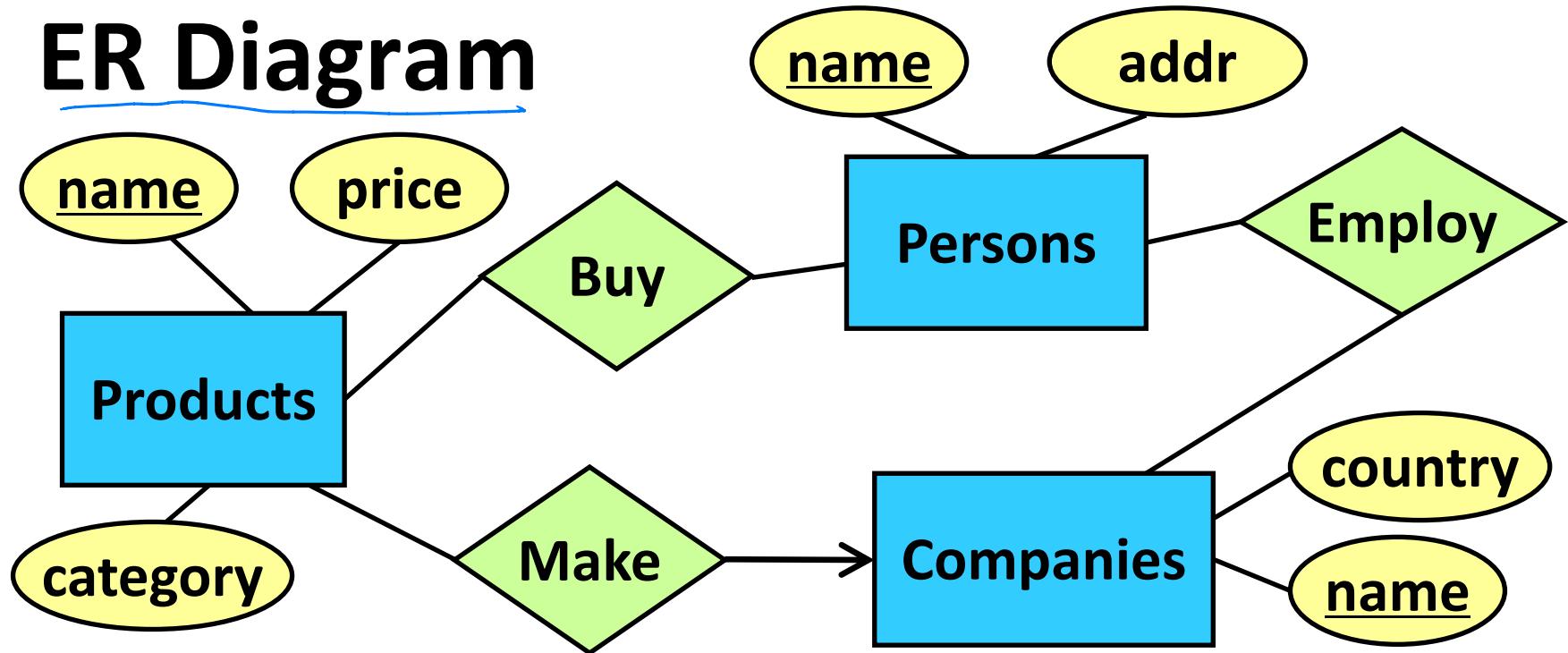
# Designing a Database for an Application

- Conceptually model the **data requirements** of the application
  - What are the things that need to be stored?
  - How do they interact with one another?
- **Tool to use: Entity-Relationship (ER) Diagrams**
  - A pictorial and intuitive way for modelling
- Translate the conceptual model into a set of tables
- Construct the tables with a DBMS

# This Lecture

- Database and DBMS
- ER diagram 
- Types of relationships
- Roles

# ER Diagram



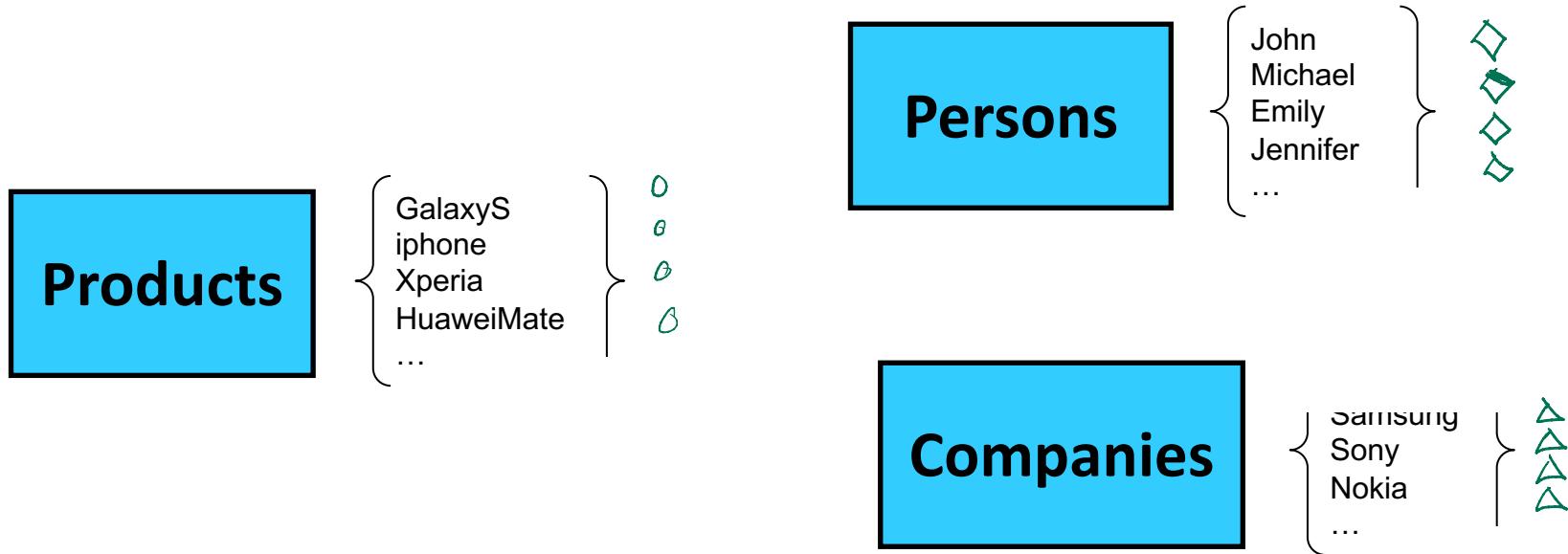
- ER diagram is a collection of visual artifacts
- Each artifact captures some data requirement or relationship

entity

relationship

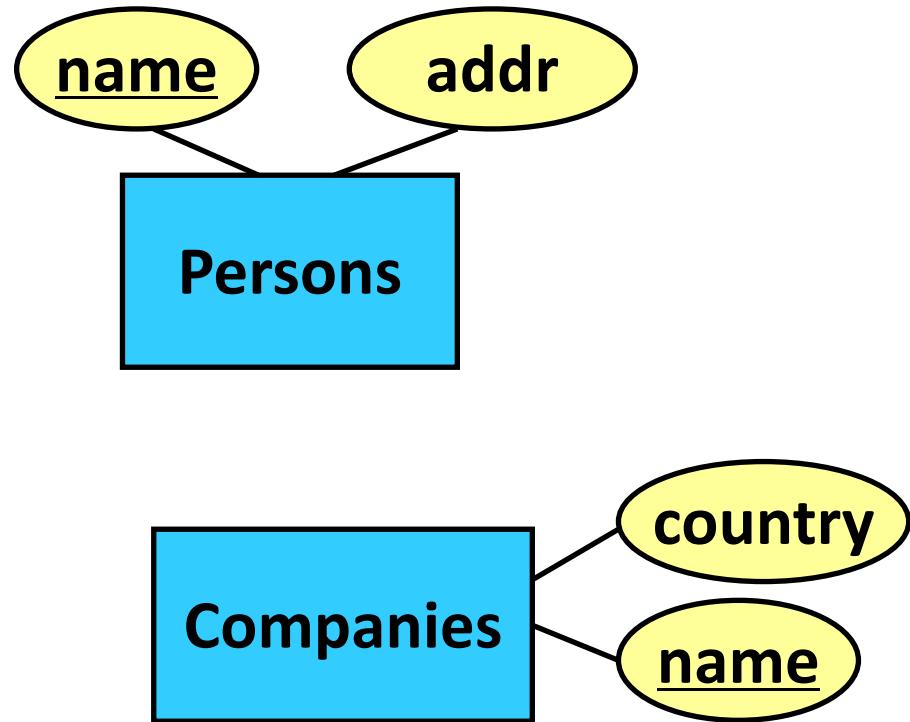
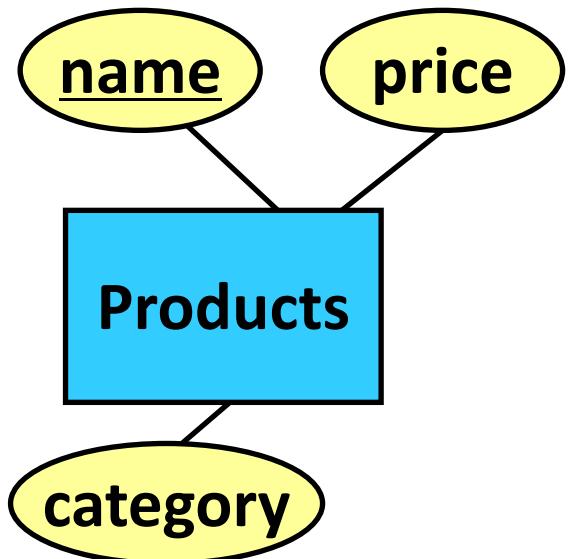
attribute

# ER Diagram



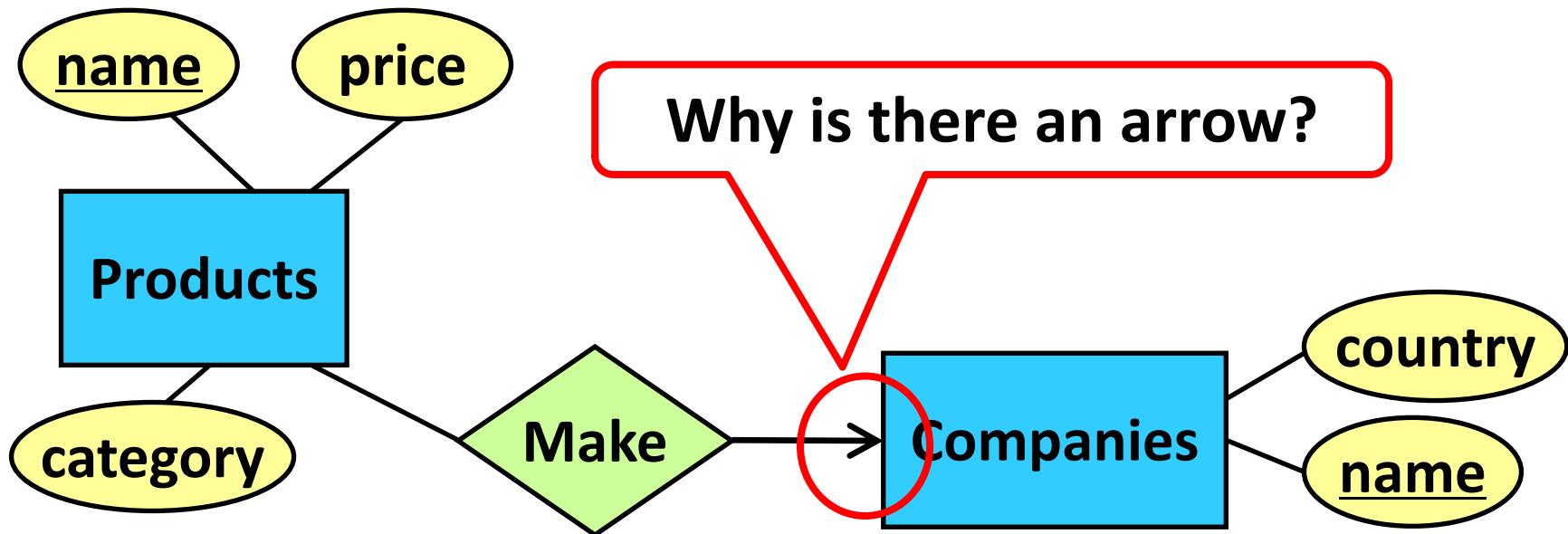
- Rectangle = **Entity Set**
- **Entity** = Real-world object (entity)
- **Entity Set** = Collection of similar objects (entities)
- Analogue: An object class in object-oriented programming language

# ER Diagram



- Oval = **Attribute** = Property of an entity set

# ER Diagram



- Diamond = **Relationship** = Connection between two entity sets
- Companies make products

# This Lecture

- Database and DBMS
- ER diagram
- Types of relationships 
- Roles

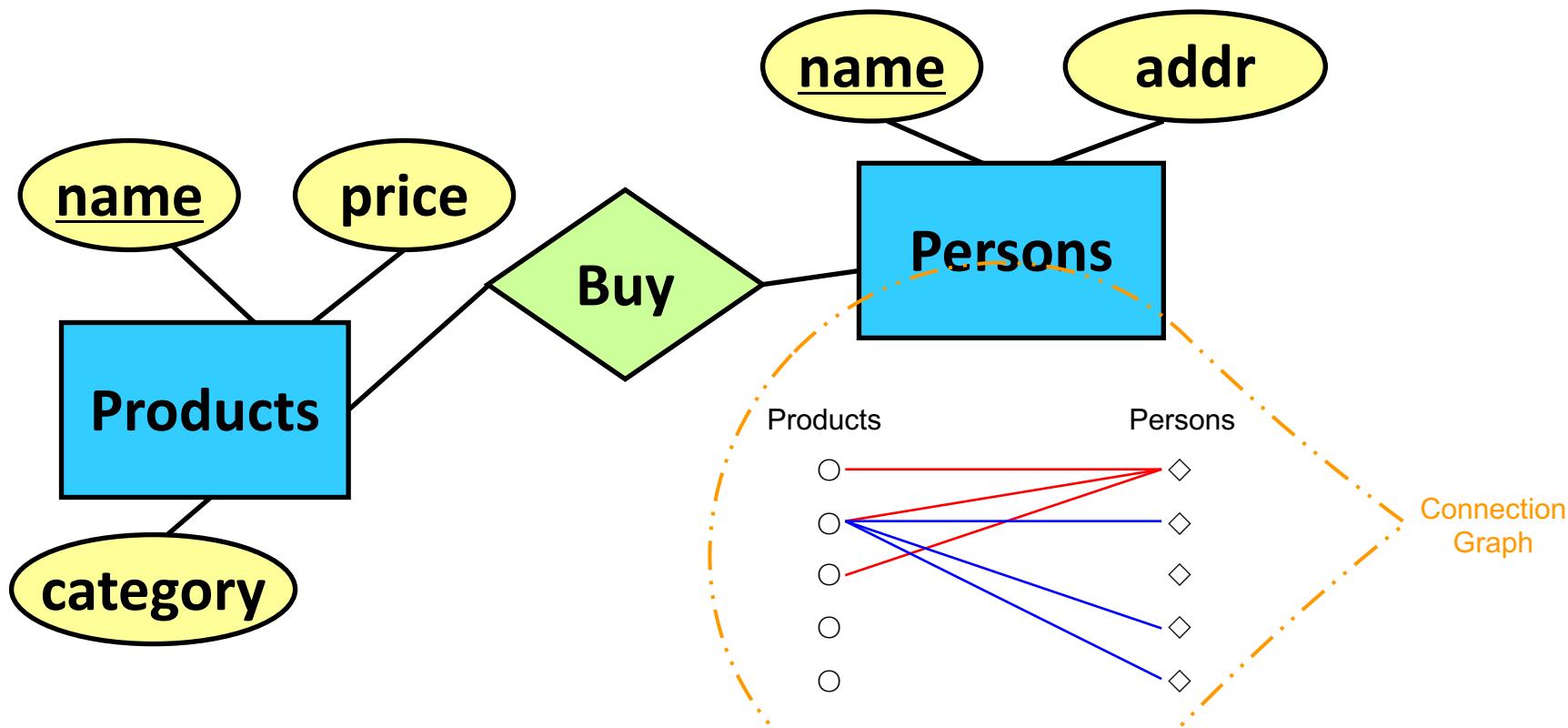
# Types of Relationships

- Many-to-Many Relationships
- Many-to-One Relationships
- One-to-One Relationships

- 1 to many example: products by 1 company
- many to one example: 1 capital to 1 country
- 1 to 1 example: 1 city

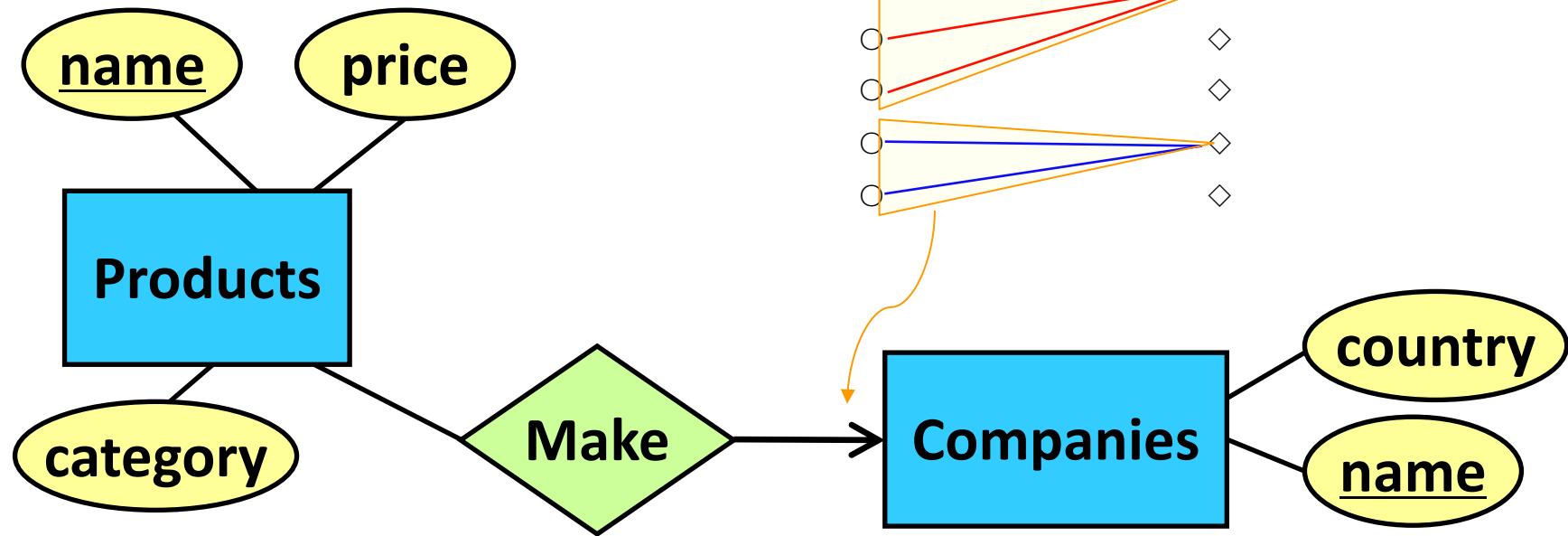


# Many-to-Many Relationship



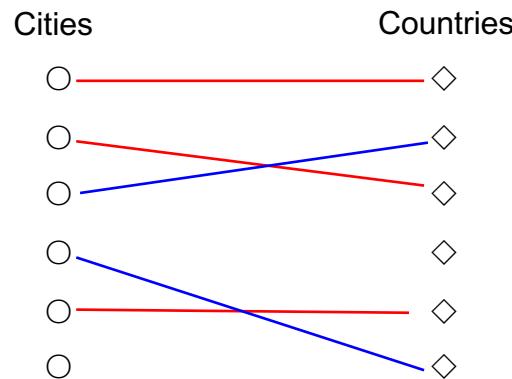
- One person can buy multiple products
- One product can be bought by multiple persons
- Note: Some Person entity is not related to Product entities, vice versa

# Many-to-One Relationship



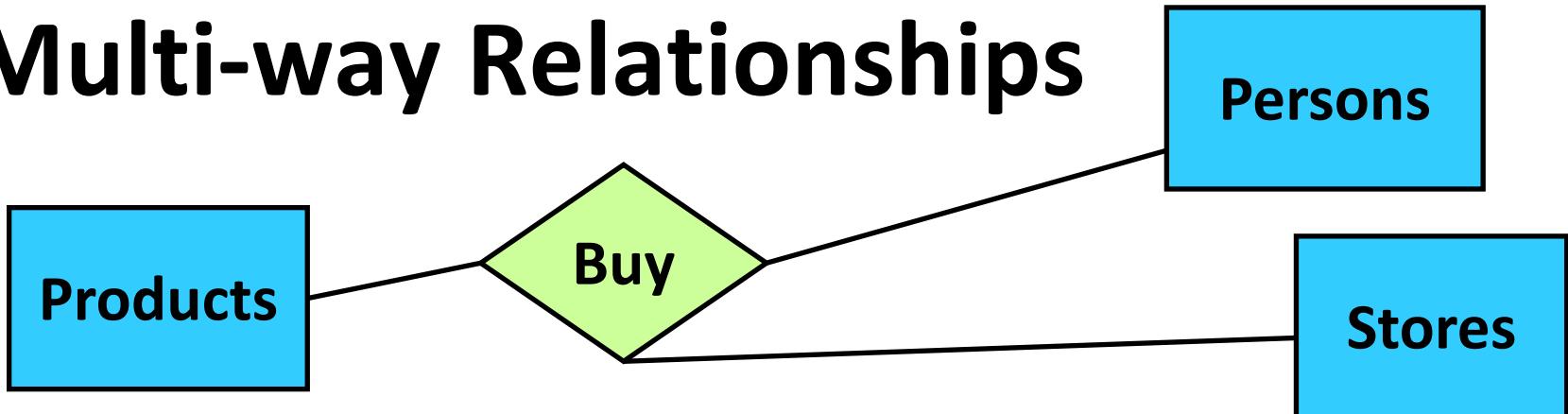
- One company can make multiple products
- But one product can only be made by one company
- Note: Some Company entity does not make any Product entity

# One-to-One Relationship



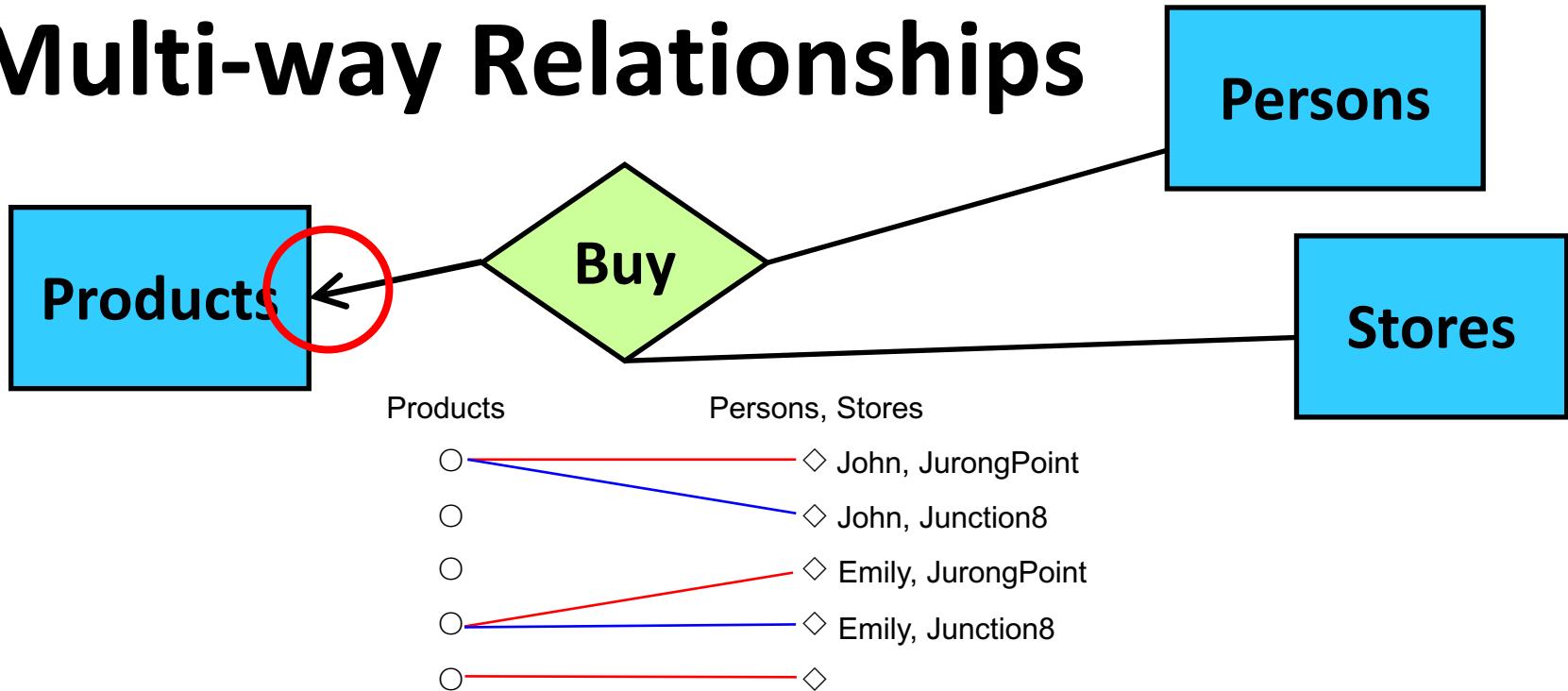
- A city can be the capital of only one country
- A country can have only one capital city
- Note: Some Country entity has no capital city, vice versa

# Multi-way Relationships



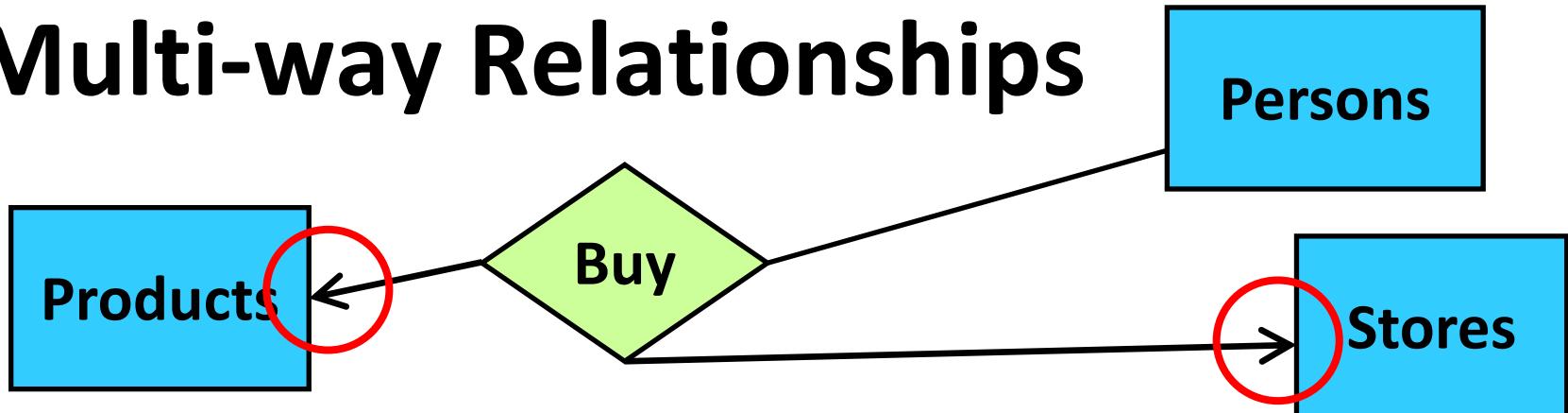
- What if we want to record the store from which the person bought the product?
- We can use a 3-way relationship
- Drawback: The arrows would be complicated

# Multi-way Relationships



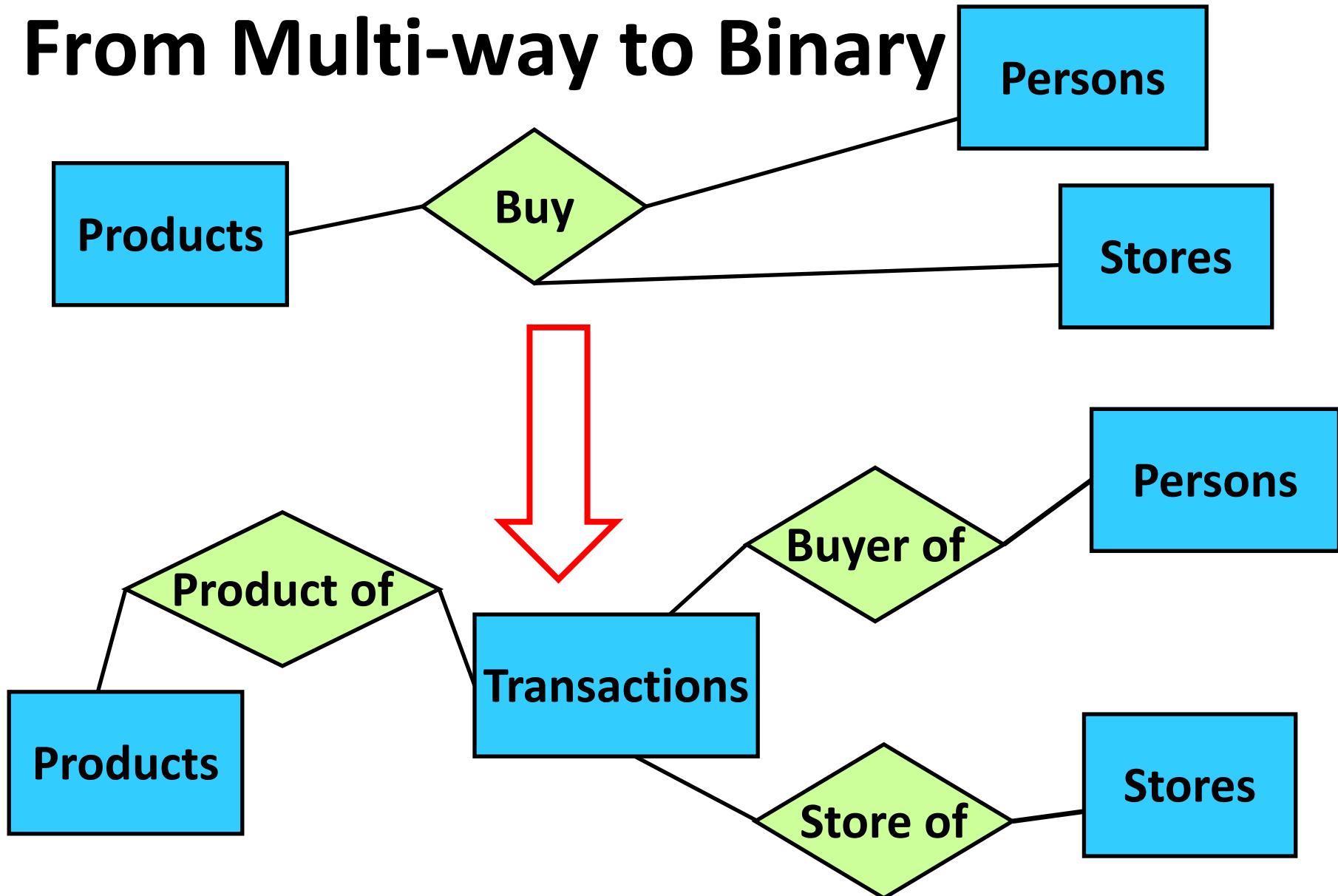
- Should we use this? What does it mean?
- One <person, store> pair to one product
- One product to many <person, store> pairs
- Meaning: A person only buys one product from one shop

# Multi-way Relationships

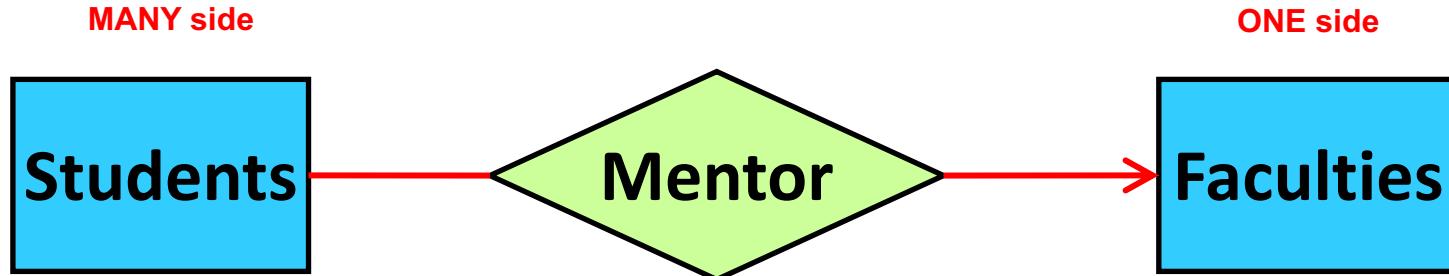


- What about this?
- <person, store> to product? many to one?
- <person, product> to store? many to one?
- Getting more complicated - avoid this

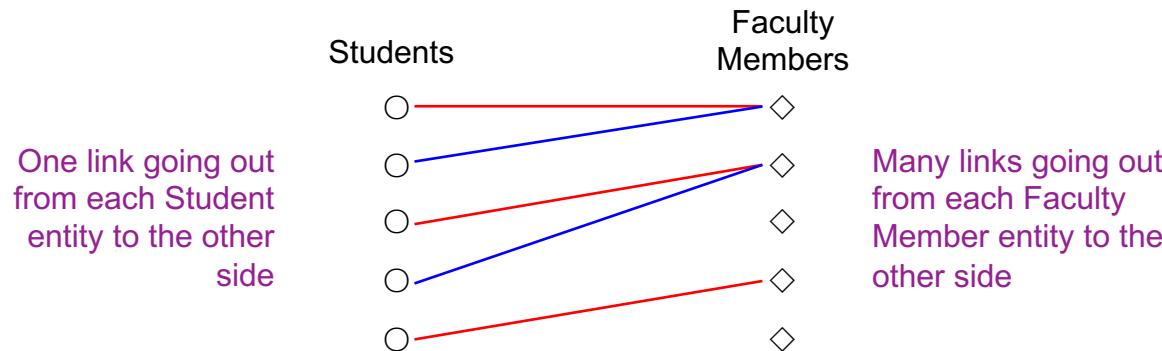
# From Multi-way to Binary



# Example



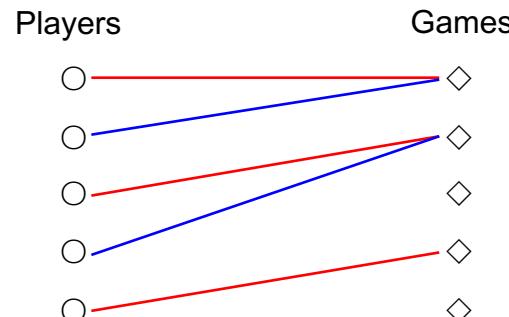
- Each student is mentored by one faculty member
- Each faculty member can mentor multiple students



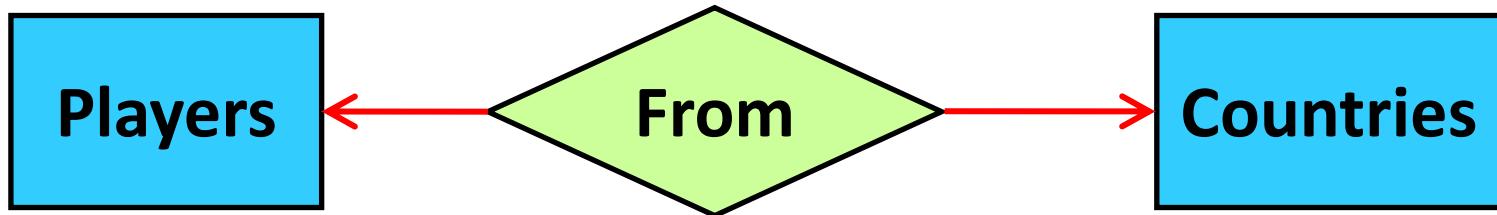
# Exercise



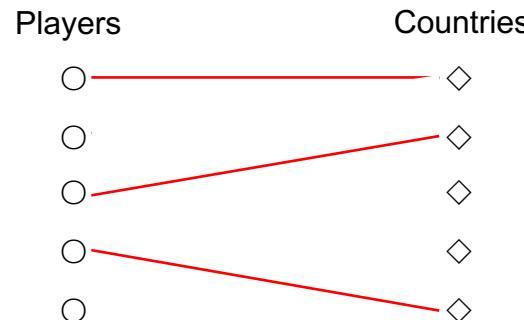
- Each player prefers only one game, but not vice versa
- Many-to-many? X
- Many-to-one? ✓
- One-to-one? X



# Exercise



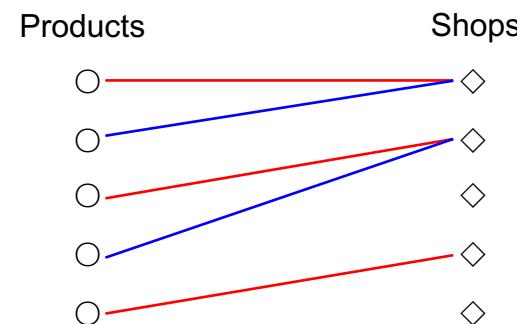
- Any two players are from exactly two different countries
- Many-to-many? X
- Many-to-one? X
- One-to-one? ✓



# Exercise



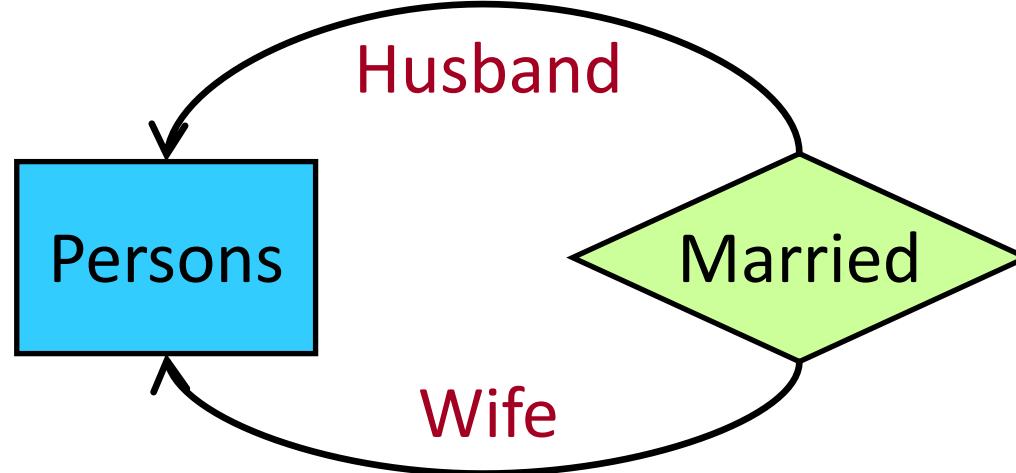
- No two shops sell the same product
- Many-to-many? X
- Many-to-one? ✓
- One-to-one? X



# This Lecture

- Database and DBMS
- ER diagram
- Types of relationships
- Roles ←

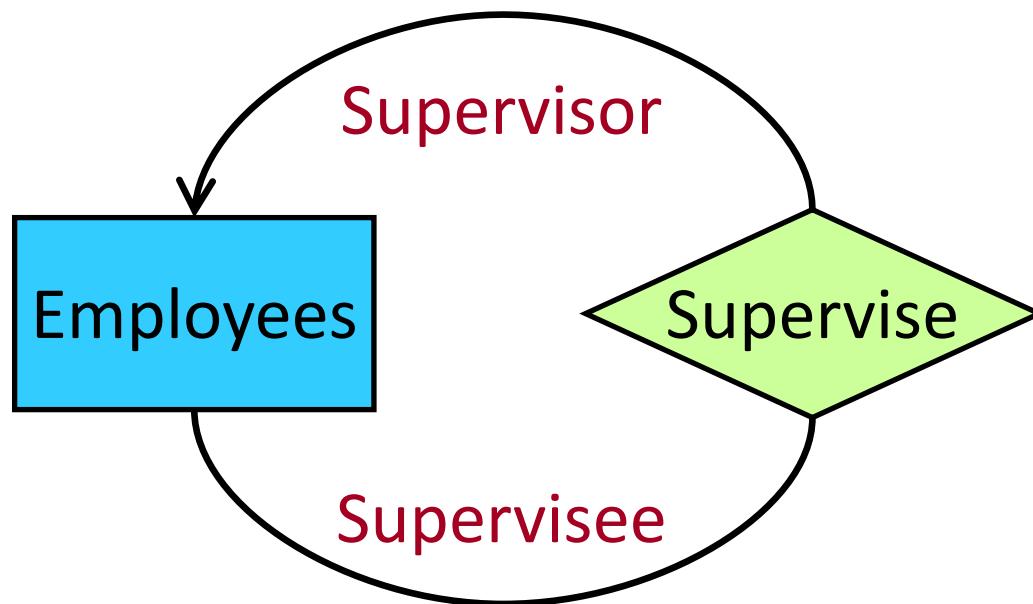
# Roles



- Sometimes an entity set may appear more than once in a relationship
- Example: some persons are married to each other
- The **role** of the person is specified on the **edge** connecting the entity set to the relationship

| Husband | Wife  |
|---------|-------|
| Bob     | Alice |
| David   | Cathy |
| ...     | ...   |

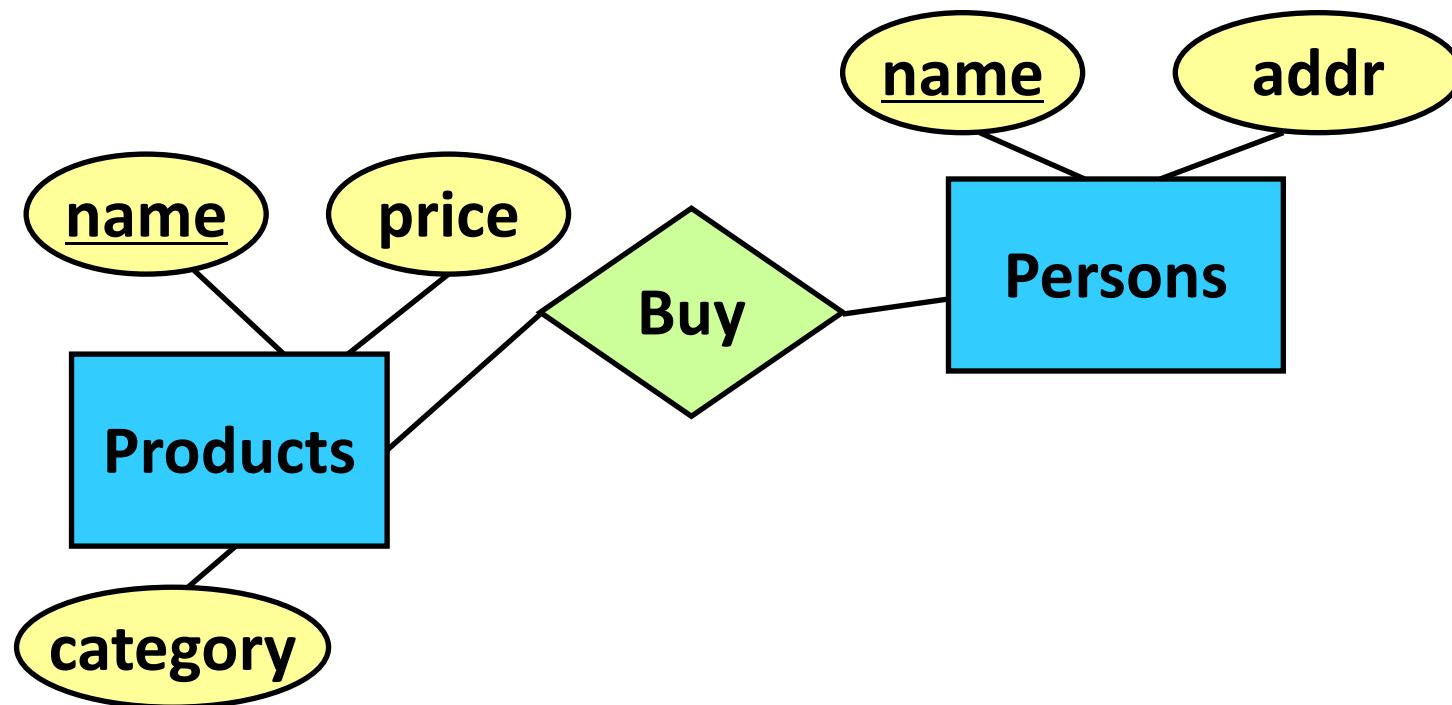
# Roles



- Example: some employee supervises other employees
- Without the roles, it is unclear whether it is many-to-one from supervisees to supervisors, or the other way around

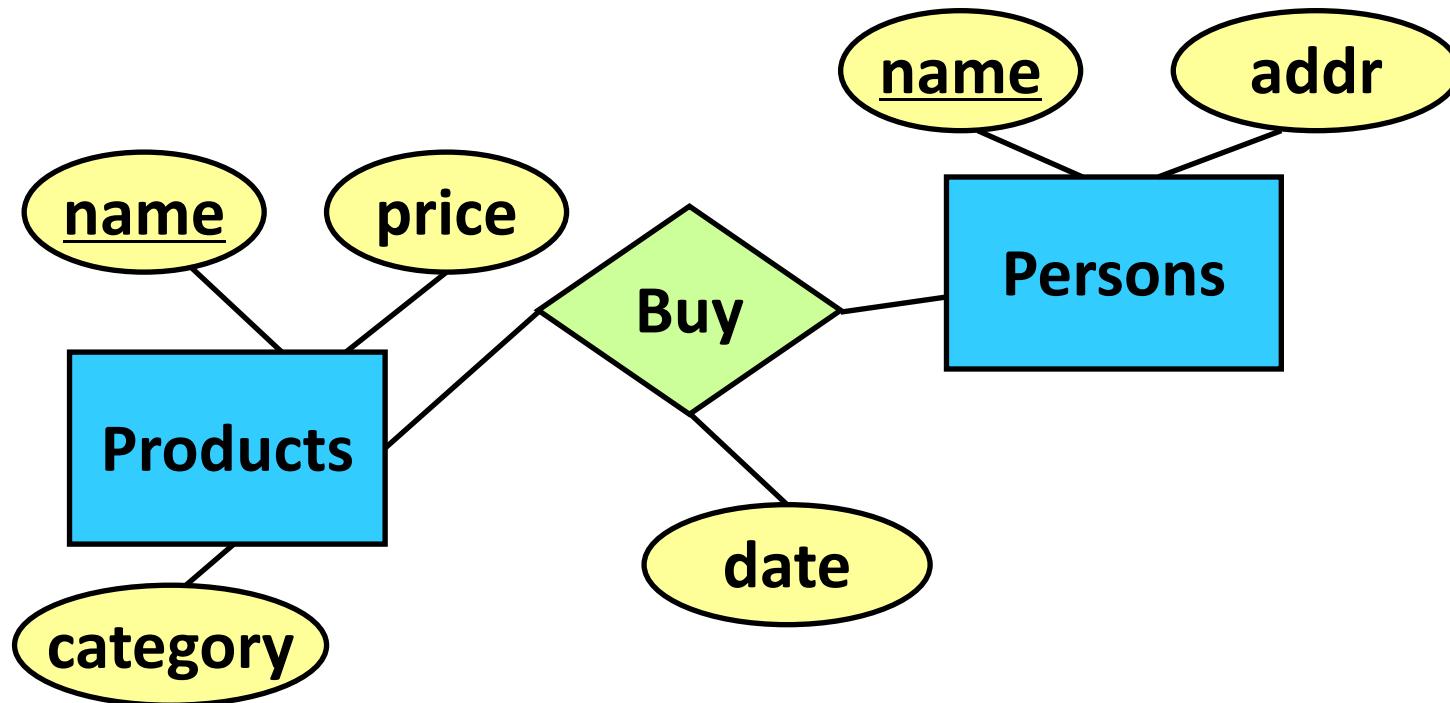
# One More Thing about Relationships

- A relationship can have its own attribute



# One More Thing about Relationships

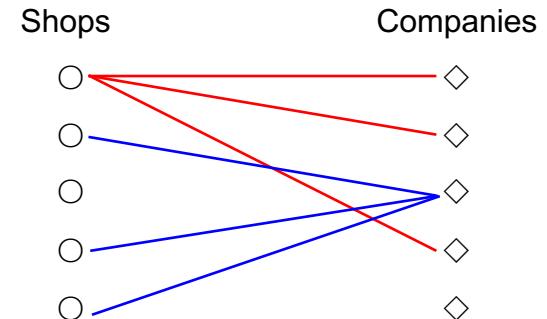
- A relationship can have its own attribute
- If we want to record the date of the purchase



# Exercise

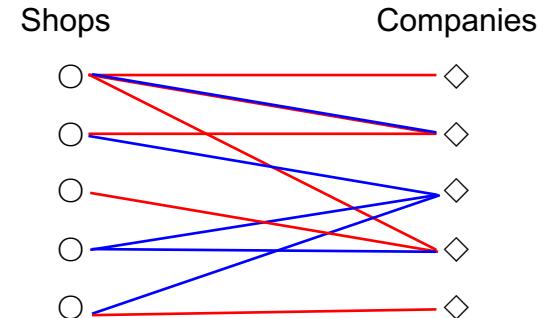
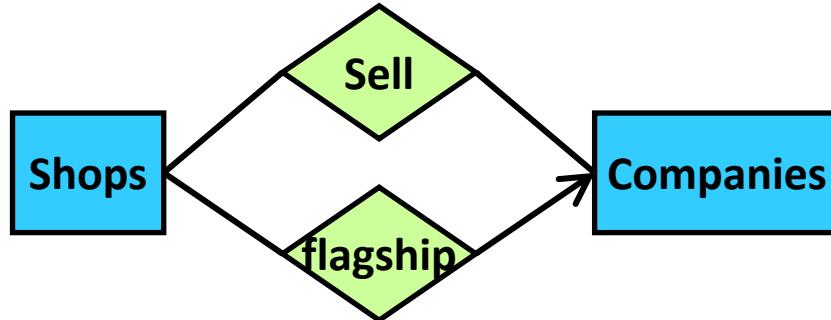
- Consider two entity sets, Shops and Companies
- Each shop sells products from at least one company
- Each company has its product sold in at least one shop
- A shop may be the flagship shop of at most one company
- Each company has at least one flagship shops
- Draw some relationships between Shops and Companies to capture the above statements

# Exercise

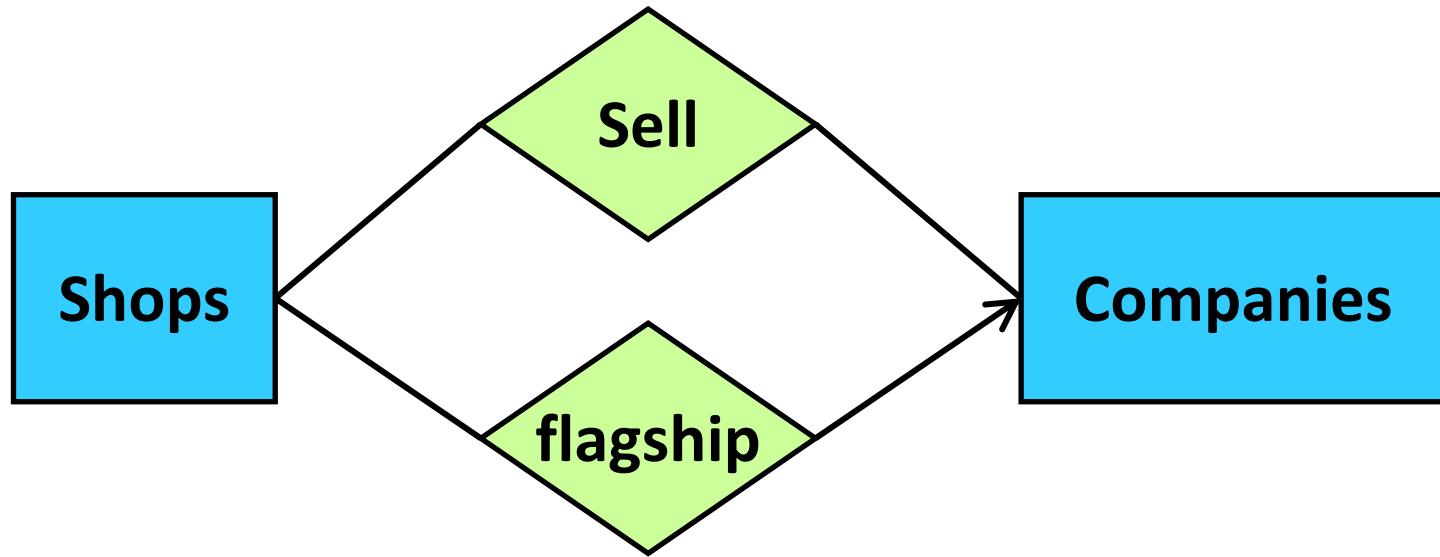


- Consider two entity sets, Shops and Companies
  - Each shop sells products from at least one company
  - Each company has its product sold in at least one shop
  - A shop may be the flagship shop of at most one company
- 
- -

# Exercise



- Consider two entity sets, Shops and Companies
- Each shop sells products from at least one company
- Each company has its product sold in at least one shop
- A shop may be the flagship shop of at most one company
- Each company has at least one flagship shops
- Draw some relationships between Shops and Companies to capture the above statements



- There can be multiple relationships between two entity sets

# **CZ2007 Introduction to Database Systems (Week 1)**

---

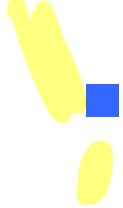
## **Topic 1: Entity Relationship Diagram (2)**



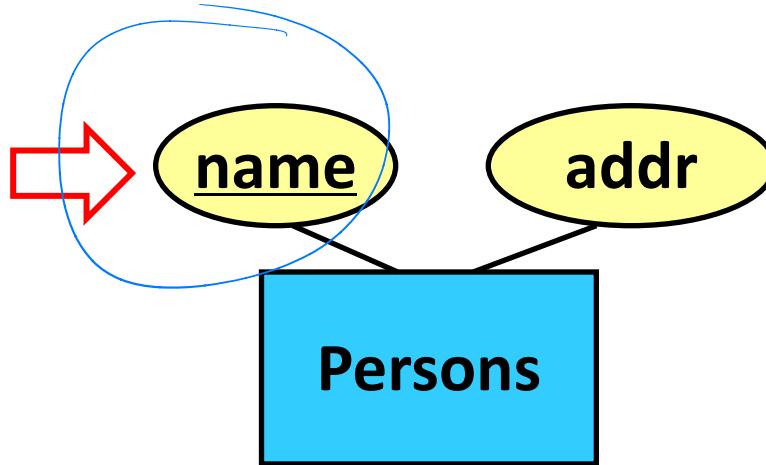
# This Lecture

- Constraints ←
- Subclasses
- Weak Entity Sets

# Constraints

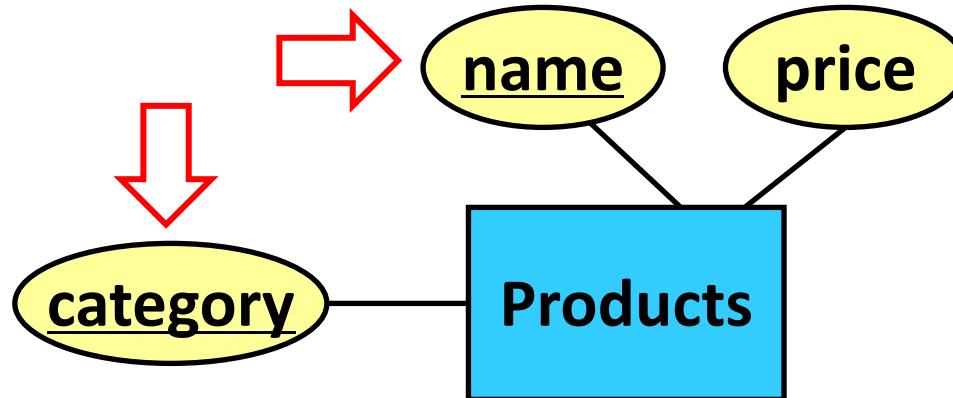
- 
- Some conditions that entity sets and relationships should satisfy
  - We will focus on three types of constraints
    - Key constraints ✓
    - Referential integrity constraints ✓
    - Degree constraints ✓

# Key



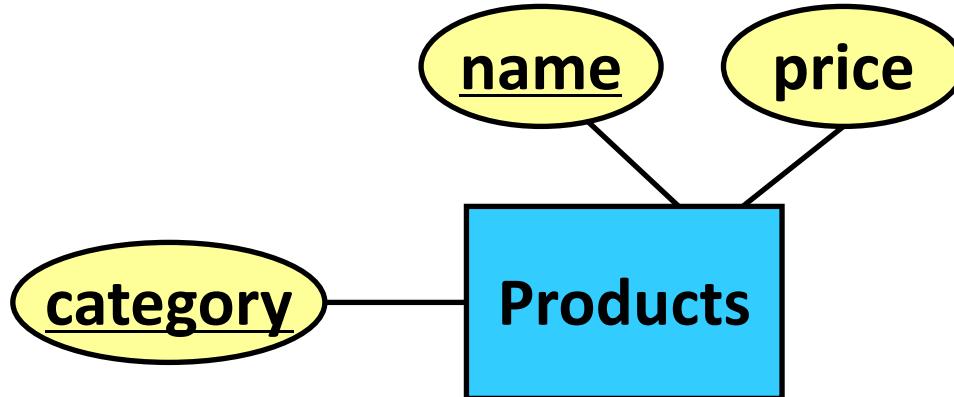
- One or more attributes that are underlined
- Meaning: They uniquely represent each entity in the entity set
- Example: The “name” uniquely represents each and every person  
i.e., each person must have a unique name

# Key



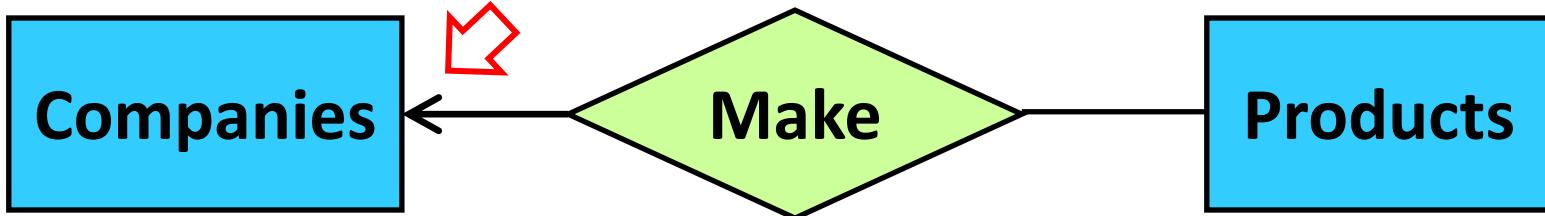
- One or more attributes that are underlined
- Each product has a unique <name, category> combination
- But there can be products with the same name, or the same category, but not both
- Example
  - Name = “**Apple**”, Category = “Fruit”, Price = “1”
  - Name = “**Apple**”, Category = “Phone”, Price = “888”

# Key



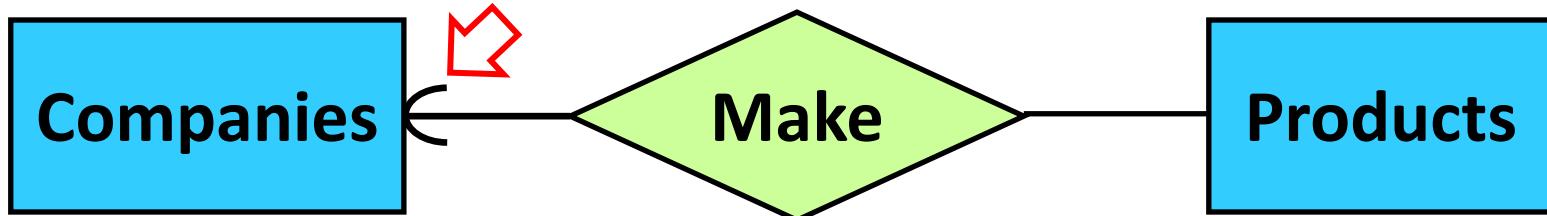
- Rule: Every entity set should have a key
  - So that we can uniquely refer to each entity in the entity set

# Referential Integrity

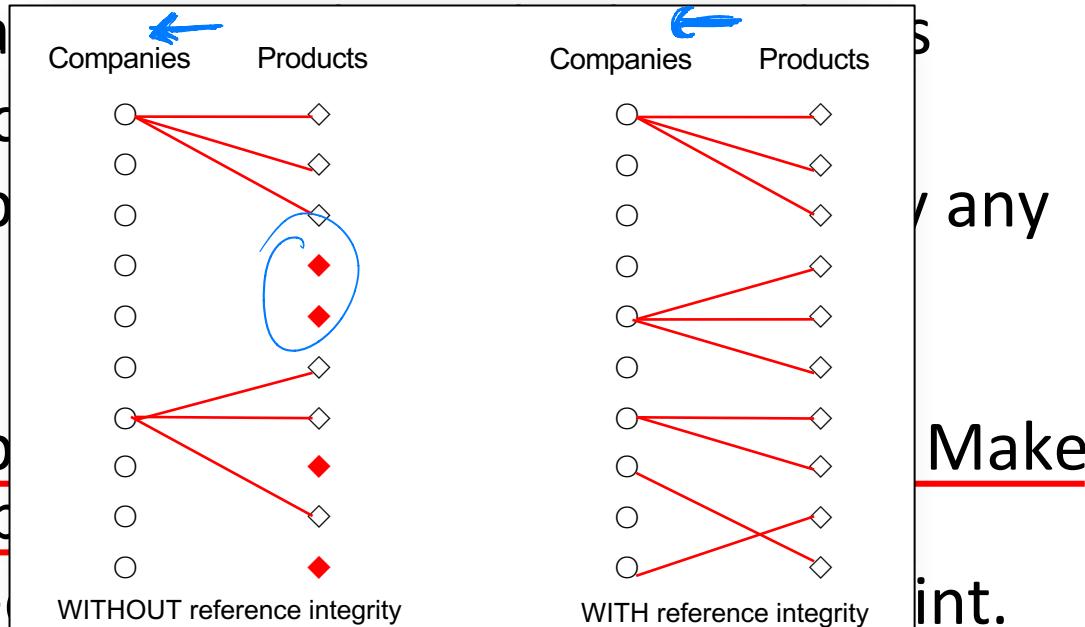


- One company may make multiple products
- One product is made by one company
- Can there be a product that is not made by any company?
- No.  
    *+ all*  
    i.e., every product must be involved in the Make relationship
- This is called a referential integrity constraint.
- How do we specify this in an ER diagram?
- Use a rounded arrow instead of a pointed arrow  
    *←*

# Referential Integrity

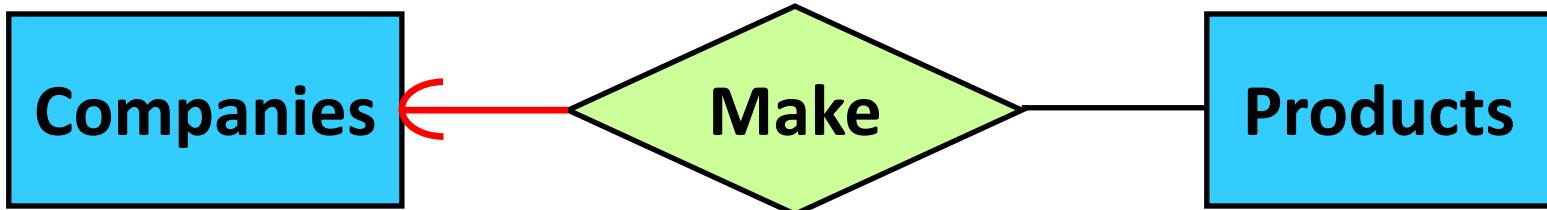


- One company can make many products
- One product can be made by many companies
- Can there be a company that makes no products?
- No.
- i.e., every product must be made by some company
- This is called referential integrity

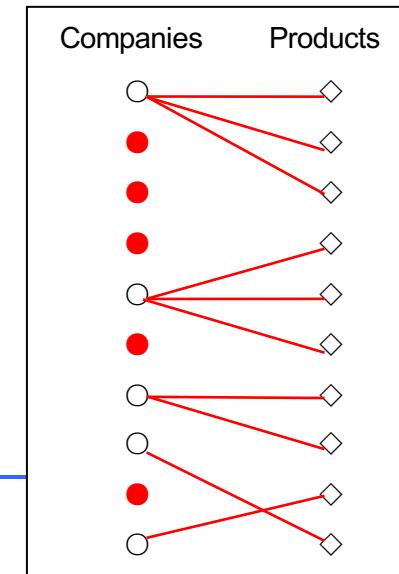


- How do we specify this in an ER diagram?
- Use a rounded arrow instead of a pointed arrow

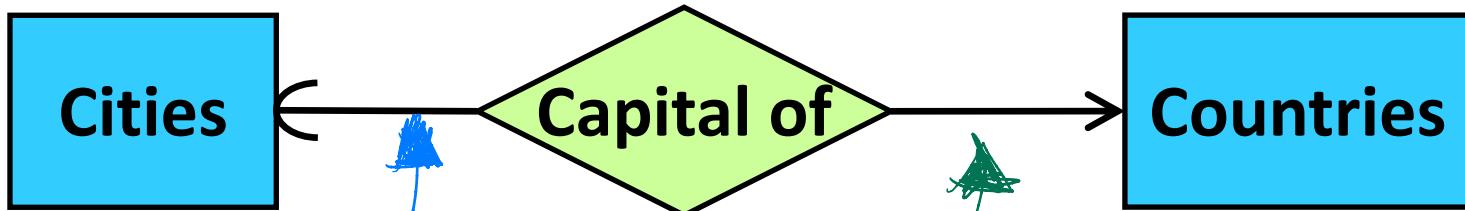
# Referential Integrity



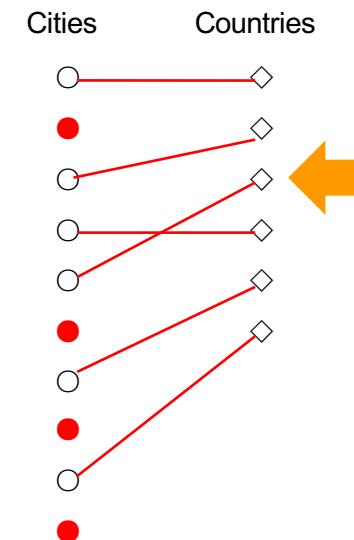
- What if every company should make at least one product?
- No arrow there .... **but we indicate using degree constraints**
- In general, a referential integrity constraint can only apply to the “one” side of
  - A many-to-one relationship, or
  - A one-to-one relationship
- 
- 



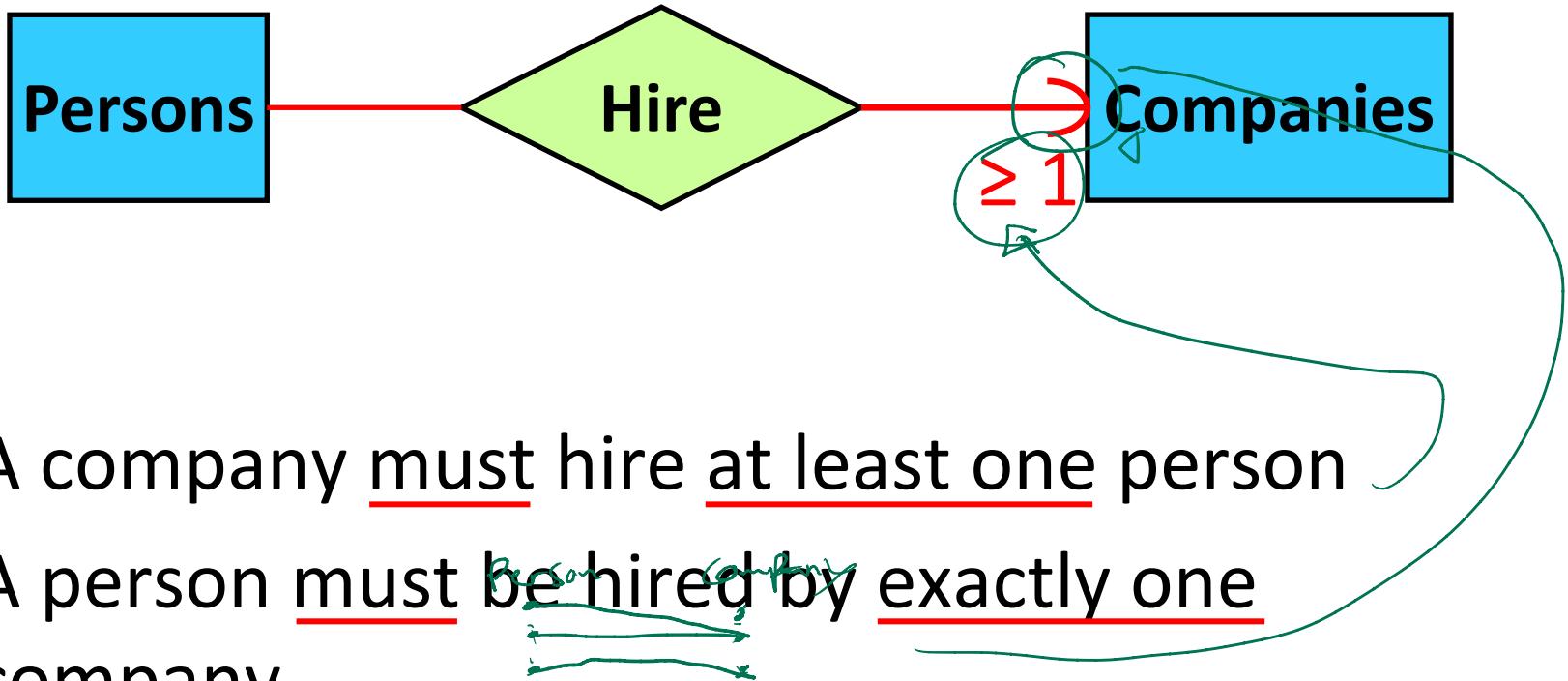
# Referential Integrity: Exercise



- A city can be the capital of only one country
- A country must have a capital

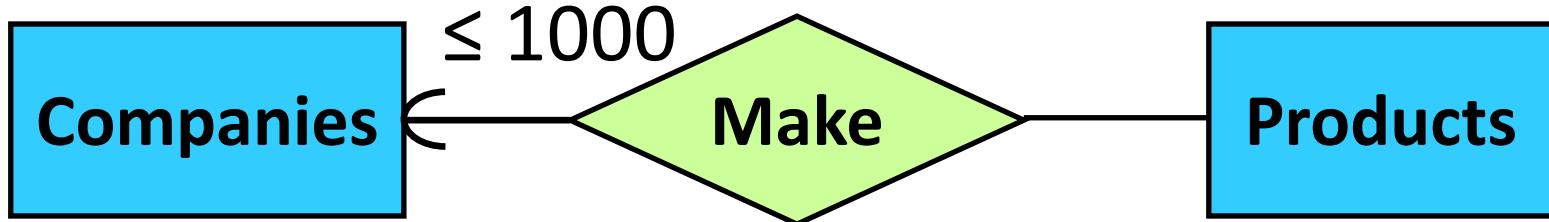


# Referential Integrity: Exercise



- A company must hire at least one person
- A person must be hired by exactly one company
- To say “Each and every company must hire at least one person”, need degree constraints

# Degree Constraint

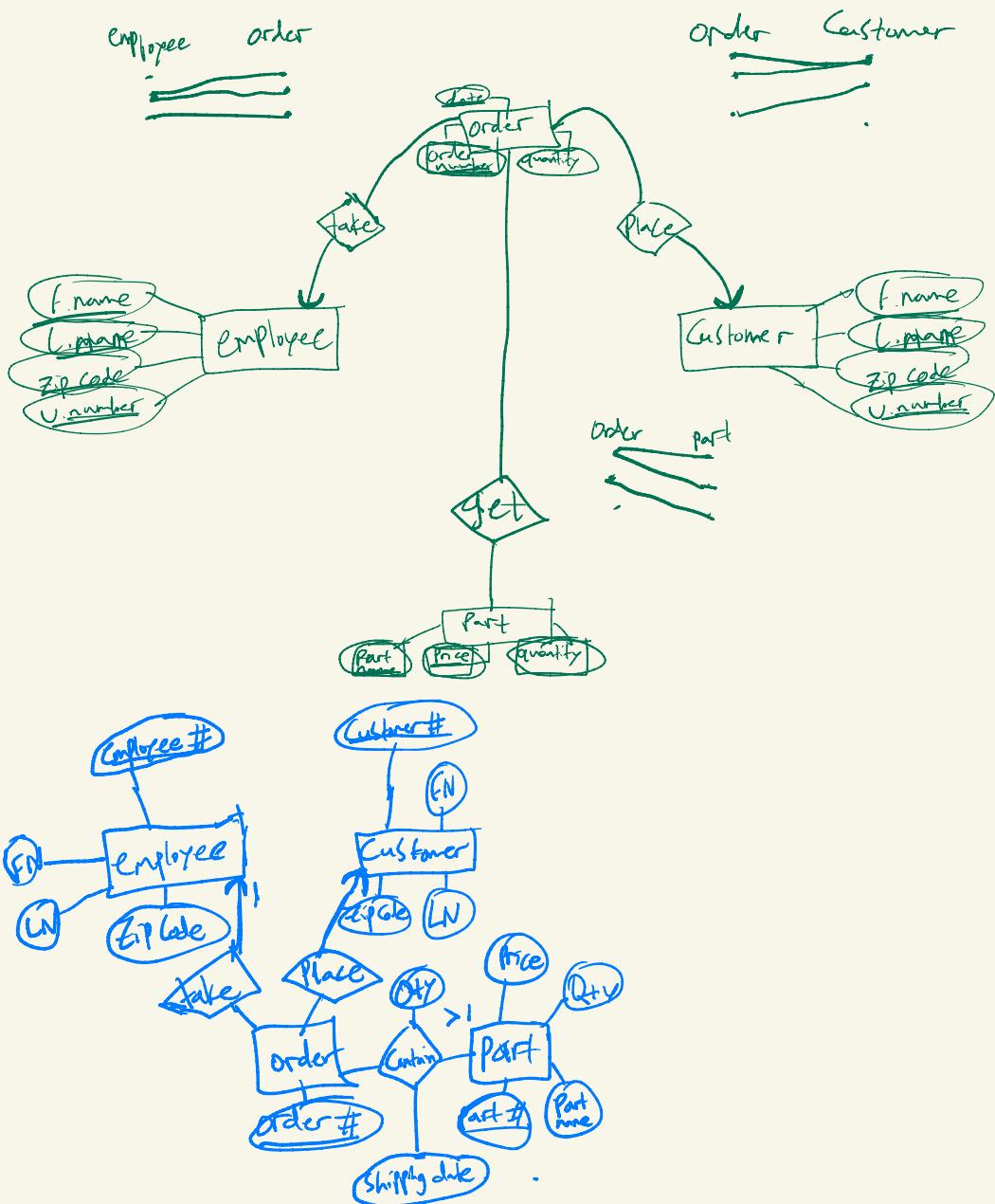


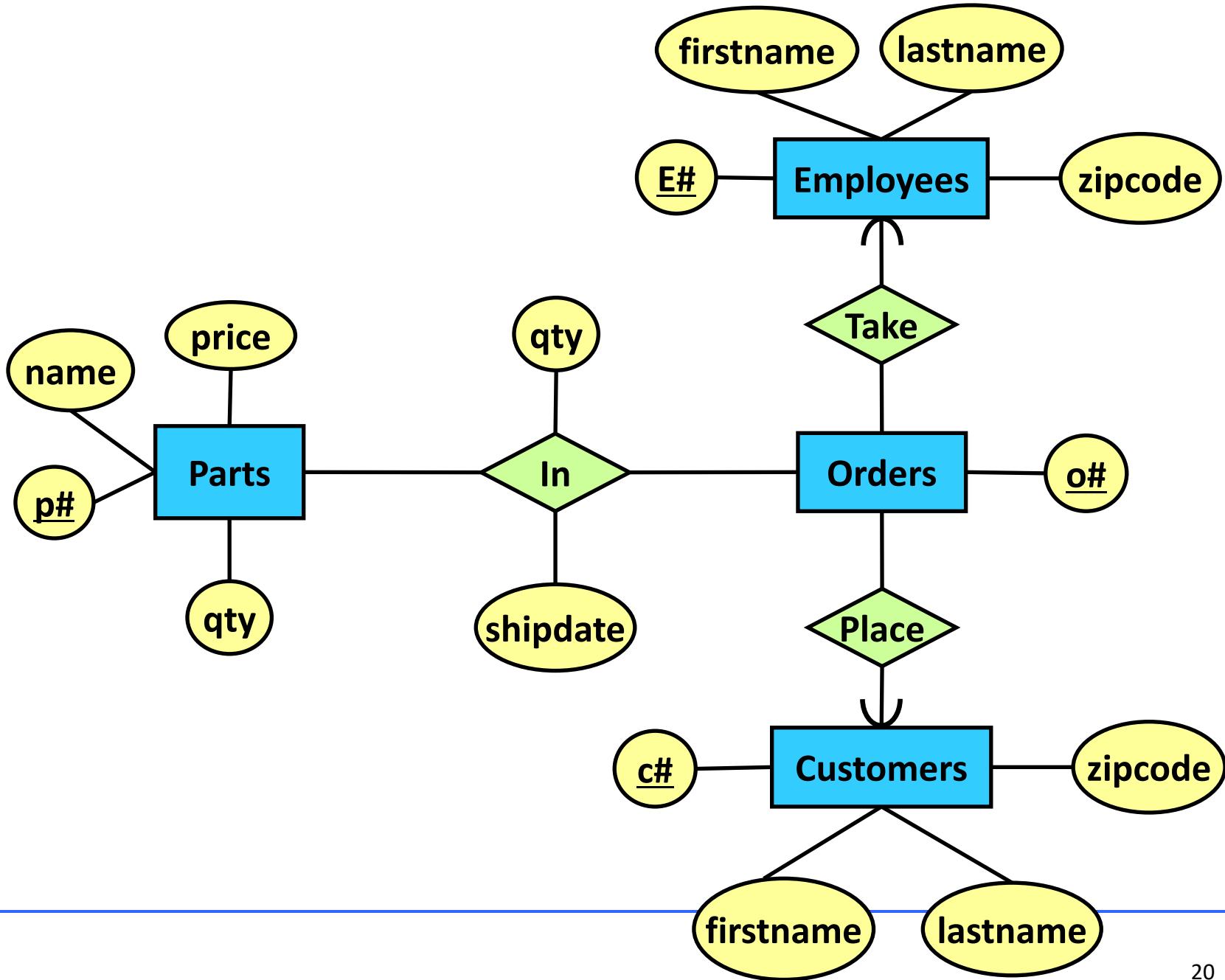
*Not required in exam*

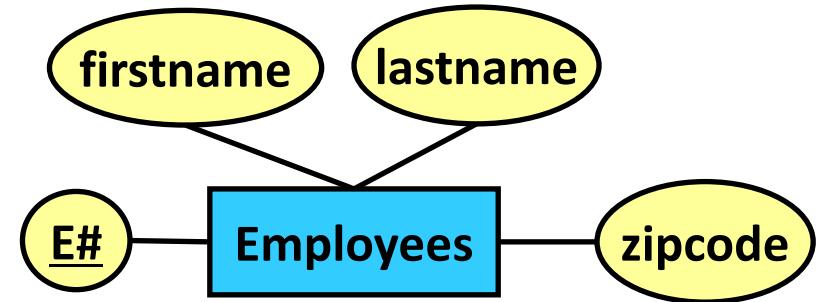
- Each (and every) company makes at most 1000 product
- Note:
  - Degree constraints are not easy to enforce in a DBMS
  - Key and referential integrity constraints can be easily enforced

# ER Diagram Design: Exercise

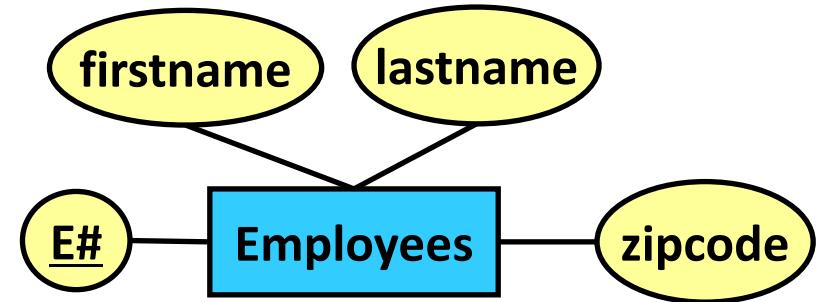
- Consider a mail order database in which employees take orders for parts from customers. The requirements are:
- Each employee is identified by a unique employee number, and has a first name, a last name, and a zip code.
- Each customer is identified by a unique customer number, and has a first name, last names, and a zip code.
- Each part being sold is identified by a unique part number. It has a part name, a price, and a quantity in stock.
- Each order placed by a customer is taken by one employee and is given a unique order number. Each order may contain certain quantities of one or more parts. The shipping date of each part is also recorded.



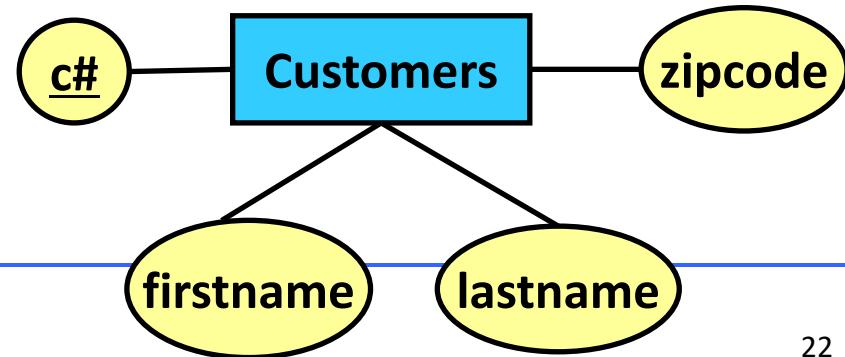


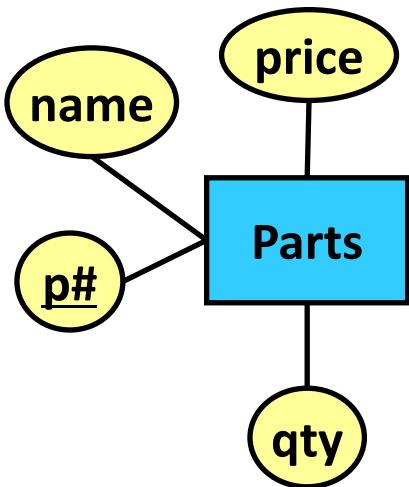


- Each employee is identified by a unique employee number, and has a first name, a last name, and a zip code.

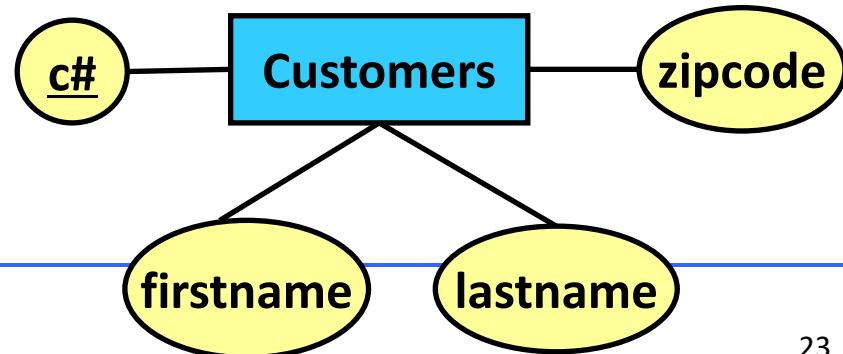
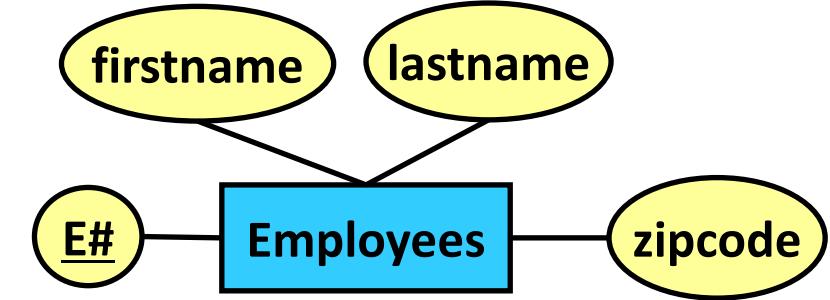


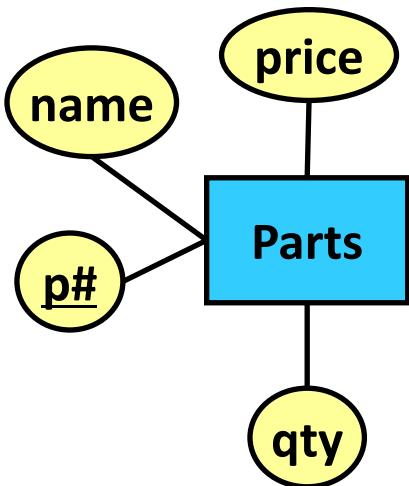
- Each customer is identified by a unique customer number, and has a first name, last names, and a zip code.



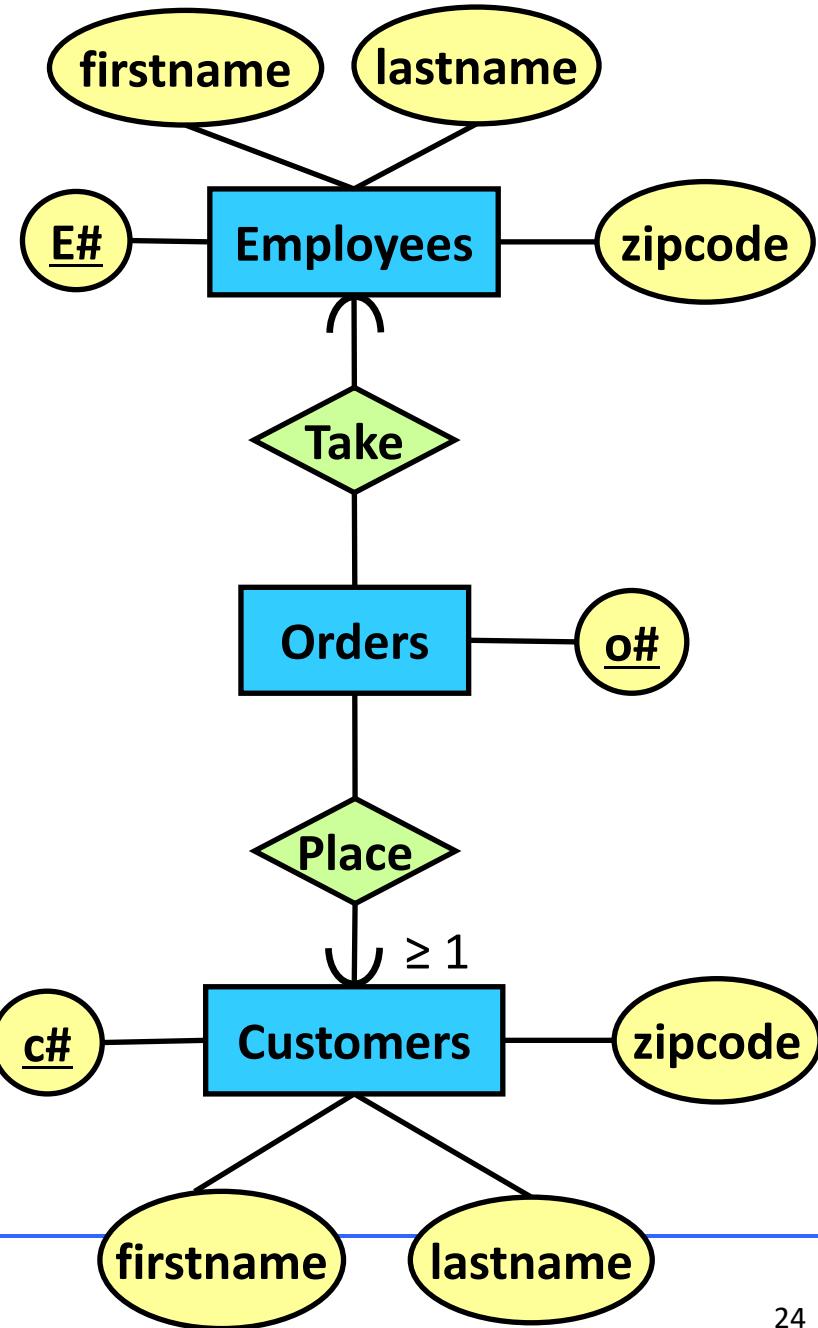


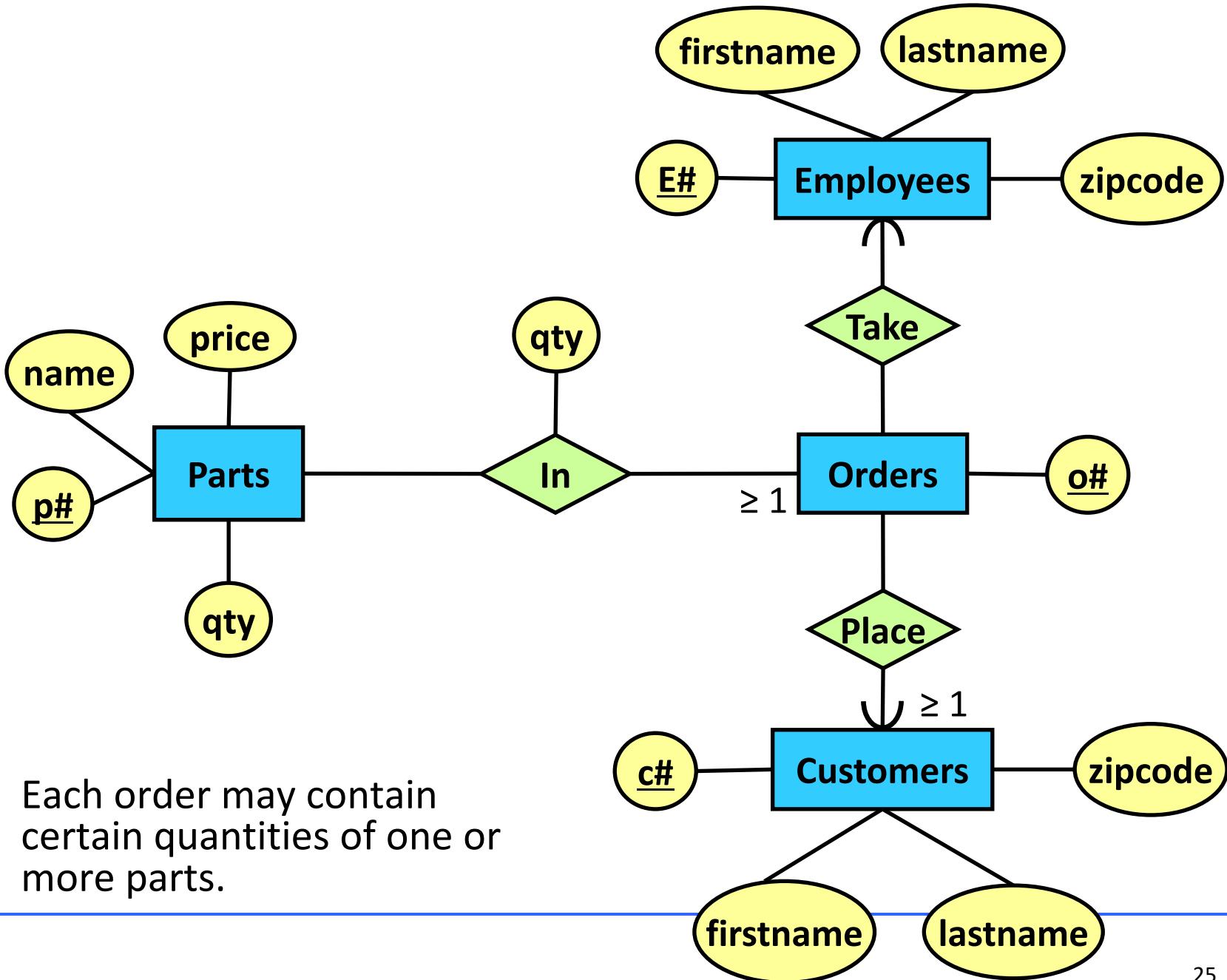
- Each part being sold is identified by a unique part number. It has a part name, a price, and a quantity in stock.

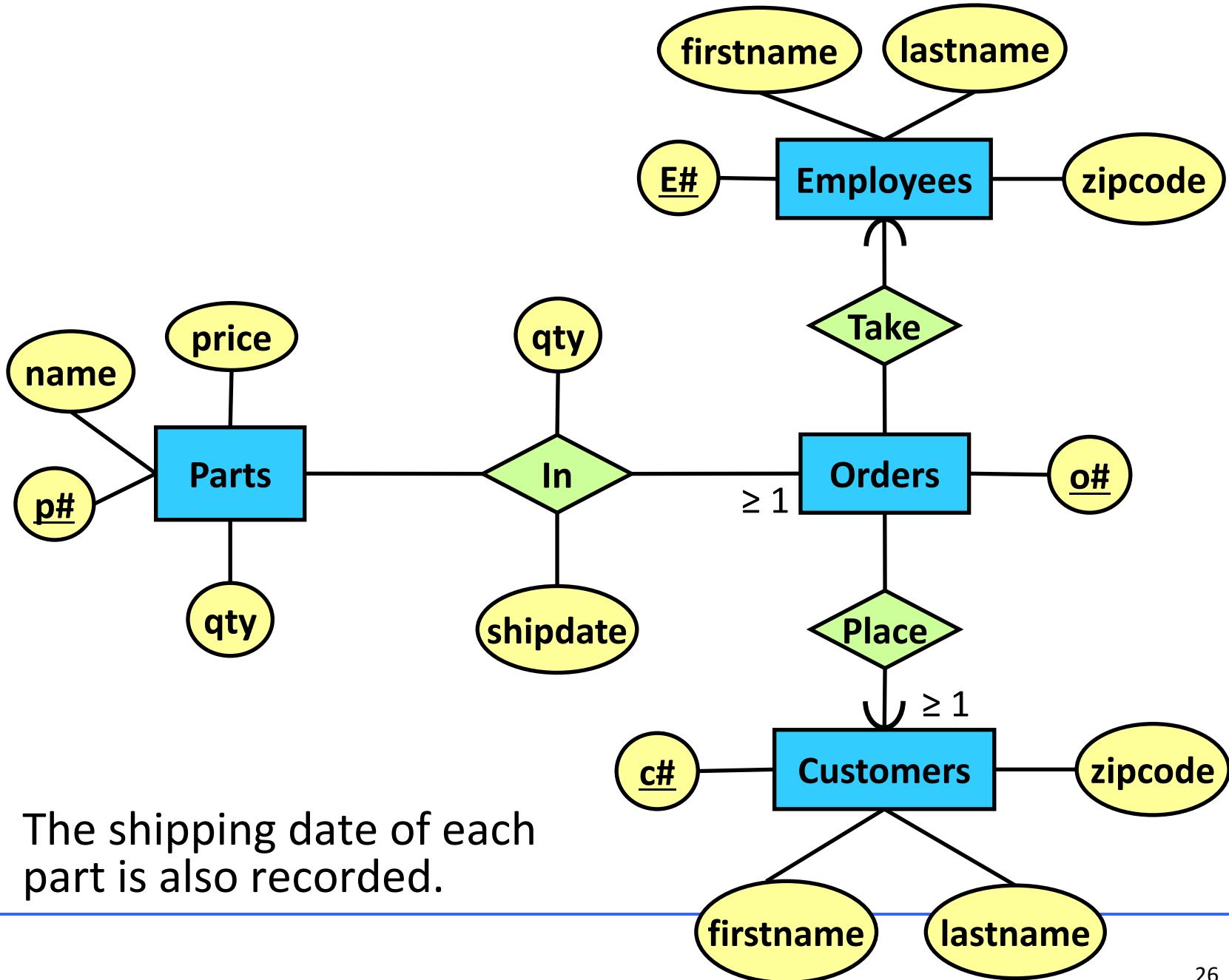




- Each order placed by a customer is taken by one employee and is given a unique order number.





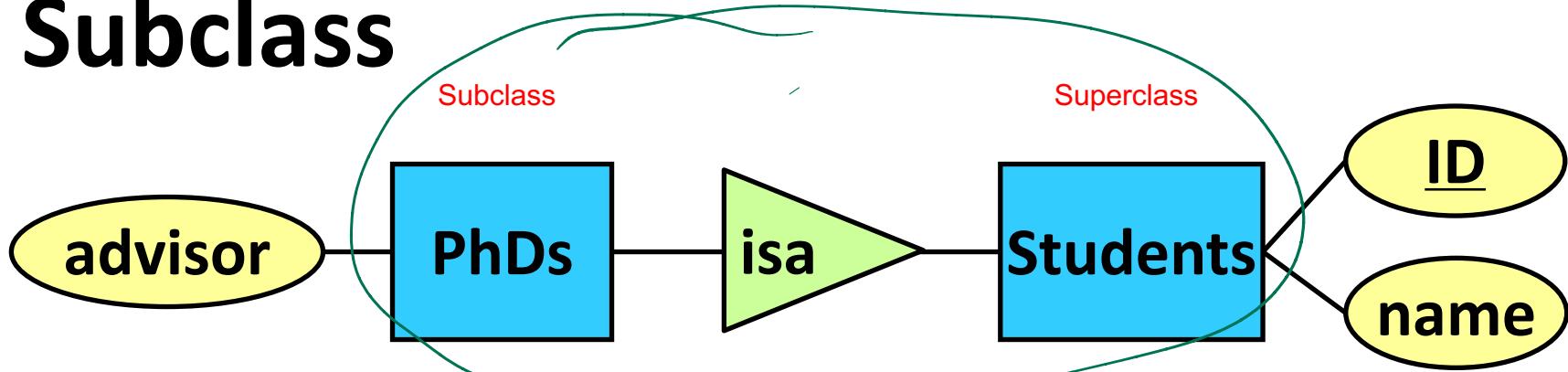


- The shipping date of each part is also recorded.

# This Lecture

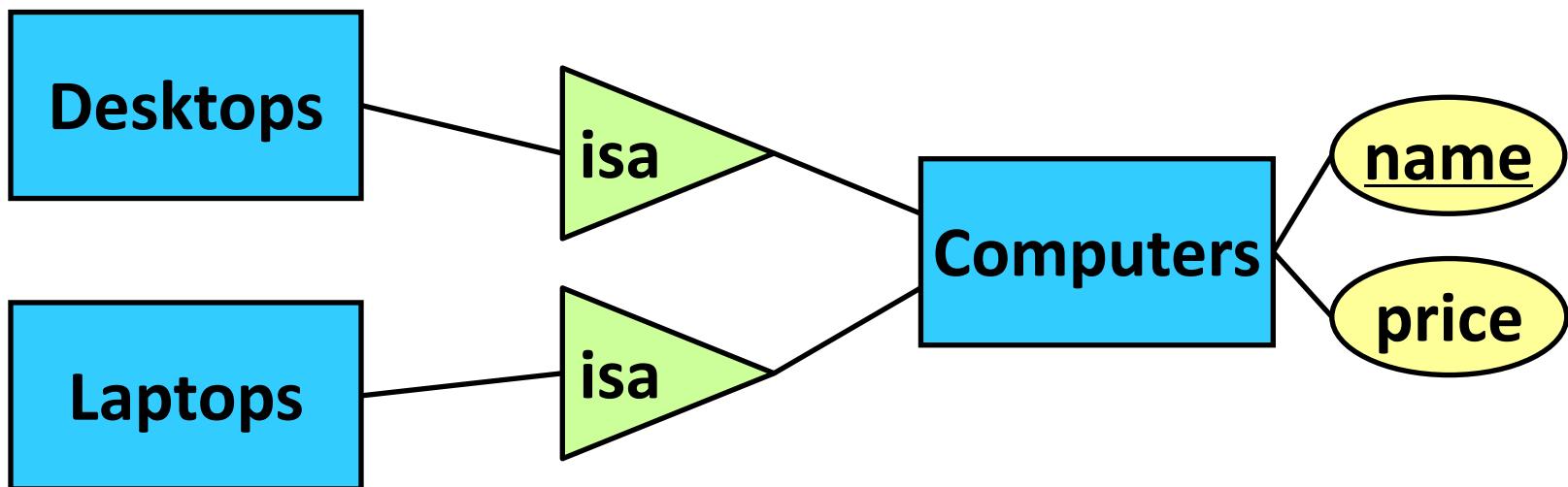
- Constraints
- Subclasses 
- Weak Entity Sets

# Subclass

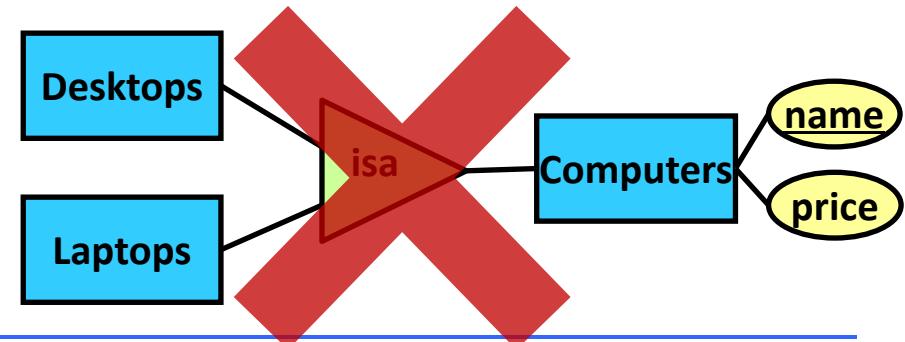


- PhDs are a special type of Students
- Subclass = Special type
- The connection between a subclass and its superclass is captured by the isa relationship, which is represented using a triangle
- Key of a subclass = key of its superclass
- Example: Key of Phds = Students.ID
- Students is referred to as the superclass of PhDs

# Subclass



- An entity set can have multiple subclasses
- Example
  - Superclass: Computers
  - Subclass 1: Desktop
  - Subclass 2: Laptop

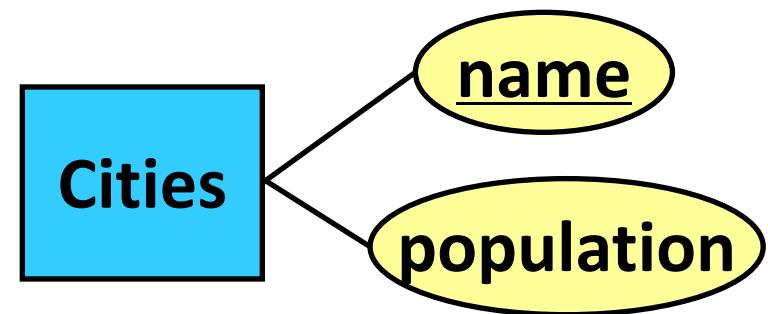


# This Lecture

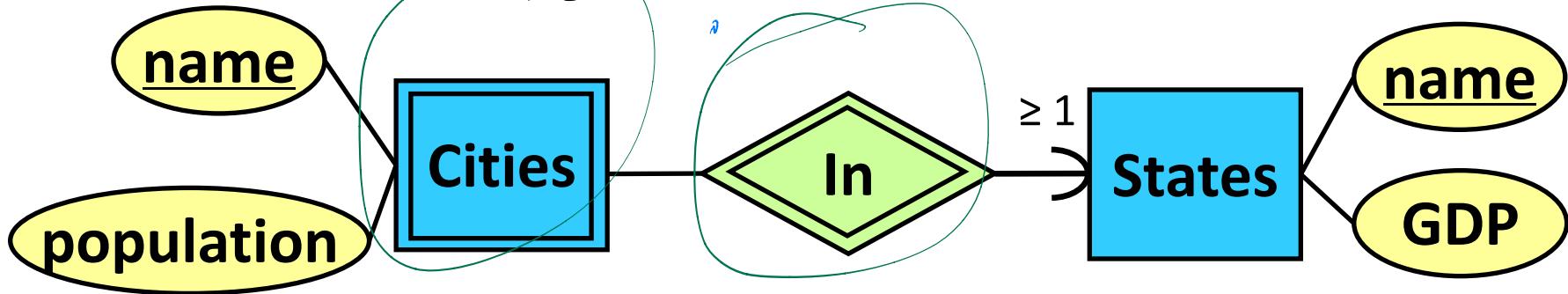
- Constraints
- Subclasses
- Weak Entity Sets 

# Weak Entity Sets

- Weak entity sets are a special type of entity sets that
  - cannot be uniquely identified by their own attributes
  - needs attributes from other entities to identify themselves
- Example: Cities in USA
- Problem: there are cities with identical names

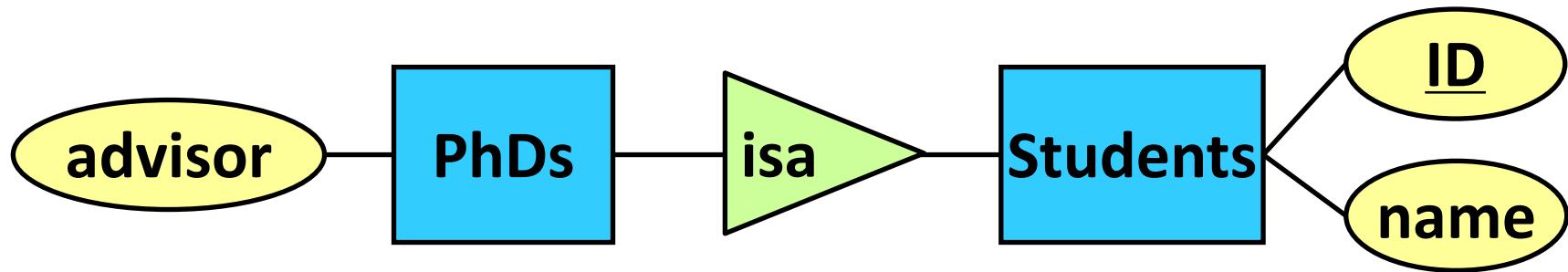


# Weak Entity Sets

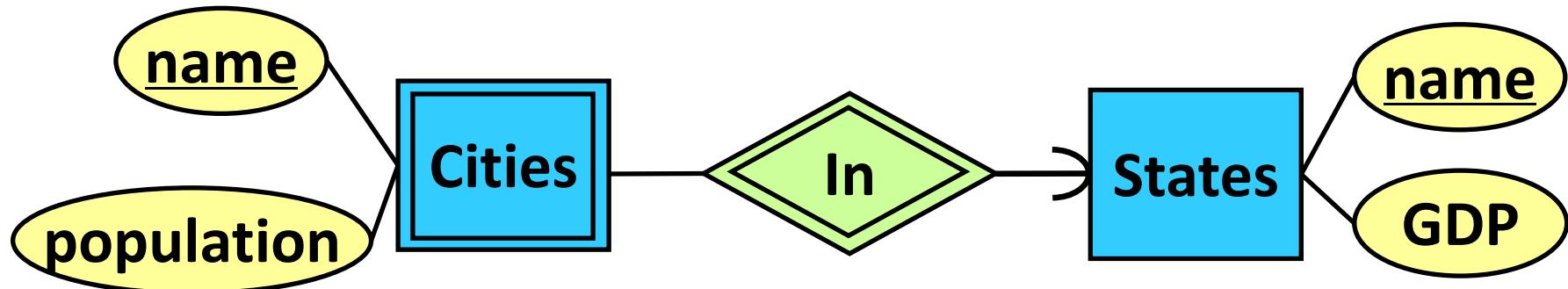


- Problem: there are cities with identical names
- Observation: cities in the same state would have different names
- Solution: make Cities a **weak entity set** associated with the entity set States
- The relationship **In** is called the **supporting relationship** of Cities
- Weak entity set = Double-lined rectangle
- Supporting relationship = Double-lined diamond
- The key of Cities = **(State.name, Cities.name)**
- This is a single composite key

# Subclass vs. Weak Entity Sets

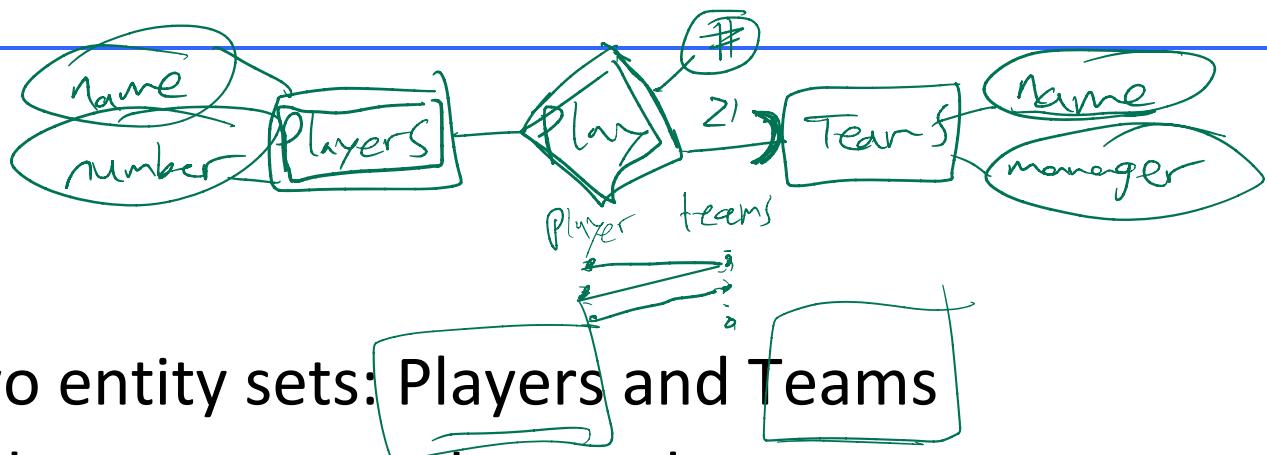


- PhDs are a special type of Students



- Cities are NOT a special type of States

# Exercise



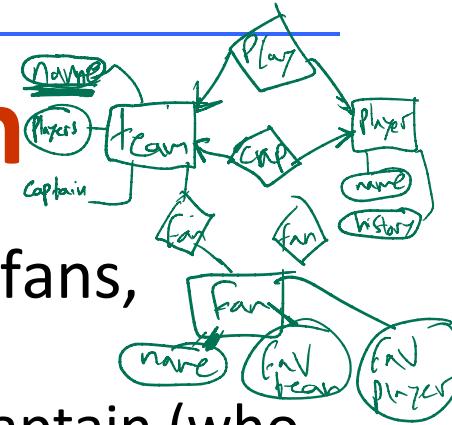
- Consider two entity sets: **Players** and **Teams**
- Each player has a name and a number
- Each team has a name and a manager
- Each player plays for exactly one team, and is uniquely identified within the team by his/her number
- Each team is uniquely identified by its name
- Different players may have the same name
- Draw a ER diagram that captures the above statements
- What is the key of Players?

# Exercise



- Consider two entity sets: Players and Teams
- Each player has a name and a number
- Each team has a name and a manager
- Each player plays for exactly one team, and is uniquely identified within the team by his/her number
- Each team is uniquely identified by its name
- Different players may have the same name
- Draw a ER diagram that captures the above statements
- What is the key of Players? (**Players.number, Teams.name**)

# Exercise: ER-Diagram Design



- Record info about teams, players, and their fans, including:
  - For each team, its name, its players, its team captain (who is also a player)
  - For each player, his/her name, and the history of teams on which he/she has played, including the start and ending dates for each team
  - For each fan, his/her name, favorite teams, favorite players
- Additional information:
  - Each team has at least one player, and exactly one captain
  - Each team has a unique name
  - Two players (or two fans) may have the same name
  - Each fan has at least one favorite team and at least one favorite player

Exercise: ER-Diagram Design

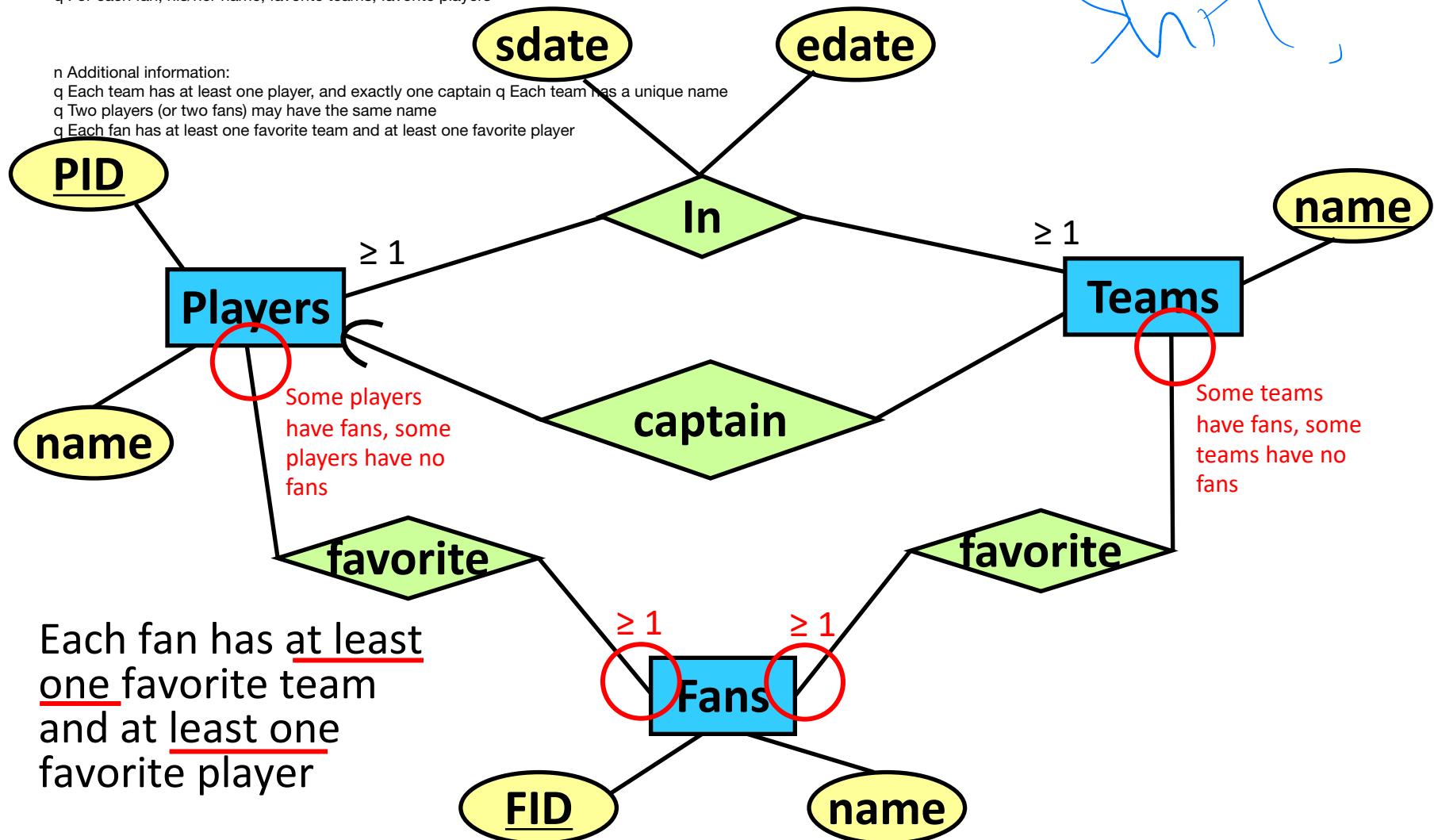
n Record info about teams, players, and their fans, including:

- q For each team, its name, its players, its team captain (who is also a player)
- q For each player, his/her name, and the history of teams on which he/she has played, including the start and ending dates for each team
- q For each fan, his/her name, favorite teams, favorite players



n Additional information:

- q Each team has at least one player, and exactly one captain q Each team has a unique name
- q Two players (or two fans) may have the same name
- q Each fan has at least one favorite team and at least one favorite player



# **SC2207/CZ2007 Introduction to Database Systems (Week 2)**

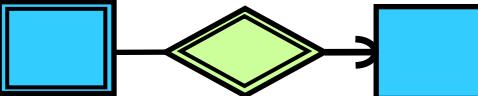
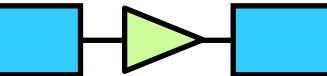
---

## **Topic 1: Entity Relationship Diagram (3)**



# So far, we learned:

- Elements of ER Diagrams

- Entities Sets 
  - Relationships 
  - Attributes 
  - Weak Entities Sets 
  - Subclasses 

- How do we design an ER Diagram for an application?

# This Lecture

- ER diagram design principles ←
- ER diagram → relational schema

# From Applications to ER Diagrams

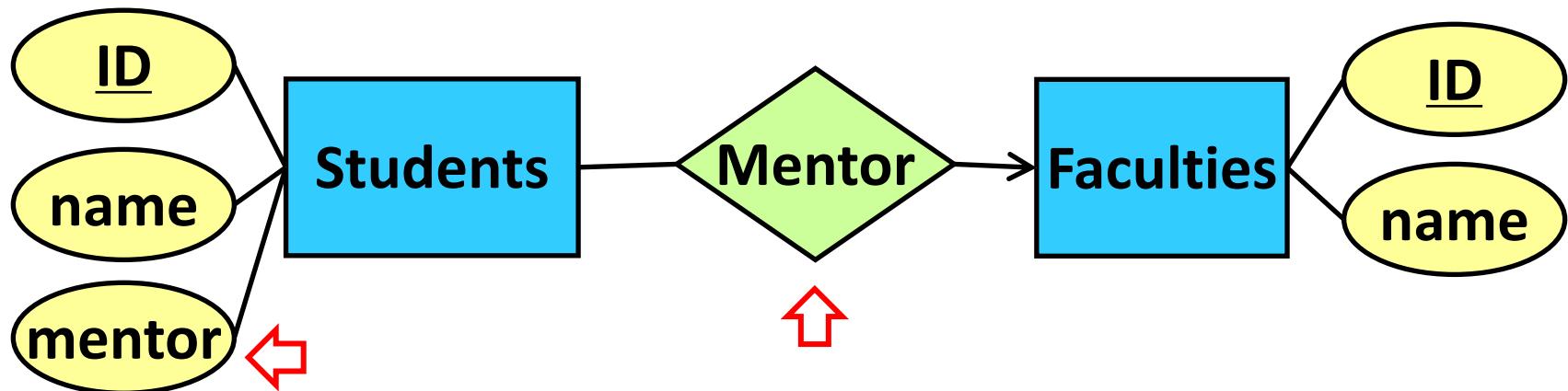
- Identify the **objects** involved in your application
- Model each type of objects as an entity set
- Identify the attributes of each entity set
- Identify the relationships among the entity sets
- Refine your design
- Example: A database for NTU
  - **Objects**: Students, Faculties, Schools, Courses...
  - **Entity sets**: Students, Faculties, Schools, Courses...
  - **Relationships**: course-enrollment, course-lecturer...

# Design Principle 1: Be Faithful

- Be faithful to the specifications of the application
- Capture the requirements as much as possible

# Design Principle 2: Avoid Redundancy

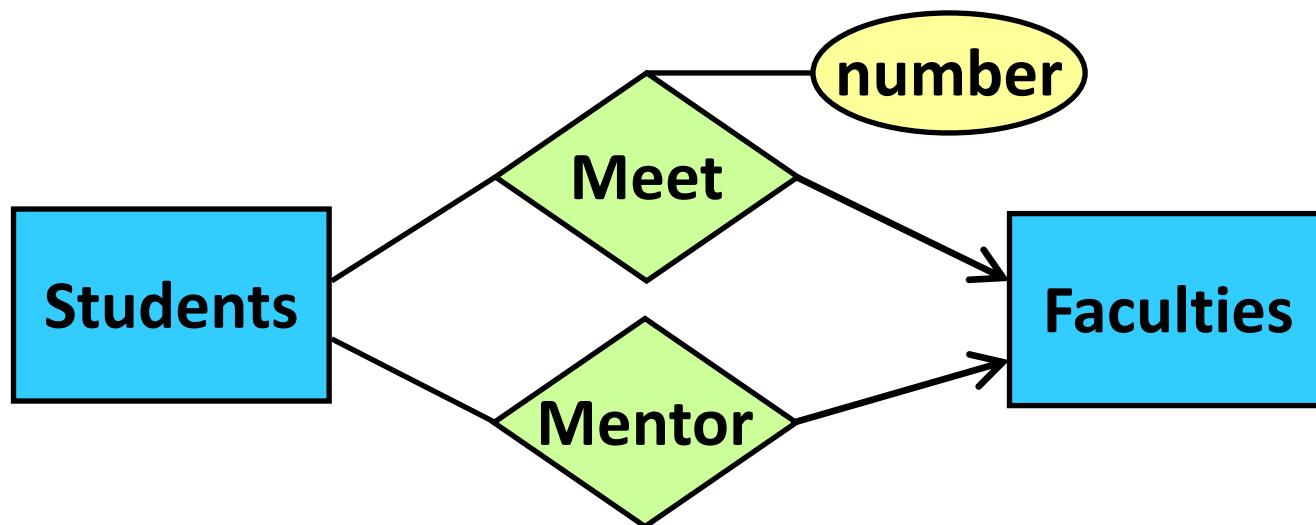
- Avoid repetition of information
- Example



- Problems that can be caused by redundancy
  - Waste of space
  - Possible inconsistency

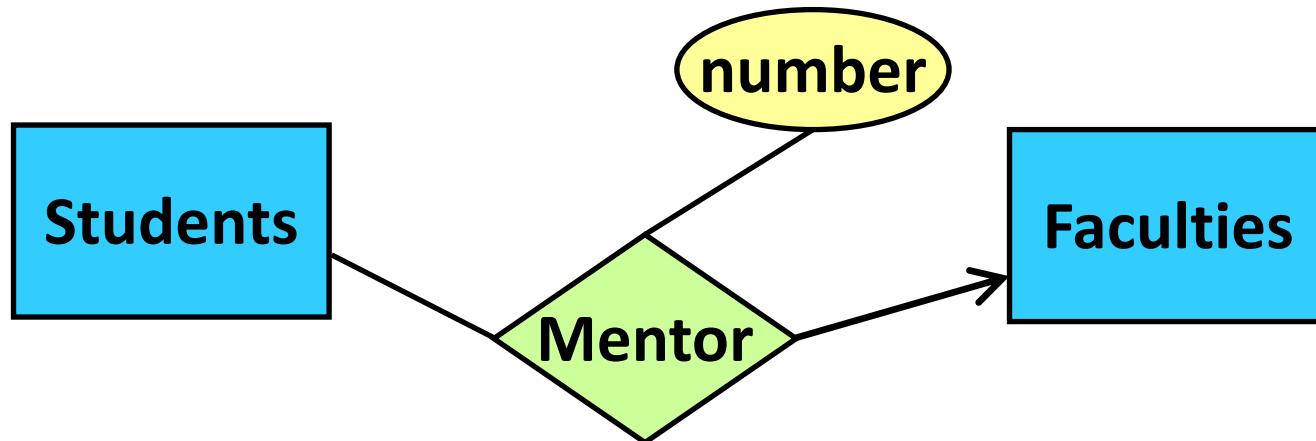
# Design Principle 3: Keep It Simple

- Each student is mentored by one faculty
- One faculty can mentor multiple students
- We also record the number of times that a mentee meets with his/her mentor
- Design below: Not wrong, but can be simplified



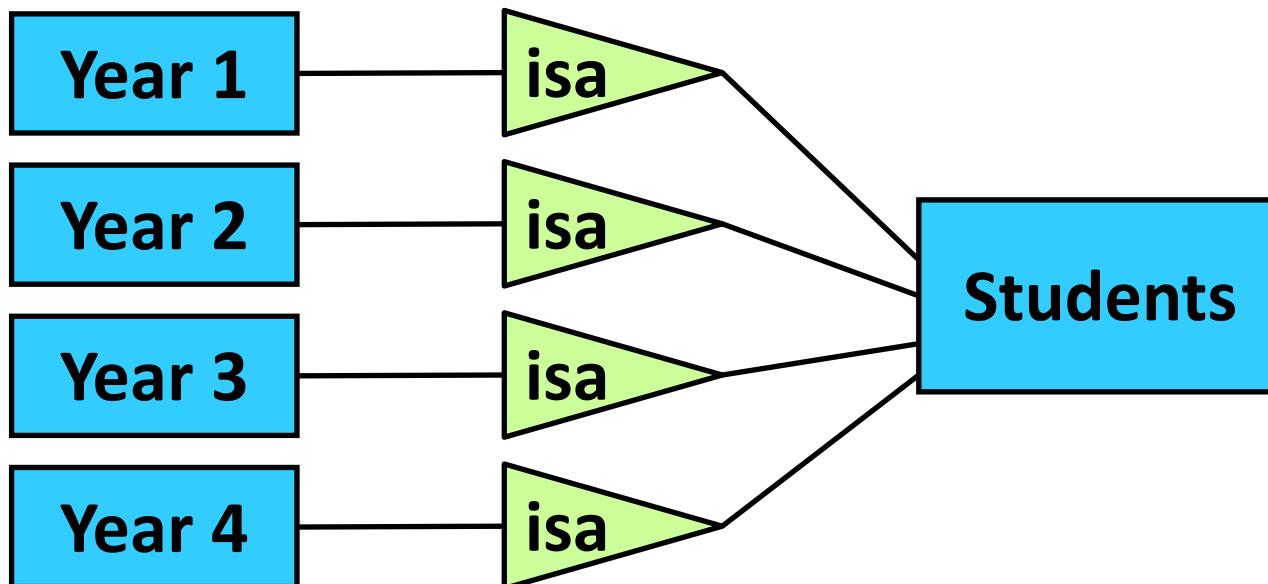
# Design Principle 3: Keep It Simple

- Each student is mentored by one faculty
- One faculty can mentor multiple students
- We also record the number of times that a mentee meets with his/her mentor
- Better Design:



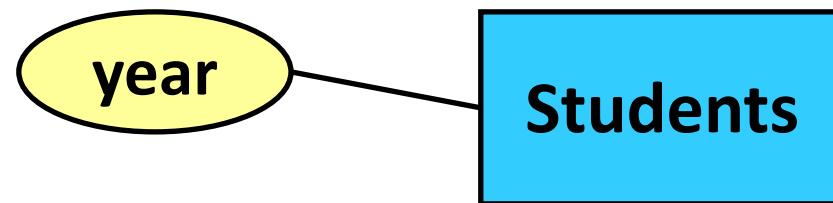
# Design Principle 3: Keep It Simple

- There are four types of students: Year 1, Year 2, Year 3, Year 4
- Design below: Not wrong, but can be simplified



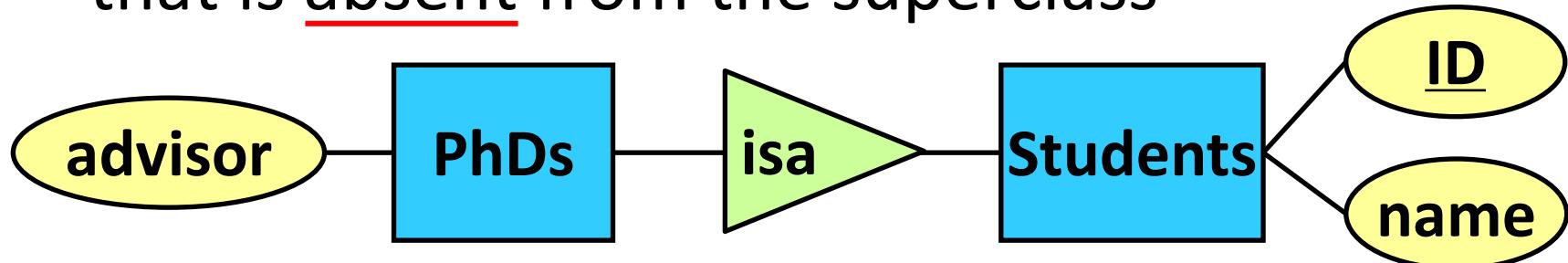
# Design Principle 3: Keep It Simple

- There are four types of students: Year 1, Year 2, Year 3, Year 4
- Better Design

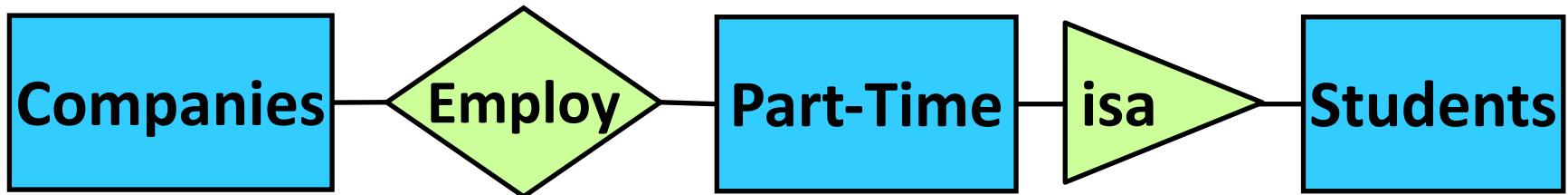


# Tips: When to Use Subclasses

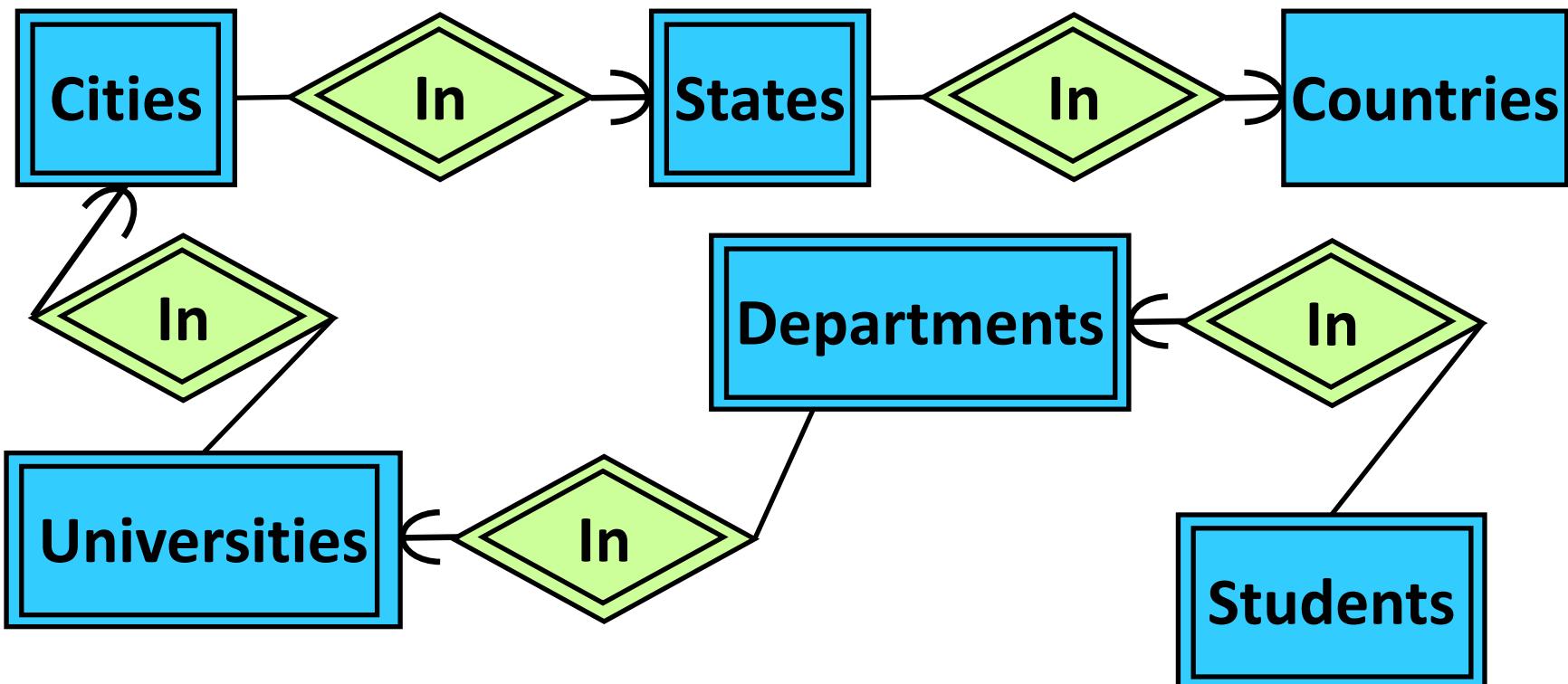
- Case 1: When a subclass has some attribute that is absent from the superclass



- Case 2: When a subclass has its own relationship with some other entity sets



# Design Principle 4: Don't Over-use Weak Entity Sets



- Too many entity sets that should not be “weak”

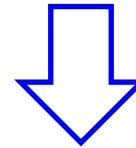
# This Lecture

- ER diagram design principles
- ER diagram → relational schema 

# Road Map

- We have discussed
  - Elements of ER Diagrams: Entity Sets, Relationships, Attributes...
  - Design principles of ER Diagrams
- These all concern the conversion below:

Real-World Application

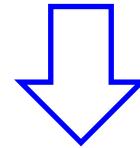


Entity-Relationship (ER) Diagrams

# Road Map

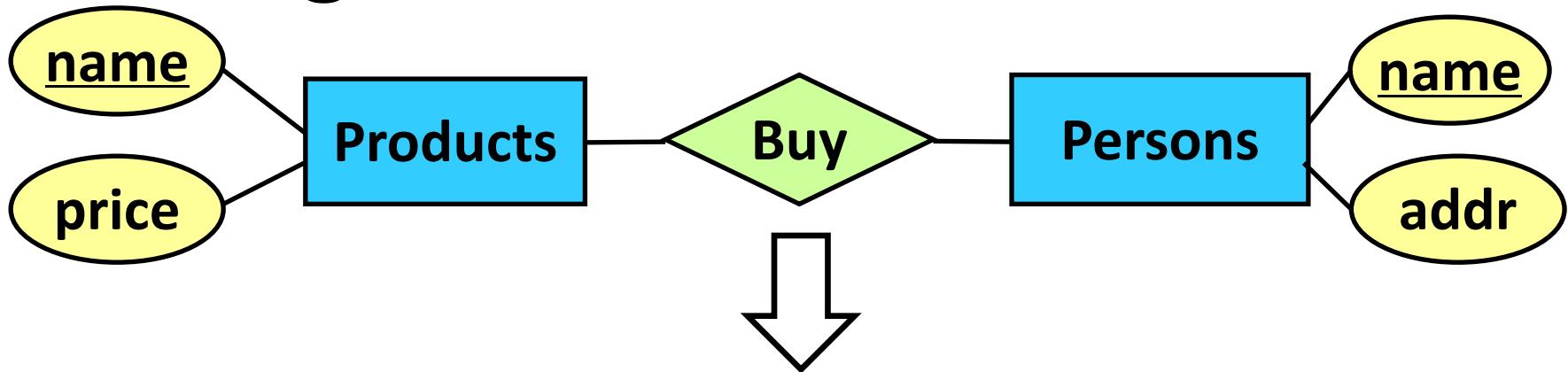
- But how do we convert the ER diagrams to a set of tables?
- We will discuss this in the next few slides

**Entity-Relationship (ER) Diagrams**



**Tables (Database Schema)**

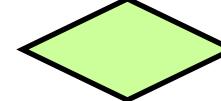
# ER Diagram → Relational Schema



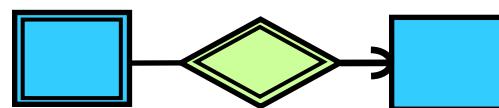
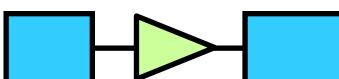
- Products (name, price)
- Persons (name, addr)
- Buy (product\_name, person\_name)
- Terminology
  - A **relation schema** = name of a table + names of its attributes
  - A **database schema** = a set of relation schemas

# ER Diagram → Relational Schema

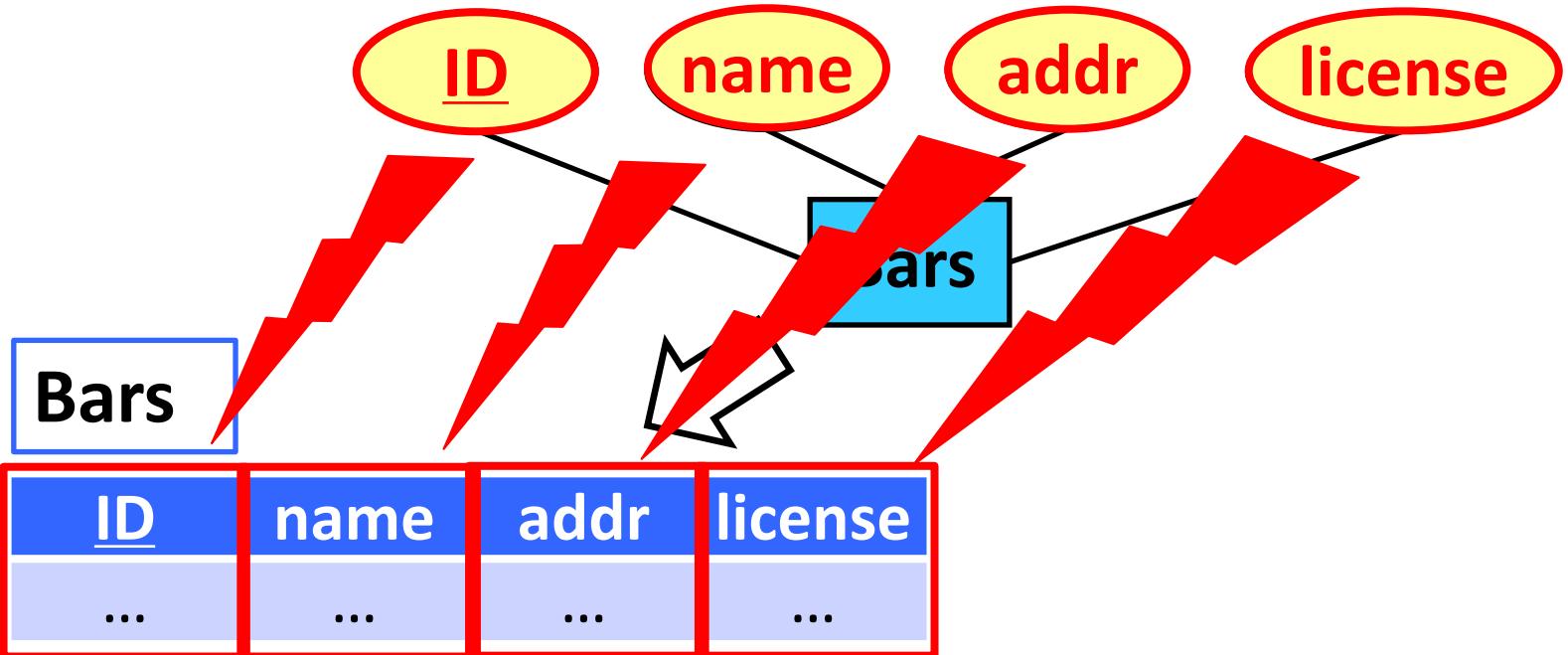
## ■ General rules:

- Each entity set  becomes a relation 
- Each many-to-many relationship  becomes a relation 

## ■ Special treatment needed for:

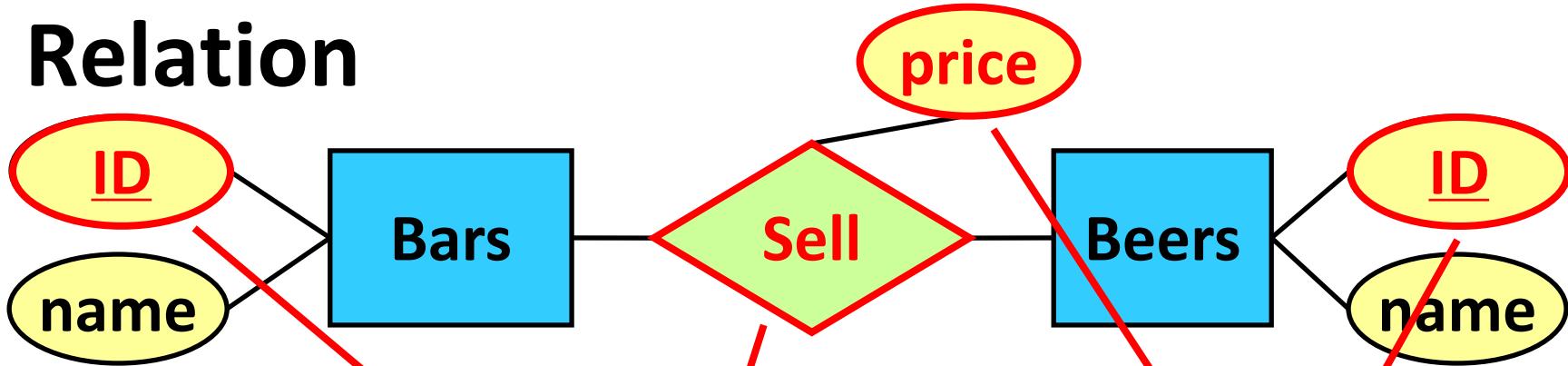
- Weak entity sets 
- Subclasses 
- Many-to-one and one-to-one relationships 

# Entity Set → Relation



- Each entity set is converted into a relation that contains all its attributes
- Key of the relation = key of the entity set

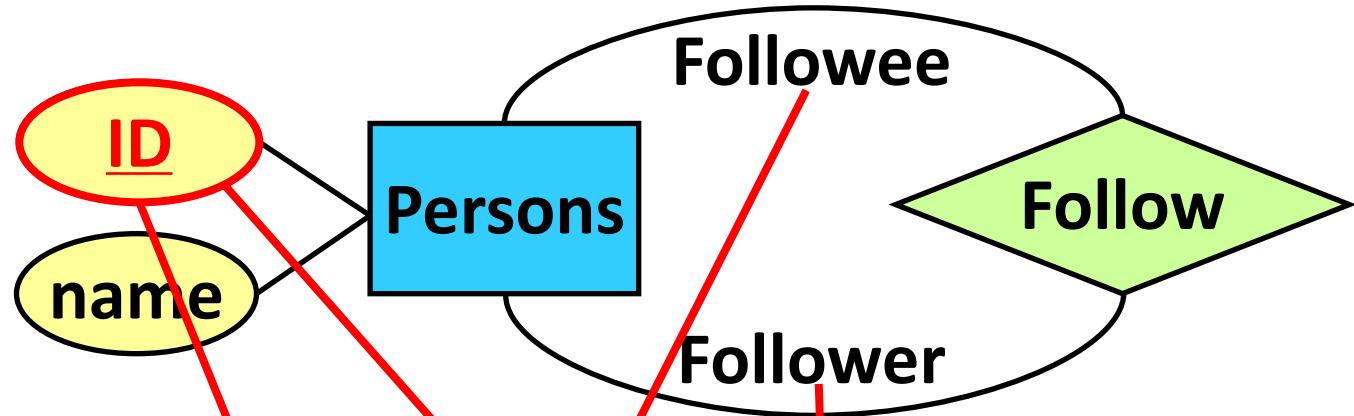
# Many-to-Many Relationship → Relation



- Converted into a relation that contains
  - all keys of the participating entity sets, and
  - the attributes of the relationship (if any)
- Key of relation = Keys of the participating entity sets

| Sell | Bars-ID | Beers-ID | price |
|------|---------|----------|-------|
| ...  | ...     | ...      | ...   |
| ...  | ...     | ...      | ...   |

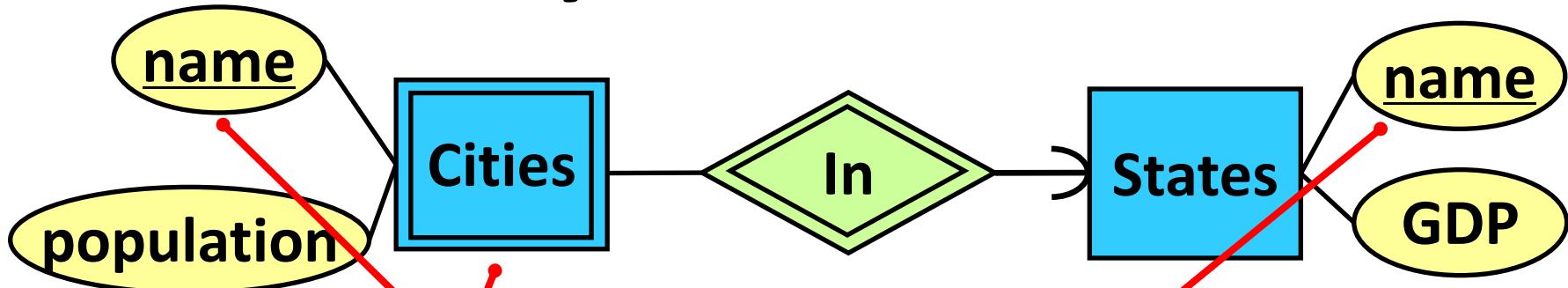
# Many-to-Many Relationship → Relation



- If an entity is involved multiple times in a relationship
  - Its key will appear in the corresponding relation multiple times
  - The key is re-named according to the corresponding role

| Follow | Followee-ID | Follower-ID |
|--------|-------------|-------------|
|        | ...         | ...         |
|        |             |             |

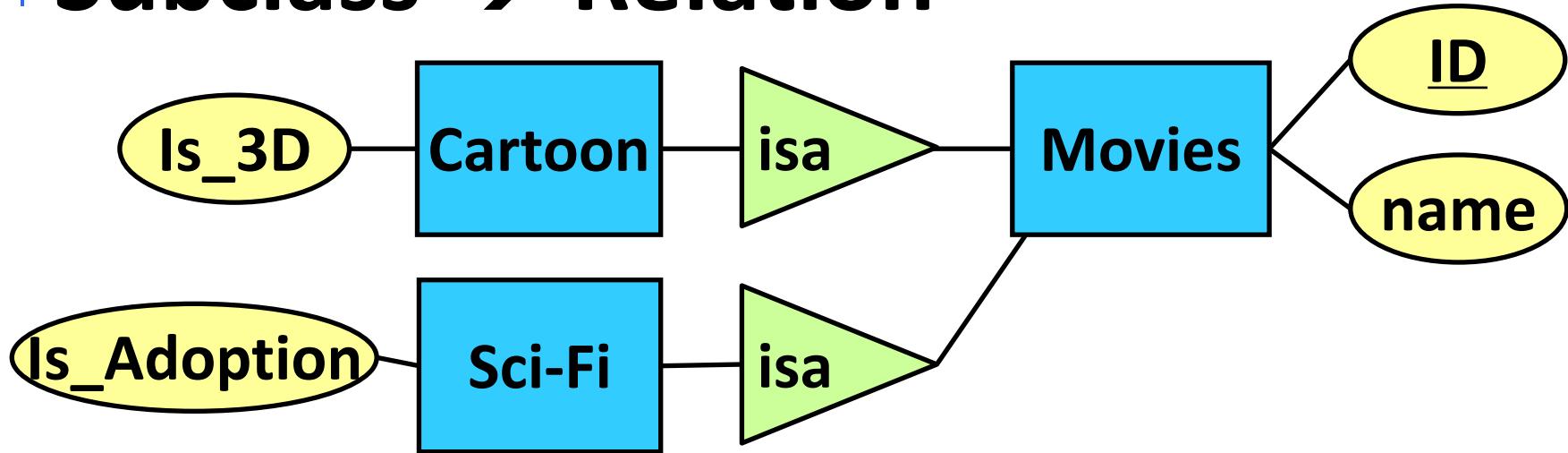
# Weak Entity Set → Relation



- Each weak entity set is converted to a relation that contains
  - all of its attributes, and
  - the key of the supporting entity set
  - attribute (if any) of supporting relationship
- The supporting relationship is ignored

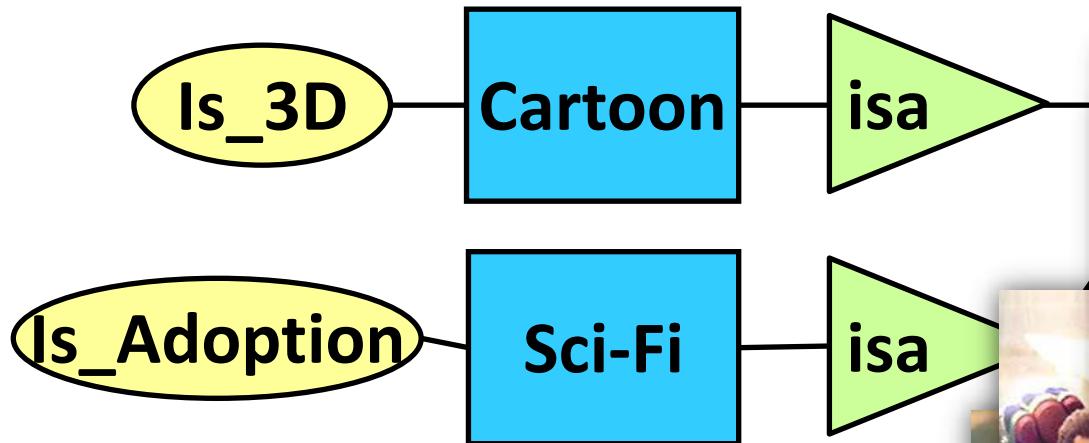
| Cities | state-name | city-name | population |
|--------|------------|-----------|------------|
| ...    | ...        | ...       | ...        |

# Subclass → Relation



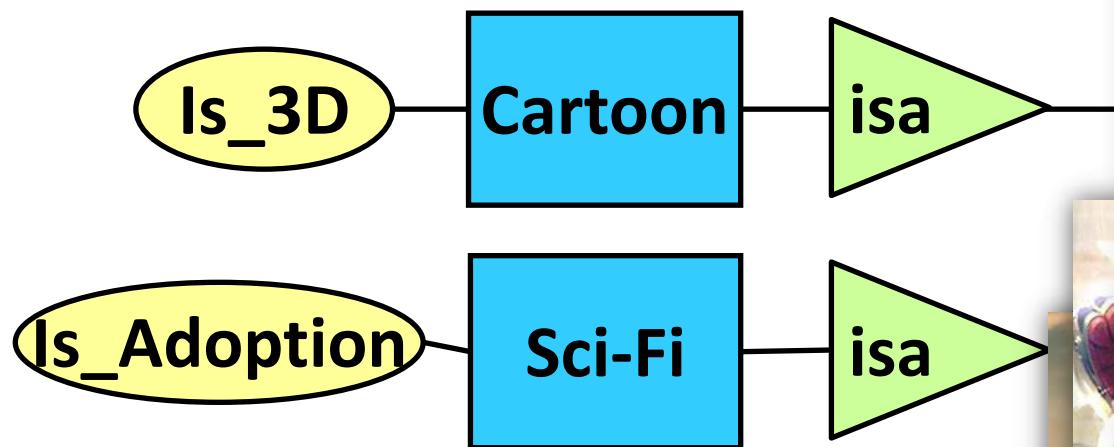
- There are three different ways
  - The ER approach **One record in multiple relations**
  - The OO approach **One record in one relation; potentially many multiple relations**
  - The NULL approach **One big relation, lots of NULLs**

# The ER approach



- One relation for each entity:
  - Movies( ID, name )
  - Cartoon( ID, Is\_3D )
  - Sci-Fi( ID, Is\_Adoption )
- A record may appear in

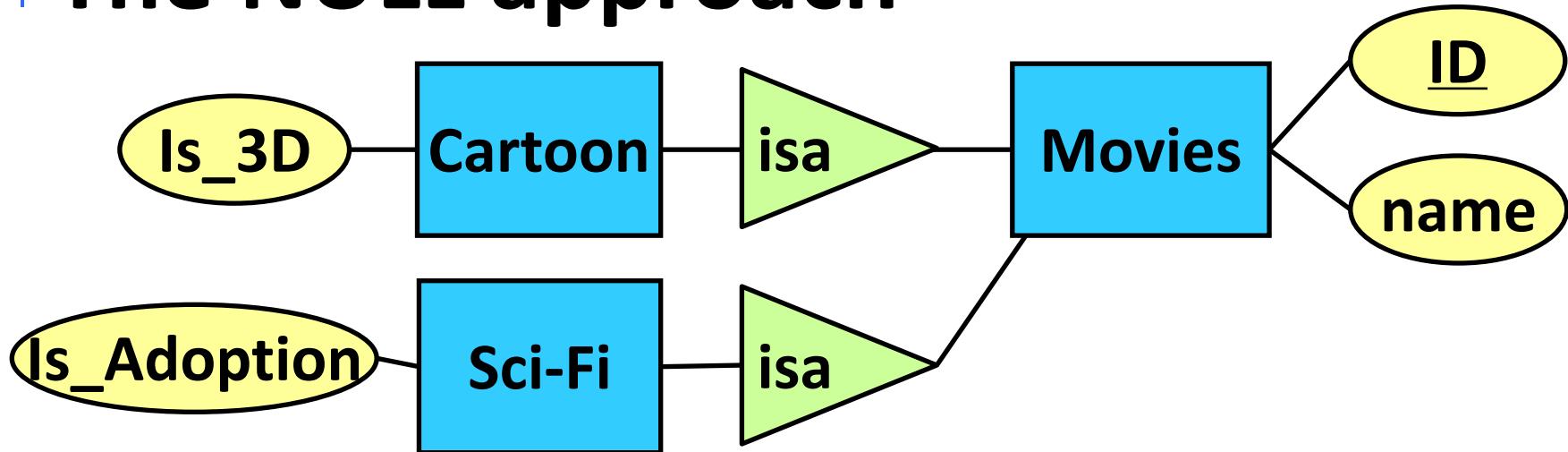
# The OO approach



- One relation for each entity set **subclass combination**
  - Movies( ID, name )
  - Cartoon( ID, name, Is\_3D )
  - Sci-Fi( ID, name, Is\_Adoption )
  - Sci-Fi-Cartoon( ID, name, Is\_3D, Is\_Adoption )
- Each record appears in only one relation



# The NULL approach



- One relation that includes everything
  - `Movies( ID, name, Is_3D, Is_Adoption )`
- For non-cartoon movies, its “Is\_3D” is `NULL`
- For non-sci-fi movies, its “Is\_Adoption” is `NULL`



# Which Approach is the Best?

- It depends
- The NULL approach
  - Advantage: Needs only one relation
  - Disadvantage: May have many NULL values
- The OO approach
  - Advantage: Good for searching subclass combinations
  - Disadvantage: May have too many tables
- The ER approach
  - A middle ground between OO and NULL

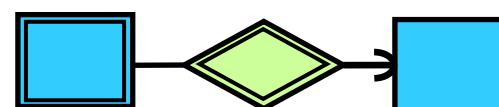
# ER Diagram → Relational Schema

- General rules:
  - Each entity set becomes a relation
  - Each many-to-many relationship becomes a relation

- Special treatment needed for:



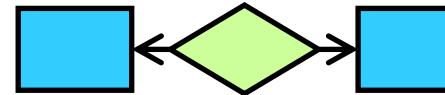
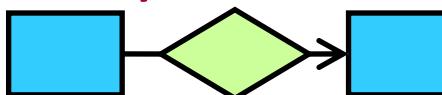
Weak entity sets



Subclasses

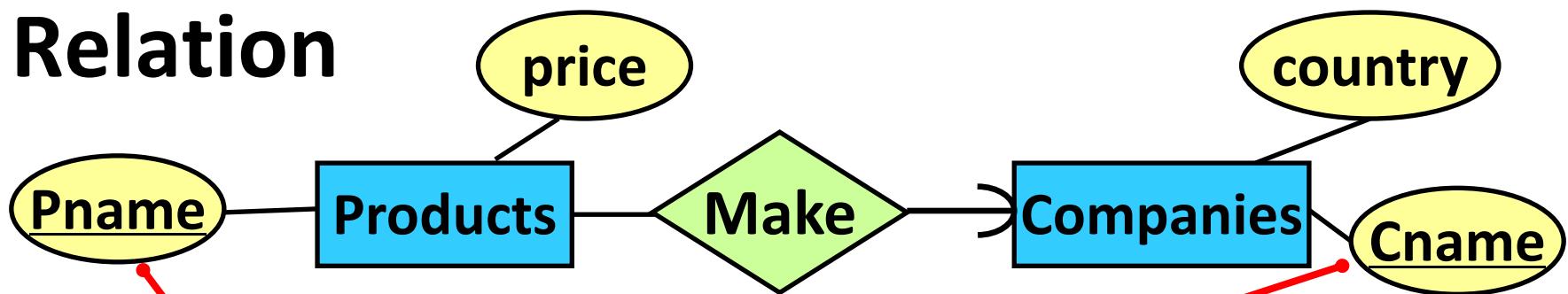


→ Many-to-one and one-to-one relationships

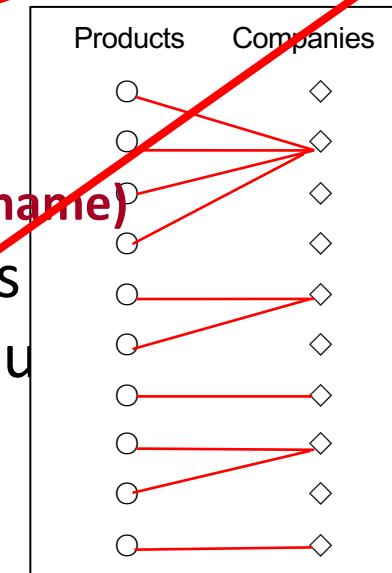


# Many-to-One Relationship →

## Relation      price

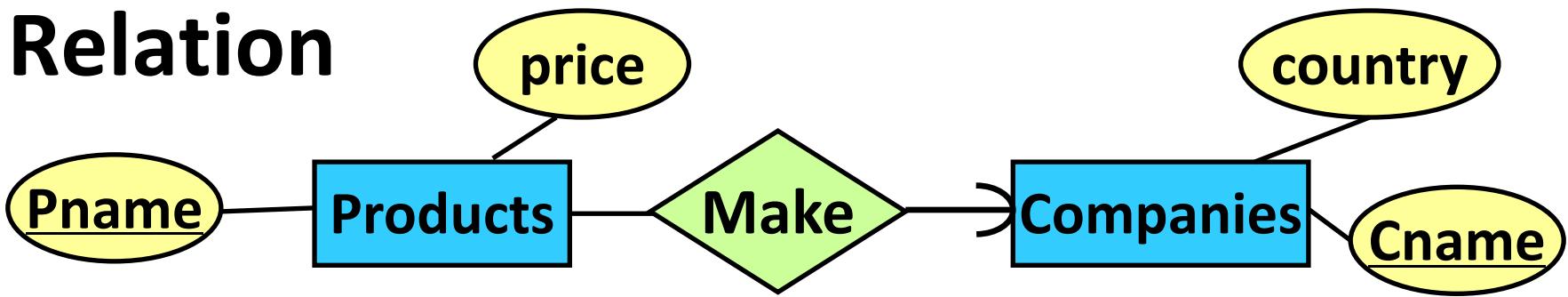


- Intuitive translation:
    - Products( Pname, price )
    - Companies( Cname, country )
    - Make( Pname, Cname ) => **Make( Pname, Cname )**
  - Observation: in “Make”, each Pname has
  - Simplification: Merge “Make” and “Produ
  - Results:
    - Products( Pname, price, Cname )
    - Companies( Cname, country )



# Many-to-One Relationship →

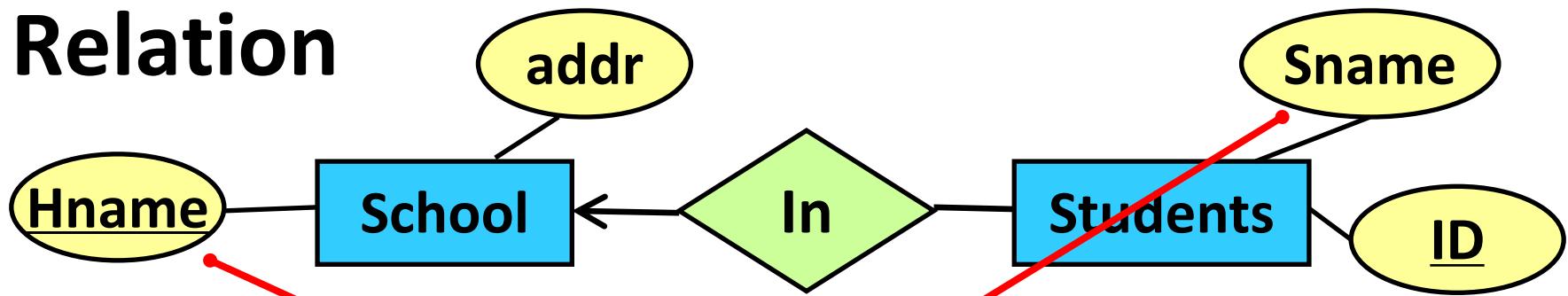
## Relation



- In general, we do not need to create a relation for a many-to-one relationship
- Instead, we only need to put the key of the “one” side into the relation of the “many” side
- If relationship has attribute, it also goes to the “many” side relation

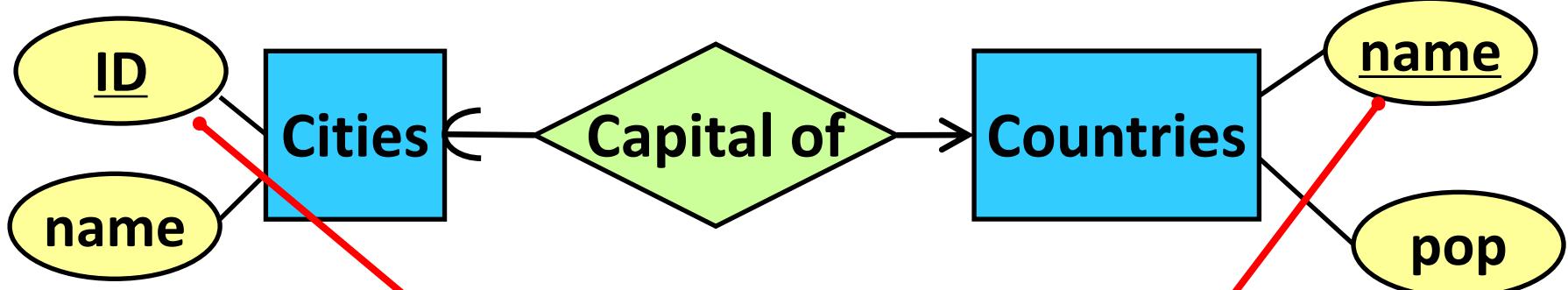
# Many-to-One Relationship →

## Relation



- Translation:
  - **School( Hname, addr )**
  - **Students( ID, Sname, Hname )**
- Only need to put the key of the “one” side into the relation of the “many” side

# One-to-One Relationship → Relation



- No need to create a relation for a one-to-one relationship
- Only need to put the key of one side into the relation of the other
- Solution 1
  - Cities( CityID, Cityname )
  - Countries( Countryname, pop, CityID)
- Solution 2
  - Cities( CityID, Cityname, Countryname )
  - Countries( Countryname, pop )

Sol 1 vs Sol 2. Which one is better?

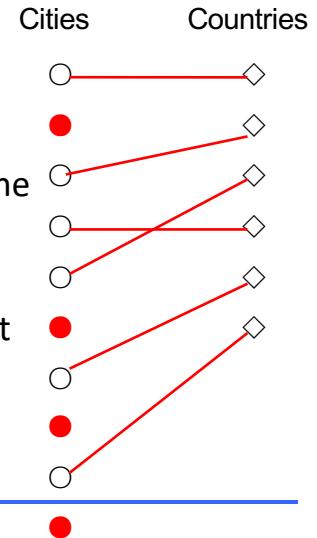
↑

Will not be null

A city can be the capital of only one country

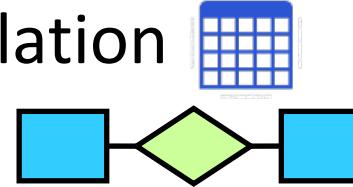
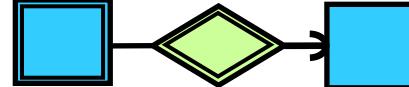
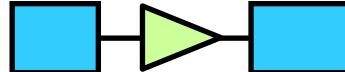
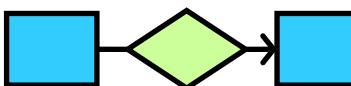
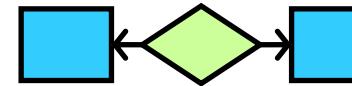
A country must have a capital

May be null



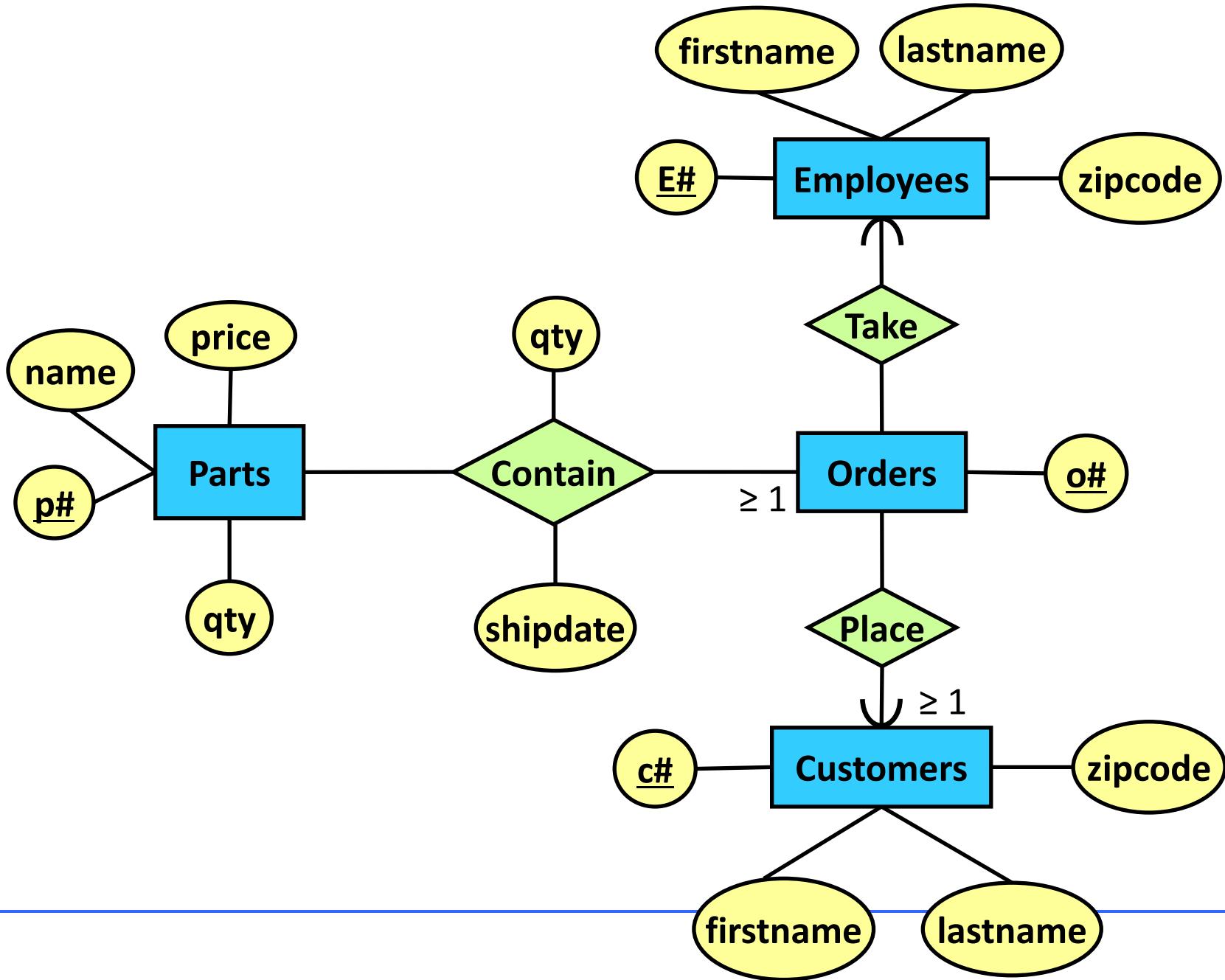
If we reverse it to many to one also correct

# Summary

- General rules:
  - Each entity set  becomes a relation 
  - Each many-to-many relationship becomes a relation 
- Special treatment needed for:
  - Weak entity sets 
  - Subclasses 
  - Many-to-one and one-to-one relationships  

# ER Diagram Design: Example

- Consider a mail order database in which employees take orders for parts from customers. The requirements are:
- Each employee is identified by a unique employee number, and has a first name, a last name, and a zip code.
- Each customer is identified by a unique customer number, and has a first name, last names, and a zip code.
- Each part being sold is identified by a unique part number. It has a part name, a price, and a quantity in stock.
- Each order placed by a customer is taken by one employee and is given a unique order number. Each order may contain certain quantities of one or more parts. The shipping date of each part is also recorded.



# Relational Schema

- Parts( p#, name, price, stock )
- Employees( e#, fname, lname, ZIP )
- Customers( c#, fname, lname, ZIP )
- Orders( o#, e#, c# ) – from two many-to-one relationships
- Contain( o#, p#, qty, shipdate ) – from many-to-many relationships

# **SC2207/CZ2007 Introduction to Database Systems (Week 2)**

---

## **Topic 2: Functional Dependencies (1)**



# This Lecture

- Data anomalies 
- Functional dependencies
- Armstrong's axioms

# Database Design

- Applications → ER diagrams → tables
- It works in general, but sometimes things may go wrong:
  - ER diagrams may not be well designed
  - Not all requirements can be represented in ER diagram
  - Conversion from ER diagrams to tables may not be properly done
- As a result, resulting tables may have various problems

# Data Anomalies

| Name  | NRIC | PhoneNumber | HomeAddress |
|-------|------|-------------|-------------|
| Alice | 1234 | 67899876    | Jurong East |
| Alice | 1234 | 83848384    | Jurong East |
| Bob   | 5678 | 98765432    | Pasir Ris   |

- Primary key of the table:  
(NRIC, PhoneNumber), one composite key
- There are several **anomalies** in the table
- First, redundancy:
  - Alice's address is duplicated, attribute values appear multiple times in different rows (columns) in table

# Data Anomalies

| Name  | NRIC | PhoneNumber | HomeAddress |
|-------|------|-------------|-------------|
| Alice | 1234 | 67899876    | Jurong East |
| Alice | 1234 | 83848384    | Jurong East |
| Bob   | 5678 | 98765432    | Pasir Ris   |

- Primary key of the table:  
(NRIC, PhoneNumber), one composite key
- Second, update anomalies:
  - We may accidentally update one of Alice's addresses, leaving the other unchanged

# Data Anomalies

| Name  | NRIC | PhoneNumber | HomeAddress |
|-------|------|-------------|-------------|
| Alice | 1234 | 67899876    | Jurong East |
| Alice | 1234 | 83848384    | Jurong East |
| Bob   | 5678 | 98765432    | Pasir Ris   |

- Primary key of the table:  
(NRIC, PhoneNumber), one composite key
- Third, deletion anomalies:
  - Bob no longer uses a phone
  - Can we remove Bob's phone number?
  - No. (Note: Primary key attributes cannot be NULL)

# Data Anomalies

| Name  | NRIC | PhoneNumber | HomeAddress |
|-------|------|-------------|-------------|
| Alice | 1234 | 67899876    | Jurong East |
| Alice | 1234 | 83848384    | Jurong East |
| Bob   | 5678 | 98765432    | Pasir Ris   |

- Primary key of the table:  
(NRIC, PhoneNumber), one composite key
- Fourth, insertion anomalies:
  - Name = Cathy, NRIC = 9394, HomeAddress = YiShun
  - Can we insert this information into the table?
  - No. (Note: Primary key attributes cannot be NULL)

# Normalization

| Name  | NRIC | PhoneNumber | HomeAddress |
|-------|------|-------------|-------------|
| Alice | 1234 | 67899876    | Jurong East |
| Alice | 1234 | 83848384    | Jurong East |
| Bob   | 5678 | 98765432    | Pasir Ris   |

- How do we get rid of those anomalies?
- **Normalize** the table (i.e., decompose it)

| Name  | NRIC | HomeAddress | NRIC | PhoneNumber |
|-------|------|-------------|------|-------------|
| Alice | 1234 | Jurong East | 1234 | 67899876    |
| Bob   | 5678 | Pasir Ris   | 1234 | 83848384    |
|       |      |             | 5678 | 98765432    |

# Effects of Normalization

| Name  | NRIC | HomeAddress |
|-------|------|-------------|
| Alice | 1234 | Jurong East |
| Bob   | 5678 | Pasir Ris   |

| NRIC | PhoneNumber |
|------|-------------|
| 1234 | 67899876    |
| 1234 | 83848384    |
| 5678 | 98765432    |

- Duplication?
  - No. (Alice's address is no longer duplicated.)
- Update anomalies?
  - No. (There is only one place where we can update the address of Alice)
- Deletion anomalies?
  - No. (We can freely delete Bob's phone number)
- Insertion anomalies?
  - No. (We can insert an individual without a phone)

# This Lecture

- Data anomalies
- Functional dependencies 
- Armstrong's axioms

# Road Map

- We will discuss
  - How to decide whether a table is good
  - If a table is not good, how to normalize it
- The plan
  - First, basic concepts (e.g., functional dependencies)
  - Second, reasoning with basic concepts
  - Finally, the real meat (e.g., normalization)

# Functional Dependencies: Intuition

| Name  | NRIC | PhoneNumber | HomeAddress |
|-------|------|-------------|-------------|
| Alice | 1234 | 67899876    | Jurong East |
| Alice | 1234 | 83848384    | Jurong East |
| Bob   | 5678 | 98765432    | Pasir Ris   |

- Why was this table bad?
- It has a lot of anomalies
- Why does it have those anomalies?
- Intuitive answer: It contains a bad combination of attributes

# Functional Dependencies: Intuition

| Name  | NRIC | PhoneNumber | HomeAddress |
|-------|------|-------------|-------------|
| Alice | 1234 | 67899876    | Jurong East |
| Alice | 1234 | 83848384    | Jurong East |
| Bob   | 5678 | 98765432    | Pasir Ris   |

- In general, how do we know whether a combination of attributes is bad?
- We need to check the **correlations** among those attributes
- What kind of correlations?
- **Functional dependencies (FD)**

# Functional Dependencies (FD)

- Consider two attributes in practice: NRIC, Name
- Given an NRIC, can we always uniquely identify the name of the person?
- Theoretically yes -- We just need help from ICA
- Given a Name, can we always uniquely identify the NRIC of the person
- In general no -- Different people can have the same name
- Therefore
  - NRIC determines Name, but not vice versa
  - Functional dependencies:  
 $\text{NRIC} \rightarrow \text{Name}$ , but not  $\text{Name} \rightarrow \text{NRIC}$

# Functional Dependencies (FD)

- Consider three attributes in practice:
  - StudentID, Name, Address, PostalCode
- Functional Dependencies:
  - $\text{StudentID} \rightarrow \text{Name, Address, PostalCode}$
  - $\text{Address} \rightarrow \text{PostalCode}$

# Formal Definition of FD

- Attributes  $A_1, A_2, \dots, A_m$ ,  $B_1, B_2, \dots, B_n$
- $A_1 A_2 \dots A_m \rightarrow B_1 B_2 \dots B_n$
- Meaning: There do not exist two objects that
  - Have the same values on  $A_1, A_2, \dots, A_m$
  - but different values on  $B_1, B_2, \dots, B_n$
- Previous example: NRIC  $\rightarrow$  Name
- Meaning: There do not exist two persons that
  - have the same values on NRIC
  - but different values on Name

| Name    | Category   | Color | Department      | Price |
|---------|------------|-------|-----------------|-------|
| Gizmo   | Gadget     | Green | Toys            | 49    |
| Tweaker | Gadget     | Black | Toys            | 99    |
| Gizmo   | Stationary | Green | Office Supplies | 59    |

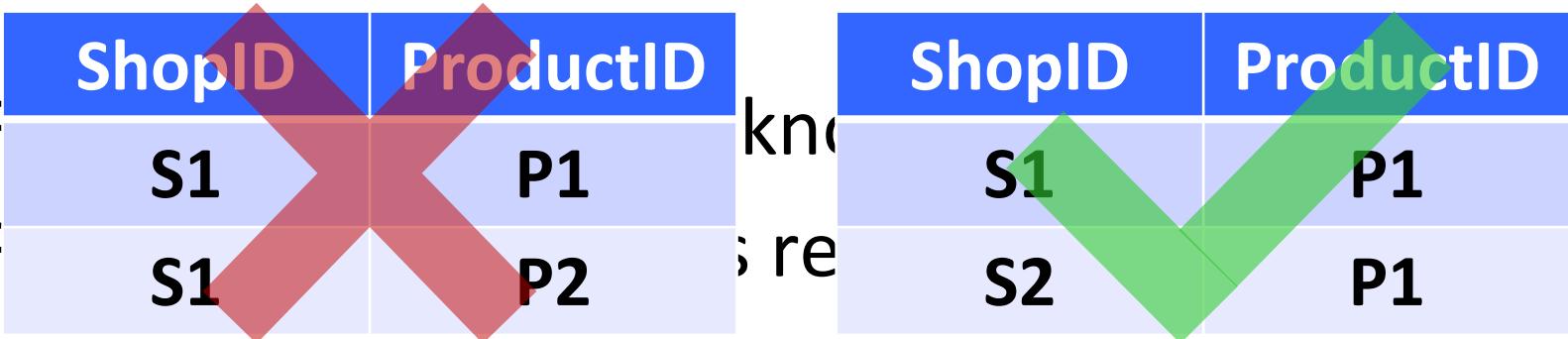
- Functional dependencies that hold on the table:

- Category  $\rightarrow$  Department ✓ ✓
- Category, Color  $\rightarrow$  Price ✓ ✓
- Price  $\rightarrow$  Color ✓ ✓
- Name  $\rightarrow$  Color ✓ ✓
- Department, Category  $\rightarrow$  Name ✗ ✗
- Color, Department  $\rightarrow$  Name, Price, Category ✓ ✓

# Where Do FDs Come From?

- From common sense, knowledge in real world
- From application's requirements
- Example
  - Purchase( CustomerID, ProductID, ShopID, Price, Date )
  - Requirement: Each shop can sell at most one product
  - FD implied:  $\text{ShopID} \rightarrow \text{ProductID}$

# Where Do FDs Come From?

- 

known  
ShopID | ProductID  
S1 | P1  
S1 | P2

ShopID | ProductID  
S1 | P1  
S2 | P1
- **Example**
  - Purchase( CustomerID, ProductID, ShopID, Price, Date )
  - Requirement: Each shop can sell at most one product
  - FD implied:  $\text{ShopID} \rightarrow \text{ProductID}$

# Where Do FDs Come From?

- Example
  - Purchase( CustomerID, ProductID, ShopID, Price, Date )
  - Requirement: No two customers buy the same product
  - FD implied:  $\text{ProductID} \rightarrow \text{CustomerID}$

# Where Do FDs Come From?

## ■ Example

- Purchase( CustomerID, ProductID, ShopID, Price, Date )
- Requirement: No two customers buy the same product
- FD implied:  $\text{ProductID} \rightarrow \text{CustomerID}$

| CustomerID | ProductID |
|------------|-----------|
| C1         | P1        |
| C2         | P1        |

| CustomerID | ProductID |
|------------|-----------|
| C1         | P1        |
| C1         | P2        |

# Where Do FDs Come From?

## ■ Example

- Purchase( CustomerID, ProductID, ShopID, Price, Date )
- Requirement: No shop will sell the same product to the same customer on the same date at two different prices
- FD implied:  
 $\text{CustomerID}, \text{ProductID}, \text{ShopID}, \text{Date} \rightarrow \text{Price}$

# Where Do FDs Come From?

- E:

| CustomerID | ProductID | ShopID | Date | Price |
|------------|-----------|--------|------|-------|
| C1         | P1        | S1     | D1   | 99    |
| C1         | P1        | S1     | D1   | 33    |

  - Purchase( CustomerID, ProductID, ShopID, Price,
  - | CustomerID | ProductID | ShopID | Date | Price |
|------------|-----------|--------|------|-------|
| C1         | P1        | S1     | D1   | 99    |
| C2         | P1        | S1     | D1   | 33    |

    - different prices
    - FD implied:  
 $\text{CustomerID}, \text{ProductID}, \text{ShopID}, \text{Date} \rightarrow \text{Price}$

# Roadmap

- To decide whether or not a table is good, we need to examine the relationships among attributes, i.e., the FDs
- Now we know what FDs are, and where they are from
- How do we use FDs to check whether a table is good?
- We need to learn how to **reason** with FDs

# This Lecture

- Data anomalies
- Functional dependencies
- Armstrong's axioms 

# Reasoning with FDs

- How to reason with FDs
- Example:
  - Given:  
 $\text{NRIC} \rightarrow \text{Address}$ ,       $\text{Address} \rightarrow \text{PostalCode}$
  - We have:  $\text{NRIC} \rightarrow \text{PostalCode}$
- In general
  - Given  $A \rightarrow B$ ,  $B \rightarrow C$
  - We always have  $A \rightarrow C$
- We will discuss how we can do this kind of derivations

# Reasoning with FDs

- Why do we care about such derivations?
- Answer: it is needed in deciding whether a table is “bad” or not
- Intuitive explanation:
  - Suppose that  $A \rightarrow C$  is an FD that makes a table “bad”
  - But we are only given  $A \rightarrow B$  and  $B \rightarrow C$
  - If we do not know that  $A \rightarrow C$  is implied by  $A \rightarrow B$  and  $B \rightarrow C$ , then we may misjudge the table to be a “good” one

# Reasoning with FDs

## ■ Armstrong's Axioms

- Three axioms for FD reasoning
- Easy to understand, but not easy to apply

# Armstrong's Axioms

- Axiom of Reflexivity
  - A set of attributes → A **subset** of the attributes
- Example
  - NRIC, Name → NRIC
  - StudentID, Name, Age → Name, Age
  - ABCD → ABC
  - ABCD → BCD
  - ABCD → AD

# Armstrong's Axioms

- Axiom of Augmentation
  - Given  $A \rightarrow B$
  - We always have  $AC \rightarrow BC$ , for any C
- Example
  - If NRIC  $\rightarrow$  Name
  - Then NRIC, Age  $\rightarrow$  Name, Age
  - and NRIC, Salary, Weight  $\rightarrow$  Name, Salary, Weight
  - and NRIC, Addr, Postal  $\rightarrow$  Name, Addr, Postal

# Armstrong's Axioms

- Axiom of Transitivity
  - Given  $A \rightarrow B$  and  $B \rightarrow C$
  - We always have  $A \rightarrow C$
- Example
  - If  $\text{NRIC} \rightarrow \text{Addr}$ , and  $\text{Addr} \rightarrow \text{Postal}$
  - Then  $\text{NRIC} \rightarrow \text{Postal}$

# Reasoning with FDs

- Given  $A \rightarrow B$ ,  $BC \rightarrow D$
- Can you prove that  $AC \rightarrow D$ ?
- Proof
  - Given  $A \rightarrow B$ , we have  $AC \rightarrow BC$  (Augmentation)
  - Given  $AC \rightarrow BC$  and  $BC \rightarrow D$ , we have  $AC \rightarrow D$  (Transitivity)

# Reasoning with FDs

- Given  $A \rightarrow B$ ,  $D \rightarrow C$
- Can you prove that  $AD \rightarrow BC$ ?
- Proof
  - Given  $A \rightarrow B$ , we have  $AD \rightarrow BD$  (Augmentation)
  - Given  $AD \rightarrow BD$ , we have  $AD \rightarrow B$  (Reflexivity)
  - Given  $D \rightarrow C$ , we have  $AD \rightarrow AC$  (Augmentation)
  - Given  $AD \rightarrow AC$ , we have  $AD \rightarrow C$  (Reflexivity)
  - In other words,  $AD$  decides  $B$  and  $C$
  - Therefore,  $AD \rightarrow BC$

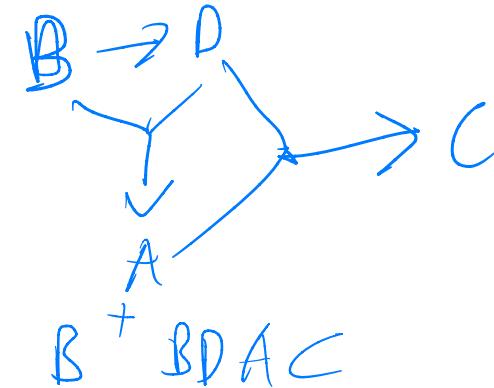
# Reasoning with FDs

- Given  $A \rightarrow C$ ,  $AC \rightarrow D$ ,  $AD \rightarrow B$
- Can you prove that  $A \rightarrow B$ ? *Yes*
- Proof
  - Given  $A \rightarrow C$ , we have  $A \rightarrow AC$  (Augmentation)
  - Given  $A \rightarrow AC$  and  $AC \rightarrow D$ , we have  $A \rightarrow D$  (Transitivity)
  - Given  $A \rightarrow D$ , we have  $A \rightarrow AD$  (Augmentation)
  - Given  $A \rightarrow AD$  and  $AD \rightarrow B$ , we have  $A \rightarrow B$  (Transitivity)

$$\begin{array}{l} A \rightarrow C \quad AC \rightarrow A \text{ reflexivity} \\ AC \rightarrow D \quad A \rightarrow P \\ AD \rightarrow D \rightarrow D \rightarrow B \\ A \rightarrow D \end{array}$$

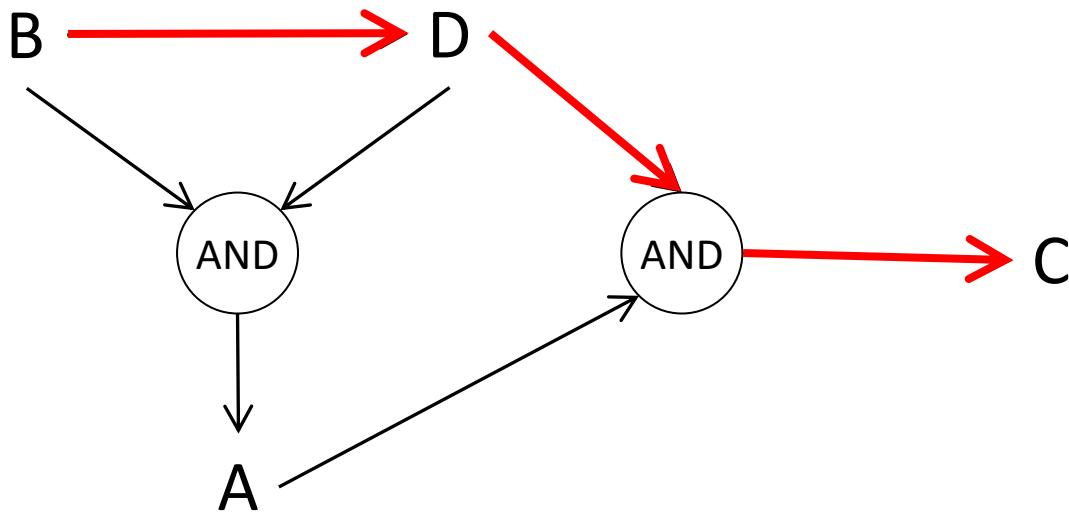
# Reasoning with FD

- Four attributes: A, B, C, D
- Given:  $B \rightarrow D$ ,  $DB \rightarrow A$ ,  $AD \rightarrow C$
- Can you prove  $B \rightarrow C$ ?
- Doable with Armstrong's axioms, but troublesome
- We will discuss a more convenient approach



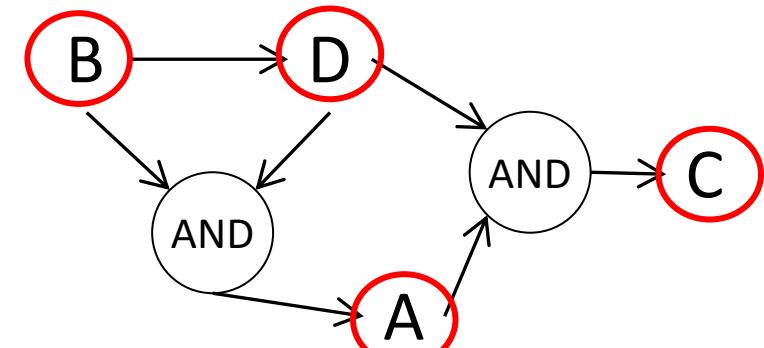
# An Intuitive Solution

- Four attributes: A, B, C, D
- Given:  $B \rightarrow D$ ,  $DB \rightarrow A$ ,  $AD \rightarrow C$
- Can you prove  $B \rightarrow C$ ?



# Steps of the Intuitive Solution

- Four attributes: A, B, C, D
- Given:  $B \rightarrow D$ ,  $DB \rightarrow A$ ,  $AD \rightarrow C$
- Can you prove  $B \rightarrow C$ ?

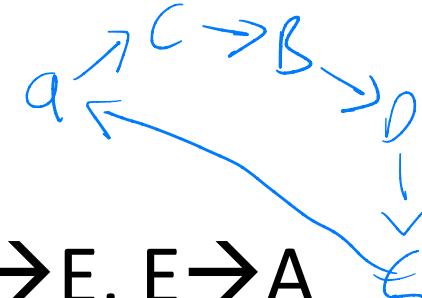


- 
- First, activate B
    - Activated set = { B }
  - Second, activate whatever B can activate
    - Activated set = { B, D }, since  $B \rightarrow D$
  - Third, use all activated elements to activate more
    - Activated set = { B, D, A }, since  $DB \rightarrow A$
  - Repeat the third step, until no more activation is possible
    - Activated set = { B, D, A, C }, since  $AD \rightarrow C$ ; done

# Exercise

$$A \rightarrow C$$

$$C \rightarrow ABE$$

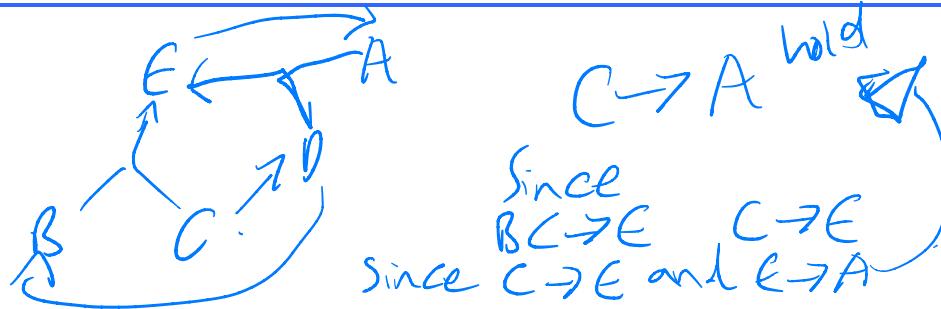


- Given:  $A \rightarrow C$ ,  $C \rightarrow B$ ,  $B \rightarrow D$ ,  $D \rightarrow E$ ,  $E \rightarrow A$
- Can you prove  $C \rightarrow ABE$ ?
- We start with  $\{C\}$
- Since  $C \rightarrow B$ , we have  $\{C, B\}$
- Since  $B \rightarrow D$ , we have  $\{C, B, D\}$
- Since  $D \rightarrow E$ , we have  $\{C, B, D, E\}$
- Since  $E \rightarrow A$ , we have  $\{C, B, D, E, A\}$
- A, B, E are all in the set, so  $C \rightarrow ABE$  holds

$$\{C\}^+ = \{C, B, D, E, A\}$$

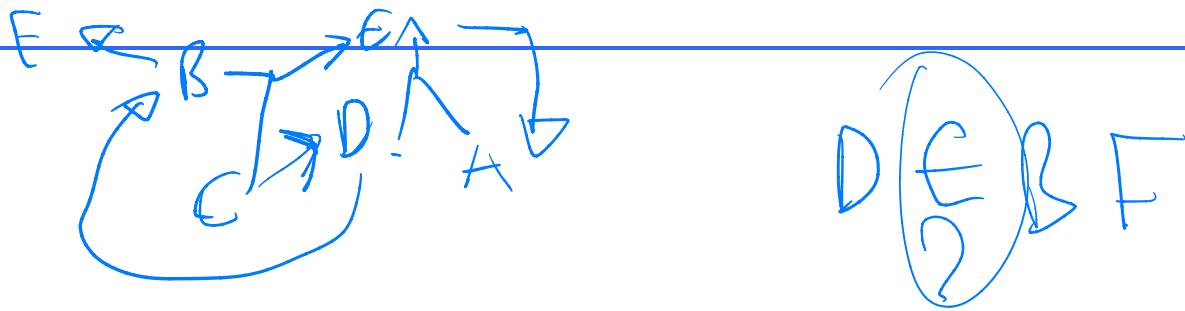
C

# Exercise



- Given:  $C \rightarrow D$ ,  $AD \rightarrow E$ ,  $BC \rightarrow E$ ,  $E \rightarrow A$ ,  $D \rightarrow B$
- Can you prove  $C \rightarrow A$ ?
- We start with  $\{C\}$
- Since  $C \rightarrow D$ , we have  $\{C, D\}$
- Since  $D \rightarrow B$ , we have  $\{C, D, B\}$
- Since  $BC \rightarrow E$ , we have  $\{C, D, B, E\}$
- Since  $E \rightarrow A$ , we have  $\{C, D, B, E, A\}$
- A is in the set, so  $C \rightarrow A$  holds

# Exercise



- Given:  $C \rightarrow D$ ,  $AD \rightarrow E$ ,  $BC \rightarrow E$ ,  $E \rightarrow A$ ,  $D \rightarrow B$ ,  $B \rightarrow F$
- Can you prove  $D \rightarrow C$ ?  $\nwarrow$
- We start with  $\{D\}$
- Since  $D \rightarrow B$ , we have  $\{D, B\}$
- Since  $B \rightarrow F$ , we have  $\{D, B, F\}$
- What else?
- No more.
- $\{D, B, F\}$  is all what can be decided by D
- We refer to  $\{D, B, F\}$  as the **closure** of D

# **SC2207/CZ2007 Introduction to Database Systems (Week 3)**

---

## **Topic 2: Functional Dependencies (2)**



# This Lecture

- Closure ←
- Keys
- Finding keys
- Normal forms

# Closure

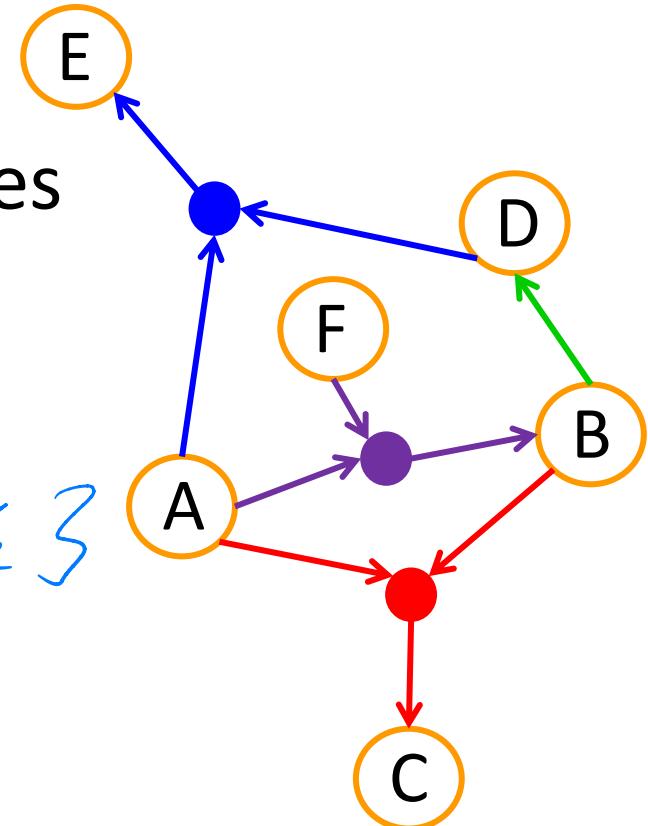
- Let  $S = \{A_1, A_2, \dots, A_n\}$  be a set  $S$  of  $n$  attributes
- Closure of  $S$  is the set of attributes that can be determined by  $A_1, A_2, \dots, A_n$
- Notation:  $\{A_1, A_2, \dots, A_n\}^+$
- Example
  - Given  $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$
  - $\{A\}^+ = \{A, B, C, D, E\}$
  - $\{B\}^+ = \{B, C, D, E\}$
  - $\{D\}^+ = \{D, E\}$
  - $\{E\}^+ = \{E\}$

$$\begin{aligned}A &= \{A, B, C, D, E\} \\B &= \{B, C, D, E\} \\C &= \{C, D, E\} \\D &= \{D, E\} \\E &= \{E\}\end{aligned}$$

# Exercise

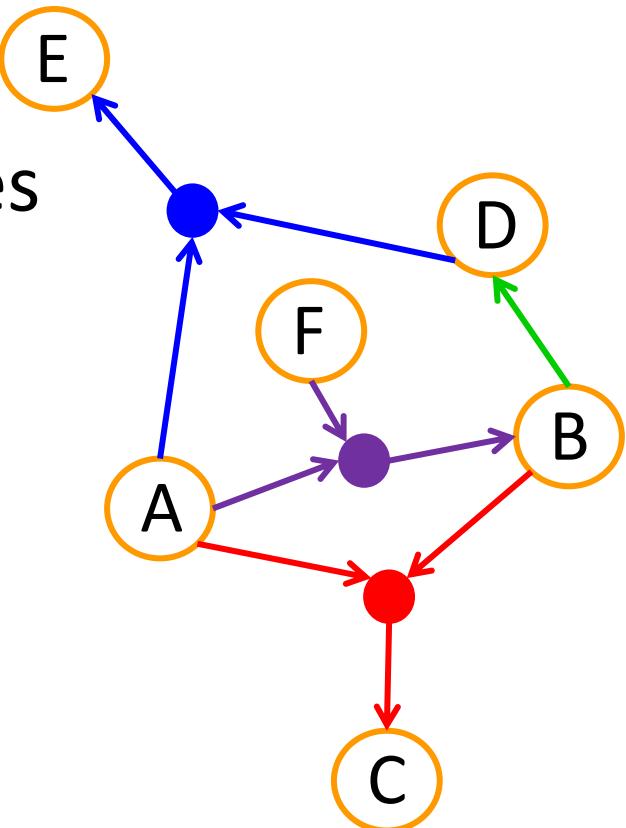
- A table with six attributes A, B, C, D, E, F
- $AB \rightarrow C$ ,  $AD \rightarrow E$ ,  $B \rightarrow D$ ,  $AF \rightarrow B$
- Compute the following closures

- $\{B, C\}^+ = \{B, C, D\}$
- $\{A, B\}^+ = \{A, B, C, D, E\}$
- $\{A, F\}^+ = \{A, F, B, C, D, E\}$



# Exercise

- A table with six attributes A, B, C, D, E, F
- $AB \rightarrow C$ ,  $AD \rightarrow E$ ,  $B \rightarrow D$ ,  $AF \rightarrow B$
- Compute the following closures
  - $\{B, C\}^+ = \{B, C, D\}$
  - $\{A, B\}^+ = \{A, B, C, D, E\}$
  - $\{A, F\}^+ = \{A, B, C, D, E, F\}$



# Closure & FD

- To prove that  $X \rightarrow Y$  holds, we only need to show that  $\{X\}^+$  contains  $Y$
- $AB \rightarrow C, AD \rightarrow E, B \rightarrow D, AF \rightarrow B$
- Prove that  $AF \rightarrow D$
- $\{AF\}^+ = \{AFBCDE\}$ , which contains  $D$
- Therefore,  $AF \rightarrow D$  holds

# Closure & FD

- To prove that  $X \rightarrow Y$  does not hold, we only need to show that  $\{X\}^+$  does not contain  $Y$
- $AB \rightarrow C, AD \rightarrow E, B \rightarrow D, AF \rightarrow B$
- Prove that  $AD \rightarrow F$  does not hold
- $\{AD\}^+ = \{ADE\}$ , which does not contain  $F$
- Therefore,  $AD \rightarrow F$  does not hold

# This Lecture

- Closure
- Keys 
- Finding keys
- Normal forms

# Roadmap

- Now we know what a closure is
- We discuss the usage of closure for finding the **keys** in a table
  - This is a necessary step for checking whether a table is good or not
- Concepts
  - Superkeys
  - Keys
  - Candidate keys
  - Primary keys / Secondary keys

Notion of **keys** in a table



**keys** in a entity set

# Superkeys of a Table

| Name  | NRIC | Postal | Address     |
|-------|------|--------|-------------|
| Alice | 1234 | 939450 | Jurong East |
| Bob   | 5678 | 234122 | Pasir Ris   |
| Cathy | 3576 | 420923 | Yishun      |

- Definition: A set of attributes in a table that determines all other attributes
- Example:
  - {NRIC} is a superkey (happens to be minimal)
  - Since NRIC → Name, Postal, Address
  - {NRIC, Name} is a superkey (not minimal)
  - Since {NRIC, Name} → Postal, Address

# Keys of a Table

| Name  | <u>NRIC</u> | Postal | Address     |
|-------|-------------|--------|-------------|
| Alice | 1234        | 939450 | Jurong East |
| Bob   | 5678        | 234122 | Pasir Ris   |
| Cathy | 3576        | 420923 | Yishun      |

- Definition: A superkey that is minimal
- i.e., if we remove any attribute from the superkey, it will not be a superkey anymore
- Example:
  - {NRIC} is a superkey (happens to be minimal)
  - Since NRIC → Name, Postal, Address
  - {NRIC, Name} is a superkey
  - Since {NRIC, Name} → Postal, Address
  - NRIC is a key, but {NRIC, Name} is not a key, but a superkey

# Keys of a Table

| Name  | <u>NRIC</u> | Postal | Address     |
|-------|-------------|--------|-------------|
| Alice | 1234        | 939450 | Jurong East |
| Bob   | 5678        | 234122 | Pasir Ris   |
| Cathy | 3576        | 420923 | Yishun      |

- Note: Not to be confused with the keys of entity sets because keys in entity sets need not be minimal.

# Candidate Keys

| Name  | NRIC | StudentID | Postal | Address     |
|-------|------|-----------|--------|-------------|
| Alice | 1234 | 1         | 939450 | Jurong East |
| Bob   | 5678 | 2         | 234122 | Pasir Ris   |
| Cathy | 3576 | 3         | 420923 | Yishun      |

- A table may have multiple keys
- In that case, each key is referred to as a candidate key
- Example:
  - {NRIC} is a key
  - Since NRIC → Name, StudentID, Postal, Address
  - {StudentID} is a key
  - Since StudentID → Name, NRIC, Postal, Address
  - Both {NRIC} and {StudentID} are candidate keys

# Primary and Secondary Keys

| Name  | NRIC | StudentID | Postal | Address     |
|-------|------|-----------|--------|-------------|
| Alice | 1234 | 1         | 939450 | Jurong East |
| Bob   | 5678 | 2         | 234122 | Pasir Ris   |
| Cathy | 3576 | 3         | 420923 | Yishun      |

- When a table have multiple keys ...
- We choose one of them as the primary key
- The others are referred to as secondary keys
- Example:
  - {NRIC} is a key
  - {StudentID} is a key
  - If we choose {NRIC} as the primary key
  - Then {StudentID} is the secondary key

# Summary

- Superkeys
  - A set of attributes that determines all other attributes in a table
- Keys
  - A minimal set of attributes that determines all other attributes in a table
- Example
  - $R(A, B, C, D)$
  - Given:  $A \rightarrow BCD$ ,  $BC \rightarrow A$
  - $A$  is a key and a superkey
  - $BC$  is also a key and a superkey
  - $AB$  is not a key; it is only a superkey

# This Lecture

- Closure
- Keys
- Finding keys 
- Normal forms

# Finding the Keys

- To check whether a table is “good”, we need to find the keys of the table
- How do we do that?
- Use functional dependencies (FDs) and closures

# Example

$\{A\}^+ = \{A, B, C\}$

- Definition of a Key: A minimal set of attributes that determines all other attributes
- A table  $R(A, B, C)$
- FDs given:  $A \rightarrow B, B \rightarrow C$
- Is  $A$  a key? Yes
- $\{A\}^+ = \{ABC\}$ , i.e.,  $A \rightarrow ABC$ . Yes.
- Is  $B$  a key?  $\{\mathbb{B}\}^+ = \{B, C\}$  no
- $\{B\}^+ = \{BC\}$ , i.e.,  $B$  does not decide  $A$ . No.
- Is  $C$  a key? No
- $\{C\}^+ = \{C\}$ . No.
- Is  $AB$  a key?  $\{\mathbb{A}, \mathbb{B}\}^+ = \{A, B, C\}$
- No, since  $A$  is already a key.
- What about  $BC, AC, ABC$ ?

$\{\mathbb{B}\mathbb{C}\}^+, \{\mathbb{B}, \mathbb{C}\}$  no

$\{\mathbb{A}\mathbb{C}\}^+ \subset \{\mathbb{A}, \mathbb{B}, \mathbb{C}\}$  no cause  $A$

$\{\mathbb{A}\mathbb{B}\mathbb{C}\}^+ \subset \{\mathbb{A}, \mathbb{B}, \mathbb{C}\}$  no

# Example

- Definition of a Key: A minimal set of attributes that determines all other attributes
- A table  $R(A, B, C)$
- FDs given:  $A \rightarrow B$
- Is  $A$  a key? *no*
- $\{A\}^+ = \{A, B\}$ . No.
- Is  $B$  or  $C$  a key? *no*
- $\{B\}^+ = \{B\}$ .  $\{C\}^+ = \{C\}$ . No.
- Is  $AB$  or  $BC$  a key?
- $\{AB\}^+ = \{AB\}$ .  $\{BC\}^+ = \{BC\}$ . No.
- Is  $AC$  a key?
- $\{AC\}^+ = \{\underline{ABC}\}$ . Yes.
- Is  $ABC$  a key?

*AB*   *no*  
*AC*   *yes*

*no*

# Finding the Keys: Algorithm

- Check all possible combinations of attributes in the table
  - Example: A, B, C, AB, BC, AC, ABC
- For each combination, compute its closure
  - Example:  $\{A\}^+ = \dots, \{B\}^+ = \dots, \{C\}^+ = \dots, \dots$
- If a closure contains ALL attributes, then the combination might be a key (or superkey)
  - Example:  $\{A\}^+ = \{ABC\}$
- Make sure that you select only keys
  - Example:  $\{A\}^+ = \{ABC\}, \{AB\}^+ = \{ABC\}$ , don't select AB

# Example

- A table  $R(A, B, C, D)$
- $AB \rightarrow C, AD \rightarrow B, B \rightarrow D$
- First, enumerate all attribute combinations:
  - $\{A\}, \{B\}, \{C\}, \{D\}$
  - $\{AB\}, \{AC\}, \{AD\}, \{BC\}, \{BD\}, \{CD\}$
  - $\{ABC\}, \{ABD\}, \{ACD\}, \{BCD\}$
  - $\{ABCD\}$

# Example

- A table R(A, B, C, D)
- $AB \rightarrow C, AD \rightarrow B, B \rightarrow D$

$$\begin{array}{ll} A = A & AB \in AB CD \\ B = BD & AC \in AC \\ C = C & AD \in ABCD \\ D = D & \end{array}$$

- Second, compute the closures:

- $\{A\}^+ = \{A\}, \{B\}^+ = \{BD\}, \{C\}^+ = \{C\}, \{D\}^+ = \{D\}$
- $\{AB\}^+ = \{\underline{ABCD}\}, \{AC\}^+ = \{AC\}, \{AD\}^+ = \{\underline{ABCD}\}$
- $\{BC\}^+ = \{BCD\}, \{BD\}^+ = \{BD\}, \{CD\}^+ = \{CD\}$
- $\{ABC\}^+ = \{ABD\}^+ = \{ACD\}^+ = \{\underline{ABCD}\}$
- $\{BCD\}^+ = \{BCD\}$
- $\{ABCD\}^+ = \{\underline{ABCD}\}$

- Finally, output the keys

$$\begin{array}{ll} BC \Rightarrow BCD \\ BD \Rightarrow BD \\ CD \Rightarrow CD \end{array}$$

# A Small Trick

- Always check small combinations first
- A table R(A, B, C, D)
- $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A$
- Compute the closures:
  - $\{A\}^+ = \underline{\{ABCD\}}, \{B\}^+ = \underline{\{ABCD\}}, \{C\}^+ = \underline{\{ABCD\}}, \{D\}^+ = \underline{\{ABCD\}}$
  - No need to check others
  - The others are all superkeys but not keys
- Keys: {A}, {B}, {C}, {D}

# Another Small Trick

- A table  $R(A, B, C, D)$
- $AB \rightarrow C, AD \rightarrow B, B \rightarrow D$
- Notice that A does not appear at the right hand side of any functional dependencies
- In that case, A must be in every key
- Keys of R: AB, AD (From the previous slide)
- In general, if an attribute that does not appear in the right hand side of any FD, then it must be in every key

# Exercise (Find the Keys)

- A table R(A, B, C, D)
- $A \rightarrow B$ ,  $A \rightarrow C$ ,  $C \rightarrow D$
- A must be in every key
- Compute the closures:
  - $\{A\}^+ = \{\underline{ABCD}\}$
  - No need to check others
- Keys: {A}

$$A = AB \subset CD$$

A

# Exercise (Find the Keys)

- A table R(A, B, C, D, E)
- $AB \rightarrow C, C \rightarrow B, BC \rightarrow D, CD \rightarrow E$
- A must be in every key
- Compute the closures:
  - $\{A\}^+ = \{A\}$
  - $\{AB\}^+ = \underline{\{ABCDE\}}$
  - $\{AC\}^+ = \underline{\{ACBDE\}}$
  - $\{AD\}^+ = \{AD\}, \{AE\}^+ = \{AE\}$
  - $\{ADE\}^+ = \{ADE\}$

$$AE^- = AE$$

# Exercise (Find the Keys)

- A table R(A, B, C, D, E, F)
- $AB \rightarrow C$ ,  $C \rightarrow B$ ,  $CBE \rightarrow D$ ,  $D \rightarrow EF$
- A must be in every key
- Compute the closures:
  - $\{A\}^+ = \{A\}$
  - $\{AB\}^+ = \{ABC\}$
  - $\{AC\}^+ = \{ACB\}$
  - $\{AD\}^+ = \{ADEF\}$
  - $\{AE\}^+ = \{AE\}$ ,  $\{AF\}^+ = \{AF\}$
  - $\{ABC\}^+ = \{ABC\}$
  - $\{ABD\}^+ = \{ABE\}^+ = \{ACD\}^+ = \{ACE\}^+ = \underline{\{ABCDEF\}}$
  - $\{ADE\}^+ = \{ADEF\}$

# This Lecture

- Closure
- Keys
- Finding keys
- Normal forms 

# Recap

- We have talked about a lot of abstract stuff
  - Functional Dependencies
  - Armstrong's Axioms
  - Closures
  - Keys
- Again, what are we doing here?
- The above stuff is needed for **normal forms** and **normalization**

# Recap

| Name  | NRIC | Phone    | Address     |
|-------|------|----------|-------------|
| Alice | 1234 | 67899876 | Jurong East |
| Alice | 1234 | 83848384 | Jurong East |
| Bob   | 5678 | 98765432 | Pasir Ris   |

- The table has a number of anomalies
  - Redundancy, insertion anomalies, etc.
- We would like to get rid of tables like this
- For this purpose, we need
  - A method to **detect** “bad tables” like this
  - A method to **fix** “bad tables” like this

# Recap

| Name  | NRIC | Phone    | Address     |
|-------|------|----------|-------------|
| Alice | 1234 | 67899876 | Jurong East |
| Alice | 1234 | 83848384 | Jurong East |
| Bob   | 5678 | 98765432 | Pasir Ris   |

- The table has a number of anomalies
  - Redundancy, insertion anomalies, etc.
- We would like to get rid of tables like this
- For this purpose, we need
  - A method to **detect** “bad tables” like this: **normal forms**
  - A method to **fix** “bad tables” like this: **normalization**

# Normal Forms

- Some conditions that a “good” table must satisfy
- Intuition
  - A student can get first-class honor, if her GPA is at least 4.0
  - “At least 4.0” is a condition that first-class-honor students must satisfy
  - Normal forms are conditions for “good” tables

# Normal Forms

- Various normal forms (in increasing order of strictness)
  - First normal form
  - Second normal form
  - Third normal form (3NF)
  - Boyce-Codd normal form (BCNF)
  - Fourth normal form
  - Fifth normal form
  - Sixth normal form
- 3NF and BCNF are most commonly used

# Normal Forms

- Various normal forms (in increasing order of strictness)
  - First normal form
  - Second normal form
  - Third normal form (3NF)
  - Boyce-Codd normal form (BCNF)
  - Fourth normal form
  - Fifth normal form
  - Sixth normal form
- 3NF and BCNF are most commonly used

# **CZ2007 Introduction to Database Systems (Week 3)**

---

## **Topic 3: Boyce-Codd Normal Form (1)**



# This Lecture

- Boyce-Codd Normal Form 
- BCNF Decomposition

# Normal Forms

- Various normal forms (in increasing order of strictness)
  - First normal form
  - Second normal form
  - Third normal form (3NF)
  - Boyce-Codd normal form (BCNF)
  - Fourth normal form
  - Fifth normal form
  - Sixth normal form
- 3NF and BCNF are most commonly used

# Boyce-Codd Normal Form (BCNF)

- A table R is in BCNF, if and only if
  - The left hand side of **every** non-trivial FD contains a key of R
- Non-trivial FD:
  - An FD that is not implied by the axiom of reflexivity
  - Example:
    - $A \rightarrow B$  -- Non-trivial
    - $AC \rightarrow BC$  -- Non-trivial
    - $AC \rightarrow A$  -- Trivial
    - $AC \rightarrow C$  -- Trivial

# Boyce-Codd Normal Form (BCNF)

- A table R is in BCNF, if and only if
  - The left hand side of **every** non-trivial FD contains a key of R
- $R(A, B)$
- Given FDs:  $A \rightarrow B$
- Key: **A**
- All FDs on R:  $A \rightarrow B$ ,  $AB \rightarrow A$ ,  $AB \rightarrow B$ ,  $AB \rightarrow AB$ 
  - $AB \rightarrow A$ ,  $AB \rightarrow B$ ,  $AB \rightarrow AB$ : trivial
  - $A \rightarrow B$ : The left hand side contains a key
- Therefore, R is in BCNF

# Boyce-Codd Normal Form (BCNF)

- A table R is in BCNF, if and only if
  - The left hand side of **every** non-trivial FD contains a key of R
- $R(A, B, C)$
- Given FDs:  $A \rightarrow B$
- Key: **AC**
- ! ■ All FDs on R:  $A \rightarrow B$ ,  $AB \rightarrow B$ ,  $AC \rightarrow C$ , ...
  - The left hand side of  $A \rightarrow B$  does not contain a key
- Therefore, R is NOT in BCNF

# BCNF: Example

| Name  | { NRIC } | Phone }  | Address     |
|-------|----------|----------|-------------|
| Alice | 1234     | 67899876 | Jurong East |
| Alice | 1234     | 83848384 | Jurong East |
| Bob   | 5678     | 98765432 | Pasir Ris   |

- NRIC → Name, Address
- Key: {NRIC, Phone} (key is a composite of two attributes)
- Not in BCNF

# **BCNF: Straightforward Checking**

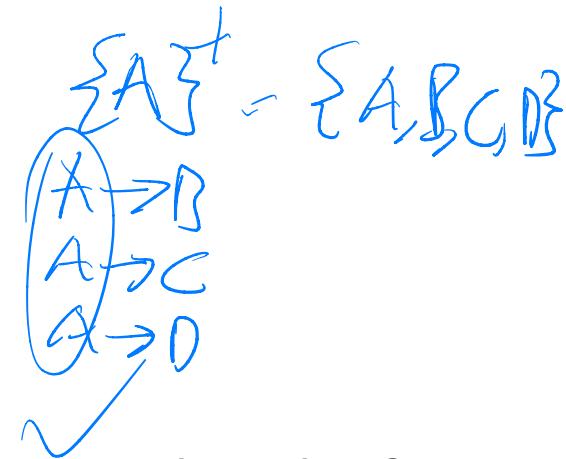
- Given: A table R, a set of FDs on R
- Step 1: Derive the keys of R
- Step 2: Derive all non-trivial FDs on R
  
- Step 3: For each non-trivial FD, check if its left hand side contains a key
- Step 4: If all FDs pass the check, then R is in BCNF; otherwise, R is not in BCNF

# BCNF: Straightforward Checking

- Given: A table R, A set of FDs on R
- Step 1: Derive the keys of R
- Step 2: Derive all non-trivial FDs on R
  - This is too time-consuming
  - Trick: Only check the FDs **given** on R instead of all FDs
- Step 3: For each non-trivial FD, check if its left hand side contains a key
- Step 4: If all FDs pass the check, then R is in BCNF; otherwise, R is not in BCNF

# BCNF Checking: Example

- R(A, B, C, D)
- Given:  $A \rightarrow B$ ,  $A \rightarrow C$ ,  $A \rightarrow D$ ,  $A \rightarrow A$
- Keys: A
- Check: For each given non-trivial FD, check if its left hand side contains a key
  - $A \rightarrow B$ : Non-trivial, LHS contains a key
  - $A \rightarrow C$ : Non-trivial, LHS contains a key
  - $A \rightarrow D$ : Non-trivial, LHS contains a key
- Therefore, R is in BCNF



# BCNF Checking: Example

- R(A, B, C, D)  $\{A\}^t \subset \{A, B, C, D\}$
- Given:  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow D$
- Keys: A
- Check: For each given non-trivial FD, check if its left hand side contains a key
  - $A \rightarrow B$ : Non-trivial, LHS contains a key
  - $B \rightarrow C$ : Non-trivial, LHS does NOT contain a key
- Therefore, R is NOT in BCNF

# BCNF Checking: Rationale

- A table R is in BCNF, if and only if for **every** FD on R,
  - Either the FD is trivial
  - Or the left hand side of the FD contains a key
- The definition says “**every**”, but we only check the ones “**given**” on R
- This leaves the “**hidden**” ones unchecked
- Why does it work?
- Rationale: If the “**given**” FDs pass the check, then all “**hidden**” ones will pass the check because all derived FDs will come from the given FDs.

# BCNF Checking: Multiple-Key Cases

- R(A, B, C, D)
- Given:  $A \rightarrow B$ ,  $B \rightarrow A$ ,  $B \rightarrow C$ ,  $B \rightarrow D$ ,  $A \rightarrow D$
- Keys: A, B (i.e., R has two keys)
- Check: For each given non-trivial FD, check if its left hand side contains a key
  - $A \rightarrow B$ ,  $A \rightarrow D$ : Non-trivial, LHS contains a key
  - $B \rightarrow A$ ,  $B \rightarrow C$ ,  $B \rightarrow D$ : Non-trivial, LHS contains a key
- Therefore, R is in BCNF

# BCNF: Intuition

- Basically, BCNF requires that there forbids any non-trivial  $X \rightarrow Y$  where  $X$  does not contain a key
- Why does this make sense?
- Intuition: Such  $X \rightarrow Y$  indicates that the table has some redundancies

# BCNF Intuition: Example

| Name  | { NRIC } | Phone }  | Address     |
|-------|----------|----------|-------------|
| Alice | 1234     | 67899876 | Jurong East |
| Alice | 1234     | 83848384 | Jurong East |
| Bob   | 5678     | 98765432 | Pasir Ris   |

- NRIC → Name, Address
- Key: {NRIC, Phone} (single composite key)
- NRIC decides Name and Address
- Therefore, every time NRIC is repeated, Name and Address would also be repeated
- Since NRIC is not a key, the same NRIC can appear multiple times in the table
- This leads to redundancy
- BCNF prevents this

# This Lecture

- Boyce-Codd Normal Form
- BCNF Decomposition 

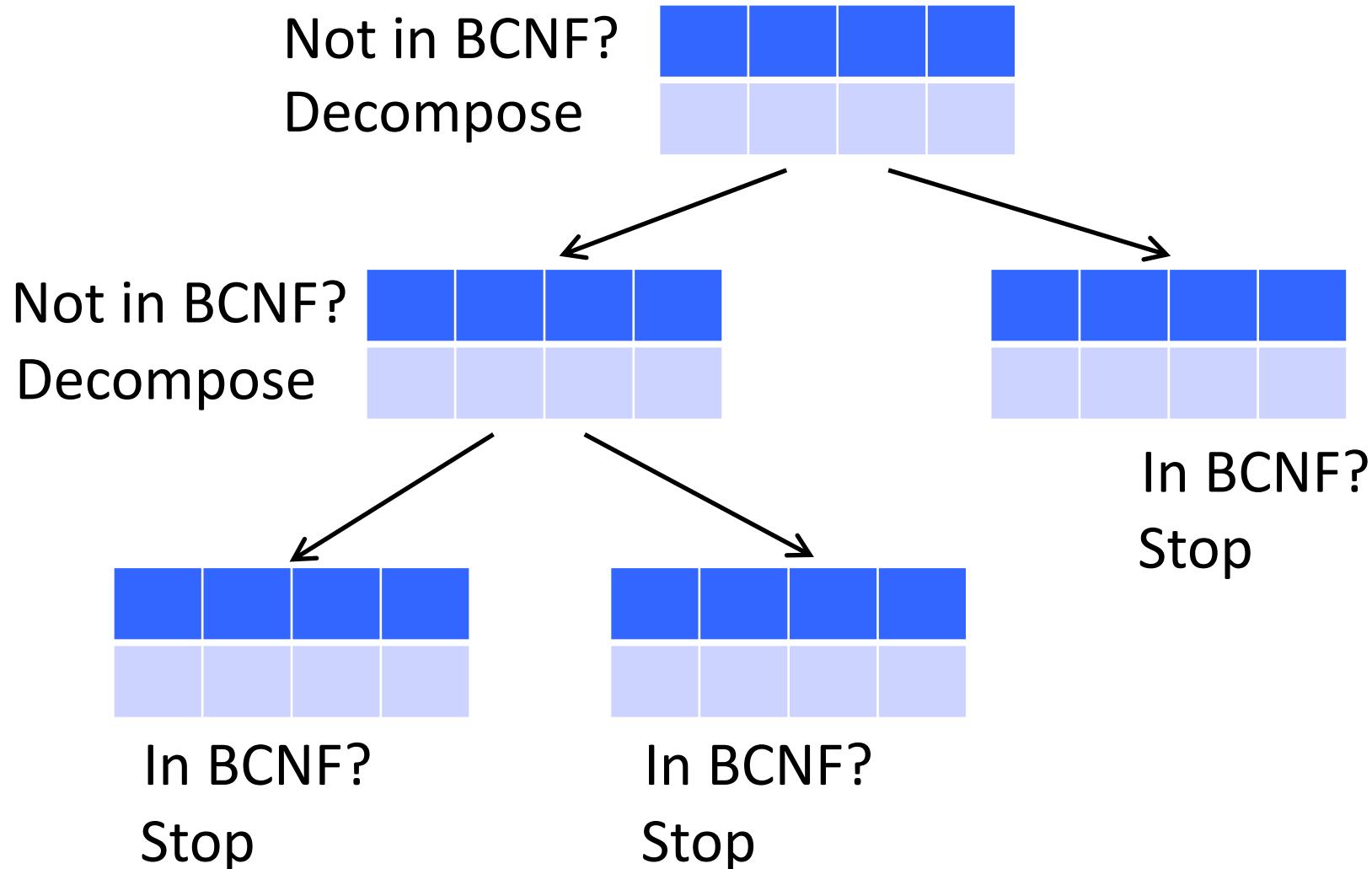
# BCNF Decomposition

| Name  | { NRIC } | Phone }  | Address     |
|-------|----------|----------|-------------|
| Alice | 1234     | 67899876 | Jurong East |
| Alice | 1234     | 83848384 | Jurong East |
| Bob   | 5678     | 98765432 | Pasir Ris   |

- What can we do if a table violates BCNF?
- Answer: Decompose it (i.e., normalize it)

| Name  | NRIC | Address     | { NRIC } | Phone }  |
|-------|------|-------------|----------|----------|
| Alice | 1234 | Jurong East | 1234     | 67899876 |
| Bob   | 5678 | Pasir Ris   | 1234     | 83848384 |

# Decompose, until all are in BCNF



# BCNF Decomposition: Example

- R(A, B, C, D)
- Given:  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow D$
- Key of R: A
- Step 1: Identify a FD that violates BCNF
- $B \rightarrow C$  is a violation, since
  - It is non-trivial
  - Its left hand side does not contain a key

# BCNF Decomposition: Example

- R(A, B, C, D)
- Given:  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow D$
- Key of R: A
- Step 1:  $B \rightarrow C$  is a violation
- Step 2: Compute the closure of the left hand side of the FD
- $\{B\}^+ = \{BCD\}$

# BCNF Decomposition: Example

- $R(A, B, C, D)$
- Given:  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow D$
- Key of  $R$ :  $A$
- Step 1:  $B \rightarrow C$  is a violation
- Step 2:  $\{B\}^+ = \{BCD\}$
- Step 3: Decompose  $R$  into two tables  $R_1$  and  $R_2$ 
  - $R_1(B, C, D)$  contains all attributes in the closure
  - $R_2(A, B)$  contains all attributes NOT in the closure plus  $B$ , LHS of violating FD
- Step 4: Check  $R_1$  and  $R_2$ , decompose if necessary

# BCNF Decomposition Algorithm

- Input: A table R BCNF
- Step 1: Find a FD  $X \rightarrow Y$  on R that violates BCNF
  - If cannot find, stop, R in BCNF
- Step 2: Compute the closure  $\{X\}^+$
- Step 3: Break R into two tables  $R_1$  and  $R_2$ 
  - $R_1$  contains all attributes in  $\{X\}^+$
  - $R_2$  contains all attributes NOT in  $\{X\}^+$  plus X
- Repeat Steps 1-3 on  $R_1$  and  $R_2$

# BCNF Decomposition: Example

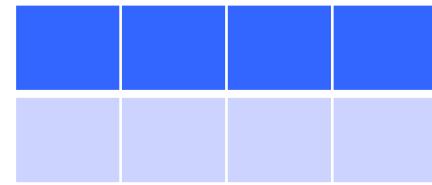
- R(A, B, C, D)
- Given: A→B, B→C, C→D
- Key of R: A
- Previous results: R<sub>1</sub>(B, C, D), R<sub>2</sub>(A, B)
- Is R<sub>2</sub> in BCNF?
  - Yes. So R<sub>2</sub> is done
- Is R<sub>1</sub> in BCNF?
  - No. Key of R<sub>1</sub> is B and C→D is a violation.
- Decompose R<sub>1</sub> into R<sub>3</sub> and R<sub>4</sub>
  - {C}<sup>+</sup> = {CD}
  - R<sub>3</sub>(C, D) contains all attributes in {C}<sup>+</sup>
  - R<sub>4</sub>(B, C) contains C and all attribute NOT in {C}<sup>+</sup>
- Are R<sub>3</sub> and R<sub>4</sub> in BCNF?
  - Yes. So we stop here.

# BCNF Decomposition: Example

- R(A, B, C, D)
- Given:  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow D$
- Key of R: A
  
- Final BCNF Decomposition
  - $R_2(A, B)$
  - $R_3(C, D)$
  - $R_4(B, C)$

# Decompose, until all are in BCNF

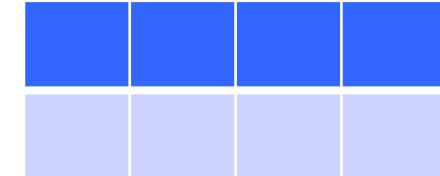
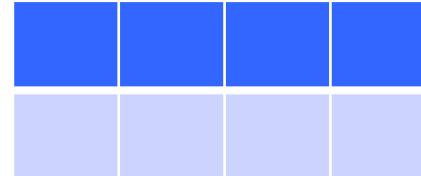
$X \rightarrow Y$  violation?  
Decompose



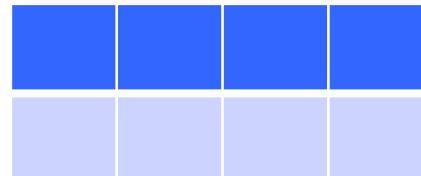
X + Attributes  
NOT in  $\{X\}^+$

Attributes in  $\{X\}^+$

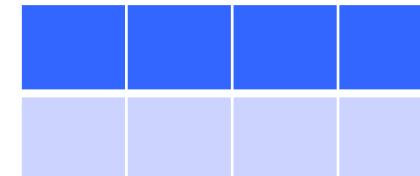
$X' \rightarrow Y'$  violation?  
Decompose



Attributes in  $\{X'\}^+$



In BCNF then  
Stop



In BCNF then  
Stop

# Notes

- The BCNF decomposition of a table may not be unique
- If a table has only two attributes, then it must be in BCNF
  - Therefore, you do not need to check tables with only two attributes

# Exercise: BCNF Decomposition

- $R(A, B, C, D, E)$
- Given:  $AB \rightarrow C$ ,  $C \rightarrow D$ ,  $D \rightarrow E$
- Key of  $R$ :  $AB$  (one single composite key)
- $D \rightarrow E$  is a violation
- Decompose  $R(A, B, C, D, E)$ 
  - $\{D\}^+ = \{D, E\}$
  - $R_1(D, E)$ ,  $R_2(A, B, C, D)$
  - $R_1$  has only two attributes. Must be in BCNF.
  - $R_2$  ... Key of  $R_2$ :  $AB$ . Therefore,  $C \rightarrow D$  is a violation
- Decompose  $R_2(A, B, C, D)$ 
  - $\{C\}^+ = \{C, D, E\}$ .  $E$  is omitted since it is not in  $R_2$
  - $R_3(C, D)$ ,  $R_4(A, B, C)$
  - $R_3$  has only two attributes. Must be in BCNF.
  - $R_4$  ... Key of  $R_4$ :  $AB$ . No violation of BCNF. Done

# **CZ2007 Introduction to Database Systems (Week 4)**

---

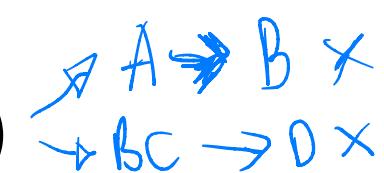
## **Topic 3: Boyce-Codd Normal Form (2)**



# This Lecture

- Tricky case of BCNF 
- Properties of BCNF

# Tricky Case of BCNF Decomposition

- $R(A, B, C, D, E)$
- $A \rightarrow B, BC \rightarrow D$
- Key of  $R$ : **ACE** (one single composite key) 
- $A \rightarrow B$  is a violation.
- Decompose  $R$ 
  - $\{A\}^+ = \{A, B\}$
  - $R_1(A, B), R_2(A, C, D, E)$
  - $R_1$  is in BCNF
  - How about  $R_2$ ? 
- **Key of  $R_2$ : ACE**
- Violations any?
- A bit tricky ...

#KOH JIA WEI#

Questions about determination of Candidate key  
To: Ng Wee Keong

00:11

Hello Professor Ng,

Can I clarify with you how you determined the candidate key?

For R1(A,B), how do we know that A is the candidate key?  
Do we check there exists a functional dependency  $A \rightarrow B$ , hence  $(A)^+ = (A,B)$ , hence the candidate key of R1 is A?

For R2(A,C,D,E), how do we know that ACE is the candidate key of R2?  
I know that candidate keys are minimal superkeys, derived from functional dependencies. I am assuming that candidate key is found through the relations  $A \rightarrow B$  and  $BC \rightarrow D$ .  
However, I am confused because the attribute "B" does not exist within the realm of the R2 relation of (A,C,D,E), hence  $BC \rightarrow D$  functional dependency cannot be used?

## Tricky Case of BCNF Decomposition

- R(A, B, C, D, E)  
 $A \rightarrow B$ ,  $BC \rightarrow D$
- Key of R: ACE (one single composite key)
- $A \rightarrow B$  is a violation.
- Decompose R
  - $\{A\}^* = \{A, B\}$
  - $R_1(A, B)$ ,  $R_2(A, C, D, E)$
  - $R_1$  is in BCNF
    - How about  $R_2$ ?
- Key of  $R_2$ : ACE
- Violations any?
- A bit tricky ...

We use the term "candidate key" when there are more than one keys. In this case, R has only one key. So, we use "key", not candidate key.

For R1, A is a key because of FD  $A \rightarrow B$ .

In fact, any two-attribute schema is in BCNF.  
(Tutorial question).

For R2, ACE is a key because ACE is the key of original relation R and A, C, E are all in R2.

By definition of key, if ACE is a key in R, it means it determines all attributes  $ACE \rightarrow A, B, C, D, E$ .

# Tricky Case of BCNF Decomposition

- In general, we may have a tricky case in BCNF decomposition, if
  - We are checking whether a table T satisfies BCNF, and there is an FD  $X \rightarrow Y$  such that
    - X contains some attribute in T, but
    - Y contains some attribute NOT in T
- Example in the previous slide:
  - We are checking  $R_2(A, C, D, E)$
  - FDs:  $A \rightarrow B$ ,  $BC \rightarrow D$
  - A is in  $R_2$ , but B is not
  - This leads to a tricky case
  - In this case, we have to use closures to check whether  $R_2$  is in BCNF

# Checking BCNF in a Tricky Case

- We are checking  $R_2(A, C, D, E)$
- FDs that we have:  $A \rightarrow B$ ,  $BC \rightarrow D$
- Check the closures:
- $\{A\}^+ = \{A, B\}$
- $B$  is not in  $R_2(A, C, D, E)$ , so the closure becomes
- $\{A\}^+ = \{A\}$
- Similarly,  $\{C\}^+ = \{C\}$ ,  $\{D\}^+ = \{D\}$ ,  $\{E\}^+ = \{E\}$
- So far, none of these indicate a violation of BCNF (**trivial FDs**)
- We check further:  $\{AC\}^+ = \{ACD\}$
- This indicates that  $AC \rightarrow D$  and  $AC$  does not contain key ACE of  $R_2$
- This means that  $R_2$  is not in BCNF
- So we need to decompose it

# Checking BCNF in a Tricky Case

- We are checking  $R_2(A, C, D, E)$
- FDs that we have:  $A \rightarrow B$ ,  $BC \rightarrow D$
- We know that  $AC \rightarrow D$  violates BCNF on  $R_2$
- Decompose  $R_2(A, C, D, E)$ 
  - $\{AC\}^+ = \{A, C, D\}$
  - $R_3(A, C, D)$ ,  $R_4(A, C, E)$
- $R_4$  is in BCNF (ACE is key)
- What about  $R_3$ ?
- Tricky case ( $A \rightarrow B$ ); need to use closure again

# Checking BCNF in a Tricky Case

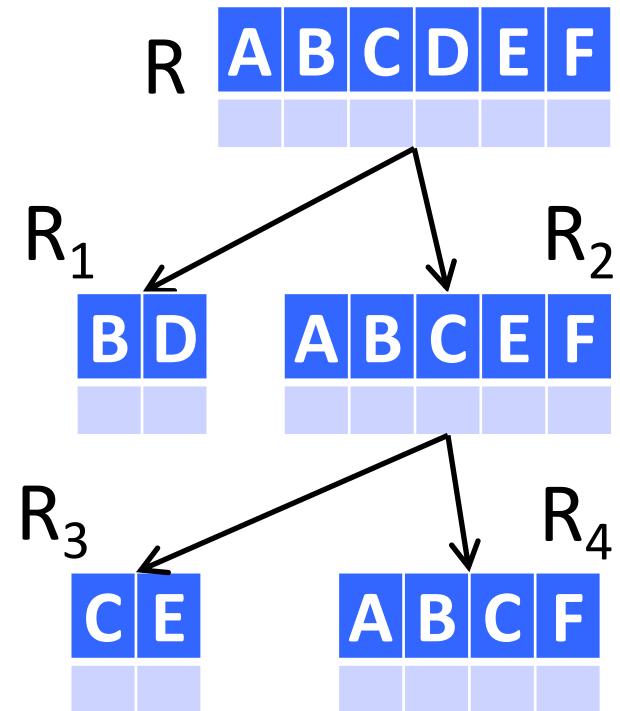
- We are checking  $R_3(A, C, D)$
- FDs that we have:  $A \rightarrow B$ ,  $BC \rightarrow D$
- $\{A\}^+ = \{A, B\}$        $\rightarrow$        $\{A\}^+ = \{A\}$  on  $R_3$
- $\{C\}^+ = \{C\}$ ,  $\{D\}^+ = \{D\}$
- $\{AC\}^+ = \{A, B, C, D\}$        $\rightarrow$        $\{AC\}^+ = \{A, C, D\}$  on  $R_3$
- $\{AD\}^+ = \{A, B, D\}$        $\rightarrow$        $\{AD\}^+ = \{A, D\}$  on  $R_3$
- $\{CD\}^+ = \{C, D\}$
- None of the closures indicate a violation of BCNF
- Therefore,  $R_3$  is in BCNF
- Final decomposition:  $R_1(A, B)$ ,  $R_3(A, C, D)$ ,  $R_4(A, C, E)$

# Summary

- Whenever we have a tricky case in checking BCNF, we need to resort to closures
- How to use closures to check that a table T is NOT in BCNF:
  - If there is a closure  $\{X\}^+ = \{Y\}$ , such that
    - Y does not contain all attributes in T, and (not all)
    - Y contains more attributes than X (more but)
- Previous example:
  - $R_2(A, C, D, E)$ , with  $A \rightarrow B$ ,  $BC \rightarrow D$
  - $\{AC\}^+ = \{A, C, D\}$
  - $\{A, C, D\}$  does not contain all attributes in  $R_2$ , but
  - $\{A, C, D\}$  contains more attributes than  $\{AC\}$

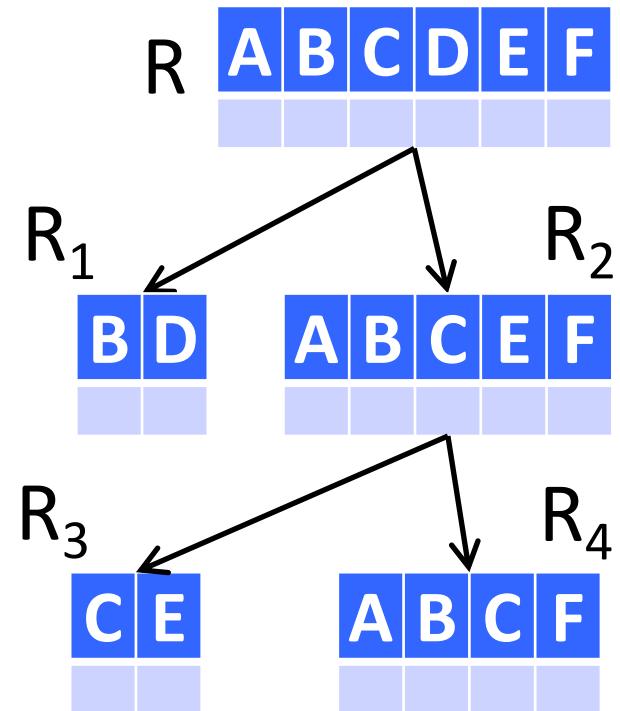
# Exercise

- $R( A, B, C, D, E, F )$
- Given FDs:  $B \rightarrow D$ ,  $C \rightarrow E$ ,  $DE \rightarrow A$
- Keys: BCF
- $B \rightarrow D$  violates BCNF
- Decompose R:
  - $R1( B, D ), R2( A, B, C, E, F )$
- R1 is in BCNF
- What about R2? Tricky case. We could address the tricky case, but . . .
- we also notice  $C \rightarrow E$  is violating. We can just decompose rightaway:
- Decompose R2,  $\{C\}^+ = \{C, E\}$ 
  - $R3( C, E ), R4( A, B, C, F )$



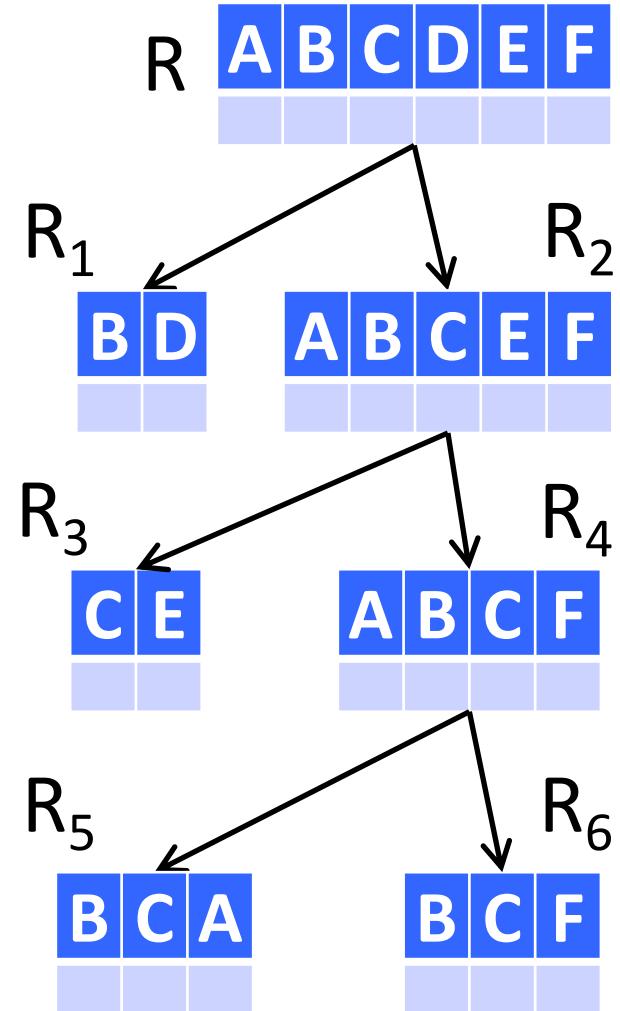
# Exercise

- $R( A, B, C, D, E, F )$
- Given FDs:  $B \rightarrow D$ ,  $C \rightarrow E$ ,  $DE \rightarrow A$
- $R_3( C, E )$  is in BCNF
- What about  $R_4( A, B, C, F )$ ?
- Tricky case
- Check closures
  - $\{A\}^+ = \{A\}$ ,  $\{B\}^+ = \{BD\}$ ,  $\{C\}^+ = \{CE\}$ ,  $\{F\}^+ = \{F\}$
  - $\{AB\}^+ = \{ABD\}$ ,  $\{AC\}^+ = \{ACE\}$ ,  $\{AF\}^+ = \{AF\}$ ,  $\{BC\}^+ = \underline{\{BCDEA\}}$
- So there is a non-trivial FD:  $BC \rightarrow A$



# Exercise

- R( A, B, C, D, E, F )
- Given FDs:  $B \rightarrow D$ ,  $C \rightarrow E$ ,  $DE \rightarrow A$
- R3( C, E ) is in BCNF
- What about R4( A, B, C, F )?
- Keys of R4: BCF
- There is a non-trivial FD:  $BC \rightarrow A$
- It violates BCNF
- Decompose R4
  - R5( B, C, A ), R6( B, C, F )
- Final decomposition:
  - R1( B, D ), R3( C, E ), R5( B, C, A ), R6( B, C, F )



# This Lecture

- Tricky case of BCNF
- Properties of BCNF 

# Properties of BCNF Decomposition

- Good properties
  - No update or deletion anomalies
  - Very small redundancy
  - The original table can always be reconstructed from the decomposed tables if functional dependencies are preserved  
(this is called the **lossless join** property)
  - Reconstruction is at the schema level only if some FDs not preserved

# Lossless Join Property

| Name  | <u>NRIC</u> | <u>Phone</u> | Address     |
|-------|-------------|--------------|-------------|
| Alice | 1234        | 67899876     | Jurong East |
| Alice | 1234        | 83848384     | Jurong East |
| Bob   | 5678        | 98765432     | Pasir Ris   |

- The table above can be perfectly reconstructed using the decomposed tables below as all FDs preserved:
- $\{NRIC, Phone\} \rightarrow Name, Address$ ,  $NRIC \rightarrow Name, Address$

| Name  | <u>NRIC</u> | Address     |
|-------|-------------|-------------|
| Alice | 1234        | Jurong East |
| Bob   | 5678        | Pasir Ris   |

| <u>NRIC</u> | <u>Phone</u> |
|-------------|--------------|
| 1234        | 67899876     |
| 1234        | 83848384     |
| 5678        | 98765432     |

# Why BCNF guarantees lossless join?

- Say we decompose a table  $R$  into two tables  $R_1$  and  $R_2$
- The decomposition guarantees lossless join, whenever the common attributes in  $R_1$  and  $R_2$  constitute a **superkey** of  $R_1$  or  $R_2$
- Example
  - $R(A, B, C)$  decomposed into  $R_1(A, B)$  and  $R_2(B, C)$ , with  $B$  being the key of  $R_2$
  - $R(A, B, C, D)$  decomposed into  $R_1(A, B, C)$  and  $R_2(B, C, D)$ , with  $BC$  being the key of  $R_1$

# Why BCNF guarantees lossless join?

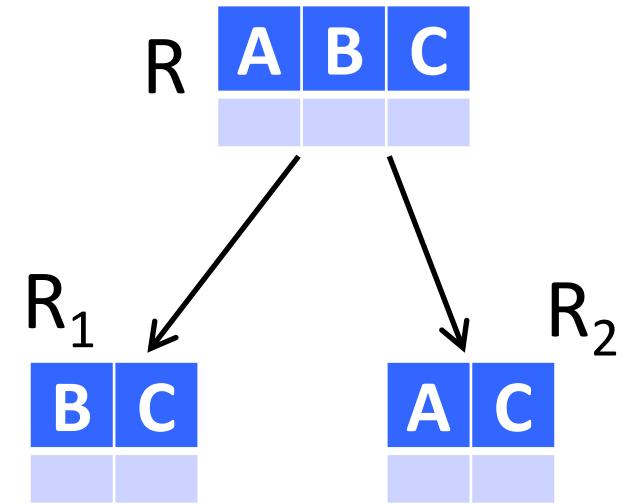
- The decomposition of R guarantees lossless join, whenever the common attributes in  $R_1$  and  $R_2$  constitute a superkey of  $R_1$  or  $R_2$
- BCNF Decomposition of R
  - Find a BCNF violation  $X \rightarrow Y$
  - Compute  $\{X\}^+$
  - $R_1$  contains all attributes in  $\{X\}^+$
  - $R_2$  contains X and all attributes NOT in  $\{X\}^+$
  - X is both in  $R_1$  and  $R_2$
  - And X is a superkey of  $R_1$
  - Therefore,  $R_1$  and  $R_2$  is a lossless decomposition of R

# Properties of BCNF Decomposition

- Good properties
  - No update or deletion anomalies
  - Very small redundancy
  - The original table can always be reconstructed from the decomposed tables if functional dependencies are preserved  
(this is called the **lossless join** property)
- Bad property
  - It may not preserve all functional dependencies

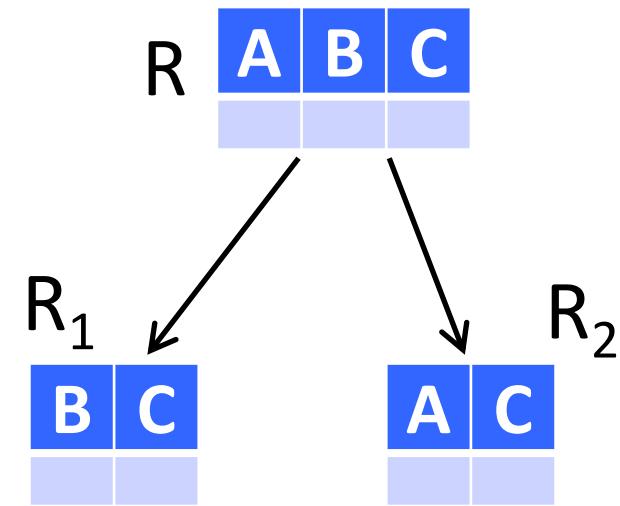
# Dependency Preservation

- Given: Table  $R(A, B, C)$ 
  - with  $AB \rightarrow C, C \rightarrow B$
- Keys:  $AB, AC$
- BCNF Decomposition
  - $R_1(B, C)$
  - $R_2(A, C)$
- Non-trivial FDs on  $R_1$ :  $C \rightarrow B$
- Non-trivial FDs on  $R_2$ : none
- The other FD,  $AB \rightarrow C$ , should hold on any individual table, but it is “lost”
- We say that a BCNF decomposition does not always **preserve** all FDs



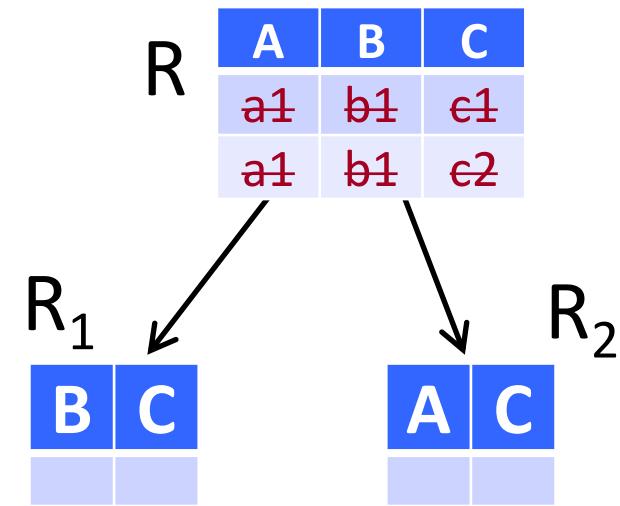
# Dependency Preservation

- Why do we want to preserve FDs?
- Because we want to make it easier to avoid “inappropriate” updates
- Previous example
  - We have two tables  $R_1(B, C)$ ,  $R_2(A, C)$
  - We have  $C \rightarrow B$  and  $AB \rightarrow C$
  - Due to  $AB \rightarrow C$ , we are not suppose to have two tuples  $(a_1, b_1, c_1)$  and  $(a_1, b_1, c_2)$
  - 
  - 
  -



# Dependency Preservation

- Why do we want to preserve FDs?
- Because we want to make it easier to avoid “inappropriate” updates
- Previous example
  - We have two tables  $R_1(B, C)$ ,  $R_2(A, C)$
  - We have  $C \rightarrow B$  and  $AB \rightarrow C$
  - Due to  $AB \rightarrow C$ , we are not suppose to have two tuples  $(a1, b1, c1)$  and  $(a1, b1, c2)$
  - But as we store A and C separately in  $R_1$  and  $R_2$ , it is not easy to check whether such two tuples exist at the same time
  - 
  -



# Dependency Preservation

- Why do we want to preserve FDs?
- Because we want to make it easier to avoid “inappropriate” updates
- Previous example
  - We have two tables  $R_1(B, C)$ ,  $R_2(A, C)$
  - We have  $C \rightarrow B$  and  $AB \rightarrow C$
  - Due to  $AB \rightarrow C$ , we are not suppose to have two tuples  $(a1, b1, c1)$  and  $(a1, b1, c2)$
  - But as we store A and C separately in  $R_1$  and  $R_2$ , it is not easy to check whether such two tuples exist at the same time
  - That is, if someone wants to insert  $(a1, b1, c2)$ , it is not easy for us to check whether  $(a1, b1, c1)$  already exists
  - This is less than ideal

| R  | A  | B  | C |
|----|----|----|---|
| a1 | b1 | c1 |   |
| a1 | b1 | c2 |   |

| $R_1$ | B  | C |
|-------|----|---|
| b1    | c1 |   |
| b1    | c2 |   |

| $R_2$ | A  | C |
|-------|----|---|
| a1    | c1 |   |
| a1    | c2 |   |

# Third Normal Form (3NF)

- A relaxation of BCNF that
  - Is less strict
  - Allows decompositions that **always** preserve functional dependencies
- Will be discussed in the next lecture

# CZ2007 Introduction to Database Systems (Week 4)

---

## Topic 4: Third Normal Form (1)



# This Lecture

- 3NF 
- 3NF Decomposition

# Third Normal Form (3NF)

- A relaxation of BCNF that
  - Is less strict
  - Allows decompositions that **always** preserve functional dependencies

# 1NF, 2NF

key = if key just 1 : A

key attribute = if key > 1 (part)

key attributes = ABC

- **Key-attribute:** An attribute in a multi-attribute key
- Key-attribute(s) = Partial key or part of a key
- 1NF: All attributes have atomic values
- 2NF: Every non-key attribute is dependent on the whole of **EVERY** candidate key
  - Even so, may still have additional dependencies, such as (non-key-attribute X) → (non-key-attribute Y) in relation

# Third Normal Form (3NF)

- Definition: A table satisfies 3NF, if and only if for every non-trivial  $X \rightarrow Y$ 
  - Either X contains a key
  - Or each attribute in Y is contained in a key
- Example:
  - Given FDs:  $C \rightarrow B$ ,  $AB \rightarrow C$ ,  $BC \rightarrow C$
  - Keys:  $\{AB\}$ ,  $\{AC\}$
  - $AB \rightarrow C$  is OK, since **AB** is a key of R
  - $C \rightarrow B$  is OK, since **B** is in a key of R
  - $BC \rightarrow C$  is OK, since **C** is in AC and in BC (it is also trivial)
  - So R is in 3NF

| R | A | B | C |
|---|---|---|---|
|   |   |   |   |
|   |   |   |   |

# 3NF, BCNF

- 3NF: 2NF + all key-attributes determined **ONLY** by candidate keys (in whole or in part)
  - (non-key-attribute X) → (non-key-attribute Y) cannot exist anymore, RHS must be key attribute in 3NF
  - But candidate keys may have **overlapping** attributes
  - May result in key-attribute(s) of one key depends on key-attribute(s) of another key (this dependency is eliminated in BCNF)
- BCNF: In all dependencies (FDs), LHS must contain key (cannot depend on partial key)

# Third Normal Form (3NF)

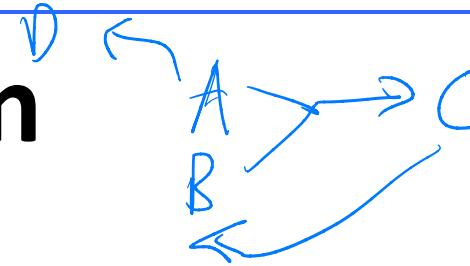
- Definition: A table satisfies 3NF, if and only if for every non-trivial  $X \rightarrow Y$ 
  - Either X contains a key
  - Or each attribute in Y is contained in a key
- Another Example:
  - Given FDs:  $A \rightarrow B$ ,  $B \rightarrow C$
  - Keys: {A}
  - $A \rightarrow B$  is OK, since **A** is a key of R
  - $B \rightarrow C$  is not OK, since **C** is NOT in a key of R, and it is NOT in the left hand side
  - So R is NOT in 3NF

| R | A | B | C |
|---|---|---|---|
|   |   |   |   |
|   |   |   |   |

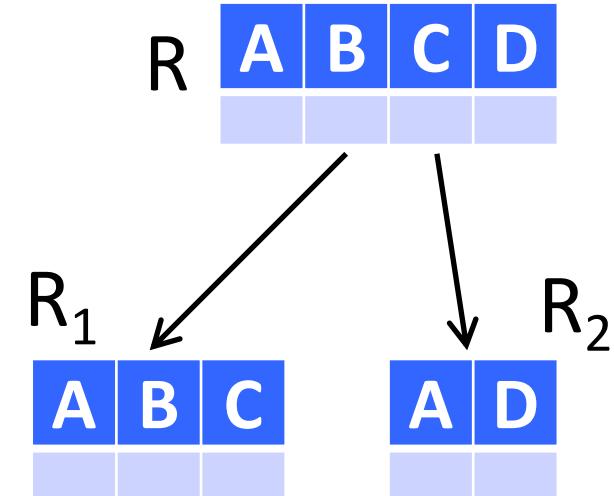
# This Lecture

- 3NF
- 3NF Decomposition 

# 3NF Decomposition



- Given: A table NOT in 3NF
- Objective: Decompose it into smaller tables that are in 3NF
- Example
  - Given:  $R(A, B, C, D)$
  - FDs:  $AB \rightarrow C$ ,  $C \rightarrow B$ ,  $A \rightarrow D$
  - Keys:  $\{AB\}$ ,  $\{AC\}$
  - **R is not in 3NF, due to  $A \rightarrow D$**
  - 3NF decomposition of R:  
 $R_1(A, B, C)$ ,  $R_2(A, D)$

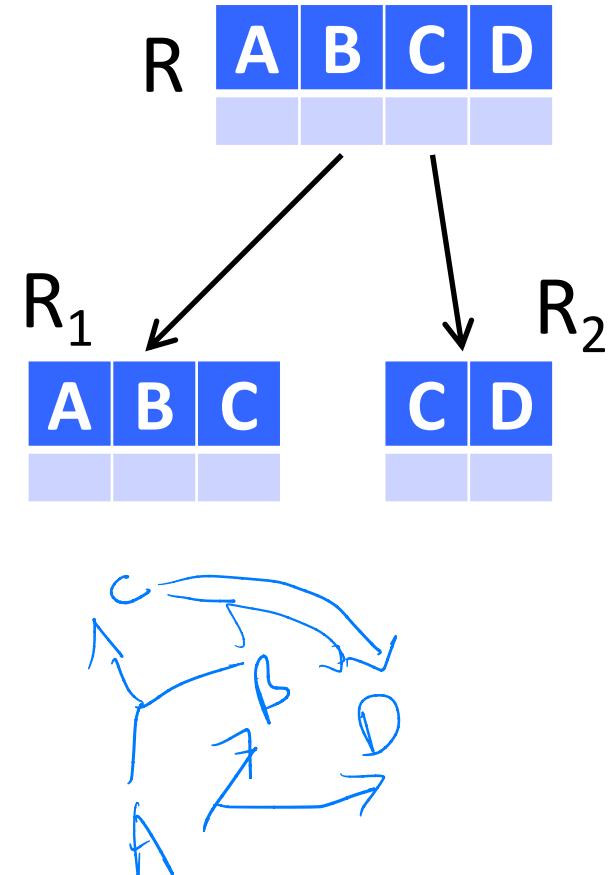


$$\begin{aligned} \{A\}^+ &= \{A, D\} \\ R_2 &\subset A, D \end{aligned}$$

$$R_1 = A, B, C$$

# 3NF Decomposition Algorithm

- Given: A table R, and a set S of FDs
  - e.g., R(A, B, C, D)  
 $S = \{A \rightarrow BD, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Step 1: Derive a **minimal basis** of S
  - e.g., a minimal basis of S is  $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$
- Step 2: In the minimal basis, combine the FDs whose left hand sides are the same
  - e.g., after combining  $A \rightarrow B$  and  $A \rightarrow C$ , we have  $\{A \rightarrow BC, C \rightarrow D\}$
- Step 3: Create a table for each FD remained
  - $R_1(A, B, C), R_2(C, D)$
- Step 4: If none of the tables contain a key of the original table R, create a table that contains a key of R
- Step 5: Remove redundant tables (schema is a subset of another)



# Minimal Basis

- Given a set  $S$  of FDs, the **minimal basis** of  $S$  is a **simplified** version of  $S$
- Previous example:
  - $S = \{A \rightarrow BD, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
  - A minimal basis:  $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$
- How simplified?
- Three conditions.
- **Condition 1: For any FD in the minimal basis, its right hand side has only one attribute.**
- Example in  $S$ :  $A \rightarrow BD$  does not satisfy this condition
- That is why  $A \rightarrow BD$  is not in the minimal basis

# Minimal Basis

- Previous example:
  - $S = \{A \rightarrow BD, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
  - A minimal basis:  $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$
- Condition 2: No FD in the minimal basis is redundant.
- That is, no FD in the minimal basis can be derived from the other FDs.
- Example in S:  $BC \rightarrow D$  can be derived from  $C \rightarrow D$
- That is why  $BC \rightarrow D$  is not in the minimal basis

# Minimal Basis

- Previous example:
  - $S = \{A \rightarrow BD, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
  - A minimal basis:  $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$
- Condition 3: For each FD in the minimal basis, none of the attributes on the left hand side is redundant
- That is, for any FD in the minimal basis, if we remove an attribute from the left hand side, then the resulting FD is a new FD that cannot be derived from the original set of FDs
- Example:
  - $S$  contains  $AB \rightarrow C$
  - If we remove  $B$  from the left hand side, we have  $A \rightarrow C$
  - $A \rightarrow C$  can be derived from  $S$ , as  $\{A\}^+ = \{ABDC\}$
  - This indicates that  $A \rightarrow C$  is “hidden” in  $S$
  - Hence, we can replace  $AB \rightarrow C$  with  $A \rightarrow C$ , as  $A \rightarrow C$  is “simpler”
  - This is why  $AB \rightarrow C$  is not in the minimal basis

# Algorithm for Minimal Basis

- Given: a set S of FDs
- Example:  $S = \{A \rightarrow BD, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Step 1: Transform the FDs, so that each right hand side contains only one attribute
- Result:  $S = \{A \rightarrow B, A \rightarrow D, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Reason:
  - Condition 1 for minimal basis:  
The right hand side of each FD contains only one attribute

# Algorithm for Minimal Basis

- Result of Step 1:
  - $S = \{A \rightarrow B, A \rightarrow D, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Step 2: Remove redundant FDs
- Is  $A \rightarrow B$  redundant?
- i.e., is  $A \rightarrow B$  implied by other FDs in  $S$ ?
- Let's check
- Without  $A \rightarrow B$ , we have  $\{A \rightarrow D, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Given those FDs, we have  $\{A\}^+ = \{AD\}$ , which does not contain  $B$
- Therefore,  $A \rightarrow B$  is not implied by the other FDs

# Algorithm for Minimal Basis

- Result of Step 1:
  - $S = \{A \rightarrow B, A \rightarrow D, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Continue Step 2: Remove redundant FDs
- Is  $A \rightarrow D$  redundant?
- i.e., is  $A \rightarrow D$  implied by other FDs in  $S$ ?
- Let's check
- Without  $A \rightarrow D$ , we have  $\{A \rightarrow B, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Given those FDs, we have  $\{A\}^+ = \{ABCD\}$ , which contains  $D$
- Therefore,  $A \rightarrow D$  is implied by the other FDs
- Hence,  $A \rightarrow D$  is redundant and should be removed
- Result:  $S = \{A \rightarrow B, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$

# Algorithm for Minimal Basis

- Result of the last step:
  - $S = \{A \rightarrow B, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Continue Step 2: Remove redundant FDs
- Is  $AB \rightarrow C$  redundant?
- i.e., is  $AB \rightarrow C$  implied by other FDs in  $S$ ?
- Let's check
- Without  $AB \rightarrow C$ , we have  $\{A \rightarrow B, C \rightarrow D, BC \rightarrow D\}$
- Given those FDs, we have  $\{AB\}^+ = \{AB\}$ , which does not contain  $C$
- Therefore,  $AB \rightarrow C$  is NOT implied by the other FDs
- Hence,  $AB \rightarrow C$  is not redundant and should not be removed

# Algorithm for Minimal Basis

- Result of the last step:
  - $S = \{A \rightarrow B, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Continue Step 2: Remove redundant FDs
- Is  $C \rightarrow D$  redundant?
- i.e., is  $C \rightarrow D$  implied by other FDs in  $S$ ?
- Let's check
- Without  $C \rightarrow D$ , we have  $\{A \rightarrow B, AB \rightarrow C, BC \rightarrow D\}$
- Given those FDs, we have  $\{C\}^+ = \{C\}$ , which does not contain  $D$
- Therefore,  $C \rightarrow D$  is NOT implied by the other FDs and should not be removed

# Algorithm for Minimal Basis

- Result of the last step:
  - $S = \{A \rightarrow B, AB \rightarrow C, C \rightarrow D, BC \rightarrow D\}$
- Continue Step 2: Remove redundant FDs
- Is  $BC \rightarrow D$  redundant?
- i.e., is  $BC \rightarrow D$  implied by other FDs in  $S$ ?
- Let's check
- Without  $BC \rightarrow D$ , we have  $\{A \rightarrow B, AB \rightarrow C, C \rightarrow D\}$
- Given those FDs, we have  $\{BC\}^+ = \{BCD\}$ , which contains D
- Therefore,  $BC \rightarrow D$  is implied by the other FDs
- Hence,  $BC \rightarrow D$  is redundant and should be removed
- Result:  $S = \{A \rightarrow B, AB \rightarrow C, C \rightarrow D\}$

# Algorithm for Minimal Basis

- Result of Step 2:
  - $S = \{A \rightarrow B, AB \rightarrow C, C \rightarrow D\}$
- **Step 3: Remove redundant attributes on the left hand side (lhs) of each FD**
- Only  $AB \rightarrow C$  has more than one attribute on the lhs
- Let's check
- Is A redundant?
- If we remove A, then  $AB \rightarrow C$  becomes  $B \rightarrow C$
- Whether this removal is OK depends on whether  $B \rightarrow C$  is “hidden” in S already
- If  $B \rightarrow C$  is “hidden” in S, then the removal of A is OK, (since the removal does not add extraneous information into S)
- Is  $B \rightarrow C$  “hidden” in S?
- Check: Given S, we have  $\{B\}^+ = \{B\}$ , which does NOT contain C
- Therefore,  $B \rightarrow C$  is not “hidden” in S, and hence, A is NOT redundant

# Algorithm for Minimal Basis

- Result of Step 2:
  - $S = \{A \rightarrow B, AB \rightarrow C, C \rightarrow D\}$
- Step 3: Remove redundant attributes on the left hand side (lhs) of each FD
- Only  $AB \rightarrow C$  has more than one attribute on the lhs
- Let's check
- Is B redundant?
- If we remove B, then  $AB \rightarrow C$  becomes  $A \rightarrow C$
- Whether this is OK depends on whether  $A \rightarrow C$  is “hidden” in S
- Is  $A \rightarrow C$  “hidden” in S?
- Check: Given S, we have  $\{A\}^+ = \{ABCD\}$ , which contains C
- Therefore,  $A \rightarrow C$  is “hidden” in S
- Hence, we can simplify  $AB \rightarrow C$  to  $A \rightarrow C$
- Final minimal basis:  $S = \{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$

# CZ2007 Introduction to Database Systems (Week 5)

---

## Topic 4: Third Normal Form (2)



# This Lecture

- 3NF Decomposition 
- Properties of 3NF

# Exercise

- $S = \{A \rightarrow C, AC \rightarrow D, AD \rightarrow B\}$
- 1. Transform the FDs to ensure that the right hand side of each FD has only one attribute
- 2. See if any FD can be derived from the other FDs. Remove those FDs one by one
- 3. Check if we can remove any attribute from the left hand side of any FD

# Exercise

- $S = \{A \rightarrow C, AC \rightarrow D, AD \rightarrow B\}$
- 1. Transform the FDs to ensure that the right hand side of each FD has only one attribute
- Results:  $M = \{A \rightarrow C, AC \rightarrow D, AD \rightarrow B\}$
- 2. See if any FD can be derived from the other FDs. Remove those FDs one by one

# Exercise

- $M = \{A \rightarrow C, AC \rightarrow D, AD \rightarrow B\}$
- 2. See if any FD can be derived from the other FDs.  
Remove those FDs one by one.
- Try  $A \rightarrow C$  first
  - If  $A \rightarrow C$  is removed, then the ones left would be  $AC \rightarrow D, AD \rightarrow B$
  - With the remaining FDs, we have  $\{A\}^+ = \{A\}$
  - Since  $\{A\}^+$  does not contain C, we know that  $A \rightarrow C$  cannot be derived from the remaining FDs
  - Therefore,  $A \rightarrow C$  cannot be removed

# Exercise

- $M = \{A \rightarrow C, AC \rightarrow D, AD \rightarrow B\}$
- 2. See if any FD can be derived from the other FDs.  
Remove those FDs one by one.
- Next, try  $AC \rightarrow D$ 
  - If  $AC \rightarrow D$  is removed, then the ones left would be  $A \rightarrow C, AD \rightarrow B$
  - With the remaining FDs, we have  $\{AC\}^+ = \{ACD\}$
  - Since  $\{AC\}^+$  does not contain D, we know that  $AC \rightarrow D$  cannot be derived from the remaining FDs
  - Therefore,  $AC \rightarrow D$  cannot be removed

# Exercise

- $M = \{A \rightarrow C, AC \rightarrow D, AD \rightarrow B\}$
- 2. See if any FD can be derived from the other FDs.  
Remove those FDs one by one.
- Next, try  $AD \rightarrow B$ 
  - If  $AD \rightarrow B$  is removed, then the ones left would be  $A \rightarrow C, AC \rightarrow D$
  - With the remaining FDs, we have  $\{AD\}^+ = \{AD\}$
  - Since  $\{AD\}^+$  does not contain B, we know that  $AD \rightarrow B$  cannot be derived from the remaining FDs
  - Therefore,  $AD \rightarrow B$  cannot be removed

# Exercise

- $M = \{A \rightarrow C, AC \rightarrow D, AD \rightarrow B\}$
- 3. Check if we can remove any attribute from the left hand side of any FD
- First, try to remove A from  $AC \rightarrow D$ 
  - It results in  $C \rightarrow D$
  - Can  $C \rightarrow D$  be derived from M?
  - $\{C\}^+ = \{C\}$  given M.
  - Since  $\{C\}^+$  does not contain D, we know that  $C \rightarrow D$  cannot be derived from M
  - Therefore, A cannot be removed from  $AC \rightarrow D$

# Exercise

- $M = \{A \rightarrow C, AC \rightarrow D, AD \rightarrow B\}$
- 3. Check if we can remove any attribute from the left hand side of any FD
  - Next, try to remove C from  $AC \rightarrow D$ 
    - It results in  $A \rightarrow D$
    - Can  $A \rightarrow D$  be derived from M?
    - $\{A\}^+ = \{ABCD\}$  given M.
    - Since  $\{A\}^+$  contains D, we know that  $A \rightarrow D$  can be derived from M
    - Therefore, C can be removed from  $AC \rightarrow D$
  - New  $M = \{A \rightarrow C, A \rightarrow D, AD \rightarrow B\}$

# Exercise

- New  $M = \{A \rightarrow C, A \rightarrow D, AD \rightarrow B\}$
- 3. Check if we can remove any attribute from the left hand side of any FD
- Next, try to remove A from  $AD \rightarrow B$ 
  - It results in  $D \rightarrow B$
  - Can  $D \rightarrow B$  be derived from M?
  - $\{D\}^+ = \{D\}$  given M.
  - Since  $\{D\}^+$  does not contain B, we know that  $D \rightarrow B$  cannot be derived from M
  - Therefore, A cannot be removed from  $AD \rightarrow B$

# Exercise

- $M = \{A \rightarrow C, A \rightarrow D, AD \rightarrow B\}$
- 3. Check if we can remove any attribute from the left hand side of any FD
- Next, try to remove D from  $AD \rightarrow B$ 
  - It results in  $A \rightarrow B$
  - Can  $A \rightarrow B$  be derived from M?
  - $\{A\}^+ = \{ABCD\}$  given M.
  - Since  $\{A\}^+$  contains B, we know that  $A \rightarrow B$  can be derived from M
  - Therefore, D can be removed from  $AD \rightarrow B$
- New  $M = \{A \rightarrow C, A \rightarrow D, A \rightarrow B\}$ ; done

# 3NF Decomposition Algorithm

- Given:
  - Table R(A, B, C, D)
  - A minimal basis  $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$
- Step 1: Combine those FDs with the same left hand side
  - Result:  $\{A \rightarrow BC, C \rightarrow D\}$
- Step 2: For each FD, create a table that contains all attributes in the FD
  - Result:  $R_1(A, B, C), R_2(C, D)$
- Step 3: Remove redundant tables (if any)
- Tricky issue: Sometimes we also need to add an additional table (see the next slide)

# 3NF Decomposition Algorithm

- Given:
  - Table R(A, B, C, D)
  - A minimal basis  $\{A \rightarrow B, C \rightarrow D\}$
- Step 1: Combine those FDs with the same left hand side
  - Result:  $\{A \rightarrow B, C \rightarrow D\}$
- Step 2: For each FD, create a table that contains all attributes in the FD
  - Result:  $R_1(A, B), R_2(C, D)$
- Step 3: Remove redundant tables (if any)
- Problem:  $R_1$  and  $R_2$  do not ensure lossless join
- Solution: Add a table that contains a key of the original table R
- Key of R:  $\{A, C\}$
- Additional table to add:  $R_3(A, C)$
- Final result:  $R_1(A, B), R_2(C, D), R_3(A, C)$

# 3NF Decomposition Algorithm

- Given:
  - Table R(A, B, C, D)
  - A minimal basis  $\{A \rightarrow B, C \rightarrow D\}$
- Step 1: Combine those FDs with the same left hand side
  - Result:  $\{A \rightarrow B, C \rightarrow D\}$
- Step 2: For each FD, create a table that contains all attributes in the FD
  - Result:  $R_1(A, B), R_2(C, D)$
- Step 3: If no table contain a key of the original table, add a table that contains a key of the original table
  - Result:  $R_1(A, B), R_2(C, D), R_3(A, C)$
- Step 4: Remove redundant tables (if any)

# This Lecture

- 3NF Decomposition
- Properties of 3NF 

# Minimal Basis is not always unique

- For given set of FDs, its minimal basis may not be unique
- Example:
  - Given  $R(A, B, C)$  and  $\{A \rightarrow B, A \rightarrow C, B \rightarrow C, B \rightarrow A, C \rightarrow A, C \rightarrow B\}$
  - Minimal basis 1:  $\{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$
  - Minimal basis 2:  $\{A \rightarrow C, B \rightarrow C, C \rightarrow A, C \rightarrow B\}$
- Different minimal basis may lead to different 3NF decompositions

# BCNF vs. 3NF

- BCNF: For any non-trivial FD
  - its left hand side (lhs) is a superkey
- 3NF: For any non-trivial FD
  - Either its lhs is a superkey
  - Or each attribute on its right hand side either appear in the lhs or in a key
- Observation: BCNF is stricter than 3NF
- Therefore
  - A table that satisfies BCNF must satisfy 3NF, but not vice versa
  - A table that violates 3NF must violate BCNF, but not vice versa

# BCNF vs. 3NF

- BCNF Decomposition:
  - Avoids insertion, deletion, and update anomalies
  - Eliminates most redundancies
  - But does not always preserve all FDs
- 3NF Decomposition:
  - Avoids insertion, deletion, and update anomalies
  - May lead to a bit more redundancy than BCNF
  - Always preserve all FDs
- So which one to use?
- A logical approach
  - Go for a BCNF decomposition first
  - If it preserves all FDs, then we are done
  - If not, then go for a 3NF decomposition instead

# Why Does 3NF Preserve All FDs?

- Given: A table R, and a set S of FDs
- Step 1: Derive a minimal basis of S
- Step 2: In the minimal basis, combine the FDs whose left hand sides are the same
- Step 3: Create a table for each FD remained
- Step 4: If none of the tables contain a key of the original table R, create a table that contains a key of R
- Step 5: Remove redundant tables
- Rationale: Because of Step 3 (**minimal basis preserves FDs; no redundant FDs**)

# **CZ2007 Introduction to Database Systems (Week 5)**

---

## **Topic 5: Relational Algebra (1)**



# This Lecture

- Motivation for relational algebra
- Relational algebraic operators
  - Selection:  $\sigma_{A > 100} R_1$
  - Projection:  $\Pi_{A, B} R_1$
  - Union:  $R_1 \cup R_2$
  - Intersection:  $R_1 \cap R_2$
  - Difference:  $R_1 - R_2$
  - Natural Join:  $R_1 \bowtie R_2$
  - Theta Join:  $R_1 \bowtie_{R1.A=R2.A \text{ AND } R1.B < R2.B} R_2$

# Relational Algebra: Motivation

- We have specification of an DB application
- We use ER-diagram for a conceptual design of database
- We transform ER-diagram into database schema (i.e., the schemas of a set of tables)
- We normalize the schema, and then insert some tuples into the tables
- Now what?
- How do we perform queries on those tables?
  - Database side: Relational Algebra (RA)
  - User side: Structured Query Language (SQL)

# Relational Algebra: Motivation



User

Query  
In SQL

Query Interface

Query  
In RA

Processing Engine

Database



# Relational Algebra

- A mathematical way to formulate queries on relations (i.e., tables)
- Has numerous **operators** for query formulation
- Example
  - Given: Two relations  $R_1(A, B, C)$ ,  $R_2(A, B, C)$
  - Selection:  $\sigma_{A > 100} R_1$
  - Projection:  $\Pi_{A, B} R_1$
  - Union:  $R_1 \cup R_2$
  - Intersection:  $R_1 \cap R_2$
  - And a few others...

# Selection $\sigma$ (row-wise operation)

| Students | ID    | Name | Age  | School |
|----------|-------|------|------|--------|
| 1234     | Alice | 20   | SCSE |        |
| 5678     | Bob   | 20   | EEE  |        |
| 3742     | Cathy | 22   | SCSE |        |
| 9413     | David | 21   | CEE  |        |

- Query: “Find me the student named Alice”
- $\sigma_{\text{Name} = \text{'Alice'}}$  Students

| Results | ID    | Name | Age  | School |
|---------|-------|------|------|--------|
| 1234    | Alice | 20   | SCSE |        |

# Selection $\sigma$

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

- Query: “Find the students in SCSE”
- $\sigma_{\text{School} = \text{'SCSE'}}$  Students

| Results | ID   | Name  | Age | School |
|---------|------|-------|-----|--------|
|         | 1234 | Alice | 20  | SCSE   |
|         | 3742 | Cathy | 22  | SCSE   |

# Selection $\sigma$

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

- Query: “Find the SCSE students under 21”
- $\sigma_{\text{School} = \text{'SCSE'} \text{ AND } \text{Age} < 21}$  Students

| Results | ID   | Name  | Age | School |
|---------|------|-------|-----|--------|
|         | 1234 | Alice | 20  | SCSE   |

# Selection $\sigma$

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

- Query: “Find the students who are either in SCSE or under 21”
- $\sigma_{\text{School} = \text{'SCSE'} \text{ OR } \text{Age} < 21} \text{ Students}$

# Projection $\Pi$ (column-wise)

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

- Query: “Find the IDs and Names of all students”
- $\Pi_{ID, Name} Students$

Results

| ID   | Name  |
|------|-------|
| 1234 | Alice |
| 5678 | Bob   |
| 3742 | Cathy |
| 9413 | David |

# Combining Operators

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

- Query: “Find the IDs and Names of all students in SCSE”
- $\Pi_{ID, Name} (\sigma_{School = 'SCSE'} Students)$

| ID   | Name  |
|------|-------|
| 1234 | Alice |
| 3742 | Cathy |

# Combining Operators

| Students | ID   | Name  | Age | School |
|----------|------|-------|-----|--------|
|          | 1234 | Alice | 20  | SCSE   |
|          | 5678 | Bob   | 20  | EEE    |
|          | 3742 | Cathy | 22  | SCSE   |
|          | 9413 | David | 21  | CEE    |

- Query: “Find the IDs and Names of all students in SCSE”
- How about  $\sigma_{\text{School} = \text{'SCSE'}}(\Pi_{\text{ID}, \text{Name}} \text{ Students})$ ?
- Wrong
- The projection goes before the selection here
- Since the projection eliminates “School”, the selection cannot be performed

# Union $\cup$

**Students**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |

**Volunteer**

| Name  | Age |
|-------|-----|
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

## Results

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

- Query: “Find the persons who are either volunteers”
- Students  $\cup$  Volunteer
-

# Union $\cup$

**Students**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |

**Volunteer**

| Name  | Age |
|-------|-----|
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

**Results**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

- Query: “Find the persons who are either students or volunteers”
- Students  $\cup$  Volunteer
- Note 1: Duplicate tuples are automatically removed

# Union $\cup$

**Students**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |

**Volunteer**

| Name  | Age |
|-------|-----|
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

## Results

| Name  |
|-------|
| Alice |
| Bob   |
| Cathy |
| David |
| Eddie |
| Fred  |

- Query: “Find the names of the persons who are either students or volunteers”
- $\Pi_{\text{Name}} (\text{Students} \cup \text{Volunteer})$
- $(\Pi_{\text{Name}} \text{ Students}) \cup (\Pi_{\text{Name}} \text{ Volunteer})$

# Union $\cup$

**Students**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |

**Volunteer**

| Name  |
|-------|
| Cathy |
| David |
| Eddie |
| Fred  |

## Results

| Name  |
|-------|
| Alice |
| Bob   |
| Cathy |
| David |
| Eddie |
| Fred  |

- Query: “Find the persons who are either students or volunteers”
- Students  $\cup$  Volunteer ?
- Wrong
- Note 2: The two sides of a union must have the same schema (i.e., the same set of attributes)
- Correct solution:  $(\Pi_{\text{Name}} \text{ Students}) \cup \text{Volunteer}$

# Intersection $\cap$

**Students**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |

**Volunteer**

| Name  | Age |
|-------|-----|
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

**Results**

| Name  | Age |
|-------|-----|
| Cathy | 22  |
| David | 21  |

- Query: “Find the persons who are both students and volunteers”
- Students  $\cap$  Volunteer
- Note 1: Duplicate tuples are automatically removed

# Intersection $\cap$

**Students**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |

**Volunteer**

| Name  |
|-------|
| Cathy |
| David |
| Eddie |
| Fred  |

**Results**

| Name  |
|-------|
| Cathy |
| David |

- Query: “Find the persons who are both students and volunteers”
- $(\Pi_{\text{Name}} \text{ Students}) \cap \text{Volunteer}$
- Note 2: The two sides of an intersection must have the same schema (i.e., the same set of attributes)

# Difference –

**Students**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |

**Volunteer**

| Name  | Age |
|-------|-----|
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

**Results**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |

- Query: “Find the persons who are students but not volunteers”
- Students – Volunteer
- Note 1: Duplicate tuples are automatically removed

# Difference –

**Students**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |

**Volunteer**

| Name  | Age |
|-------|-----|
| Cathy | 22  |
| David | 21  |
| Eddie | 43  |
| Fred  | 35  |

**Results**

| Name  | Age |
|-------|-----|
| Eddie | 43  |
| Fred  | 35  |

- Query: “Find the persons who are volunteers but not students”
- Volunteer – Students

# Difference –

**Students**

| Name  | Age |
|-------|-----|
| Alice | 20  |
| Bob   | 21  |
| Cathy | 22  |
| David | 21  |

**Volunteer**

| Name  |
|-------|
| Cathy |
| David |
| Eddie |
| Fred  |

**Results**

| Name  |
|-------|
| Alice |
| Bob   |

- Query: “Find the persons who are students but not volunteers”
- $(\Pi_{\text{Name}} \text{ Students}) - \text{Volunteer}$
- Note 2: The two sides of a difference must have the same schema (i.e., the same set of attributes)

# Exercise

## Grades

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Bob   | DB     | B     |

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DM     | C     |

| Name  | Course | Grade |
|-------|--------|-------|
| Bob   | AI     | B     |
| Cathy | CG     | A     |

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | AI     | B     |
| Cathy | CG     | A     |
| David | NN     | C     |

- Query: “Find the students who have taken DB and DM, but not AI or CG”
- $((\sigma_{\text{Course} = \text{'DB'}} \text{ Grades}) \cap (\sigma_{\text{Course} = \text{'DM'}} \text{ Grades})) - ((\sigma_{\text{Course} = \text{'AI'}} \text{ Grades}) \cup (\sigma_{\text{Course} = \text{'CG'}} \text{ Grades}))$
- Result is empty set
- Wrong

# Exercise

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Bob   | DB     | B     |

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DM     | C     |

| Name  | Course | Grade |
|-------|--------|-------|
| Bob   | AI     | B     |
| Cathy | CG     | A     |

| Grades |        |       |
|--------|--------|-------|
| Name   | Course | Grade |
| Alice  | DB     | A     |
| Alice  | DM     | C     |
| Bob    | DB     | B     |
| Bob    | AI     | B     |
| Cathy  | CG     | A     |
| David  | NN     | C     |

- Query: “Find the students who have taken DB and DM, but not AI or CG”
- $((\Pi_{\text{Name}} \sigma_{\text{Course} = 'DB'} \text{Grades}) \cap (\Pi_{\text{Name}} \sigma_{\text{Course} = 'DM'} \text{Grades})) - ((\Pi_{\text{Name}} \sigma_{\text{Course} = 'AI'} \text{Grades}) \cup (\Pi_{\text{Name}} \sigma_{\text{Course} = 'CG'} \text{Grades}))$

# Exercise

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Bob   | DB     | B     |
| Bob   | AI     | B     |
| Cathy | CG     | A     |
| David | NN     | C     |

## Grades

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | AI     | B     |
| Cathy | CG     | A     |
| David | NN     | C     |

- Query: “Find the students who have never taken DM”
- $\sigma_{\text{Course} \neq \text{'DM'}} \text{Grades}$
- Alice has taken DM but still appear in the result
- Wrong

# Exercise

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Bob   | DB     | B     |
| Bob   | AI     | B     |
| Cathy | CG     | A     |
| David | NN     | C     |

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DM     | C     |

## Grades

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | AI     | B     |
| Cathy | CG     | A     |
| David | NN     | C     |

- Query: “Find the students who have never taken DM”
- Grades – ( $\sigma_{\text{Course} = \text{'DM'}}$  Grades)
- Alice has taken DM but still appear in the result
- Wrong

# Exercise

| Name  |
|-------|
| Alice |
| Bob   |
| Bob   |
| Cathy |
| David |

| Name  |
|-------|
| Alice |

Grades

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | AI     | B     |
| Cathy | CG     | A     |
| David | NN     | C     |

- Query: “Find the students who have never taken DM”
- $(\Pi_{\text{Name}} \text{ Grades}) - (\Pi_{\text{Name}} \sigma_{\text{Course} = 'DM'} \text{ Grades})$

# Natural Join $\bowtie$

Students

| NRIC | Name  |
|------|-------|
| 11   | Alice |
| 2    | Bob   |
| 33   | Cathy |
| 4    | David |

Phones

| NRIC | Number  |
|------|---------|
| 11   | 9123234 |
| 11   | 8635168 |
| 33   | 8213654 |
| 5    | 9653154 |

Results

| NRIC | Name  | Number  |
|------|-------|---------|
| 11   | Alice | 9123234 |
| 11   | Alice | 8635168 |
| 33   | Cathy | 8213654 |

- Query: “Find the NRIC, Name, and Phone of each student, omitting those without a phone” (those without phone will not appear in table)
- Students  $\bowtie$  Phones
- Note 1: The join is performed based on the common attributes of the two relations
- Note 2: Each common attribute appears only once in the result

# Natural Join ⚡

| Students |        | Donations |        | Results |        |        |
|----------|--------|-----------|--------|---------|--------|--------|
| Name     | School | Name      | Amount | Name    | School | Amount |
| Alice    | SCSE   | Cathy     | 100    | Cathy   | CEE    | 100    |
| Bob      | EEE    | David     | 200    | David   | SCSE   | 200    |
| Cathy    | CEE    | Eddie     | 300    |         |        |        |
| David    | SCSE   | Fred      | 400    |         |        |        |

- Students ⚡ Donations
- Meaning: “For those students who have made donation, find their names, schools, and amounts of their donations”

# Natural Join $\bowtie$

Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

Results

| Name  | School | Amount |
|-------|--------|--------|
| David | SCSE   | 200    |

- $(\sigma_{\text{School} = \text{'SCSE'}} \text{ Students}) \bowtie \text{ Donations}$
- Meaning: “For those SCSE students who have made a donation, find their names, schools, and amounts of their donations”

# Exercise

## Results

Name

Alice

Grades

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | NN     | B     |
| Cathy | SP     | B     |
| Cathy | NN     | A     |

CrsSch

| Course | School |
|--------|--------|
| DB     | SCSE   |
| DM     | SCSE   |
| AI     | SCSE   |
| NN     | EEE    |
| SP     | EEE    |

- Query: “Find the students who have taken SCSE courses but not EEE courses”
- $(\Pi_{Name} \text{ Grades} \bowtie (\sigma_{School = 'SCSE'} \text{ CrsSch})) - (\Pi_{Name} \text{ Grades} \bowtie (\sigma_{School = 'EEE'} \text{ CrsSch}))$

# Exercise

Grades

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | NN     | B     |
| Cathy | SP     | B     |
| Cathy | NN     | A     |

CrsSch

| Course | School |
|--------|--------|
| DB     | SCSE   |
| DM     | SCSE   |
| AI     | SCSE   |
| NN     | EEE    |
| SP     | EEE    |

- Query: “Find the students who have only taken EEE courses”
- How to eliminate Bob who has taken SCSE courses?

# Exercise

## Results

Name

Cathy

Grades

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | NN     | B     |
| Cathy | SP     | B     |
| Cathy | NN     | A     |

CrsSch

| Course | School |
|--------|--------|
| DB     | SCSE   |
| DM     | SCSE   |
| AI     | SCSE   |
| NN     | EEE    |
| SP     | EEE    |

- Query: “Find the students who have only taken EEE courses”
- $\Pi_{\text{Name}} \text{ Grades} - (\Pi_{\text{Name}} \text{ Grades} \bowtie (\sigma_{\text{School} \leftrightarrow \text{'EEE'}} \text{ CrsSch}))$

# Theta Join $\bowtie$ condition

Students

| SName | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

Results

| SName | Name  | School | Amount |
|-------|-------|--------|--------|
| Cathy | Cathy | CEE    | 100    |
| David | David | SCSE   | 200    |

- Query: “For those students who have made donations, find their names, schools, and amounts of their donations”
- Students  $\bowtie_{Sname=Name}$  Donations
- Difference from natural join: Duplicate attributes will NOT be removed from the results
- In general, the join condition in a theta join can also be inequalities

# Theta Join $\bowtie$ condition

Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

Quiz2

| Name  | Score |
|-------|-------|
| Alice | 80    |
| Bob   | 90    |
| Cathy | 90    |
| David | 70    |

Results

| Name  | Score | Name  | Score |
|-------|-------|-------|-------|
| Alice | 70    | Alice | 80    |
| Cathy | 80    | Cathy | 90    |

- Query: “Find the students who score higher in quiz 2 than quiz 1”
- Quiz1  $\bowtie$  Quiz1.Name = Quiz2.Name AND Quiz1.Score < Quiz2.Score Quiz2
- Note: In the join condition, whenever there are ambiguous attribute names (e.g., Score), we need to add the table names along with the attribute names to eliminate the ambiguity (e.g., by using Quiz1.Score instead of Score)

# Cartesian Product ×

| Students |     |
|----------|-----|
| Name     | Age |
| Alice    | 19  |
| Bob      | 22  |

| Courses |      |
|---------|------|
| ID      | Name |
| C1      | DB   |
| C2      | Algo |

## Results

| Name  | Age | ID | Name |
|-------|-----|----|------|
| Alice | 19  | C1 | DB   |
| Alice | 19  |    | Algo |
| Bob   | 22  | C1 | DB   |
|       | 22  |    | Algo |

- Effect: Theta join without a condition
- Query: “Create a table that provides all possible student-course combinations”
- Students × Donations

# CZ2007 Introduction to Database Systems (Week 6)

---

## Topic 5: Relational Algebra (2)



# Last Lecture: Relational Algebra

- Given: Two relations  $R_1(A, B, C)$ ,  $R_2(A, B, C)$
- Selection:  $\sigma_{A > 100} R_1$
- Projection:  $\Pi_{A, B} R_1$
- Union:  $R_1 \cup R_2$
- Intersection:  $R_1 \cap R_2$
- Difference:  $R_1 - R_2$
- Natural Join:  $R_1 \bowtie R_2$
- Theta Join:  $R_1 \bowtie_{R1.A=R2.A \text{ AND } R1.B < R2.B} R_2$

# This Lecture

- Assignment:  $T_1 := \sigma_{A > 100} R_1$
- Rename:  $\rho_{\text{test}(A', B', C')} R_1$
- Duplicate Elimination  $\delta$
- Extended Projection  $\Pi$
- Grouping and Aggregation  $\gamma$

# Assignment :=

**Quiz1**

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

**Evaluation1**

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

**Over85**

| Name  | Score |
|-------|-------|
| Bob   | 90    |
| David | 100   |

- Conceptually: Make another copy of the table and give it a new name
- Example
  - Evaluation1 := Quiz1
  - Over85 :=  $\sigma_{Score > 85}$  Quiz1
- Note: All attribute names are retained

# Assignment :=

- Useful to break down steps
- Example:
  - $(\Pi_{\text{Name}} \text{ Students}) \cup (\Pi_{\text{Name}} \text{ Volunteer})$
- Equivalent Representation
  - $R1 := \Pi_{\text{Name}} \text{ Students}$
  - $R2 := \Pi_{\text{Name}} \text{ Volunteer}$
  - $R1 \cup R2$
- This makes your solution easier to write and easier for others to understand

# Rename $\rho$

**Quiz1**

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

**Evaluation1**

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

**Eval1**

| SName | QScore |
|-------|--------|
| Alice | 70     |
| Bob   | 90     |
| Cathy | 80     |
| David | 100    |

- Similar to assignment, but allows change of attribute names
- Example
  - $\rho_{\text{Evaluation1}} \text{ Quiz1}$
  - $\rho_{\text{Eval1(SName, QScore)}} \text{ Quiz1}$

# Exercise

Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

R1

| Name  | Score |
|-------|-------|
| Cathy | 80    |

R2

| Name1 | Score1 | Name2 | Score2 |
|-------|--------|-------|--------|
| Bob   | 90     | Cathy | 80     |
| David | 100    | Cathy | 80     |

R3

| Name1 | Score1 |
|-------|--------|
| Bob   | 90     |
| David | 100    |

- Find the students who score higher than Cathy in Quiz1
- $R1 := \sigma_{\text{Name}=\text{'Cathy'}} \text{Quiz1}$
- $\rho_{R2(\text{Name1}, \text{Score1}, \text{Name2}, \text{Score2})} \text{Quiz1} \bowtie_{\text{Quiz1.Score} > R1.\text{Score}} R1$
- $R3 := \Pi_{\text{Name1}, \text{Score1}} R2$

# Exercise

Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |

R1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |

R2

| Name  | Score | Name  | Score |
|-------|-------|-------|-------|
| Alice | 70    | Bob   | 90    |
| Alice | 70    | Cathy | 80    |
| Cathy | 80    | Bob   | 90    |

R4

Name  
Bob

- Find the students who score the highest in Quiz1
- R1 := Quiz1
- R2 := Quiz1  $\bowtie_{\text{Quiz1.Name} \neq \text{R1.Name} \text{ AND } \text{Quiz1.Score} < \text{R1.Score}}$  R1
- R3 :=  $\Pi_{\text{Quiz1.Name}}$  R2
- R4 :=  $\Pi_{\text{Quiz1.Name}}$  Quiz1 – R3

# Exercise

Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

Quiz2

| Name  | Score |
|-------|-------|
| Alice | 80    |
| Bob   | 90    |
| Cathy | 90    |
| David | 70    |

Quiz3

| Name  | Score |
|-------|-------|
| Alice | 90    |
| Bob   | 90    |
| Cathy | 80    |
| David | 70    |

- Query: “Find the students whose scores in Quizzes 1, 2, and 3 keep increasing”
- $(\text{Quiz1} \bowtie_{\text{Quiz1.Name} = \text{Quiz2.Name} \text{ AND } \text{Quiz1.Score} < \text{Quiz2.Score}} \text{Quiz2}) \bowtie_{\text{Quiz2.Name} = \text{Quiz3.Name} \text{ AND } \text{Quiz2.Score} < \text{Quiz3.Score}} \text{Quiz3}$

# Duplicate Elimination $\delta$

Purchase

| Name  | Product | Date       |
|-------|---------|------------|
| Alice | iPhone  | 2017.01.01 |
| Bob   | Xbox    | 2017.01.01 |
| Cathy | iPhone  | 2017.01.01 |
| David | Xbox    | 2017.02.17 |

R1

| Product |
|---------|
| iPhone  |
| Xbox    |
| iPhone  |

R2

| Product |
|---------|
| iPhone  |
| Xbox    |

- Effect: Eliminate duplicate tuples
- Query: Find the list of products sold on 2017.01.01
- $R1 := \Pi_{\text{Product}} (\sigma_{\text{Date}='2017.01.01'} \text{ Purchase})$
- $R2 := \delta(R1)$

# Extended Projection $\Pi$

Scores

| Name  | Quiz1 | Quiz2 |
|-------|-------|-------|
| Alice | 70    | 90    |
| Bob   | 90    | 80    |
| Cathy | 80    | 100   |
| David | 100   | 90    |

Results

| Name  | Total |
|-------|-------|
| Alice | 160   |
| Bob   | 170   |
| Cathy | 180   |
| David | 190   |

- Similar to ordinary projection, but allows the creation of new attributes via arithmetic
- Query: “For each student, find his/her total score in Quiz 1 and 2”
- $\Pi_{\text{Name}, \text{Quiz1} + \text{Quiz2} \rightarrow \text{Total}} \text{ Scores}$
- The left hand side of “ $\rightarrow$ ” gives the arithmetic performed
- The right hand side gives an attribute name to the result

# Extended Projection $\Pi$

Scores

| Name  | Quiz1 | Quiz2 |
|-------|-------|-------|
| Alice | 70    | 90    |
| Bob   | 90    | 80    |
| Cathy | 80    | 100   |
| David | 100   | 90    |

Results

| Name  | Average |
|-------|---------|
| Alice | 80      |
| Bob   | 85      |
| Cathy | 90      |
| David | 95      |

- Similar to ordinary projection, but allows the creation of new attributes via arithmetic
- Query: “For each student, find his/her average score in Quiz 1 and 2”
- $\Pi_{\text{Name}, (\text{Quiz1} + \text{Quiz2})/2} \rightarrow \text{Average}$  Scores

# Grouping and Aggregation $\gamma$

Quiz1

| Name  | School | Score |
|-------|--------|-------|
| Alice | SCSE   | 90    |
| Bob   | EEE    | 80    |
| Cathy | EEE    | 100   |
| David | SCSE   | 90    |

Results

MaxScore

100

- Query: “Find the highest score in Quiz1”
- $\gamma_{\text{MAX}(Score)} \rightarrow \text{MaxScore}$  Quiz1
- The attribute name on right hand side of “ $\rightarrow$ ” can be arbitrary

# Grouping and Aggregation $\gamma$

Quiz1

| Name  | School | Score |
|-------|--------|-------|
| Alice | SCSE   | 90    |
| Bob   | EEE    | 80    |
| Cathy | EEE    | 100   |
| David | SCSE   | 90    |

Results

MinScore

80

- Query: “Find the lowest score in Quiz1”
- $\gamma_{\text{MIN}(\text{Score})} \rightarrow \text{MinScore}$  Quiz1

# Grouping and Aggregation $\gamma$

Quiz1

| Name  | School | Score |
|-------|--------|-------|
| Alice | SCSE   | 90    |
| Bob   | EEE    | 80    |
| Cathy | EEE    | 100   |
| David | SCSE   | 90    |

Results

AvgScore

90

- Query: “Find the average score in Quiz1”
- $\gamma_{\text{AVG}(\text{Score})} \rightarrow \text{AvgScore}$  Quiz1

# Grouping and Aggregation $\gamma$

Quiz1

| Name  | School | Score |
|-------|--------|-------|
| Alice | SCSE   | 90    |
| Bob   | EEE    | 80    |
| Cathy | EEE    | 100   |
| David | SCSE   | 90    |

Results

SumScore

360

- Query: “Find the sum of scores in Quiz1”
- $\gamma_{\text{SUM}(\text{Score})} \rightarrow \text{SumScore}$  Quiz1

# Grouping and Aggregation $\gamma$

Quiz1

| Name  | School | Score |
|-------|--------|-------|
| Alice | SCSE   | 90    |
| Bob   | EEE    | 80    |
| Cathy | EEE    | 100   |
| David | SCSE   | 90    |

Results

NumStu

4

- Query: “Find the number of students in Quiz1”
- $\gamma_{\text{COUNT}(\text{Name})} \rightarrow \text{NumStu}$  Quiz1
- $\gamma_{\text{COUNT}(\text{School})} \rightarrow \text{NumStu}$  Quiz1
- $\gamma_{\text{COUNT}(\text{Score})} \rightarrow \text{NumStu}$  Quiz1
- All three queries above give the number of tuples in Quiz1

# Aggregate Functions

- MAX( ... )
- MIN( ... )
- AVG( ... )
- SUM( ... )
- COUNT( ... )

# Grouping and Aggregation $\gamma$

## Quiz1

| Name  | School | GPA |
|-------|--------|-----|
| Alice | SCSE   | 4   |
| Bob   | EEE    | 3   |
| Cathy | EEE    | 3.4 |
| David | SCSE   | 3.6 |

## Results

| School | AvgGPA |
|--------|--------|
| SCSE   | 3.8    |
| EEE    | 3.2    |

- Query: “Find the average GPA in each school”
- $\gamma_{\text{School}, \text{AVG}(\text{GPA})} \rightarrow \text{AvgGPA}$  Quiz1
- Effect: Divide tuples into separate groups based on their “School” value, and then compute the average GPA in each group

# Grouping and Aggregation $\gamma$

## Quiz1

| Name  | School | GPA |
|-------|--------|-----|
| Alice | SCSE   | 4   |
| Bob   | EEE    | 3   |
| Cathy | EEE    | 3.4 |
| David | SCSE   | 3.6 |

## Results

| School | AvgGPA | MaxGPA |
|--------|--------|--------|
| SCSE   | 3.8    | 4      |
| EEE    | 3.2    | 3.4    |

- Query: “Find the average GPA and highest GPA in each school”
- $\gamma_{\text{School}, \text{AVG}(\text{GPA})} \rightarrow \text{AvgGPA}$ ,  $\text{MAX}(\text{GPA}) \rightarrow \text{MaxGPA}$  Quiz1

# Grouping and Aggregation $\gamma$

## Quiz1

| Name  | School | Year | GPA |
|-------|--------|------|-----|
| Alice | SCSE   | 3    | 4   |
| Bob   | EEE    | 1    | 3   |
| Cathy | EEE    | 2    | 3.4 |
| David | SCSE   | 3    | 3.6 |

## Results

| School | Year | GPA |
|--------|------|-----|
| SCSE   | 3    | 3.8 |
| EEE    | 1    | 3   |
| EEE    | 2    | 3.4 |

- $\gamma_{\text{School}, \text{Year}, \text{AVG}(\text{GPA})} \rightarrow \text{AvgGPA}$  Quiz1
- Effect: Divide tuples into separate groups based on their “School, year” value combination, and then compute the average GPA in each group

# Example

Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

R1

| MaxScore |
|----------|
| 100      |

R2

| Name  | Score | MaxScore |
|-------|-------|----------|
| David | 100   | 100      |

R3

| Name  |
|-------|
| David |

- Query: “Find the student that scores the highest in Quiz1”
- $\sigma_{\text{Score} = \text{MAX}(\text{Score})} \text{Quiz1}$  ?
- Wrong: Aggregate functions can only be used with the aggregation operation  $\gamma$
- $R1 := \gamma_{\text{MAX}(\text{Score}) \rightarrow \text{MaxScore}}(\text{Quiz1})$
- $R2 := \text{Quiz1} \bowtie_{\text{Score} = \text{MaxScore}} R1$
- $R3 := \Pi_{\text{Name}}(R2)$

# Exercise

Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

R1

| MaxScore |
|----------|
| 100      |

R2

| Name  | Score | MaxScore |
|-------|-------|----------|
| David | 100   | 100      |

R3

| Name  | Score |
|-------|-------|
| David | 100   |

- Query: “Find the student that scores the second highest in Quiz1”
- $R1 := \gamma_{MAX(Score)} \rightarrow MaxScore(Quiz1)$
- $R2 := Quiz1 \bowtie_{Score = MaxScore} R1$
- $R3 := \Pi_{Name, Score}(R2)$
- $R4 := Quiz1 - R3$
- $R5 := \gamma_{MAX(Score)} \rightarrow 2ndMaxScore(R4)$
- $R6 := R4 \bowtie_{Score = 2ndMaxScore} R5$

# Exercise

R1

| Name  | Score | School |
|-------|-------|--------|
| Alice | 70    | SCSE   |
| Bob   | 90    | EEE    |
| Cathy | 80    | EEE    |
| David | 100   | SCSE   |

Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | EEE    |
| David | SCSE   |

- Query: “For each school, find the student that scores the highest in Quiz1”
- R1 := Quiz1  $\bowtie$  Student
- R2 :=  $\gamma_{\text{School}, \text{MAX}(\text{Score}) \rightarrow \text{MaxScore}}(R1)$
- 
-

# Exercise

R2

| School | MaxScore |
|--------|----------|
| SCSE   | 100      |
| EEE    | 90       |

Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | EEE    |
| David | SCSE   |

- Query: “For each school, find the student that scores the highest in Quiz1”
- R1 := Quiz1  $\bowtie$  Student
- R2 :=  $\gamma_{\text{School}, \text{MAX}(\text{Score}) \rightarrow \text{MaxScore}}(\text{R1})$
- R3 := R1  $\bowtie_{\text{R1.School} = \text{R2.School} \text{ AND } \text{Score} = \text{MaxScore}}$  R2
-

# Exercise

R3

| Name  | Score | School | MaxScore |
|-------|-------|--------|----------|
| Bob   | 90    | EEE    | 90       |
| David | 100   | SCSE   | 100      |

Quiz1

| Name  | Score |
|-------|-------|
| Alice | 70    |
| Bob   | 90    |
| Cathy | 80    |
| David | 100   |

Students

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | EEE    |
| David | SCSE   |

- Query: “For each school, find the student that scores the highest in Quiz1”
- R1 := Quiz1  $\bowtie$  Student
- R2 :=  $\gamma_{\text{School}, \text{MAX}(\text{Score}) \rightarrow \text{MaxScore}}(R1)$
- R3 := R1  $\bowtie_{R1.\text{School} = R2.\text{School} \text{ AND } \text{Score} = \text{MaxScore}}$  R2
- R3 :=  $\Pi_{\text{Name}, \text{Score}}(R3)$

# Exercise

**Grades**

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | NN     | B     |
| Cathy | SP     | B     |
| Cathy | NN     | A     |

**CrsSch**

| Course | School |
|--------|--------|
| DB     | SCSE   |
| DM     | SCSE   |
| NN     | EEE    |
| SP     | EEE    |

- Query: “Find the students who have taken all courses from SCSE”
- $R1 := \sigma_{\text{School} = \text{'SCSE'}} \text{CrsSch}$
- $R2 := \text{Grades} \bowtie R1$
- $R3 := \gamma_{\text{Name}, \text{COUNT}(\text{Course}) \rightarrow \text{CrsCNT}} (R2)$
- $R4 := \gamma_{\text{COUNT}(\text{Course}) \rightarrow \text{ScseCNT}} (R1)$
- $R5 := R3 \bowtie_{\text{CrsCNT} = \text{ScseCNT}} R4$
- $R6 := \Pi_{\text{Name}} (R5)$

# Exercise

**Grades**

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | NN     | B     |
| Cathy | SP     | B     |
| Cathy | NN     | A     |

**CrsSch**

| Course | School |
|--------|--------|
| DB     | SCSE   |
| DM     | SCSE   |
| NN     | EEE    |
| SP     | EEE    |

- Query: “For each school, find the students who have taken all courses in the school”
- $R1 := \text{Grades} \bowtie \text{CrsSch}$
- $R2 := \gamma_{\text{Name}, \text{School}, \text{COUNT}(\text{Course})} \rightarrow \text{CrsCNT} (R1)$
- $R3 := \gamma_{\text{School}, \text{COUNT}(\text{Course})} \rightarrow \text{CrsCNT} (\text{CrsSch})$
- $R4 := R2 \bowtie_{R2.\text{School} = R3.\text{School} \text{ AND } R2.\text{CrsCNT} = R3.\text{CrsCNT}} R3$

# CZ2007 Introduction to Database Systems (Week 6)

---

## Topic 5: Relational Algebra (3)



# Last Lecture

- Assignment:  $T_1 := \sigma_{A > 100} R_1$
- Rename:  $\rho_{\text{test}(A', B', C')} R_1$
- Duplicate Elimination  $\delta$
- Extended Projection  $\Pi$
- Grouping and Aggregation  $\gamma$

# This Lecture

- Division:  $\div$
- Left Outerjoin:  $\bowtie_L^o$  condition
- Right Outerjoin:  $\bowtie_R^o$  condition
- Full Outerjoin:  $\bowtie^o$

# Division $\div$

Owns

| Name  | Product |
|-------|---------|
| Alice | iPad    |
| Alice | iPhone  |
| Bob   | iPhone  |
| Cathy | iPad    |

AppleP

| Product |
|---------|
| iPhone  |
| iPad    |

Results

| Name  |
|-------|
| Alice |

- Query: “Find each person that owns all Apple products”
- $\text{Owns} \div \text{AppleP}$
- In general,  $R_1(A, B) \div R_2(B)$  returns a table that contains only A
- The table contains each A value in  $R_1$  that is associated with every B value in  $R_2$
- Intuitive interpretation: “Find the A that R<sub>1</sub> all the B in R<sub>2</sub>”
- Example: “Find the Name that Owns all the Product in AppleP”

# Division $\div$

Joins

| Name  | Club |
|-------|------|
| Alice | ABC  |
| Bob   | DEF  |
| Bob   | ABC  |
| Cathy | DEF  |

Clubs

| Club |
|------|
| ABC  |
| DEF  |

Results

| Name |
|------|
| Bob  |

- Query: “Find each person that has joined all clubs”
- Joins  $\div$  Clubs

# Division $\div$

Joins

| Name  | Club |
|-------|------|
| Alice | ABC  |
| Bob   | DEF  |
| Bob   | ABC  |
| Cathy | DEF  |

Clubs

| Name |
|------|
| ABC  |
| DEF  |

Results

Club

- Joins  $\div$  Clubs ?
- No result.
- In general,  $R_1(A, B) \div R_2(B)$  returns a table that contains only A

# Division $\div$

Owns

| Name  | Product |
|-------|---------|
| Alice | iPad    |
| Alice | iPhone  |
| Bob   | iPhone  |
| Cathy | iPad    |

AppleP

| Product | Price |
|---------|-------|
| iPhone  | 999   |
| iPad    | 699   |

Results

| Name  |
|-------|
| Alice |

- Query: “Find each person that owns all Apple products”
- Owns  $\div$  AppleP ?
- Wrong, since “Price” does not appear in “Owns”
- In general,  $R_1(A, B) \div R_2(B)$  returns a table that contains only A
- Correct answer: Owns  $\div (\Pi_{\text{Product}} \text{AppleP})$

# Division $\div$

**Owns**

| Name  | Since | Product |
|-------|-------|---------|
| Alice | 2013  | iPhone  |
| Alice | 2013  | iPad    |
| Bob   | 2013  | iPhone  |
| Bob   | 2010  | iPad    |

**AppleP**

| Product |
|---------|
| iPhone  |
| iPad    |

**Results**

| Name  | Since |
|-------|-------|
| Alice | 2013  |

- $\text{Owns} \div \text{AppleP}$
- The result is a table with attributes in **Owns** but not in **AppleP**, i.e., **Name** and **Since**
- The table contains every {**Name**, **Since**} combination that is associated with all **Product** in **AppleP**

# Exercise

Grades

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | NN     | B     |
| Cathy | SP     | B     |
| Cathy | NN     | A     |

CrsSch

| Course | School |
|--------|--------|
| DB     | SCSE   |
| DM     | SCSE   |
| NN     | EEE    |
| SP     | EEE    |

- Query: “Find the students who have taken all courses from SCSE”
- $R1 := \sigma_{School = 'SCSE'} CrsSch$
- $R2 := \Pi_{Course} R1$
- $R3 := \Pi_{Name, Course} (Grades)$
- $R4 := R3 \div R2$

# Exercise

**Grades**

| Name  | Course | Grade |
|-------|--------|-------|
| Alice | DB     | A     |
| Alice | DM     | C     |
| Bob   | DB     | B     |
| Bob   | NN     | B     |
| Cathy | SP     | B     |
| Cathy | NN     | A     |

**CrsSch**

| Course | School |
|--------|--------|
| DB     | SCSE   |
| DM     | SCSE   |
| NN     | EEE    |
| SP     | EEE    |

- Query: “Find the students who have taken all courses from SCSE but not all courses from EEE”
- $R1 := \Pi_{\text{Course}} (\sigma_{\text{School} = \text{'SCSE'}} \text{ CrsSch})$
- $R2 := \Pi_{\text{Name, Course}} (\text{Grades})$
- $R3 := R2 \div R1$
- $R4 := \Pi_{\text{Course}} (\sigma_{\text{School} = \text{'EEE'}} \text{ CrsSch})$
- $R5 := \Pi_{\text{Name, Course}} (\text{Grades})$
- $R6 := R3 - (R5 \div R4)$

# Left Outerjoin $\bowtie_L$ condition

**Students**

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

**Donations**

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

**Results**

| Name  | School | Amount |
|-------|--------|--------|
| Alice | SCSE   | NULL   |
| Bob   | EEE    | NULL   |
| Cathy | CEE    | 100    |
| David | SCSE   | 200    |

- Query: “For each student, find the amount of his/her donation”
- Students  $\bowtie_L$  Donations
- All tuples in Students are retained in the results
- For each student who has not made a donation, a “NULL” value is given as his/her Amount

# Left Outerjoin $\bowtie_L$ condition

Students

| SName | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

Results

| SName | Name  | School | Amount |
|-------|-------|--------|--------|
| Alice | NULL  | SCSE   | NULL   |
| Bob   | NULL  | EEE    | NULL   |
| Cathy | Cathy | CEE    | 100    |
| David | David | SCSE   | 200    |

- Query: “For each student, find the amount of his/her donation”
- Students  $\bowtie_L$  Sname = Name Donations
- Similar to theta joins in that all attributes are retained

# Right Outerjoin $\bowtie_R$ condition

| Students |        |
|----------|--------|
| Name     | School |
| Alice    | SCSE   |
| Bob      | EEE    |
| Cathy    | CEE    |
| David    | SCSE   |

| Donations |        |
|-----------|--------|
| Name      | Amount |
| Cathy     | 100    |
| David     | 200    |
| Eddie     | 300    |
| Fred      | 400    |

| Results |        |        |
|---------|--------|--------|
| Name    | School | Amount |
| Cathy   | CEE    | 100    |
| David   | SCSE   | 200    |
| Eddie   | NULL   | 300    |
| Fred    | NULL   | 400    |

- Query: “For each donator, find the school he/she is in”
- Students  $\bowtie_R$  Donations
- All tuples in Donations are retained in the results
- For each donator who is not students, a “NULL” value is given as his/her School

# Right Outerjoin $\bowtie_R$ condition

Students

| SName | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

Donations

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

Results

| SName | Name  | School | Amount |
|-------|-------|--------|--------|
| Cathy | Cathy | CEE    | 100    |
| David | David | SCSE   | 200    |
| NULL  | Eddie | NULL   | 300    |
| NULL  | Fred  | NULL   | 400    |

- Query: “For each donator, find the school he/she is in”
- Students  $\bowtie_R$  Sname = Name Donations
- Similar to theta joins in that all attributes are retained

# Full Outerjoin $\bowtie$ condition

**Students**

| Name  | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

**Donations**

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

**Results**

| Name  | School | Amount |
|-------|--------|--------|
| Alice | SCSE   | NULL   |
| Bob   | EEE    | NULL   |
| Cathy | CEE    | 100    |
| David | SCSE   | 200    |
| Eddie | NULL   | 300    |
| Fred  | NULL   | 400    |

- The combination of left and right outerjoins
- Students  $\bowtie$  Donations

# Full Outerjoin $\bowtie$ condition

**Students**

| SName | School |
|-------|--------|
| Alice | SCSE   |
| Bob   | EEE    |
| Cathy | CEE    |
| David | SCSE   |

**Donations**

| Name  | Amount |
|-------|--------|
| Cathy | 100    |
| David | 200    |
| Eddie | 300    |
| Fred  | 400    |

**Results**

| SName | Name  | School | Amount |
|-------|-------|--------|--------|
| Alice | NULL  | SCSE   | NULL   |
| Bob   | NULL  | EEE    | NULL   |
| Cathy | Cathy | CEE    | 100    |
| David | David | SCSE   | 200    |
| NULL  | Eddie | NULL   | 300    |
| NULL  | Fred  | NULL   | 400    |

- Students  $\bowtie_{Sname = Name}$  Donations

# Example

CastIn

| Name  | Movie     | Year |
|-------|-----------|------|
| John  | Batman    | 2012 |
| Steve | Batman    | 2012 |
| Meg   | The Women | 2008 |

Stars

| Name  | Gender | Birth |
|-------|--------|-------|
| John  | Male   | 1980  |
| Meg   | Female | 1981  |
| Steve | Male   | 1990  |

- For each movie, count the number of male movie stars that were cast in the movie
- $R1 := \sigma_{\text{Gender} = \text{'Male'}} \text{ Stars}$
- $R2 := \text{CastIn} \bowtie R1$
- Incomplete!
- “All female cast” movies not included

| Name  | Movie  | Year | Gender | Birth |
|-------|--------|------|--------|-------|
| John  | Batman | 2012 | Male   | 1980  |
| Steve | Batman | 2012 | Male   | 1990  |

# Example

CastIn

| Name  | Movie     | Year |
|-------|-----------|------|
| John  | Batman    | 2012 |
| Steve | Batman    | 2012 |
| Meg   | The Women | 2008 |

Stars

| Name  | Gender | Birth |
|-------|--------|-------|
| John  | Male   | 1980  |
| Meg   | Female | 1981  |
| Steve | Male   | 1990  |

- For each movie, count the number of male movie stars that were cast in the movie
- $R1 := \sigma_{\text{Gender} = \text{'Male'}} \text{ Stars}$
- $R2 := \text{CastIn} \bowtie_L R1$
- $R3 := \gamma_{\text{Movie}, \text{COUNT}(\text{Gender}) \rightarrow \text{MaleStars}} (R2)$

# Example

## CastIn

| Name  | Movie     | Year |
|-------|-----------|------|
| John  | Batman    | 2012 |
| Steve | Batman    | 2012 |
| Meg   | The Women | 2008 |

## R2

| Name  | Movie     | Year | Gender | Birth |
|-------|-----------|------|--------|-------|
| John  | Batman    | 2012 | Male   | 1980  |
| Steve | Batman    | 2012 | Male   | 1990  |
| Meg   | The Women | 2008 | NULL   | NULL  |

## Stars

| Name  | Gender | Birth |
|-------|--------|-------|
| John  | Male   | 1980  |
| Meg   | Female | 1981  |
| Steve | Male   | 1990  |

## R3

| Movie     | MaleStars |
|-----------|-----------|
| Batman    | 2         |
| The Women | 1         |



- $R2 := \text{CastIn} \circ_L R1$
- $R3 := \gamma_{\text{Movie}, \text{COUNT}(\text{Gender}) \rightarrow \text{MaleStars}}(R2)$

# Example

- For each movie, count the number of male movie stars that were cast in the movie
- $R1 := \sigma_{\text{Gender} = \text{'Male'}} \text{ Stars}$
- $R2 := \gamma_{\text{Name}, \text{COUNT}(\text{Gender}) \rightarrow \text{Male}} (R1)$
- $R3 := \text{CastIn} \bowtie_L R2$
- $R4 := \gamma_{\text{Movie}, \text{SUM}(\text{Male}) \rightarrow \text{MaleStars}} (R3)$

R2

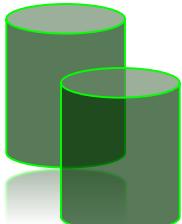
| Name  | Male |
|-------|------|
| John  | 1    |
| Steve | 1    |

R3

| Name  | Movie     | Year | Male |
|-------|-----------|------|------|
| John  | Batman    | 2012 | 1    |
| Steve | Batman    | 2012 | 1    |
| Meg   | The Women | 2008 | NULL |

R4

| Movie     | MaleStars |
|-----------|-----------|
| Batman    | 2         |
| The Women | 0         |



CZ2007

# Introduction to Databases

---

## Querying Relational Databases using SQL **Part-1**

---

**Cong Gao**

Professor

School of Computer Science and Engineering  
Nanyang Technological University, Singapore

# Why Should You Study Databases?

- Make more \$\$\$:
  - Startups need DB talent right away
  - Massive industry...



ORACLE

Microsoft

Google™

Spark 

- Intellectual (Research):
  - Science: data poor to data rich
    - No idea how to handle the data!
  - Fundamental ideas to/from all of CS:
    - Systems, theory, AI, logic, stats, analysis....

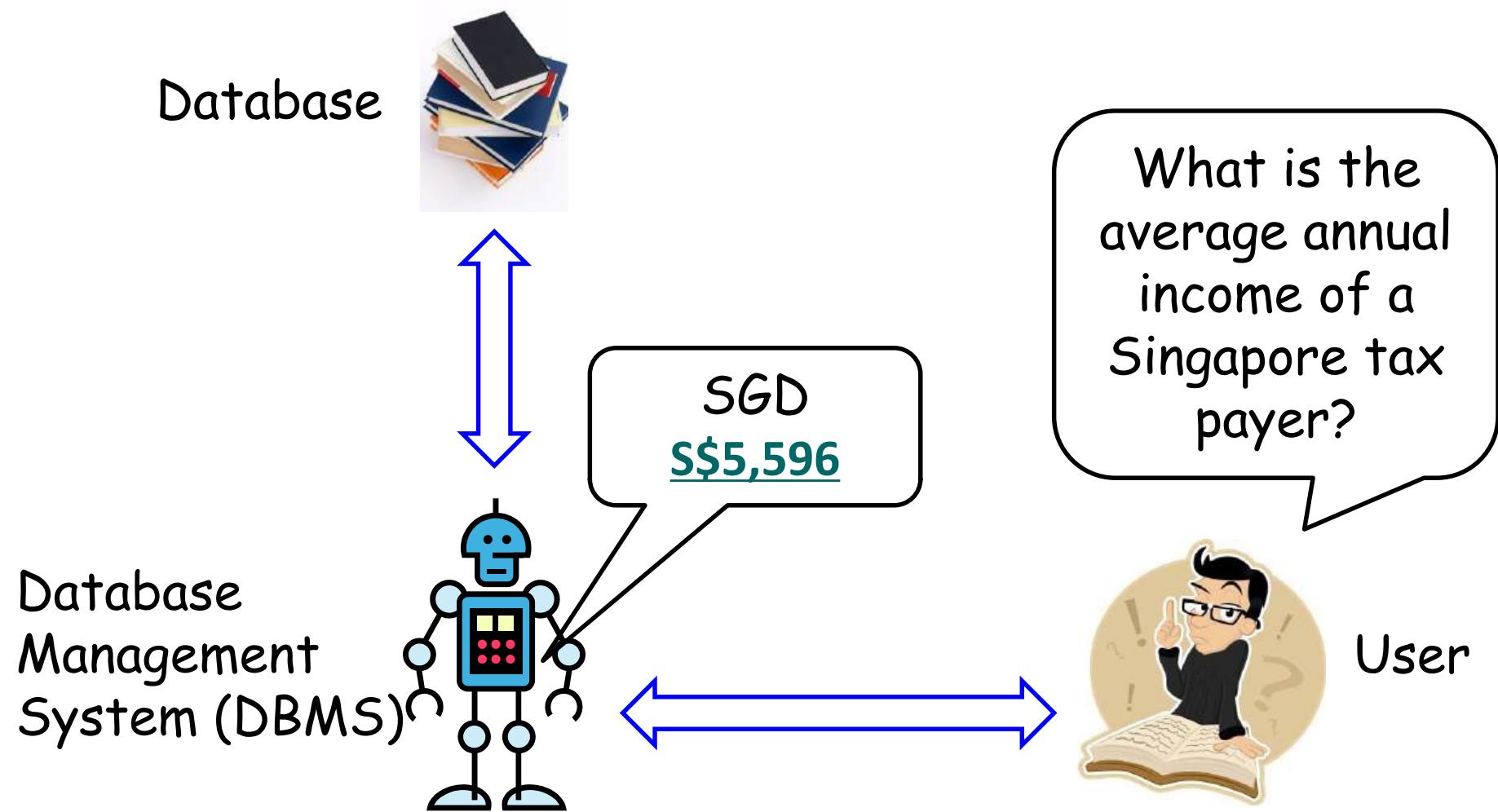
Many great computer systems ideas started in DB.

# Ask Questions!

The important thing is not to  
stop questioning.

Albert Einstein

# Database and DBMS



# Tables, Relations, Relational Model

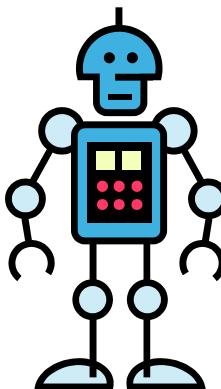
Database



| Taxpayer_ID | Annual_Income |
|-------------|---------------|
| 51248297    | 100000        |
| 33891634    | 50000         |
| ...         | ...           |

Income\_Table

Database  
Management  
System



User

# Tables, Relations, Relational Model

Database



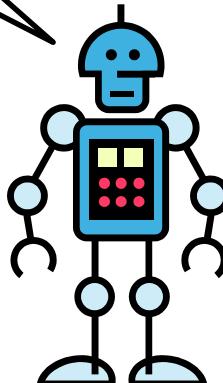
| Taxpayer_ID | Annual_Income |
|-------------|---------------|
| 51248297    | 100000        |
| 33891634    | 50000         |
| ...         | ...           |

Income\_Table

???

What is the average annual income of a Singapore tax payer?

Database Management System



User



# Structured Query Language (SQL)

Database

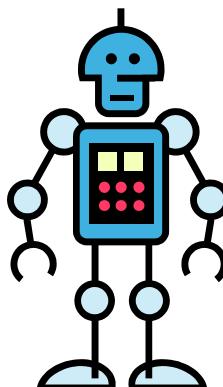


| Taxpayer_ID | Annual_Income |
|-------------|---------------|
| 51248297    | 100000        |
| 33891634    | 50000         |
| ...         | ...           |

Income\_Table

```
SELECT avg(Annual_Income)  
FROM Income_Table
```

Database Management System



User

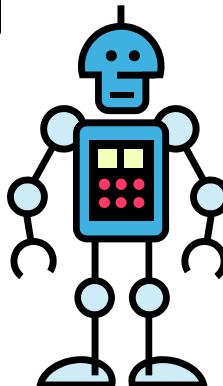
# Structured Query Language (SQL)

Database



More details about SQL will be covered in the course

Database Management System



| Taxpayer_ID | Annual_Income |
|-------------|---------------|
| 51248297    | 100000        |
| 33891634    | 50000         |
| ...         | ...           |

Income\_Table

```
SELECT avg(Annual_Income)  
FROM Income_Table
```

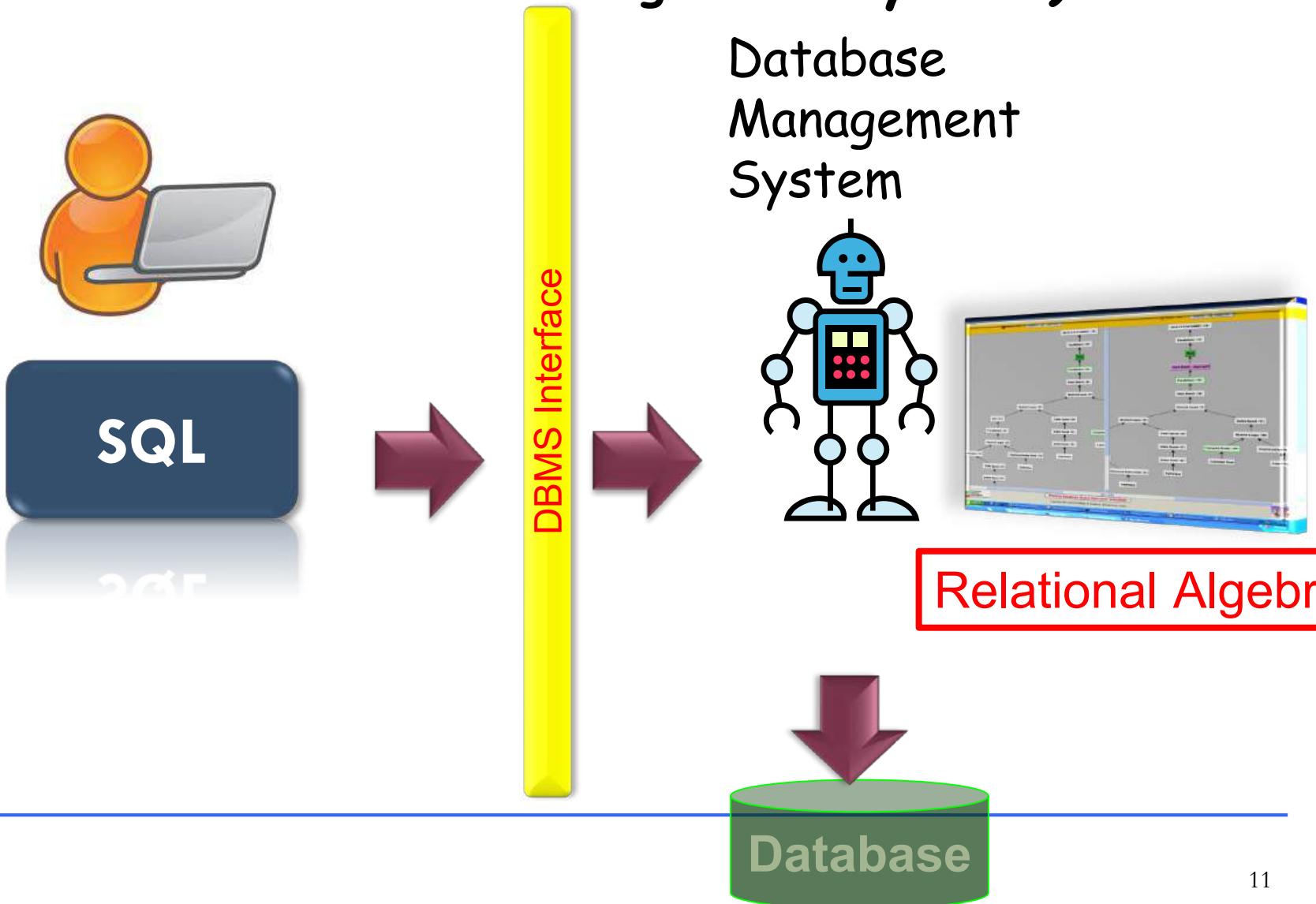


User



# Querying RDBMS

## (Relational Database Management System)



# Roadmap

- Next 9 lectures
  - Introduction to SQL
  - SELECT  
FROM  
WHERE
  - Eliminating duplicates
  - Renaming attributes
  - Expressions in SELECT Clause
  - Patterns for Strings
  - Ordering
  - Joins
  - ...

# What is SQL?

- Structured Query Language (SQL) – standard query language for relational databases. Pronounced “S-Q-L” or “sequel”
- A brief history:
  - First proposal of SEQUEL (IBM Research, System R, 1974)
  - First implementation in SQL/DS (IBM) and Oracle (1981)
  - Around 1983 there is a “de facto standard”
  - Became official standard in 1986 – defined by the American National Standards Institute (ANSI), and in 1987 – by the International Organization for Standardization (ISO)
  - ANSI SQL89
  - ANSI SQL92 (SQL2)
  - ANSI SQL99 (SQL3)
  - ANSI SQL 2003 (added OLAP, XML, etc.)
  - ANSI SQL 2006 (added more XML)
  - ... .... ...
  - ANSI SQL 2016 (added pattern matching, JSON, etc.)



# Present Days: Big Data

## Infrastructure

### Analytics



### Operational



### As A Service



### Structured DB



### Technologies



New technology. *Same SQL Principles*

# What SQL we shall study?

- All major database vendors (Oracle, IBM, Microsoft, Sybase) conform to SQL standard



- Although database companies have added “proprietary” extensions (**different dialects**)
- Commercial systems offer features that are not part of the standard
  - Incompatibilities between systems
  - Incompatibilities with newer standards (e.g. triggers in SQL:1999)

- We concentrate more on the principles
- (mostly) We will study SQL92 - a basic subset

# Good Practice for learning SQL

- Install a DBMS in your machine, such as PostgreSQL, mySQL, etc
  - Set up a database and tables with example data
  - Run SQL, debug yourself
- SQL server in school lab
- Google error (stackoverflow)
- Consult textbook
- Other sources:
  - An easy to use website:
    - <https://www.w3schools.com/sql/default.asp>
    - Can try to run SQL using the example database there
  - Comparison of different SQL implementations - by Troels Arvin (<http://troels.arvin.dk/db/rdbms/>)

# What we want to do with SQL?

- Manage and query the database (a set of relations / tables)

## What we want to do on the relations?

- Retrieve
- Insert
- Delete
- Update

# More about SQL

## Declarative Language

- SQL is a ***declarative language*** (non-procedural).
- A SQL query specifies *what* to retrieve but not *how* to retrieve it.



## What is a procedural language ??

- Procedure/ Functions
- Write instructions on *how* to do it
- C, C++, Java



## SQL is Not a complete programming language

- It does not have control or iteration commands.

# Stuffs supported by SQL

## Data Manipulation Language (DML)

- Perform queries
- Perform updates (add/ delete/ modify)



## Data Definition Language (DDL)

- Creates databases, tables, indices
- Create views
- Specify integrity constraints



## Embedded SQL

Wrap a high-level programming language around DML to do more sophisticated queries/updates

We shall not study  
this!

# Tables in SQL

- A relation or table is a multiset of tuples (rows) having the attributes specified by the schema
  - Schema: the name of a relation + the set of attributes

A multiset is an unordered list (or: a set with multiple duplicate instances allowed)

List: [1, 1, 2, 3]  
Set: {1, 2, 3}  
Multiset: {1, 1, 2, 3}

## Product

| <b>PName</b> | <b>Price</b> | <b>Category</b> | <b>Manufacturer</b> |
|--------------|--------------|-----------------|---------------------|
| iPhone x     | 888          | Phone           | Apple               |
| iPad         | 668          | Tablet          | Apple               |
| Mate 10      | 798          | Phone           | Huawei              |
| EOS 550D     | 1199         | Camera          | Canon               |

# Attributes (Columns) in a Table

An attribute (or column) is a **typed** data entry present in each tuple in the relation

## Product

| <u>PName</u> | Price | Category | Manufacturer |
|--------------|-------|----------|--------------|
| iPhone x     | 888   | Phone    | Apple        |
| iPad         | 668   | Tablet   | Apple        |
| Mate 10      | 798   | Phone    | Huawei       |
| EOS 550D     | 1199  | Camera   | Canon        |

# Tuples (Rows) in a Table

A tuple or row is a single entry in the table having the attributes specified by the schema

Also referred to sometimes as a record

## Product

| <u>PName</u> | Price | Category | Manufacturer |
|--------------|-------|----------|--------------|
| iPhone x     | 888   | Phone    | Apple        |
| iPad         | 668   | Tablet   | Apple        |
| Mate 10      | 798   | Phone    | Huawei       |
| EOS 550D     | 1199  | Camera   | Canon        |

# Data Types in SQL

- Character strings
  - CHAR(20)
  - VARCHAR(50)
  - ...
- Numbers
  - INT
  - FLOAT
  - ...
- Others
  - BOOLEAN
  - DATETIME
  - ...

Every attribute must have a type

| Product  |       |          |
|----------|-------|----------|
| PName    | Price | Category |
| iPhone x | 888   | Phone    |
| iPad     | 668   | Tablet   |
| Mate 10  | 798   | Phone    |
| EOS 550D | 1199  | Camera   |

# Key of a Table

## Product

| <u>PName</u> | Price | Category | Manufacturer |
|--------------|-------|----------|--------------|
| iPhone x     | 888   | Phone    | Apple        |
| iPad         | 668   | Tablet   | Apple        |
| Mate 10      | 798   | Phone    | Huawei       |
| EOS 550D     | 1199  | Camera   | Canon        |

- A **key** is an attribute whose values are unique; we underline a primary key

Product(Pname, Price, Category, Manufacturer)

# Principle Form of SQL

## Basic Structure of SQL

SELECT desired attributes ( $A_1, A_2, \dots, A_n$ )  
FROM one or more tables ( $R_1, R_2, \dots, R_m$ )  
WHERE condition about tuples of the tables ( $P$ )

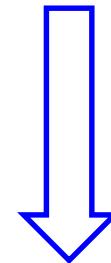
## Mapping to Relational Algebra

$$\Pi_{A_1, A_2, \dots, A_n} (\sigma_P (R_1 \times R_2 \times \dots \times R_m))$$

# Simple SQL Query

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT *
FROM Product
WHERE Category = 'Phone'
```



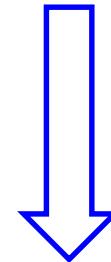
"selection"

| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone x | 888   | Phone    | Apple        |
| Mate 10  | 798   | Phone    | Huawei       |

# Simple SQL Query

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT *
FROM Product
WHERE Category <> 'Phone'
```

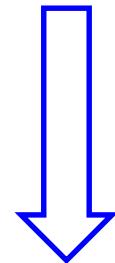


| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPad     | 668   | Tablet   | Apple        |
| EOS 550D | 1199  | Camera   | Canon        |

# Simple SQL Query

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT *
FROM Product
WHERE Category = 'Phone' AND Price > 800
```

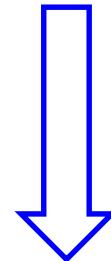


| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone x | 888   | Phone    | Apple        |

# Simple SQL Query

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT *
FROM Product
WHERE Category = 'Tablet' OR Price > 1000
```



| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPad     | 668   | Tablet   | Apple        |
| EOS 550D | 1199  | Camera   | Canon        |

# Simple SQL Query (**selection and projection**)

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT PName, Price, Manufacturer  
FROM Product  
WHERE Price > 800
```

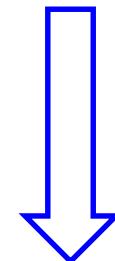
“selection  
and  
projection”

| PName    | Price | Manufacturer |
|----------|-------|--------------|
| iPhone x | 888   | Apple        |
| EOS 550D | 1199  | Canon        |

# Simple SQL Query (WHERE Clause: BETWEEN)

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT PName, Price, Manufacturer  
FROM Product  
WHERE Price BETWEEN 800 AND 1200
```

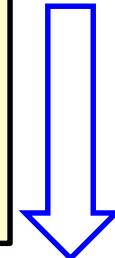


| PName    | Price | Manufacturer |
|----------|-------|--------------|
| iPhone x | 888   | Apple        |
| EOS 550D | 1199  | Canon        |

# Simple SQL Query (WHERE Clause: IN)

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT PName, Price, Manufacturer  
FROM Product  
WHERE Manufacturer IN ('Huawei', 'Canon')
```



| PName    | Price | Manufacturer |
|----------|-------|--------------|
| Mate 10  | 798   | Huawei       |
| EOS 550D | 1199  | Canon        |

# SQL Syntax

- There is a set of *reserved words* that cannot be used as names for table name or attribute name. For example, **SELECT, FROM, WHERE**, etc.
- Use single quotes for constants:
  - 'abc' - Okay
  - "abc" - Not okay
- SQL is generally *case-insensitive*.
  - Exception: is string constants. 'FRED' not the same as 'fred'.
- White-space is ignored
- All statements end with a semicolon (;

# Summary and Roadmap

- Introduction to SQL
- SELECT  
FROM  
WHERE

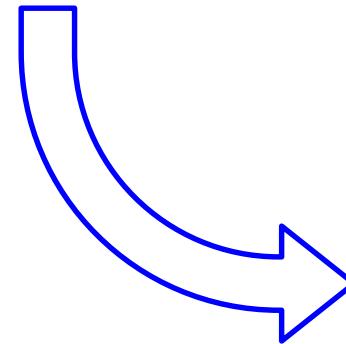
- Next
  - Eliminating duplicates
  - Renaming attributes
  - Expressions in SELECT Clause
  - Patterns for Strings
  - Ordering
  - Joins
  - ...

Reference: Chapter 6.1 of the Book  
"Database Systems: The Complete Book;  
Hector Garcia-Molina Jeffrey D. Ullman,  
Jennifer Widom

# Eliminating Duplicates

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT Category  
FROM Product
```

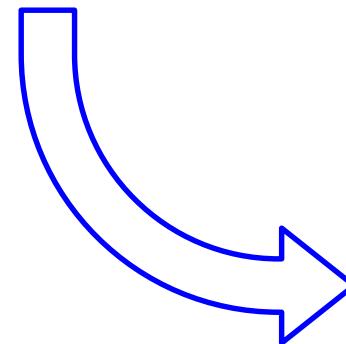


| Category |
|----------|
| Phone    |
| Tablet   |
| Phone    |
| Camera   |

# Eliminating Duplicates (cont.)

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT DISTINCT Category  
FROM Product
```

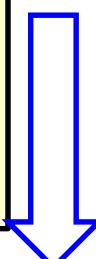


| Category |
|----------|
| Phone    |
| Tablet   |
| Camera   |

# AS: Renaming Attributes

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT PName AS Product, Price AS Cost, Manufacturer  
FROM Product  
WHERE Category = 'Phone'
```



| Product  | Cost | Manufacturer |
|----------|------|--------------|
| iPhone x | 888  | Apple        |
| EOS 550D | 1199 | Canon        |

# Expressions in SELECT Clause

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

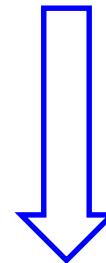
```
SELECT PName, Price*1.4 AS Cost_IN_SGD, Manufacturer  
FROM Product  
WHERE Category = 'Phone'
```

| Product  | Cost_IN_SGD | Manufacturer |
|----------|-------------|--------------|
| iPhone x | 1243.2      | Apple        |
| EOS 550D | 1678.6      | Canon        |

# Patterns for Strings

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT *
FROM Product
WHERE PName LIKE 'iPh%'
```



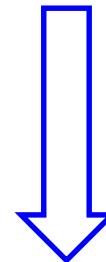
| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone x | 888   | Phone    | Apple        |

% stands for "any string"

# Patterns for Strings

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT *
FROM Product
WHERE PName LIKE '%Phone x%'
```



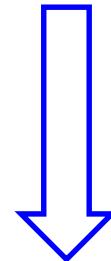
| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone x | 888   | Phone    | Apple        |

% stands for "any string"

# Patterns for Strings

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT *
FROM Product
WHERE PName LIKE '%P%e%'
```



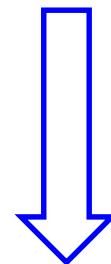
| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone x | 888   | Phone    | Apple        |

% stands for "any string"

# Patterns for Strings

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT *
FROM Product
WHERE PName LIKE '_Phone x'
```



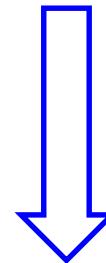
| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone x | 888   | Phone    | Apple        |

\_ stands for "any character"

# Patterns for Strings

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT *
FROM Product
WHERE PName LIKE '_Phone_'
```



| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone x | 888   | Phone    | Apple        |

\_ stands for "any single character"

# Patterns for Strings

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT *
FROM Product
WHERE PName NOT LIKE '_Phone_'
```

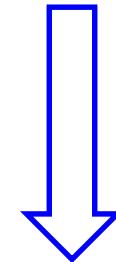
# More example patterns

- '\_\_\_' - Matches any string of exactly three characters
- '\_\_\_%' - Matches any string of at least three characters
- 'ab\%cd%' - Match all strings beginning with "ab%cd"

# Ordering the Results (cont.)

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT PName, Price  
FROM Product  
WHERE Price < 800  
ORDER BY PName DESC
```

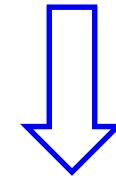


| PName   | Price |
|---------|-------|
| iPad    | 668   |
| Mate 10 | 798   |

# Ordering the Results (cont.)

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT PName, Category  
FROM Product  
WHERE Price < 1000  
ORDER BY Category, PName
```

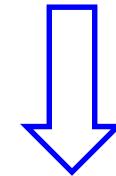


| PName     | Category |
|-----------|----------|
| Milestone | Phone    |
| iPhone x  | Phone    |
| iPad      | Tablet   |

# Ordering the Results (cont.)

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT PName, Category  
FROM Product  
WHERE Price < 1000  
ORDER BY Category DESC,  
PName
```

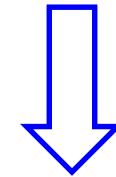


| PName    | Category |
|----------|----------|
| iPad     | Tablet   |
| Mate 10  | Phone    |
| iPhone x | Phone    |

# Ordering the Results (cont.)

| Product | PName    | Price | Category | Manufacturer |
|---------|----------|-------|----------|--------------|
|         | iPhone x | 888   | Phone    | Apple        |
|         | iPad     | 668   | Tablet   | Apple        |
|         | Mate 10  | 798   | Phone    | Huawei       |
|         | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT PName, Category  
FROM Product  
WHERE Price < 1000  
ORDER BY Category DESC,  
PName DESC
```



| PName    | Category |
|----------|----------|
| iPad     | Tablet   |
| iPhone x | Phone    |
| Mate 10  | Phone    |

# Exercise

Product

|  | PName    | Price | Category | Manufacturer |
|--|----------|-------|----------|--------------|
|  | iPhone x | 888   | Phone    | Apple        |
|  | iPad     | 668   | Tablet   | Apple        |
|  | Mate 10  | 798   | Phone    | Motorola     |
|  | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT DISTINCT Category  
FROM Product  
ORDER BY Category
```

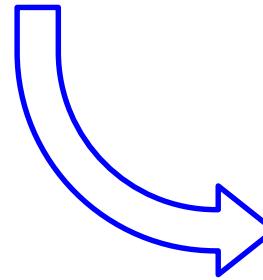
?

# Exercise

## Product

| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone x | 888   | Phone    | Apple        |
| iPad     | 668   | Tablet   | Apple        |
| Mate 10  | 798   | Phone    | Huawei       |
| EOS 550D | 1199  | Camera   | Canon        |

```
SELECT DISTINCT Category  
FROM Product  
ORDER BY Category
```



| Category |
|----------|
| Camera   |
| Phone    |
| Tablet   |

# Exercise

Product

|  | PName    | Price | Category | Manufacturer |
|--|----------|-------|----------|--------------|
|  | iPhone x | 888   | Phone    | Apple        |
|  | iPad     | 668   | Tablet   | Apple        |
|  | Mate 10  | 798   | Phone    | Huawei       |
|  | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
WHERE Price < 1000
```

?

# Exercise

## Product

|  | PName    | Price | Category | Manufacturer |
|--|----------|-------|----------|--------------|
|  | iPhone x | 888   | Phone    | Apple        |
|  | iPad     | 668   | Tablet   | Apple        |
|  | Mate 10  | 798   | Phone    | Huawei       |
|  | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT DISTINCT Category  
FROM Product  
ORDER BY Category  
WHERE Price < 1000
```

Error!

- “WHERE” should always proceed “ORDER BY”

# Exercise

Product

|  | PName    | Price | Category | Manufacturer |
|--|----------|-------|----------|--------------|
|  | iPhone x | 888   | Phone    | Apple        |
|  | iPad     | 668   | Tablet   | Apple        |
|  | Mate 10  | 798   | Phone    | Huawei       |
|  | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT DISTINCT Category  
FROM Product  
ORDER BY PName
```

?

# Exercise

Product

|  | PName    | Price | Category | Manufacturer |
|--|----------|-------|----------|--------------|
|  | iPhone x | 888   | Phone    | Apple        |
|  | iPad     | 668   | Tablet   | Apple        |
|  | Mate 10  | 798   | Phone    | Huawei       |
|  | EOS 550D | 1199  | Camera   | Canon        |

```
SELECT DISTINCT Category  
FROM Product  
ORDER BY PName
```

Error!

- “ORDER BY” items must appear in the select list if “SELECT DISTINCT” is specified

# Joins

Company

Product

| CName  | StockPrice | Country |
|--------|------------|---------|
| Canon  | 45         | Japan   |
| Huawei | 1          | China   |
| Apple  | 374        | USA     |

| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone x | 888   | Phone    | Apple        |
| iPad     | 668   | Tablet   | Apple        |
| Mate 10  | 798   | Phone    | Huawei       |
| EOS 550D | 1199  | Camera   | Canon        |

- A user wants to know the names and prices of all products by Japan companies. How?

# Joins

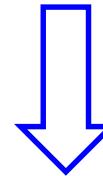
Product

| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone x | 888   | Phone    | Apple        |
| iPad     | 668   | Tablet   | Apple        |
| Mate 10  | 798   | Phone    | Huawei       |
| EOS 550D | 1199  | Camera   | Canon        |

Company

| CName  | Stock Price | Country |
|--------|-------------|---------|
| Apple  | 374         | USA     |
| Huawei | 1           | China   |
| Canon  | 45          | Japan   |

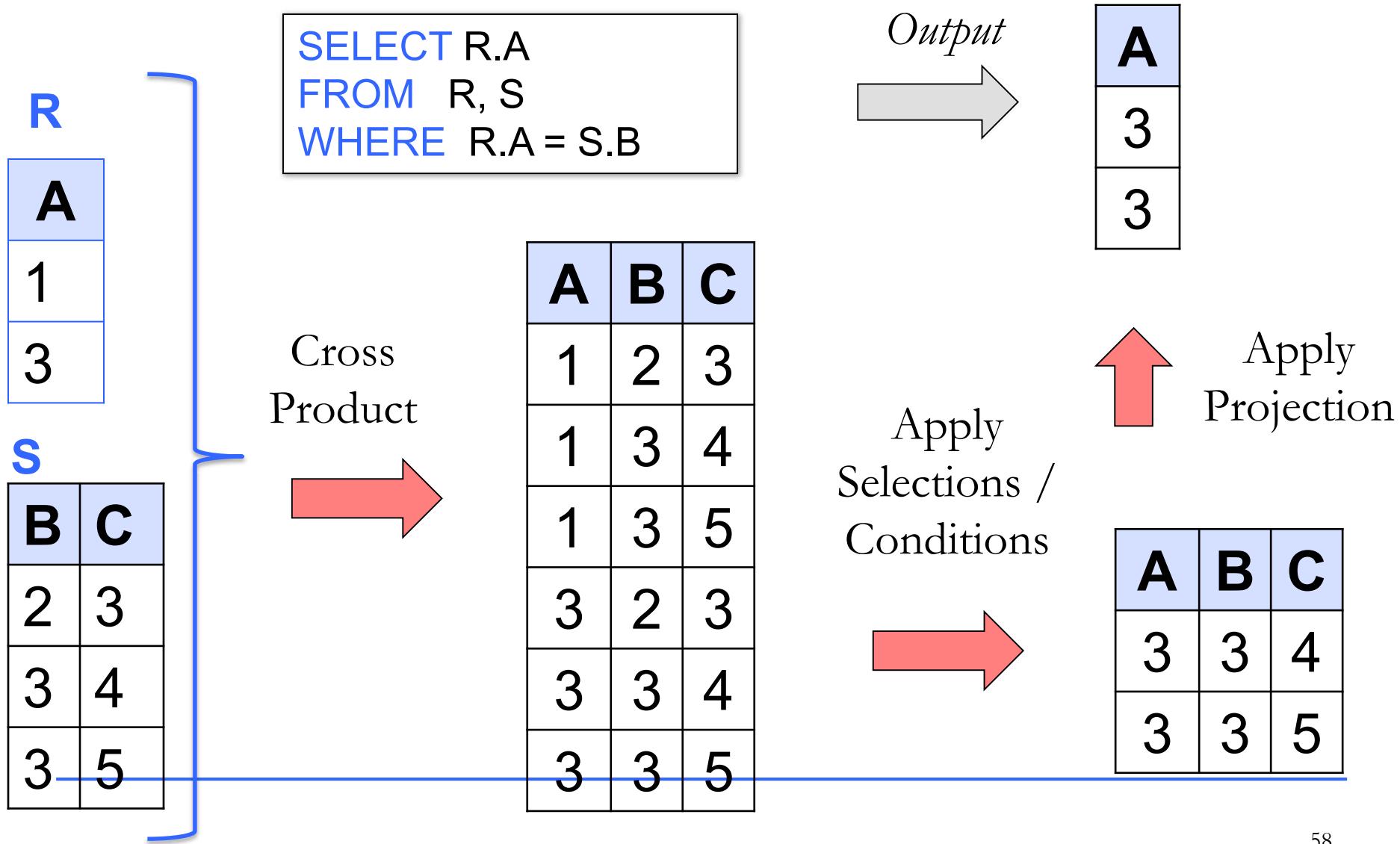
- SELECT PName, Price  
FROM Product, Company  
WHERE Country = 'Japan'  
AND Manufacturer = CName



| PName    | Price |
|----------|-------|
| EOS 550D | 1199  |

# Meaning (Semantics) of Join

## – An Example



# Summary of Meaning (Semantics) of Join

1. Take **cross product**:

$$X = R \times S$$

```
SELECT R.A  
FROM R, S  
WHERE R.A = S.B
```

2. Apply **selections / conditions**:

= Filtering!

3. Apply **projections** to get final output:

= Returning only *some* attributes

Remembering this order is critical

# How Join is Actually Executed in a Database System?

- The preceding slides show what a join means (i.e., semantics)
- Not actually how the DBMS executes it under the covers

We shall not study it in this course  
– will be discussed in CZ 4031

# Joins

Person

| PName | Address | WorksFor |
|-------|---------|----------|
| ...   | ...     | ...      |

Company

| CName | Address | Country |
|-------|---------|---------|
| ...   | ...     | ...     |

- Find the names of the persons who work for companies in USA
- ```
SELECT PName  
      FROM Person, Company  
     WHERE Country = 'USA'  
       AND WorksFor = CName
```

# Joins

Person

| PName | Address | WorksFor |
|-------|---------|----------|
| ...   | ...     | ...      |

Company

| CName | Address | Country |
|-------|---------|---------|
| ...   | ...     | ...     |

- Find the names of the persons who work for companies in USA, as well as their company addresses
- SELECT PName, Address  
FROM Person, Company  
WHERE Country = 'USA'  
AND WorksFor = CName

Error!

# Joins

Person

| PName | Address | WorksFor |
|-------|---------|----------|
| ...   | ...     | ...      |

Company

| CName | Address | Country |
|-------|---------|---------|
| ...   | ...     | ...     |

- Find the names of the persons who work for companies in USA, as well as their company addresses
- ```
SELECT PName, Company.Address
  FROM Person, Company
 WHERE Country = 'USA'
   AND WorksFor = CName
```

# Joins

Person

| PName | Address | CName |
|-------|---------|-------|
| ...   | ...     | ...   |

Company

| CName | Address | Country |
|-------|---------|---------|
| ...   | ...     | ...     |

- Find the names of the persons who work for companies in USA, as well as their company addresses
- ```
SELECT PName, Company.Address  
FROM Person, Company  
WHERE Country = 'USA'  
AND CName = CName
```

Error!

# Joins

## Person

| <u>PName</u> | Address | CName |
|--------------|---------|-------|
| ...          | ...     | ...   |

## Company

| <u>CName</u> | Address | Country |
|--------------|---------|---------|
| ...          | ...     | ...     |

- Find the names of the persons who work for companies in USA, as well as their company addresses
- ```
SELECT PName, Company.Address
      FROM Person, Company
     WHERE Country = 'USA'
          AND Person.CName = Company.CName
```

# Joins

Person

| PName | Address | CName |
|-------|---------|-------|
| ...   | ...     | ...   |

Company

| CName | Address | Country |
|-------|---------|---------|
| ...   | ...     | ...     |

- Find the names of the persons who work for companies in USA, as well as their company addresses
- ```
SELECT X.PName, Y.Address
FROM Person AS X, Company AS Y
WHERE Y.Country = 'USA'
AND X.CName = Y.CName
```

# Joins

Person

| <u>PName</u> | Address | CName |
|--------------|---------|-------|
| ...          | ...     | ...   |

Company

| <u>CName</u> | Address | Country |
|--------------|---------|---------|
| ...          | ...     | ...     |

- Find the names of the persons who work for companies in USA, as well as their company addresses
- ```
SELECT X.PName, Y.Address
FROM Person X, Company Y
WHERE Y.Country = 'USA'
AND X.CName = Y.CName
```

# Exercise

| Company | CName | StockPrice | Country |
|---------|-------|------------|---------|
| ...     | ...   | ...        | ...     |

| Product | PName | Price | Category | Manufacturer |
|---------|-------|-------|----------|--------------|
| ...     | ...   | ...   | ...      | ...          |

- Exercise: Find the names of the companies in China that produce products in the 'tablet' category
- ```
SELECT DISTINCT CName
  FROM Company, Product
 WHERE Manufacturer = CName
   AND Country = 'China'
   AND Category = 'Tablet'
```

# Exercise

| Company |
|---------|
| ...     |

| CName | StockPrice | Country |
|-------|------------|---------|
| ...   | ...        | ...     |

| Product |
|---------|
| ...     |

| PName | Price | Category | Manufacturer |
|-------|-------|----------|--------------|
| ...   | ...   | ...      | ...          |

- Exercise: Find the names of the companies in China that produce products in the 'tablet' or 'phone' category
- ```
SELECT DISTINCT CName
  FROM Company, Product
 WHERE Manufacturer = CName
   AND Country = 'China'
   AND (Category = 'Tablet'
        OR Category = 'Phone')
```

# Exercise

Product

| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone X | 888   | Phone    | Apple        |
| iPad     | 668   | Tablet   | Apple        |
| Mate 10  | 798   | Phone    | Huawei       |
| EOS 550D | 1199  | Camera   | Canon        |

- Exercise: Find the manufacturers that produce products in both the 'tablet' and 'phone' categories
- ```
SELECT DISTINCT Manufacturer
FROM Product
WHERE Category = 'Tablet'
      AND Category = 'Phone'
```

Error!

# Exercise

Product

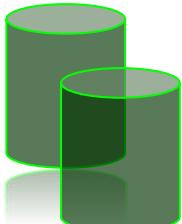
| PName    | Price | Category | Manufacturer |
|----------|-------|----------|--------------|
| iPhone X | 888   | Phone    | Apple        |
| iPad     | 668   | Tablet   | Apple        |
| Mate 10  | 798   | Phone    | Huawei       |
| EOS 550D | 1199  | Camera   | Canon        |

- Exercise: Find the manufacturers that produce products in both the 'tablet' and 'phone' categories
- ```
SELECT DISTINCT X.Manufacturer
FROM Product AS X, Product AS Y
WHERE X.Manufacturer = Y.Manufacturer
AND X.Category = 'Tablet'
AND Y.Category = 'Phone'
```

# Summary and roadmap

- Introduction to SQL
  - SELECT  
FROM  
WHERE
  - Eliminating duplicates
  - Renaming attributes
  - Expressions in SELECT Clause
  - Patterns for Strings
  - Ordering
  - Joins
- Next
    - Subquery
    - Aggregations
    - UNION, INTERSECT, EXCEPT
    - NULL
    - Outerjoin
    - ...

Reference: Chapter 6.1&6.2 of the Book  
"Database Systems: The Complete Book;  
Hector Garcia-Molina Jeffrey D. Ullman,  
Jennifer Widom



CZ2007

# Introduction to Databases

---

## Querying Relational Databases using SQL

### Part-2

---

**Cong Gao**

Professor

School of Computer Science and Engineering  
Nanyang Technological University, Singapore

# Summary and roadmap

- Introduction to SQL
- SELECT  
FROM  
WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Next
  - Subquery
  - Aggregations
  - UNION, INTERSECT, EXCEPT
  - NULL
  - Outerjoin
  - ...

# Subqueries

- A subquery is an SQL query nested inside a larger query
- Queries with subqueries are referred to as **nested queries**
- A subquery may occur in
  - SELECT
  - FROM **SQL subquery**
  - WHERE **SQL subquery**

# A special subquery: Scalar Subquery

## Scalar Subquery

- return a **single value** which is then used in a comparison.
- If query is written so that it expects a subquery to return a single value, and it returns multiple values or no values, a **run-time error** occurs.

## Example Query

From **Sells(bar, beer, price)**, find the bars that serve **Heineken** for the same price **Junior bar** charges for **Tiger**.

# Example Scalar Subquery

Sells

Find the price **Junior** charges for **Tiger**.

| Bar    | Beer     | Price |
|--------|----------|-------|
| Clinic | Heineken | 8.00  |
| Clinic | Tiger    | 6.60  |
| Junior | Tiger    | 7.90  |
| MOS    | Heineken | 7.90  |
| Junior | Heineken | 8.00  |

SELECT  
FROM  
WHERE



| Price |
|-------|
| 7.90  |

price  
Sells  
bar = 'Junior'  
AND beer = 'Tiger';

Find the bars that serve **Heineken** at that price.

SELECT  
FROM  
WHERE  
AND

bar  
Sells  
beer = 'Heineken'

= 7.90;

| Bar |
|-----|
| MOS |

# Example Scalar Subquery

```
SELECT    bar
FROM      Sells
WHERE     beer = 'Heineken' AND
          price = (SELECT price
                    FROM    Sells
                    WHERE   bar = 'Junior'
                            AND beer = 'Tiger') ;
```

# Without using Scalar Subquery, how ?

```
SELECT      S1.bar  
FROM        Sells S1, Sells S2  
WHERE       S1.beer = 'Heineken'  
AND         S2.bar = 'Junior'  
AND         S2.beer = 'Tiger'  
AND         S1.price = S2.price;
```

Use two copies of the  
table

# Subqueries in FROM

| Company | CName | StockPrice | Country  |       |
|---------|-------|------------|----------|-------|
| ...     | ...   | ...        | ...      |       |
| Product | PName | Price      | Category | CName |
| ...     | ...   | ...        | ...      | ...   |

- Find all products in the 'phone' category with prices under 1000
- ```
SELECT X.PName
FROM (SELECT *
      FROM Product
      WHERE category = 'Phone') AS X
WHERE X.Price < 1000
```

# Subqueries in FROM (cont.)

| Company |
|---------|
|         |

|     | CName | StockPrice | Country |
|-----|-------|------------|---------|
| ... | ...   | ...        | ...     |

| Product |
|---------|
|         |

|     | PName | Price | Category | CName |
|-----|-------|-------|----------|-------|
| ... | ...   | ...   | ...      | ...   |

- Find all products in the 'phone' category with prices under 1000
- ```
SELECT PName
  FROM Product
 WHERE Category = 'Phone'
       AND Price < 1000
```
- This is a much more efficient solution

# Subqueries in WHERE (cont.)

| Company | CName | StockPrice | Country  |       |
|---------|-------|------------|----------|-------|
| ...     | ...   | ...        | ...      |       |
| Product | PName | Price      | Category | CName |
| ...     | ...   | ...        | ...      | ...   |

- Find all companies that make some products with price < 100
- ```
SELECT DISTINCT CName
FROM Company AS X
WHERE X.CName IN
      (SELECT Y.CName
       FROM Product AS Y
       WHERE Y.Price < 100)
```

# Subqueries in WHERE (cont.)

Company

| CName | StockPrice | Country |
|-------|------------|---------|
| ...   | ...        | ...     |

Product

| PName | Price | Category | CName |
|-------|-------|----------|-------|
| ...   | ...   | ...      | ...   |

- Find all companies that make some products with price < 100
- ```
SELECT DISTINCT CName
FROM Company AS X
WHERE X.CName IN
      (SELECT *
       FROM Product AS Y
       WHERE Y.Price < 100)
```

Error!

- The number of attributes in the **SELECT** clause in the subquery must match the number of attributes compared to with the comparison operator.<sup>11</sup>

# Subqueries in WHERE (cont.)

Company

| CName | StockPrice | Country |
|-------|------------|---------|
| ...   | ...        | ...     |

Product

| PName | Price | Category | CName |
|-------|-------|----------|-------|
| ...   | ...   | ...      | ...   |

- Find all companies that make some products with price < 100
- ```
SELECT DISTINCT CName
FROM Company AS X
WHERE EXISTS
    (SELECT * FROM Product AS Y
     WHERE X.CName = Y.Cname
       AND Y.Price < 100)
```
- A nested query is **correlated** with the outer query if it contains a reference to an attribute in the outer query.
- A nested query is **correlated** with the outside query if it must be re-computed for every tuple produced by the outside query.

# Subqueries in WHERE (cont.)

| Company |
|---------|
| ...     |

| CName | StockPrice | Country |
|-------|------------|---------|
| ...   | ...        | ...     |

| Product |
|---------|
| ...     |

| PName | Price | Category | CName |
|-------|-------|----------|-------|
| ...   | ...   | ...      | ...   |

- Find all companies that make some products with price < 100
- ```
SELECT DISTINCT CName
FROM Company AS X
WHERE 100 > ANY
      (SELECT Price FROM Product AS Y
       WHERE X.CName = Y.Cname)
```

# Subqueries in WHERE (cont.)

| Company |
|---------|
| ...     |

|     | CName | StockPrice | Country |
|-----|-------|------------|---------|
| ... | ...   | ...        | ...     |

| Product |
|---------|
| ...     |

|     | PName | Price | Category | CName |
|-----|-------|-------|----------|-------|
| ... | ...   | ...   | ...      | ...   |

- Find all companies that make some products with price < 100
- ```
SELECT DISTINCT CName
  FROM Product
 WHERE Price < 100
```
- This is more efficient than the previous solutions

# Operators in Subqueries

## IN

$x \in R$  is true if and only if the tuple  $x$  is a member of the relation  $R$ .

## EXISTS

- $\exists t \in R \text{ such that } P(t)$  is true if and only if the relation  $R$  is not empty.
- Returns true if the nested query has 1 or more tuples.

## ANY/SOME

$x = \text{ANY}(R)$  is a boolean cond. meaning that  $x$  equals at least one tuple in the relation  $R$ .

## ALL

$x \neq \text{ALL}(R)$  is true if and only if for every tuple  $t$  in the relation,  $x$  is not equal to  $t$ .

**Note:** The keyword NOT can proceed any of the operators ( $s$  NOT IN  $R$ )

# Avoiding Nested Queries

- In general, nested queries tend to be more inefficient than un-nested queries
  - query optimizers of DBMS **do not generally do a good job** at optimizing queries containing subqueries
- Therefore, they should be avoided whenever possible
- But there are cases where avoiding nested queries is hard...

# Subqueries in WHERE (cont.)

| Company |
|---------|
| ...     |

| CName | StockPrice | Country |
|-------|------------|---------|
| ...   | ...        | ...     |

| Product |
|---------|
| ...     |

| PName | Price | Category | CName |
|-------|-------|----------|-------|
| ...   | ...   | ...      | ...   |

- Find all companies that do not make any product with price < 100
- ```
SELECT DISTINCT CName
FROM Company AS X
WHERE NOT EXISTS
    (SELECT * FROM Product AS Y
     WHERE X.CName = Y.Cname
       AND Y.Price < 100)
```

# Subqueries in WHERE (cont.)

| Company |
|---------|
| ...     |

| CName | StockPrice | Country |
|-------|------------|---------|
| ...   | ...        | ...     |

| Product |
|---------|
| ...     |

| PName | Price | Category | CName |
|-------|-------|----------|-------|
| ...   | ...   | ...      | ...   |

- Find all companies that do not make any product with price < 100
- ```
SELECT DISTINCT CName
  FROM Company AS X
 WHERE 100 <= ALL
        (SELECT Price FROM Product AS Y
 WHERE X.CName = Y.Cname)
```

# Subquery - Rules to Remember

- The **ORDER BY** clause may not be used in a subquery.
- Column names in a subquery refer to the table name in the **FROM** clause of the subquery by default.

# Summary and roadmap

- Introduction to SQL
- SELECT  
FROM  
WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Next
  - Aggregations
  - UNION, INTERSECT, EXCEPT
  - NULL
  - Outerjoin
  - ...

Reference: Chapter 6.3 of our TextBook

# Aggregation

Cars

| Model   | Maker  | Price |
|---------|--------|-------|
| Corolla | Toyota | 1000  |
| E89     | BMW    | 2000  |
| ...     | ...    | ...   |

- What is the average price of the models from Toyota?
- How many models are there from BMW?

# Aggregation: Count

| Cars | Model   | Maker  | Price |
|------|---------|--------|-------|
|      | Corolla | Toyota | 1000  |
|      | E89     | BMW    | 2000  |
|      | ...     | ...    | ...   |

- Count the number of car models from Toyota:
- `SELECT COUNT(*)  
FROM Cars  
WHERE Maker = 'Toyota'`

# Aggregation: Count (cont.)

| Cars | Model   | Maker  | Price |
|------|---------|--------|-------|
|      | Corolla | Toyota | 1000  |
|      | E89     | BMW    | 2000  |
|      | i8      | BMW    | 50    |

- Count the number of car makers:
- SELECT COUNT(Maker)  
FROM Cars

Error!

# Aggregation: Count (cont.)

| Cars | Model   | Maker  | Price |
|------|---------|--------|-------|
|      | Corolla | Toyota | 1000  |
|      | E89     | BMW    | 2000  |
|      | i8      | BMW    | 50    |

- Count the number of car makers:
- `SELECT COUNT(DISTINCT Maker)  
FROM Cars`

# Aggregation: Average

| Cars | Model   | Maker  | Price |
|------|---------|--------|-------|
|      | Corolla | Toyota | 1000  |
|      | E89     | BMW    | 2000  |
|      | ...     | ...    | ...   |

- Compute the average price of car models from Toyota:
- ```
SELECT AVG(Price)
FROM Cars
WHERE Maker = 'Toyota'
```

# Aggregation: Min

| Cars | Model   | Maker  | Price |
|------|---------|--------|-------|
|      | Corolla | Toyota | 1000  |
|      | E89     | BMW    | 2000  |
|      | ...     | ...    | ...   |

- Compute the minimum price of car models from Toyota:
- `SELECT MIN(Price)  
FROM Cars  
WHERE Maker = 'Toyota'`

# Aggregation: Max

| Cars | Model   | Maker  | Price |
|------|---------|--------|-------|
|      | Corolla | Toyota | 1000  |
|      | E89     | BMW    | 2000  |
|      | ...     | ...    | ...   |

- Compute the maximum price of car models from Toyota:
- `SELECT MAX(Price)  
FROM Cars  
WHERE Maker = 'Toyota'`

# Aggregation: Sum

| Cars | Model   | Maker  | Price |
|------|---------|--------|-------|
|      | Corolla | Toyota | 1000  |
|      | E89     | BMW    | 2000  |
|      | ...     | ...    | ...   |

- Compute the sum of prices of car models from Toyota:
- `SELECT SUM(Price)  
FROM Cars  
WHERE Maker = 'Toyota'`

# Aggregation: Sum (cont.)

Purchase

| Product | Date     | Price | Quantity |
|---------|----------|-------|----------|
| Orange  | 2011.1.1 | 3     | 10       |
| Banana  | 2011.1.1 | 2     | 5        |
| Orange  | 2011.1.2 | 5     | 10       |
| Banana  | 2011.1.2 | 1     | 20       |
| Banana  | 2011.1.3 | 4     | 15       |

- Compute the gross sales of oranges:
- ```
SELECT SUM(Price * Quantity)
FROM Purchase
WHERE Product = 'Orange'
```

# Aggregation: Sum (cont.)

Purchase

| Product | Date     | Price | Quantity |
|---------|----------|-------|----------|
| Orange  | 2011.1.1 | 3     | 10       |
| Banana  | 2011.1.1 | 2     | 5        |
| Orange  | 2011.1.2 | 5     | 10       |
| Banana  | 2011.1.2 | 1     | 20       |
| Banana  | 2011.1.3 | 4     | 15       |

- Is it possible to obtain this?
- Yes
- Use **GROUP BY**

| Product | GrossSales |
|---------|------------|
| Orange  | 80         |
| Banana  | 90         |

# Aggregation: Group By

Purchase

| Product | Date     | Price | Quantity |
|---------|----------|-------|----------|
| Orange  | 2011.1.1 | 3     | 10       |
| Banana  | 2011.1.1 | 2     | 5        |
| Orange  | 2011.1.2 | 5     | 10       |
| Banana  | 2011.1.2 | 1     | 20       |
| Banana  | 2011.1.3 | 4     | 15       |

- SELECT Product, SUM(Price \* Quantity)  
AS GrossSales  
FROM Purchase  
GROUP BY Product

| Product | GrossSales |
|---------|------------|
| Orange  | 80         |
| Banana  | 90         |

# Aggregation: Group By (cont.)

| Purchase | Product | Date     | Price | Quantity |
|----------|---------|----------|-------|----------|
|          | Orange  | 2011.1.1 | 3     | 10       |
|          | Banana  | 2011.1.1 | 2     | 5        |
|          | Orange  | 2011.1.2 | 5     | 10       |
|          | Banana  | 2011.1.2 | 1     | 20       |
|          | Banana  | 2011.1.3 | 4     | 15       |

- SELECT Product, Date, SUM(Price \* Quantity) AS GrossSales,  
FROM Purchase  
GROUP BY Product, Date

| Product | Date     | GrossSales |
|---------|----------|------------|
| Banana  | 2011.1.1 | 10         |
| Orange  | 2011.1.1 | 30         |
| ...     | ...      | ...        |

# Aggregation: Group By (cont.)

Purchase

|  | Product | Date     | Price | Quantity |
|--|---------|----------|-------|----------|
|  | Orange  | 2011.1.1 | 3     | 10       |
|  | Banana  | 2011.1.1 | 2     | 5        |
|  | Orange  | 2011.1.2 | 5     | 10       |
|  | Banana  | 2011.1.2 | 1     | 20       |
|  | Banana  | 2011.1.3 | 4     | 15       |

- SELECT Product, SUM(Price \* Quantity)  
AS GrossSales  
FROM Purchase  
**WHERE Price >= 2**  
GROUP BY Product

| Product | GrossSales |
|---------|------------|
| Orange  | 80         |
| Banana  | 70         |

# Aggregation: Group By (cont.)

Purchase

| Product | Date     | Price | Quantity |
|---------|----------|-------|----------|
| Orange  | 2011.1.1 | 3     | 10       |
| Banana  | 2011.1.1 | 2     | 5        |
| Orange  | 2011.1.2 | 5     | 10       |
| Banana  | 2011.1.2 | 1     | 20       |
| Banana  | 2011.1.3 | 4     | 15       |

- SELECT Product, SUM(Quantity) AS TotalQuan,  
Max(Price) as MaxPrice  
FROM Purchase  
GROUP BY Product

| Product | TotalQuan | MaxPrice |
|---------|-----------|----------|
| Orange  | 20        | 5        |
| Banana  | 40        | 4        |

# Aggregation: Having (cont.)

Purchase

| Product | Date     | Price | Quantity |
|---------|----------|-------|----------|
| Orange  | 2011.1.1 | 3     | 10       |
| Banana  | 2011.1.1 | 2     | 5        |
| Orange  | 2011.1.2 | 5     | 10       |
| Banana  | 2011.1.2 | 1     | 20       |
| Banana  | 2011.1.3 | 4     | 15       |

- SELECT Product, SUM(Quantity) AS TotalQuan  
FROM Purchase  
GROUP BY Product  
**HAVING Max(Price) > 4**

| Product | TotalQuan | MaxPrice |
|---------|-----------|----------|
| Orange  | 20        | 5        |

# Aggregation: Having (cont.)

Purchase

| Product | Date     | Price | Quantity |
|---------|----------|-------|----------|
| Orange  | 2011.1.1 | 3     | 10       |
| Banana  | 2011.1.1 | 2     | 5        |
| Orange  | 2011.1.2 | 5     | 10       |
| Banana  | 2011.1.2 | 1     | 20       |
| Banana  | 2011.1.3 | 4     | 15       |

- SELECT Product, SUM(Quantity) AS TotalQuan  
FROM Purchase  
GROUP BY Product  
**HAVING** Max(Price) > 4  
OR TotalQuan > 20

| Product | TotalQuan | MaxPrice |
|---------|-----------|----------|
| Orange  | 20        | 5        |
| Banana  | 40        | 4        |

# Common Mistakes

Purchase

| Product | Date     | Price | Quantity |
|---------|----------|-------|----------|
| Orange  | 2011.1.1 | 3     | 10       |
| Banana  | 2011.1.1 | 2     | 5        |
| Orange  | 2011.1.2 | 5     | 10       |
| Banana  | 2011.1.2 | 1     | 20       |
| Banana  | 2011.1.3 | 4     | 15       |

- SELECT Product, Date,  
SUM(Quantity) AS TotalQuan  
FROM Purchase  
GROUP BY Product Error!
- Anything in the SELECT list should either be (i)  
an aggregate function or (ii) in the GROUP BY list

# Common Mistakes (cont.)

Purchase

| Product | Date     | Price | Quantity |
|---------|----------|-------|----------|
| Orange  | 2011.1.1 | 3     | 10       |
| Banana  | 2011.1.1 | 2     | 5        |
| Orange  | 2011.1.2 | 5     | 10       |
| Banana  | 2011.1.2 | 1     | 20       |
| Banana  | 2011.1.3 | 4     | 15       |

- SELECT Product, Date,  
SUM(Quantity) AS TotalQuan  
FROM Purchase  
GROUP BY Product  
HAVING Price > 2 Error!
- The HAVING clause specifies conditions on each group,  
but not conditions on each tuple. Anything in Having  
should either be (i) an aggregate function or (ii) in the  
GROUP BY list

# HAVING vs. Where

Purchase(product, date, price, quantity)

- Find total sales after 10/1/2005 per product, for those products that have more than 100 buyers.

```
SELECT product, SUM(price*quantity)
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
HAVING SUM(quantity) > 100
```

HAVING clauses contains conditions on **groups**

Whereas WHERE clauses condition on *individual tuples*...

# A summary of Grouping and Aggregation

```
SELECT      S
FROM        R1,...,Rn
WHERE       C1
GROUP BY    a1,...,ak
HAVING      C2
```

## Evaluation steps:

1. Evaluate **FROM-WHERE**: apply condition  $C_1$  on the attributes in  $R_1, \dots, R_n$
2. **GROUP BY** the attributes  $a_1, \dots, a_k$
3. **Apply condition  $C_2$  to each group (may have aggregates)**
4. Compute aggregates in  $S$  and return the result

# Exercise

Author

| UserName | RealName |
|----------|----------|
| ...      | ...      |

Wrote

| UserName | Article |
|----------|---------|
| ...      | ...     |

- Find the number of usernames under each real name
- ```
SELECT COUNT(*)
FROM Author
GROUP BY RealName
```

# Exercise

Author

| UserName | RealName |
|----------|----------|
| ...      | ...      |

Wrote

| UserName | Article |
|----------|---------|
| ...      | ...     |

- Find the number of articles written by each user, group by real names
- ```
SELECT RealName, COUNT(*)
  FROM Author AS A, Wrote AS W
 WHERE A.UserName = W.UserName
 GROUP BY RealName
```

# Exercise

Author

| UserName | RealName |
|----------|----------|
| ...      | ...      |

Wrote

| UserName | Article |
|----------|---------|
| ...      | ...     |

- Find the real names of the persons who wrote more than 10 articles
- ```
SELECT RealName  
FROM Author AS A, Wrote AS W  
WHERE A.UserName = W.UserName  
GROUP BY RealName  
HAVING COUNT(*) > 10
```

# Exercise

Author

| UserName | RealName |
|----------|----------|
| ...      | ...      |

Wrote

| UserName | Article |
|----------|---------|
| ...      | ...     |

- Find the real names of the persons who wrote more than 10 articles
- ```
SELECT DISTINCT RealName
FROM Author AS A
WHERE 10 <
      (SELECT COUNT(Wrote.Article)
       FROM Wrote AS W
       WHERE A.UserName = W.UserName)
```

Error: it only counts the number of articles of each username.  
However, a realname may have multiple usernames

# Summary and roadmap

- Introduction to SQL
- SELECT  
FROM  
WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Aggregations
- Next
  - UNION, INTERSECT, EXCEPT
  - NULL
  - Outerjoin
  - ...

# Union

Author

| UserName | RealName |
|----------|----------|
| ...      | ...      |

Wrote

| UserName | Article |
|----------|---------|
| ...      | ...     |

- Find the usernames (i) with real names starting with 'Chris' OR (ii) have written more than 10 articles
- (SELECT A.UserName  
FROM Author AS A  
WHERE RealName LIKE 'Chris%')  
**UNION**  
(SELECT W.UserName  
FROM Wrote AS W  
GROUP BY W.UserName  
HAVING COUNT(\*) > 10)  
(subquery)  
**UNION**  
(subquery)

# Union

## Author

| UserName | RealName |
|----------|----------|
| ...      | ...      |

## Wrote

| UserName | Article |
|----------|---------|
| ...      | ...     |

- Find the usernames (i) with real names starting with 'Chris' OR (ii) have written more than 10 articles
- ```
(SELECT A.UserName  
FROM Author AS A  
WHERE RealName LIKE 'Chris%')  
UNION  
(SELECT W.UserName  
FROM Wrote AS W  
GROUP BY W.UserName  
HAVING COUNT(*) > 10)
```

Note: UNION  
automatically  
removes  
duplicates

# Union

Author

| UserName | RealName |
|----------|----------|
| ...      | ...      |

Wrote

| UserName | Article |
|----------|---------|
| ...      | ...     |

- Find the usernames (i) with real names starting with 'Chris' OR (ii) have written more than 10 articles
- ```
(SELECT *
  FROM Author AS A
  WHERE RealName LIKE 'Chris%')
UNION
(SELECT *
  FROM Wrote AS W
  GROUP BY W.UserName
  HAVING COUNT(*) > 10)
```

Error!

# Intersect (Not in some DBMS)

Author

| UserName | RealName |
|----------|----------|
| ...      | ...      |

Wrote

| UserName | Article |
|----------|---------|
| ...      | ...     |

- Find the usernames (i) with real names starting with 'Chris' AND (ii) have written more than 10 articles

- (SELECT A.UserName  
FROM Author AS A  
WHERE RealName LIKE 'Chris%')

**INTERSECT**

(SELECT W.UserName  
FROM Wrote AS W  
GROUP BY W.UserName  
HAVING COUNT(\*) > 10)

# Except (Not in some DBMS)

Author

| UserName | RealName |
|----------|----------|
| ...      | ...      |

Wrote

| UserName | Article |
|----------|---------|
| ...      | ...     |

- Find the usernames who wrote more than 10 articles but do not have a real name starting with 'Chris'
- ```
(SELECT W.UserName  
      FROM Wrote AS W  
      GROUP BY W.UserName  
      HAVING COUNT(*) > 10)  
EXCEPT  
(SELECT A.UserName  
      FROM Author AS A  
      WHERE A.RealName LIKE 'Chris%')
```

# Alternative solution?

Author

| UserName | RealName |
|----------|----------|
| ...      | ...      |

Wrote

| UserName | Article |
|----------|---------|
| ...      | ...     |

- Find the usernames who wrote more than 10 articles but do not have a real name starting with 'Chris'
- ```
SELECT W.UserName
  FROM Wrote AS W, Author AS A
 WHERE RealName NOT LIKE 'Chris%'
       AND W.UserName = A.UserName
 GROUP BY W.UserName
 HAVING COUNT(*) > 10)
```

# Exercise

- Player (PID, PName, Ranking, Age)  
Court (CID, Type, Location)  
Reserves (PID, CID, Date)
- Find the names of the players who have reserved courts of 'Clay' type
- ```
SELECT PName
  FROM Player, Court, Reserves
 WHERE Player.PID = Reserves.PID AND
       Court.CID = Reserves.CID AND
       Type='Clay';
```

# Exercise

- Player (PID, PName, Ranking, Age)  
Court (CID, Type, Location)  
Reserves (PID, CID, Date)
- Find the PID of the players who have reserved 'Clay' courts but not 'Grass' courts
- (SELECT R1.PID  
FROM Court AS C1, Reserves AS R1  
WHERE R1.CID = C1.CID AND Type = 'Clay')  
EXCEPT  
(SELECT R2.PID  
FROM Court AS C2, Reserves AS R2  
WHERE R2.CID = C2.CID AND Type = 'Grass')

# Exercise

- Player (PID, PName, Ranking, Age)  
Court (CID, Type, Location)  
Reserves (PID, CID, Date)
- Find the PIDs of players who have a ranking of 3 or who have reserved court with CID 100
- ```
(SELECT Player.PID
  FROM Player
 WHERE Ranking = 3)
UNION
(SELECT Reserves.PID
  FROM Reserves
 WHERE CID = 100)
```

# Exercise

- Player (PID, PName, Ranking, Age)  
Court (CID, Type, Location)  
Reserves (PID, CID, Date)
- Find the PIDs of players who have NOT reserved a 'Clay' court before
- ```
(SELECT Player.PID
      FROM   Player)
EXCEPT
(SELECT Reserves.PID
      FROM   Reserves, Court
      WHERE  Reserves.CID = Court.CID
            AND Type = 'Clay')
```

# Exercise

- Player (PID, PName, Ranking, Age)  
Court (CID, Type, Location)  
Reserves (PID, CID, Date)
- Find the PIDs of players who have reserved each court at least once
- ```
SELECT P.PID
FROM Player AS P
WHERE NOT EXISTS
  ((SELECT C.CID
    FROM Court AS C
    EXCEPT
    (SELECT R.CID
      FROM Reserves AS R
      WHERE R.PID = P.PID) )
```

# A bit theory: Bag Semantics vs. Set Semantics

- Set semantics → No duplicates, each item appears only once
  - Default for **UNION**, **INTERSECT**, and **EXCEPT** is **set**
- Bag semantics → Duplicates allowed, i.e., a multiset
  - Default for **SELECT-FROM-WHERE** is **bag**

## How to change the default?

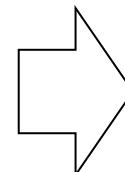
- Force set semantics with **DISTINCT** after **SELECT**
- Force bag semantics with **ALL** after **UNION**, etc.

# DISTINCT: Change Bag Semantics to Set Semantics

## Product

| PName    | Price | Category |
|----------|-------|----------|
| iPhone x | 888   | Phone    |
| iPad     | 668   | Tablet   |
| Mate 10  | 798   | Phone    |
| EOS 550D | 1199  | Camera   |

```
SELECT DISTINCT Category  
FROM Product
```

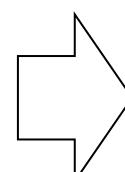


## Category

|        |
|--------|
| Phone  |
| Tablet |
| Camera |

Versus

```
SELECT Category  
FROM Product
```



## Category

|        |
|--------|
| Phone  |
| Tablet |
| Phone  |
| Camera |

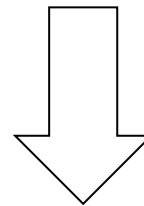
# ALL: Change Set Semantics to BAG Semantics

Product\_A

| PName    | Price |
|----------|-------|
| iPhone x | 888   |
| iPad     | 668   |
| Mate 10  | 798   |
| EOS 550D | 1199  |

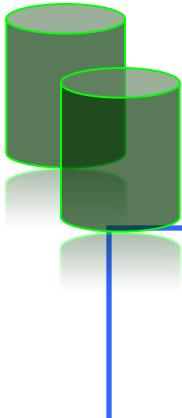
Product\_B

| PName    | Price |
|----------|-------|
| iPhone x | 888   |
| Mate 20  | 798   |



(SELECT \*  
FROM Product\_A)  
UNION ALL  
(SELECT \*  
FROM Product\_)

| PName    | Price |
|----------|-------|
| iPhone x | 888   |
| iPad     | 668   |
| Mate 10  | 798   |
| EOS 550D | 1199  |
| iPhone x | 888   |
| Mate 20  | 798   |



# CZ2007

# Introduction to Databases

---

## Querying Relational Databases using SQL

### Part--3

---

Cong Gao

Professor

School of Computer Science and Engineering  
Nanyang Technological University, Singapore

# Summary and roadmap

- Introduction to SQL
- SELECT  
FROM  
WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Aggregations
- UNION, INTERSECT, EXCEPT
- Next
  - NULL
  - Outerjoin
  - Insert/Delete tuples
  - Create/Alter/Delete tables
  - Constraints (primary key)
  - Views
  - More constraints
  - Triggers
  - Indexes

# NULL in SQL

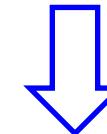
- In SQL, whenever we want to leave a value blank, we set it as **NULL**
- The DBMS regards NULL as “an unknown value”
- This makes sense but it leads to a lot of complications...

# Issues with NULL

- Any arithmetic operations involving NULL would result in NULL

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |

- ```
SELECT Price * 10
      FROM Product
     WHERE PName = 'iPad 2'
```



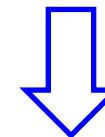
| Price |
|-------|
| 6680  |

# Issues with NULL (cont.)

- Any arithmetic operations involving NULL would result in NULL

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |

- ```
SELECT Price * 10
      FROM Product
     WHERE PName = 'iPhone xx'
```



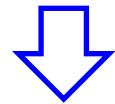
| Price |
|-------|
| NULL  |

# Issues with NULL (cont.)

- Any comparison involving NULL results in FALSE

Product

| PName     | Price |
|-----------|-------|
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |



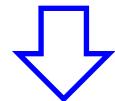
- SELECT \*  
FROM Product  
WHERE Price < 1000

| PName    | Price |
|----------|-------|
| iPhone 4 | 888   |
| iPad 2   | 668   |

# Issues with NULL (cont.)

- Any comparison involving NULL results in FALSE

| Product      |       |
|--------------|-------|
| <u>PName</u> | Price |
| iPhone 4     | 888   |
| iPad 2       | 668   |
| iPhone xx    | NULL  |
| EOS 550D     | 1199  |



- ```
SELECT *
  FROM Product
 WHERE Price >= 1000
```

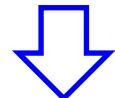
| <u>PName</u> | Price |
|--------------|-------|
| EOS 550D     | 1199  |

# Issues with NULL (cont.)

- Any comparison involving NULL results in FALSE

```
SELECT *  
FROM Product  
WHERE Price < 1000  
    OR Price >= 1000
```

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |



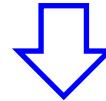
| PName    | Price |
|----------|-------|
| iPhone 4 | 888   |
| iPad 2   | 668   |
| EOS 550D | 1199  |

# Issues with NULL (cont.)

- Any comparison involving NULL results in FALSE

```
■ SELECT *  
    FROM Product  
   WHERE Price < 1000  
     OR Price >= 1000  
     OR Price = NULL
```

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |



| PName    | Price |
|----------|-------|
| iPhone 4 | 888   |
| iPad 2   | 668   |
| EOS 550D | 1199  |

# Issues with NULL (cont.)

- Any comparison involving NULL results in FALSE
- Use IS NULL to check whether a value is NULL
- ```
SELECT *
FROM Product
WHERE Price < 1000
      OR Price >= 1000
      OR Price IS NULL
```

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |

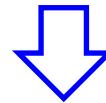
↓

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |

# Issues with NULL (cont.)

- Any comparison involving NULL results in FALSE

| Product      |       |
|--------------|-------|
| <u>PName</u> | Price |
| iPhone 4     | 888   |
| iPad 2       | 668   |
| iPhone xx    | NULL  |
| EOS 550D     | 1199  |



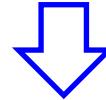
| <u>PName</u> | Price |
|--------------|-------|
|--------------|-------|

- ```
SELECT *
  FROM Product
 WHERE Price <> NULL
```

# Issues with NULL (cont.)

- Any comparison involving NULL results in FALSE
- Use IS NOT NULL to check whether a value is not NULL
- ```
SELECT *  
FROM Product  
WHERE Price IS NOT NULL
```

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |



| PName    | Price |
|----------|-------|
| iPhone 4 | 888   |
| iPad 2   | 668   |
| EOS 550D | 1199  |

# Issues with NULL (cont.)

- What about GROUP BY?
- NULLs are taken into account in group formation
- ```
SELECT Price,
      COUNT(*) AS Cnt
FROM Product
GROUP BY Price
```

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |

↓

| Price | Cnt |
|-------|-----|
| NULL  | 1   |
| 668   | 1   |
| 888   | 1   |
| 1199  | 1   |

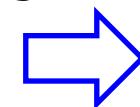
# Issues with NULL (cont.)

- What about joins?

| Phone     |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPhone xx | NULL  |

| Tablet  |       |
|---------|-------|
| PName   | Price |
| ipad 2  | 668   |
| IdeaPad | NULL  |

- SELECT P.PName, T.PName  
FROM Phone P, Tablet T  
WHERE P.Price > T.Price



| PName    | PName  |
|----------|--------|
| iPhone 4 | ipad 2 |

# Issues with NULL (cont.)

- What about joins?

| Phone     |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPhone xx | NULL  |

| Tablet  |       |
|---------|-------|
| PName   | Price |
| ipad 2  | 668   |
| IdeaPad | NULL  |

- SELECT P.PName, T.PName  
FROM Phone P, Tablet T  
WHERE P.Price = T.Price



# Issues with NULL (cont.)

- NULLs are ignored in
  - SUM,
  - MIN,
  - MAX

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |

- SELECT SUM(Price) as SumPrice  
FROM Product

→ 

| SumPrice |
|----------|
| 2755     |

# Issues with NULL (cont.)

- NULLs are ignored in AVG

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| iPhone xx | NULL  |
| EOS 550D  | 1199  |

- SELECT AVG(Price) as AvgPrice  
FROM Product

→

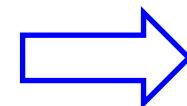
| AvgPrice |
|----------|
| 918      |

# Issues with NULL (cont.)

- SQL ignores NULLs when counting the number of values in a column

| Product  |       |
|----------|-------|
| PName    | Price |
| iPhone 4 | 888   |
| iPad 2   | 668   |
| NULL     | NULL  |
| EOS 550D | 1199  |

- `SELECT COUNT(Price)  
FROM Product`



3

# Issues with NULL (cont.)

- But NULLs are still counted in COUNT(\*)

| Product  |       |
|----------|-------|
| PName    | Price |
| iPhone 4 | 888   |
| iPad 2   | 668   |
| NULL     | NULL  |
| EOS 550D | 1199  |

- SELECT COUNT(\*)  
FROM Product

→ 4

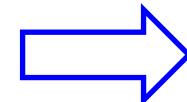
# Issues with NULL (cont.)

- But NULLs are still counted in COUNT(\*)

Avoid NULLs in tables whenever possible.  
This can usually be achieved with proper schema design.

| Product   |       |
|-----------|-------|
| PName     | Price |
| iPhone 4  | 888   |
| iPad 2    | 668   |
| Milestone | NULL  |
| EOS 550D  | 1199  |

- SELECT COUNT(\*)  
FROM Product



4

# Summary and roadmap

- Introduction to SQL
  - SELECT  
FROM  
WHERE
  - Eliminating duplicates
  - Renaming attributes
  - Expressions in SELECT Clause
  - Patterns for Strings
  - Ordering
  - Joins
  - Subquery
  - Aggregations
  - UNION, INTERSECT, EXCEPT
  - NULL
- Next
- Outerjoin
  - Insert/Delete tuples
  - Create/Alter/Delete tables
  - Constraints (primary key)
  - Views
  - More constraints
  - Triggers
  - Indexes

Reference: Chapter 6.1 of our TextBook

# Join

Product

PName

Price

iPhone 4

888

Sold

PName

Shop

iPhone 4

Suntec

- SELECT P.PName, Price, Shop  
FROM Product AS P, Sold AS S  
WHERE P.PName = S.PName

| PName    | Price | Shop   |
|----------|-------|--------|
| iPhone 4 | 888   | Suntec |

- SELECT P.PName, Price, Shop  
FROM Product AS P **JOIN** Sold AS S  
**ON** P.PName = S.PName

# Join

Product

| PName    | Price |
|----------|-------|
| iPhone 4 | 888   |
| iPad 2   | 668   |

Sold

PName

| PName    | Shop   |
|----------|--------|
| iPhone 4 | Suntec |

| PName    | Price | Shop   |
|----------|-------|--------|
| iPhone 4 | 888   | Suntec |

- SELECT P.PName, Price, Shop  
FROM Product AS P **JOIN** Sold AS S  
**ON** P.PName = S.PName

# Outerjoins

Product

PName

Price

iPhone 4

888

iPad 2

668

Sold

PName

Shop

iPhone 4

Suntec

How?

| PName    | Price | Shop   |
|----------|-------|--------|
| iPhone 4 | 888   | Suntec |
| iPad 2   | 668   | NULL   |

- SELECT P.PName, Price, Shop  
FROM Product AS P **JOIN** Sold AS S  
**ON** P.PName = S.PName

# Outerjoins (cont.)

Product

PName

Price

iPhone 4

888

iPad 2

668

Sold

PName

Shop

iPhone 4

Suntec

Include the left tuples even when there is no match

| PName    | Price | Shop   |
|----------|-------|--------|
| iPhone 4 | 888   | Suntec |
| iPad 2   | 668   | NULL   |

- SELECT P.PName, Price, Shop  
FROM Product AS P **LEFT OUTER JOIN** Sold AS S  
**ON** P.PName = S.PName

# Outerjoins (cont.)

Product

PName

Price

iPhone 4

888

Sold

PName

Shop

iPhone 4

Suntec

NULL

ION

Include the right tuples even when there is no match

| PName    | Price | Shop   |
|----------|-------|--------|
| iPhone 4 | 888   | Suntec |
| NULL     | NULL  | ION    |

- SELECT P.PName, Price, Shop  
FROM Product AS P **RIGHT OUTER JOIN** Sold AS S  
**ON** P.PName = S.PName

# Outerjoins (cont.)

Product

PName

Price

iPhone 4

888

iPad 2

668

Sold

PName

Shop

iPhone 4

Suntec

NULL

ION

Include both left  
and right tuples  
even if there is  
no match

| PName    | Price | Shop   |
|----------|-------|--------|
| iPhone 4 | 888   | Suntec |
| iPad 2   | 668   | NULL   |
| NULL     | NULL  | ION    |

- SELECT P.PName, Price, Shop  
FROM Product AS P **FULL OUTER JOIN** Sold AS S  
**ON** P.PName = S.PName

# More join type: Inner Join

## Syntax

- R INNER JOIN S USING (<attribute list>)
- R INNER JOIN S ON *R.column\_name* = *S.column\_name*

## Example

TableA

| Column1 | Column2 |
|---------|---------|
| 1       | 2       |

TableB

| Column1 | Column3 |
|---------|---------|
| 1       | 3       |

The INNER JOIN of **TableA** and **TableB** on Column1 will return:

| TableA.Column1 | TableA.Column2 | TableB.Column1 | TableB.Column3 |
|----------------|----------------|----------------|----------------|
| 1              | 2              | 1              | 3              |

**SELECT \* FROM TableA INNER JOIN TableB USING (Column1)**

**SELECT \* FROM TableA INNER JOIN TableB ON TableA.Column1 = TableB.Column1**

# Natural Join

## Syntax

R NATURAL JOIN S

## Example

TableA

| Column1 | Column2 |
|---------|---------|
| 1       | 2       |

TableB

| Column1 | Column3 |
|---------|---------|
| 1       | 3       |

The NATURAL JOIN of **TableA** and **TableB** will return:

| Column1 | Column2 | Column3 |
|---------|---------|---------|
| 1       | 2       | 3       |

**SELECT \* FROM TableA NATURAL JOIN TableB**

- The repeated columns are avoided.
- One can not specify the joining columns in a natural join.

# Summary and roadmap

- Introduction to SQL
  - SELECT  
FROM  
WHERE
  - Eliminating duplicates
  - Renaming attributes
  - Expressions in SELECT Clause
  - Patterns for Strings
  - Ordering
  - Joins
  - Subquery
  - Aggregations
  - UNION, INTERSECT, EXCEPT
  - NULL
  - Outerjoin
- Next
- Insert/Delete tuples
  - Create/Alter/Delete tables
  - Constraints (primary key)
  - Views
  - More constraints
  - Triggers
  - Indexes

Reference: Chapter 6.3 of our TextBook

# Inserting One Tuple

- **INSERT INTO** Product  
**VALUES**( 'iPhone 5', 999 )
- Alternative approaches:
- **INSERT INTO**  
Product( PName, Price)  
**VALUES**( 'iPhone 5', 999 )
- **INSERT INTO**  
Product( Price, PName, )  
**VALUES**( 999, 'iPhone 5' )

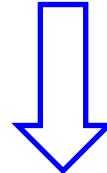
| Product  |       |
|----------|-------|
| PName    | Price |
| iPhone 4 | 888   |
| iPad 2   | 668   |
|          |       |
| PName    |       |
| PName    | Price |
| iPhone 4 | 888   |
| iPad 2   | 668   |
| iPhone 5 | 999   |

# Partial Insertion

- **INSERT INTO**  
Product( PName )  
**VALUES**( 'iPhone 5' )

Product

| PName    | Price |
|----------|-------|
| iPhone 4 | 888   |
| iPad 2   | 668   |

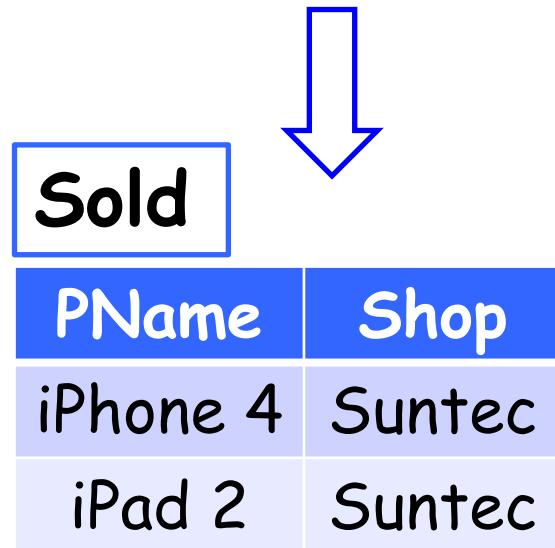


| PName    | Price |
|----------|-------|
| iPhone 4 | 888   |
| iPad 2   | 668   |
| iPhone 5 | NULL  |

# Tuple Insertion via Subqueries

- The 'Sold' table is initially empty.
- INSERT INTO Sold  
SELECT PName, 'Suntec'  
FROM Product**

| Product  |       |
|----------|-------|
| PName    | Price |
| iPhone 4 | 888   |
| iPad 2   | 668   |

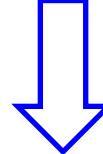


# Tuple Insertion via Subqueries

- Assume that a new shop at the ION sells all products sold at the Suntec shop

| Sold     |        |
|----------|--------|
| PName    | Shop   |
| iPhone 4 | Suntec |
| iPad 2   | Suntec |

- INSERT INTO** Sold  
**SELECT PName, 'ION'**  
**FROM** Sold

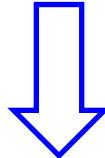


| PName    | Shop   |
|----------|--------|
| iPhone 4 | Suntec |
| iPad 2   | Suntec |
| iPhone 4 | ION    |
| iPad 2   | ION    |

# Tuple Deletion

- DELETE FROM Sold  
WHERE PName = 'iPad 2'

| Sold     |        |
|----------|--------|
| PName    | Shop   |
| iPhone 4 | Suntec |
| iPad 2   | Suntec |



| PName    | Shop   |
|----------|--------|
| iPhone 4 | Suntec |

# Tuple Deletion (cont.)

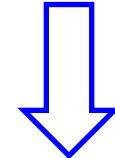
Product

| PName    | Price |
|----------|-------|
| iPhone 4 | 888   |
| iPad 2   | 668   |

Sold

| PName    | Shop   |
|----------|--------|
| iPhone 4 | Suntec |
| iPad 2   | Suntec |

- Remove from the Suntec shop all products over 800 dollars
- DELETE FROM Sold  
WHERE Shop = 'Suntec'  
AND Sold.PName IN  
(SELECT P.PName FROM Product AS P  
WHERE Price > 800)

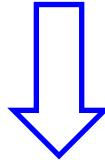


| PName  | Shop   |
|--------|--------|
| iPad 2 | Suntec |

# Deleting All Tuples

- DELETE FROM Sold;

| Sold     |        |
|----------|--------|
| PName    | Shop   |
| iPhone 4 | Suntec |
| iPad 2   | Suntec |



| PName | Shop |
|-------|------|
|       |      |

# Exercise

- Remove (i) any product from Suntec that is also sold at ION and (ii) any product from ION that is also sold at Suntec
- ```
DELETE FROM Sold
WHERE (Sold.Shop = 'Suntec' AND
      Sold.PName IN
      (Select S1.PName FROM Sold AS S1
       WHERE S1.Shop = 'ION'))
      OR (Sold.Shop = 'ION' AND Sold.PName IN
          (Select S2.PName FROM Sold AS S2
           WHERE S2.Shop = 'Suntec'))
```

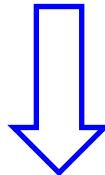
| Sold     |        |
|----------|--------|
| PName    | Shop   |
| iPhone 4 | Suntec |
| iPad 2   | Suntec |
| iPhone 4 | ION    |

# Tuple Update

- Assume that the price of iPhone 4 should be reduced to 777

- UPDATE** Product  
**SET** Price = 777  
**WHERE** PName = 'iPhone 4'

| Product  |       |
|----------|-------|
| PName    | Price |
| iPhone 4 | 888   |
| iPad 2   | 668   |



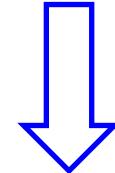
| PName    | Price |
|----------|-------|
| iPhone 4 | 777   |
| iPad 2   | 668   |

# Tuple Update (cont.)

- Assume that Google buys Apple and reduces the prices of all its products by 100

| Product  |       |         |
|----------|-------|---------|
| PName    | Price | Company |
| iPhone 4 | 888   | Apple   |
| iPad 2   | 668   | Apple   |

- UPDATE** Product  
**SET** Price = Price - 100,  
Company = 'Google'  
**WHERE** Company =  
'Apple'



| PName    | Price | Company |
|----------|-------|---------|
| iPhone 4 | 788   | Google  |
| iPad 2   | 568   | Google  |

# Tuple Update (cont.)

Maker

| Product | PName     | Price |
|---------|-----------|-------|
|         | iPhone 4  | 888   |
|         | iPad 2    | 668   |
|         | Milestone | 798   |

| PName     | Company  |
|-----------|----------|
| iPhone 4  | Apple    |
| iPad 2    | Apple    |
| Milestone | Motorola |

- Reduce the price of all Apple products by 10%
- **UPDATE** Product  
SET Price = Price \* 0.9  
**WHERE** Product.PName IN  
(SELECT Maker.PName FROM Maker  
**WHERE** Company = 'Apple')

# Tuple Update (cont.)

| Product | PName     | Price | Maker | Company  |
|---------|-----------|-------|-------|----------|
|         | iPhone 4  | 888   |       | Apple    |
|         | iPad 2    | 668   |       | Apple    |
|         | Milestone | 798   |       | Motorola |

- Reduce the price of all products by 10%
- **UPDATE** Product  
**SET** Price = Price \* 0.9

# Tuple Update (cont.)

| Product | PName     | Price |
|---------|-----------|-------|
|         | iPhone 4  | 888   |
|         | iPad 2    | 668   |
|         | Milestone | 798   |

- Set the price of every product to half of the price of iPhone 4
- **UPDATE** Product  
**SET** Price =  
( SELECT P.Price / 2  
FROM Product AS P  
WHERE P.PName = 'iPhone 4')

# Exercise

Beer

| Name | Maker | Price |
|------|-------|-------|
|------|-------|-------|

Wine

| Name | Maker | Price |
|------|-------|-------|
|------|-------|-------|

- Update the price of every beer to the average price of the wine by the same maker
- UPDATE Beer  
SET Beer.Price =  
(SELECT AVG(Wine.Price)  
FROM Wine  
WHERE Wine.Maker = Beer.Maker)

# Exercise

Beer

| Name | Maker | Price |
|------|-------|-------|
|------|-------|-------|

Wine

| Name | Maker | Price |
|------|-------|-------|
|------|-------|-------|

- Delete any beer by a maker that does not produce any wine
- **DELETE FROM Beer  
WHERE NOT EXISTS  
(SELECT \*  
FROM Wine  
WHERE Wine.Maker = Beer.Maker)**

# Exercise

Beer

| Name | Maker | Price |
|------|-------|-------|
|------|-------|-------|

Wine

| Name | Maker | Price |
|------|-------|-------|
|------|-------|-------|

- For each beer, if there does not exist a wine with the same name, then create a wine with the same name and maker, but twice the price
- ```
INSERT INTO Wine
SELECT B.Name, B.Maker, B.Price * 2
FROM Beer AS B
WHERE NOT EXISTS
    (SELECT * FROM Wine
     WHERE Wine.Name = B.Name)
```

# Table Creation

Product

| PName | Price |
|-------|-------|
|-------|-------|

- **CREATE TABLE** Product(  
PName VARCHAR(30),  
Price INT);
- In general:  
**CREATE TABLE** <table name> (  
<column name 1> <column type 1>,  
<column name 2> <column type 2>,  
... );

# Column Types

- INT or INTEGER (synonyms)
- REAL or FLOAT (synonyms)
- CHAR(n): fixed-length string of n characters
- VARCHAR(n): variable-length string of up to n characters
- DATE: = In '**yyyy-mm-dd**' format
- ...

# Summary and roadmap

- Introduction to SQL
- SELECT  
FROM  
WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Aggregations
- UNION, INTERSECT, EXCEPT
- NULL
- Outerjoin
- ~~Insert/Delete tuples~~
- Create/Alter/Delete tables

- Next

- Constraints (primary key)
- Views
- More constraints
- Triggers
- Indexes

# Table Creation (cont.)

Product

PName

Price

- **CREATE TABLE Product( PName VARCHAR(30), Price INT );**
- What if we want to make sure that all products have distinct names?
- Solution: declare PName as **PRIMARY KEY** or **UNIQUE**

# PRIMARY KEY

Product

PName

Price

- **CREATE TABLE Product( PName VARCHAR(30), Price INT );**
- What if we want to make sure that all products have distinct names?
- **CREATE TABLE Product( PName VARCHAR(30), Price INT,  
PRIMARY KEY (PName) );**

# UNIQUE

Product

PName

Price

- **CREATE TABLE Product( PName VARCHAR(30), Price INT );**
- What if we want to make sure that all products have distinct names?
- **CREATE TABLE Product( PName VARCHAR(30), Price INT,  
UNIQUE(PName) );**

# PRIMARY KEY (cont.)

Catalog

| PName | Shop | Price |
|-------|------|-------|
|-------|------|-------|

- We want to make sure that there is no duplicate PName with the same Shop
- **CREATE TABLE Catalog(**  
PName VARCHAR(30),  
Shop VARCHAR(30),  
Price INT,  
**PRIMARY KEY (PName, Shop) );**

# UNIQUE (cont.)

Catalog

| PName | Shop | Price |
|-------|------|-------|
|-------|------|-------|

- We want to make sure that there is no duplicate PName with the same Shop
- **CREATE TABLE Catalog(**  
PName VARCHAR(30),  
Shop VARCHAR(30),  
Price INT,  
**UNIQUE (PName, Shop) );**

# PRIMARY KEY vs. UNIQUE

Catalog

- Difference 1

| PName | Shop | Price |
|-------|------|-------|
|-------|------|-------|

- Only ONE set of attributes in a table can be declared as PRIMARY KEY
  - But we can declare multiple sets of attributes as UNIQUE
- CREATE TABLE Catalog( PName VARCHAR(30), Shop VARCHAR(30), Price INT, UNIQUE (PName, Shop), UNIQUE (Shop, Price) );**

# PRIMARY KEY vs. UNIQUE

Catalog

- Difference 2

- If set of attributes are declared as PRIMARY KEY, then none of these attributes can be NULL
  - UNIQUE attributes still allow NULLs

- CREATE TABLE Catalog(**  
PName VARCHAR(30),  
Shop VARCHAR(30),  
Price INT,  
**UNIQUE** (PName, Shop) );

| PName | Shop | Price |
|-------|------|-------|
|-------|------|-------|

# NOT NULL

Catalog

| PName | Shop | Price |
|-------|------|-------|
|-------|------|-------|

- We want to make sure that the price of each product is not NULL
- **CREATE TABLE Catalog(**  
PName VARCHAR(30),  
Shop VARCHAR(30),  
Price INT **NOT NULL**);

# NOT NULL (cont.)

Catalog

| PName | Shop | Price |
|-------|------|-------|
|-------|------|-------|

- We want to make sure that the price as well as PName of each product is not NULL
- **CREATE TABLE Catalog(**  
PName VARCHAR(30) **NOT NULL**,  
Shop VARCHAR(30),  
Price INT **NOT NULL**);

# NOT NULL (cont.)

Catalog

| PName | Shop | Price |
|-------|------|-------|
|-------|------|-------|

- **CREATE TABLE Catalog(**  
PName VARCHAR(30) **NOT NULL**,  
Shop VARCHAR(30),  
Price INT **NOT NULL**);
- NOT NULL may prevent partial insertions
- INSERT INTO Product(PName)  
Values( 'iPhone 5' )

Error!

# DEFAULT

Catalog

| PName | Shop | Price |
|-------|------|-------|
|-------|------|-------|

- We want to specify that, by default, the shop and price of a product is 'Suntec' and 1, respectively
- **CREATE TABLE Catalog(**  
PName VARCHAR(30) **DEFAULT 'Suntec'**,  
Shop VARCHAR(30),  
Price INT **DEFAULT 1**);

# Combination

Catalog

| PName | Shop | Price |
|-------|------|-------|
|-------|------|-------|

- We want to specify that, by default, the shop and price of a product is 'Suntec' and 1, respectively. In addition, the shop should not be NULL
- **CREATE TABLE Catalog(**  
PName VARCHAR(30) **NOT NULL**  
**DEFAULT** 'Suntec',  
Shop VARCHAR(30),  
Price INT **DEFAULT** 1);

# Table Deletion

Catalog

| PName | Shop | Price |
|-------|------|-------|
|-------|------|-------|

- DROP TABLE Catalog

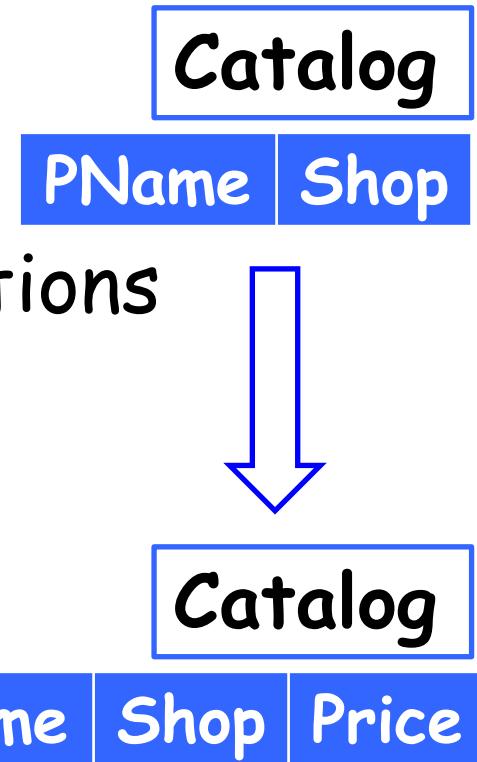
# Table Modification

- Adding a new attribute

- **ALTER TABLE Catalog**  
**ADD Price INT**

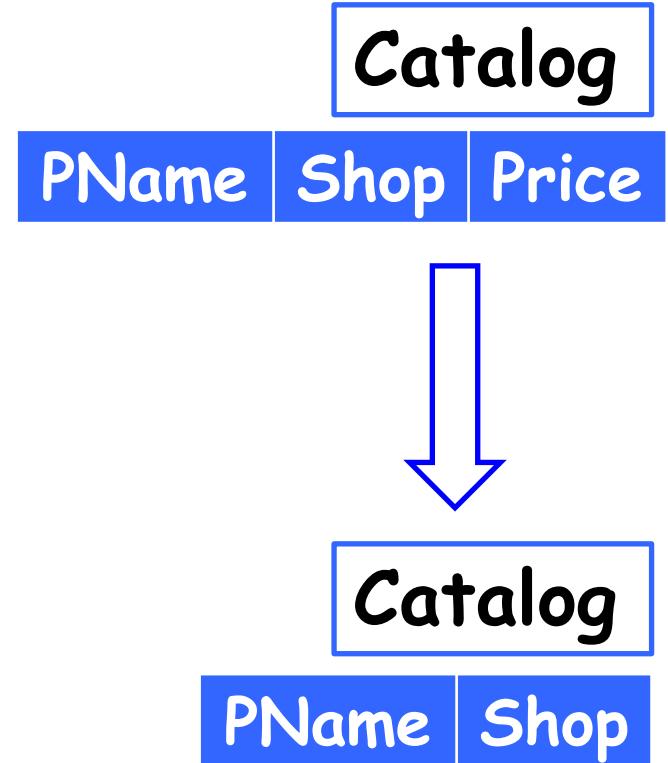
- We can also add some declarations

- **ALTER TABLE Catalog**  
**ADD Price INT NOT NULL**  
**DEFAULT 1**



# Table Modification

- Deleting an attribute
- **ALTER TABLE Catalog**  
**DROP Price**

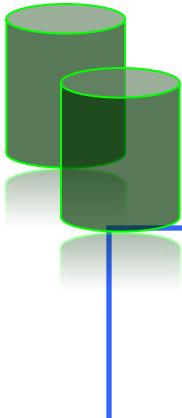


# Summary and roadmap

- Introduction to SQL
- SELECT
- FROM
- WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Aggregations
- UNION, INTERSECT, EXCEPT
- NULL
- Outerjoin
- Insert/Delete tuples
- Create/Alter/Delete tables
- Constraints (primary key)

- Next
  - Views
  - More constraints
    - FOREIGN KEY
    - CHECK
    - ASSERTION
    - Triggers
  - Indexes

Reference: Chapter 6.5 of our TextBook



# CZ2007

# Introduction to Databases

---

## Querying Relational Databases using SQL

### Part--4

---

Cong Gao

Professor

School of Computer Science and Engineering  
Nanyang Technological University, Singapore

# Summary and roadmap

- Introduction to SQL
  - SELECT
  - FROM
  - WHERE
  - Eliminating duplicates
  - Renaming attributes
  - Expressions in SELECT Clause
  - Patterns for Strings
  - Ordering
  - Joins
  - Subquery
  - Aggregations
  - UNION, INTERSECT, EXCEPT
  - NULL
  - Outerjoin
  - Insert/Delete tuples
  - Create/Alter/Delete tables
  - Constraints (primary key)
- 
- Next
    - More constraints
      - FOREIGN KEY
      - CHECK
      - ASSERTION
      - Trigger
    - Views
    - Indexes

# Views -- Motivation

Author

| PenName | RealName |
|---------|----------|
| ...     | ...      |

Wrote

| PenName | Article |
|---------|---------|
| ...     | ...     |

| RealName | Article |
|----------|---------|
| ...      | ...     |

- SELECT RealName, Article  
FROM Author, Wrote  
WHERE Author.PenName = Wrote.PenName
- Assume that we frequently need to check the real names of the authors of some articles
- Can we somehow make the above join results more easily accessible?

# Views

## Author

| PenName | RealName |
|---------|----------|
| ...     | ...      |

## Wrote

| PenName | Article |
|---------|---------|
| ...     | ...     |

## RealAuthor

| RealName | Article |
|----------|---------|
| ...      | ...     |

- **CREATE VIEW** RealAuthor **AS**  
SELECT RealName, Article  
FROM Author, Wrote  
WHERE Author.PenName = Wrote.PenName;
- This view can then be used as a table
- **SELECT Article FROM RealAuthor**  
**WHERE RealName = 'Chris'**

# Tables vs. Views

**Author**

| <u>PenName</u> | <u>RealName</u> |
|----------------|-----------------|
| ...            | ...             |

**Wrote**

| <u>PenName</u> | <u>Article</u> |
|----------------|----------------|
| ...            | ...            |

**RealAuthor**

| <u>RealName</u> | <u>Article</u> |
|-----------------|----------------|
| ...             | ...            |

- Tables are physically stored in the database
- Views are NOT physically stored but are computed on the fly

# Queries on Views

Author

PenName

RealName

- We have a view as follows:

**CREATE VIEW** RealAuthor **AS**

SELECT RealName, Article  
FROM Author, Wrote  
WHERE Author.PenName = Wrote.PenName;

- We have a query like this:

SELECT Article FROM RealAuthor  
WHERE RealName = 'Chris'

- What the DBMS would do:

SELECT RealName, Article  
FROM Author, Wrote  
WHERE Author.PenName = Wrote.PenName  
AND RealName = 'Chris';

Wrote

PenName

Article

RealAuthor

RealName

Article

Query Rewriting

# Deleting Views

Author

PenName

RealName

Wrote

PenName

Article

- DROP VIEW RealAuthor;

RealAuthor

RealName

Article

# Summary on Views

## What is view?

A **view** is a query over the **base relations** to produce another relation.

## View is virtual

It is considered ***virtual*** because it does not usually exist physically.

## To the user

A view appears just like any other table and can be present in any SQL query where a table is present.

# View Materialization

## The idea

Physically creating and keeping a temporary table  
Syntax: CREATE MATERIALIZED VIEW ... AS..

## Pro and Con

Pro: Other queries defined on the view can be answered efficiently

Con: Maintaining correspondence between the base table and the view when the base table is updated

# Views vs. Temporary Views

**Author**

| <u>PenName</u> | <u>RealName</u> |
|----------------|-----------------|
| ...            | ...             |

**Wrote**

| <u>PenName</u> | <u>Article</u> |
|----------------|----------------|
| ...            | ...            |

- **CREATE VIEW** RealAuthor **AS**  
SELECT      RealName, Article  
FROM        Author, Wrote  
WHERE      Author.PenName = Wrote.PenName;
- **WITH** RealAuthor **AS**  
(SELECT      RealName, Article  
FROM        Author, Wrote  
WHERE      Author.PenName = Wrote.PenName)  
SELECT      Article  
FROM        RealAuthor  
WHERE      RealAuthor.RealName = 'Chris';

# Views vs. Temporary Views

- A view persists after its creation
- It will be there unless you **explicitly** delete it
- **CREATE VIEW** RealAuthor **AS**  
SELECT RealName, Article  
FROM Author, Wrote  
WHERE Author.PenName =  
Wrote.PenName;
- **DROP VIEW** RealAuthor;

# Views vs. Temporary Views

- A temporary view only persists during the execution of a query
- It will not be there once the query is done
- **WITH** RealAuthor **AS**  
(SELECT RealName, Article  
FROM Author, Wrote  
WHERE Author.PenName = Wrote.PenName)  
SELECT Article  
FROM RealAuthor  
WHERE RealAuthor.RealName = 'Cedric';
- Here, the temporary view RealAuthor is gone once the query is finished

# SELECT INTO vs. VIEWS

- SELECT RealName, Article

```
INTO RealAuthor
```

```
FROM Author, Wrote
```

```
WHERE Author.PenName =
```

```
Wrote.PenName
```

A Table is  
created  
(Physically)

- CREATE VIEW RealAuthor AS

```
SELECT RealName, Article
```

```
FROM Author, Wrote
```

```
WHERE Author.PenName =
```

```
Wrote.PenName;
```

A Definition  
is created  
(Virtually)

# Summary and roadmap

- Introduction to SQL
- SELECT
- FROM
- WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Aggregations
- UNION, INTERSECT, EXCEPT
- NULL
- Outerjoin
- Insert/Delete tuples
- Create/Alter/Delete tables
- Constraints: primary key
- Views

- Next
  - More constraints
    - FOREIGN KEY
    - CHECK
    - ASSERTION
    - Trigger
  - Views
  - Indexes

# Constraints

- A constraint is a requirement on tuples/tables that the DBMS needs to enforce
- Examples
  - PRIMARY KEY
  - UNIQUE
  - NOT NULL
- More to be discussed
  - FOREIGN KEY
  - CHECK
  - ASSERTION
  - Trigger

# FOREIGN KEY Example

| Beers | Sells |      |       |
|-------|-------|------|-------|
| Name  | Bar   | Beer | Price |
| manf  |       |      |       |

- Any value in Sells.Beer should appear in Beer.Name
  - values for certain values must make sense
- How to ensure this when we create Sells?

# Example

Requirement: Beers.Name **must be** declared as either PRIMARY KEY or UNIQUE

```
CREATE TABLE Beers (
    name      CHAR(30)  PRIMARY KEY,
    manf      VARCHAR(50)
);
```

```
CREATE TABLE Sells (
    bar       CHAR(20),
    beer     CHAR(30),
    price    REAL,
    FOREIGN KEY beer REFERENCES Beers(name)
);
```

# Foreign Key

## What it is?

- Consider **Sells**(Bar, Beer, Price)
- The beer value might be a “real” beer
  - Can be found in the **Beers** relation
- A constraint that requires a beer in **Sells** to be a beer in **Beers** is called a **foreign key constraint**

## Expressing Foreign Key

- Use the keyword **REFERENCES**. Two methods:
  1. Within the declaration of an attribute, when FK has only one attribute
  2. As a separate element of Create Table: **FOREIGN KEY (<attribute list>) REFERENCES <relation> (<attributes>)**
- **Referenced attributes** must be declared as **Primary Key** or **UNIQUE** in the other **relation**

# FOREIGN KEY (cont.)

Visits

Purchase

| Name | Shop |
|------|------|
|------|------|

| Name | Shop | Product |
|------|------|---------|
|------|------|---------|

- Any value combination in Purchase(Name, Shop) should appear in Visits(Name, Shop)
- CREATE TABLE Purchase(  
Name      VARCHAR(30),  
Shop      VARCHAR(30),  
Product    VARCHAR(30),  
**FOREIGN KEY** (Name, Shop) **REFERENCES**  
Visits(Name, Shop) );
- Requirement: Visits(Name, Shop) is declared  
as either PRIMARY KEY or UNIQUE

# Foreign Key Constraints

Beers

| Name | Brand |
|------|-------|
| B1   | Tiger |

Sells

| Bar   | Beer | Price |
|-------|------|-------|
| Lotus | B1   | 10    |

- Sells(Beer) is a foreign key referencing Beer(Name)
- INSERT INTO Sells VALUES( 'Lotus', B2, 20)
- This insertion will be rejected

INSERT or UPDATE a Sells tuple so it refers to a nonexistent beer

Always rejected

# Foreign Key Constraints (cont.)

Beers

| Name | Brand |
|------|-------|
| B1   | Tiger |

Sells

| Bar   | Beer | Price |
|-------|------|-------|
| Lotus | B1   | 10    |

- Sells(Beer) is a foreign key referencing Beer(Name)
- DELETE FROM Beers  
WHERE Name = 'B1'
- What is going to happen?
- Three possibilities
  - Default: Reject the deletion
  - Cascade: Delete the corresponding tuple in Sells
  - Set NULL: Set the corresponding values to NULL

# Default

| Beers |
|-------|
|       |

| Name | Brand |
|------|-------|
| B1   | Tiger |

## Sells

| Bar   | Beer | Price |
|-------|------|-------|
| Lotus | B1   | 10    |

- Sells(Beer) is a foreign key referencing Beer(Name)
- DELETE FROM Beers  
WHERE Name = 'B1' Reject!
- UPDATE Beers  
SET Name = 'B2'  
Where Name = 'B1' Reject!

# Cascade

| Beers | Name | Brand |
|-------|------|-------|
|       | B1   | Tiger |

| Sells | Bar   | Beer | Price |
|-------|-------|------|-------|
|       | Lotus | B1   | 10    |

- Sells(Beer) is a foreign key referencing Beer(Name)
- DELETE FROM Beers  
WHERE Name = 'B1'
- Delete all tuples from sells where Beer = 'B1'
- UPDATE Beers  
SET Name = 'B2'  
Where Name = 'B1'
- Update Sells: for any tuple with Beer = 'B1', set its Beer value to 'B2'

# Set NULL

| Beers | Name | Brand |
|-------|------|-------|
|       | B1   | Tiger |

| Sells | Bar   | Beer | Price |
|-------|-------|------|-------|
|       | Lotus | B1   | 10    |

- Sells(Beer) is a foreign key referencing Beer(Name)
- DELETE FROM Beers  
WHERE Name = 'B1'
- Change all 'B1' Beer values in Sells to NULL
- UPDATE Beers  
SET Name = 'B2'  
Where Name = 'B1'
- Change all 'B1' Beer values in Sells to NULL

# Choosing a Policy

| Beers | Name | Brand |
|-------|------|-------|
|       | B1   | Tiger |

| Sells | Bar   | Beer | Price |
|-------|-------|------|-------|
|       | Lotus | B1   | 10    |

- We can specify whether to default, cascade, or set null when we declare a foreign key
- Example:  
CREATE TABLE Sells(  
 Bar VARCHAR(30),  
 Beer VARCHAR(30),  
 Price FLOAT,  
 FOREIGN KEY (Beer) REFERENCES Beers(Name)  
 ON DELETE SET NULL  
 ON UPDATE CASCADE );
- Default is taken when SET NULL and CASCADE are absent

# Summary and roadmap

- Introduction to SQL
- SELECT  
FROM  
WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Aggregations
- UNION, INTERSECT, EXCEPT
- NULL
- Outerjoin
- Insert/Delete tuples
- Create/Alter/Delete tables
- Constraints: primary key
- Views
- Constraints: Foreign key

## • Next

- More constraints
  - CHECK
  - ASSERTION
  - Trigger
- Indexes

# attribute-based check

We want to make sure that any beer does not sell over 100 dollars. How?

```
CREATE TABLE Sells (
    bar      CHAR(20),
    beer     CHAR(30) ,
    price    REAL          CHECK (price <= 100),
);
```

- **CHECK (<condition>)** must be added to the declaration for the attribute
- An attribute-based check is checked only when a value for that attribute is **inserted** or **updated** (but not deleted)

# Tuple-Based Check

- **CHECK (<condition>)** may be added as separate element in a table declaration
  - The condition can refer to any attribute of the relation,

Example: Only Lotus can sell beer for more than \$10

```
CREATE TABLE Sells (
    bar      CHAR(20),
    beer     CHAR(30),
    price    REAL,
    CHECK (bar = 'Lotus' OR price <= 10.00)
);
```

- Checked whenever a tuple is inserted or updated

# SQL Assertions

## What is it?

- Database schema constraints
  - Not available with majorDBMS, but in SQL standard
- Condition can refer to any relation or attribute in the database schema
- We must check every **ASSERTION** after every modification to any relation of the database

## Syntax

```
CREATE ASSERTION <name>  
CHECK (condition);
```

# Assertions

| Beers | Name  | Brand |
|-------|-------|-------|
| B1    | Tiger |       |

| Sells | Bar | Beer | Price |
|-------|-----|------|-------|
| Lotus | B1  | 10   |       |

- A more flexible type of checks
  - But not available with most of DBMS
- Example: There should not be more bars than beers
- **CREATE ASSERTION** MoreBars **CHECK**

```
(  
    (SELECT COUNT(*) FROM Beers)  
    >=  
    (SELECT COUNT(DISTINCT Bar) FROM Sells)  
)
```

# Example

## Constraint

In `Sells(Bar, Beer, Price)` no bar may charge an average of more than \$10.

```
CREATE ASSERTION          NoRipOffBars CHECK (
    NOT EXISTS (
        SELECT      bar
        FROM        Sells
        GROUP BY   bar
        HAVING     10.00 < AVG(price)
    ));

```

# Motivation for Trigger

## Attributes and Tuple-based checks

Limited capabilities

## Assertions

- Sufficiently general for most constraint applications
- Hard to implement efficiently!
- We must check every **ASSERTION** after every modification to any relation of the database

# Trigger

| Beers | Name  | Brand |
|-------|-------|-------|
| B1    | Tiger |       |

| Sells | Bar | Beer | Price |
|-------|-----|------|-------|
| Lotus | B1  | 10   |       |

- Every time there is a new beer from Tiger, the Lotus Bar would sell it at 10 dollars.
- How to capture this?
- we can use **triggers**
  - Triggering event
  - Action

# Trigger (AFTER INSERT)

| Beers | Name | Brand | Bar   | Beer | Price |
|-------|------|-------|-------|------|-------|
|       | B1   | Tiger | Lotus | B1   | 10    |

- Every time there is a new beer from Tiger, the Lotus Bar would sell it at 10 dollars.
- CREATE TRIGGER BeerTrig  
AFTER INSERT ON Beers  
REFERENCING NEW ROW AS NewTuple  
FOR EACH ROW  
WHEN (NewTuple.Brand = 'Tiger')  
INSERT INTO Sells  
VALUES ('Loctus', NewTuple.Name, 10);

# Trigger (AFTER DELETE)

| Beers | Name  | Brand | Sells |    |
|-------|-------|-------|-------|----|
| Bar   | Beer  | Price |       |    |
| B1    | Tiger | Lotus | B1    | 10 |

- Every time the Lotus Bar stops selling a beer, the Cheetah Bar would sell it at its last price
- CREATE TRIGGER  
AFTER DELETE ON  
REFERENCING OLD ROW AS OldTuple  
FOR EACH ROW  
WHEN (OldTuple.Bar = 'Lotus')  
INSERT INTO Sells  
VALUES ('Cheetah', OldTuple.Beer,  
OldTuple.Price);

# Trigger (AFTER UPDATE ON)

Beers

| Name | Brand |
|------|-------|
| B1   | Tiger |

| Bar   | Beer | Price |
|-------|------|-------|
| Lotus | B1   | 10    |

- Every time the some bar raises the price of a beer by over 10 dollars, the Cheetah Bar would sell it at its old price
- CREATE TRIGGER  
**AFTER UPDATE ON** BeerTrig  
REFERENCING  
OLD ROW AS OldTuple  
NEW ROW AS NewTuple  
FOR EACH ROW  
WHEN (OldTuple.Price + 10 < NewTuple.Price)  
INSERT INTO Sells  
VALUES ('Cheetah', OldTuple.Beer,  
OldTuple.Price);

duplicate beers at the Cheetah Bar. How to avoid?

# Trigger (cont.)

| Beers |
|-------|
| B1    |

| Name | Brand |
|------|-------|
| B1   | Tiger |

| Sells |
|-------|
|       |

| Bar   | Beer | Price |
|-------|------|-------|
| Lotus | B1   | 10    |

- Every time the some bar raises the price of a beer by over 10 dollars, the Cheetah Bar would sell it at its old price. (**Note: Avoid duplicate beers at the CheeTah Bar**)
- CREATE TRIGGER BeerTrig  
AFTER UPDATE ON Sells  
REFERENCING OLD ROW AS OldTuple  
NEW ROW AS NewTuple  
FOR EACH ROW  
WHEN (OldTuple.Price + 10 < NewTuple.Price)  
BEGIN  
DELETE FROM Sells  
WHERE Bar = 'Cheetah' AND Beer = OldTuple.Beer;  
INSERT INTO Sells  
VALUES ('CheeTah', OldTuple.Beer, OldTuple.Price);  
END

# A summary of Trigger

- A triggering event can be the execution of an SQL **INSERT, DELETE, and UPDATE** statement
- **WHEN** clause is optional
  - If it is missing, then the action is executed whenever the trigger is awakened
- If the When condition of the trigger is satisfied, the action associated with the trigger is performed by the DBMS

## Types of actions

- An SQL query, a DELETE, INSERT, UPDATE, ROLLBACK, SQL/PSM
- There can be **more than one** SQL statements
- Queries make no sense in an action
  - So only DB modification

# Exercise (1)

| Beers | Name  | Brand |
|-------|-------|-------|
| B1    | Tiger |       |

| Sells | Bar   | Beer | Price |
|-------|-------|------|-------|
|       | Lotus | B1   | 10    |

- Every time the Lotus Bar sells a new bear, the Cheetah Bar would sell it with 1 dollars less
- CREATE TRIGGER  
AFTER INSERT ON  
REFERENCING  
    NEW ROW AS NewTuple  
    FOR EACH ROW  
    WHEN       (NewTuple.Bar = 'Lotus')  
    BEGIN  
        DELETE FROM           Sells  
        WHERE       Bar = 'Cheetah' AND Beer = NewTuple.Beer;  
        INSERT INTO     Sells  
                 VALUES ('CheeTah', NewTuple.Beer,  
                  NewTuple.Price - 1);  
    END

# Exercise (2)

| Beers | Name  | Brand |
|-------|-------|-------|
| B1    | Tiger |       |

| Sells | Bar   | Beer | Price |
|-------|-------|------|-------|
|       | Lotus | B1   | 10    |

- Every time the Lotus Bar cut the price of a beer by  $x$  dollars, the Cheetah Bar would cut the price by  $2 * x$  dollars
- CREATE TRIGGER BeerTrig  
AFTER UPDATE ON Sells  
REFERENCING OLD ROW AS OldTuple  
NEW ROW AS NewTuple  
FOR EACH ROW  
WHEN (OldTuple.Price > NewTuple.Price AND OldTuple.Bar = 'Lotus')  
BEGIN  
    UPDATE Sells  
    SET Price = Price - 2 \* (OldTuple.Price - NewTuple.Price)  
    WHERE Bar = 'Cheetah' AND Beer = OldTuple.Beer;  
END

# Exercise (3)

| Beers | Name  | Brand |
|-------|-------|-------|
| B1    | Tiger |       |

| Sells | Bar   | Beer | Price |
|-------|-------|------|-------|
|       | Lotus | B1   | 10    |

- Every time the Lotus Bar stops selling a beer, if the Cheetah bar sells the beer, it would set the price of the beer to the average price of the Tiger beers sold at the Lotus Bar
- CREATE TRIGGER BeerTrig  
AFTER DELETE ON Sells  
REFERENCING OLD ROW AS OldTuple  
FOR EACH ROW  
WHEN (OldTuple.Bar = 'Lotus')  
BEGIN  
    UPDATE Sells  
    SET Price = (SELECT AVG(Price)  
                FROM Beers, Sells  
                WHERE Bar = 'Lotus' AND Brand = Tiger and  
                     name = beer)  
    WHERE Bar = 'Cheetah' AND Beer = OldTuple.Beer;  
END

# Question

- All customers must have an account balance of at least \$1000 in their account.
  - Do you use Check or Trigger ?
- All new customers opening an account must have a balance of \$1000
  - Do you use Check or Trigger ?

# Other syntax of triggers

- Before { [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
  - WHEN condition is tested before the triggering event is executed
- Instead of { [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
  - It overrides Insert, Update, Delete
  - Mostly used for updating a view---to update base tables of view

# BEFORE - Trigger Execution (Example)

## Employee

| First_Name | Last_Name | Phone |
|------------|-----------|-------|
| ...        | ...       | ...   |

```
CREATE TRIGGER Last_Name_Upper
BEFORE INSERT ON Employee
REFERENCING NEW ROW AS N
FOR EACH ROW
    N.Last_Name = Upper(N.Last_Name);
```

- No When Clause. The trigger action is executed.
- Then the triggering event ( Insert N into Employee) is executed

# BEFORE - Trigger Execution

1. WHEN condition is tested before the triggering event
    - If the condition is true then the action of the trigger is executed
  2. Next, the triggering event is executed, regardless of whether the condition is true
- Action:
    - update or validate record values (the same record in the triggering event) before they are saved to the database
    - Not allowed to modify the database

# INSTEAD OF - Trigger Execution (Example)

Likes

| Drinker | Beer |
|---------|------|
| John    | A1   |

Frequent

| Drinker | Bar |
|---------|-----|
| John    | B2  |

Sells

| Bar   | Beer | Price |
|-------|------|-------|
| Lotus | B1   | 10    |

CREATE VIEW  
SELECT  
FROM  
WHERE

Synergy AS  
Likes.drinker, Likes.beer, Sells.bar  
Likes, Sells, Frequent  
Likes.drinker = Frequent.drinker AND  
Likes.beer = Sells.beer AND  
Sells.bar = Frequent.bar

view Synergy has (drinker, beer, bar) triples: the bar serves the beer and the drinker frequents the bar and likes the beer

Use a INSTEAD OF trigger to turn a (drinker, beer, bar) triple into three insertions of projected pairs

# INSTEAD OF - Trigger Execution (Example)

```
CREATE TRIGGER ViewTrig  
INSTEAD OF INSERT ON Synergy  
REFERENCING  
    NEW ROW AS n  
FOR EACH ROW  
BEGIN  
    INSERT INTO Likes VALUES(n.drinker, n.beer);  
    INSERT INTO Sells(bar,beer) VALUES(n.bar, n.beer);  
    INSERT INTO Frequents VALUES(n.drinker, n.bar);  
END;
```

- Semantic: Trigger defined on view. instead of triggering event, we implement action
- Generally it is impossible to modify views because it doesn't exists physically.

# Trigger - Syntax

```
CREATE TRIGGER triggerName
{BEFORE | AFTER| INSTEAD OF}
  {INSERT | DELETE | UPDATE [OF column-name-list]}

ON table-name

[ REFERENCING [ OLD ROW AS var-to-refer-to-old-tuple
                [ NEW ROW AS var-to-refer-to-new-tuple]]
  [ OLD TABLE AS name-to-refer-to-new-table]
  [ NEW TABLE AS name-to-refer-to-old-table]]

[ FOR EACH { ROW | STATEMENT } ] Granularity

[ WHEN (precondition) ]
[BEGIN]
statement-list;
[END];
```

# Trigger Execution Granularity (Example)

```
CREATE TRIGGER
AFTER INSERT ON
REFERENCING NEW ROW AS
FOR EACH ROW
WHEN (newTuple.beer
       INSERT INTO
       VALUES (newTuple.beer);
```

BeerTrig  
Sells  
*newTuple*

NOT IN
(*SELECT name FROM Beers*)
Beers(*name*)

```
CREATE TRIGGER
AFTER UPDATE OF
FOR EACH STATEMENT
INSERT INTO
VALUES (Current_Date, SELECT AVG(Salary) FROM Employee)
```

RecordNewAvg
Salary ON Employee
Log

# Trigger Execution Granularity

## Row level

- **FOR EACH ROW** indicates row-level
- Row-level triggers are executed once for each modified (inserted, updated, or deleted) tuple/ row

## Statement level

- Executed once for an SQL statement, regardless of the number of tuples modified
  - even if 0 tuple is modified: An **UPDATE** statement that makes no changes (condition in the **WHERE** clause does not affect any tuples)
- **If neither is specified, default is “Statement level”**

# Trigger - Syntax

```
CREATE TRIGGER triggerName
{BEFORE | AFTER| INSTEAD OF}
  {INSERT | DELETE | UPDATE [OF column-name-list]}
```

ON table-name

## Referencing

```
[ REFERENCING [ OLD ROW AS var-to-refer-to-old-tuple]
  [ NEW ROW AS var-to-refer-to-new-tuple]]
  [ OLD TABLE AS name-to-refer-to-new-table]]
  [ NEW TABLE AS name-to-refer-to-old-table]]
```

```
[ FOR EACH { ROW | STATEMENT } ]
```

```
[ WHEN (precondition) ]
```

```
[BEGIN]
```

```
statement-list;
```

```
[END];
```

# Trigger Referencing a table

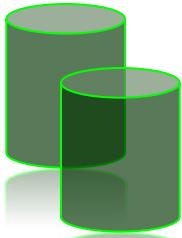
```
CREATE TRIGGER FLIGHTS_DELETE
AFTER DELETE ON FLIGHTS
REFERENCING OLD TABLE AS DELETED_FLIGHTS
FOR EACH STATEMENT
BEGIN
DELETE FROM FLIGHT_AVAILABILITY
WHERE FLIGHT_ID IN
(SELECT FLIGHT_ID
FROM DELETED_FLIGHTS);
END
```

- The **OLD TABLE** maps those rows affected by the triggering event (i.e., only deleted rows of FLIGHT due to the execution of the triggering SQL statement).  
□ **NOT** the old version of FLIGHT table before deletion

# Summary and roadmap

- Introduction to SQL
- SELECT
- FROM
- WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Aggregations
- UNION, INTERSECT, EXCEPT
- NULL
- Outerjoin
- Insert/Delete tuples
- Create/Alter/Delete tables
- Constraints: primary key
- Views
- Constraints:
  - Foreign key
  - CHECK
  - ASSERTION
  - Trigger

- Next
- Indexes



# CZ2007

# Introduction to Databases

## Querying Relational Databases using SQL

### Part--5

Cong Gao

Professor

School of Computer Science and Engineering  
Nanyang Technological University, Singapore

# Summary and roadmap

- Introduction to SQL
  - SELECT
  - FROM
  - WHERE
  - Eliminating duplicates
  - Renaming attributes
  - Expressions in SELECT Clause
  - Patterns for Strings
  - Ordering
  - Joins
  - Subquery
  - Aggregations
  - UNION, INTERSECT, EXCEPT
  - NULL
  - Outerjoin
  - Insert/Delete tuples
  - Create/Alter/Delete tables
  - Constraints: primary key
  - Views
  - Constraints:
    - Foreign key
    - CHECK
    - ASSERTION
    - Trigger
- Next
  - Indexes

## Example

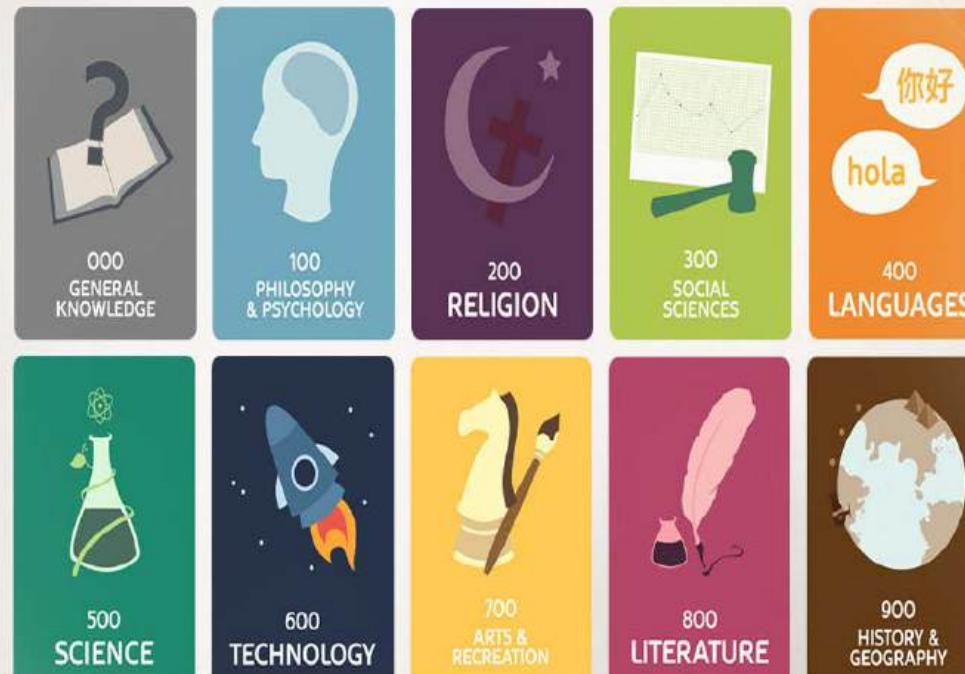
### Find Book in Library



#### Design choices?

- Scan through each aisle
- Lookup pointer to book location, with librarian's organizing scheme

# *the* DEWEY DECIMAL SYSTEM



## Understanding the Dewey Decimal System

Division  
(Drawing/Decorative Arts)

**746.43**

Section  
(Textile Arts)

Main Class  
(Art)

Classification  
Within the  
Section  
(Type of Textile)

AVONLA PARK CAMPUS LIBRARY • MACEDON MARCH 2016

www.mylibrary.com

## Algorithm for book titles

- Find right category
- Lookup Index, find location
- Walk to aisle. Scan book titles. Faster if books are sorted

# Latency numbers every engineer should know

## Ballpark timings

|                                     |  |
|-------------------------------------|--|
| execute typical instruction         | 1/1,000,000,000 sec = 1 nanosec        |
| fetch from L1 cache memory          | 0.5 nanosec                            |
| fetch from L2 cache memory          | 7 nanosec                              |
| Mutex lock/unlock                   | 25 nanosec                             |
| fetch from main memory              | 100 nanosec                            |
| send 2K bytes over 1Gbps network    | 20,000 nanosec                         |
| read 1MB sequentially from memory   | 250,000 nanosec                        |
| fetch from new disk location (seek) | 8,000,000 nanosec                      |
| read 1MB sequentially from disk     | 20,000,000 nanosec                     |
| send packet US to Europe and back   | 150 milliseconds = 150,000,000 nanosec |



(~0.25 msecs)

(~10 msecs)

(~20 msecs) )

# Example: Search for books

Billion\_Books

| BID  | Title                | Author      | Published | Full_text |
|------|----------------------|-------------|-----------|-----------|
|      | ....                 | ....        | ....      |           |
| 7003 | Harry Potter         | Rowling     | 1999      | ...       |
| 1001 | War and Peace        | Tolstoy     | 1869      | ...       |
| 1002 | Crime and Punishment | Dostoyevsky | 1866      | ...       |
| 1003 | Anna Karenina        | Tolstoy     | 1877      | ...       |
|      | ....                 |             |           |           |

All books written by Rowling?'

```
SELECT *
FROM Billion_Books
WHERE Author like
'Rowling'
```

# Example: Search for books

## Design Choices



### 1. Data in RAM

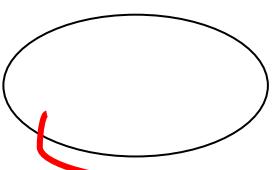
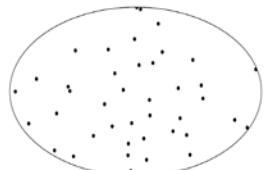
- Scan RAM sequentially & filter
  - Scan Time:  $1000 \text{ GB} * 0.25 \text{ msecs}/1\text{MB} = \underline{250 \text{ secs}}$
  - Cost (@100\$/16GB) ~ = 6000\$ of RAM

### 2. Data in disk (random spots)

- Seek each record on disk & filter
  - Scan Time: (Seek)  $10 \text{ msecs} * 1\text{Billion records} + (\text{Scan}) 1 \text{ TB} / 100 \text{ MB-sec}$ 
    - =  $10^7 \text{ secs} (115 \text{ days}) + 10^4 \text{ secs} \sim = \underline{115 \text{ days}}$
  - Cost (@100\$/TB of disk) = 100\$ of disk

### 3. Data in disk (sequentially organized)

- Seek to table, and sequentially scan records on disk & filter
  - Scan Time: (Seek)  $10 \text{ msecs} + (\text{Scan}) 1 \text{ TB} / 100 \text{ MB-sec}$ 
    - =  $10^4 \text{ secs} \sim = \underline{3 \text{ hrs}}$
  - Cost (@100\$/TB of disk) = 100\$ of disk



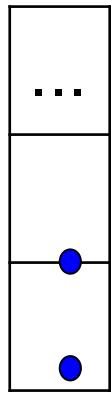
```
SELECT *
FROM Billion_Books
WHERE Author like 'Rowling'
```

## Input: Data size

1 Billion books  
Each record =  
1000 bytes  
(i.e., 1000 GBs or  
1 TB)

# Example: Search for books

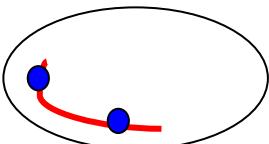
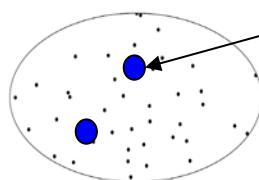
```
SELECT *  
FROM Billion_Books  
WHERE Author like 'Rowling'
```



Index in RAM

| Location | Author  |
|----------|---------|
|          | Rowling |
|          | Tolstoy |
|          | Rowling |
|          | ...     |

<Disk block,  
position>



Index => Maintain location of record

- Memory block
- Disk block (seek positions)

## Notes:

- $O(n)$  seeks for 'n' results
- RAM index costs \$\$ but speeds up
- Or index on disk (cz4031)
- Or index on index on index....(cz4031r)



# Indexes on a table

- An index speeds up selections on search key(s)
  - Any subset of fields
- Example

Books(BID, name, author, price, year, text)

On which attributes would you build indexes?

# Example

## Billion\_Books

| BID  | Title                       | Author      | Published | Full_text |
|------|-----------------------------|-------------|-----------|-----------|
| 1001 | <i>War and Peace</i>        | Tolstoy     | 1869      | ...       |
| 1002 | <i>Crime and Punishment</i> | Dostoyevsky | 1866      | ...       |
| 1003 | <i>Anna Karenina</i>        | Tolstoy     | 1877      | ...       |

```
SELECT *
FROM Billion_Books
WHERE Published > 1867
```

# Example

| By_Yr_Index |      | Billion_Books |                             |             |           |           |
|-------------|------|---------------|-----------------------------|-------------|-----------|-----------|
| Published   | BID  | BID           | Title                       | Author      | Published | Full_text |
| 1866        | 1002 | 1001          | <i>War and Peace</i>        | Tolstoy     | 1869      | ...       |
| 1869        | 1001 | 1002          | <i>Crime and Punishment</i> | Dostoyevsky | 1866      | ...       |
| 1877        | 1003 | 1003          | <i>Anna Karenina</i>        | Tolstoy     | 1877      | ...       |
| ...         |      | ...           |                             |             |           |           |

Maintain an index for this, and search over that!

Why might just keeping the table sorted by year not be good enough?

# Example

| By_Yr_Index |      |
|-------------|------|
| Published   | BID  |
| 1866        | 1002 |
| 1869        | 1001 |
| 1877        | 1003 |

Russian\_Novels

| BID  | Title                       | Author      | Published | Full_text |
|------|-----------------------------|-------------|-----------|-----------|
| 1001 | <i>War and Peace</i>        | Tolstoy     | 1869      | ...       |
| 1002 | <i>Crime and Punishment</i> | Dostoyevsky | 1866      | ...       |
| 1003 | <i>Anna Karenina</i>        | Tolstoy     | 1877      | ...       |

By\_Author\_Title\_Index

| Author      | Title                | BID  |
|-------------|----------------------|------|
| Dostoyevsky | Crime and Punishment | 1002 |
| Tolstoy     | Anna Karenina        | 1003 |
| Tolstoy     | War and Peace        | 1001 |

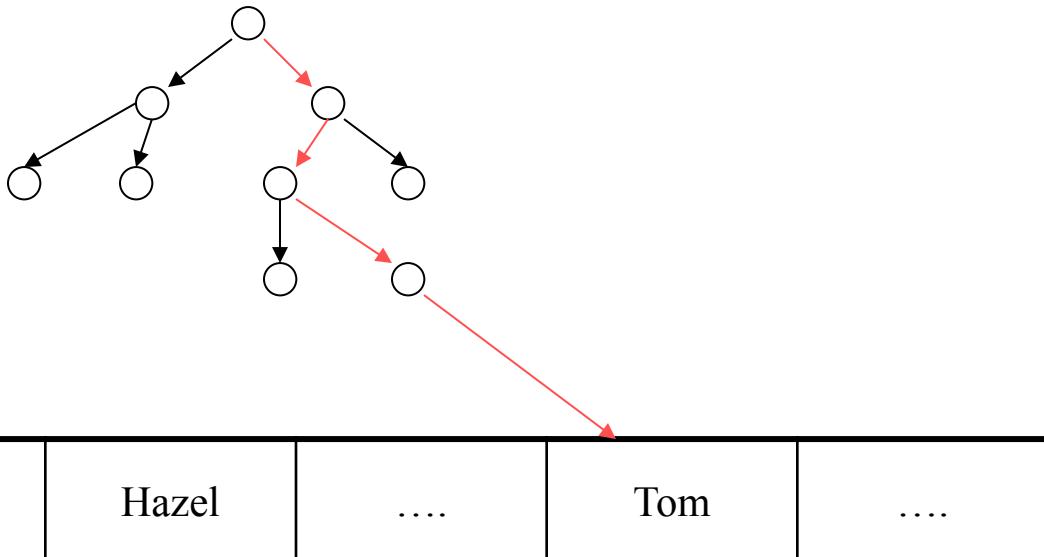
Can have multiple indexes to support multiple search keys

Indexes shown here as tables, but in reality we will use more efficient data structures...(CZ4031)

# Creating Indexes in Databases

## Indexes in databases

- Tree-structured (think of binary search tree)
- Hash-based



# Covering Indexes

By\_Yr\_Index

| Published | BID  |
|-----------|------|
| 1866      | 1002 |
| 1869      | 1001 |
| 1877      | 1003 |

An index **covers** for a specific query if the index contains all the needed attributes- *meaning the query can be answered using the index alone!*

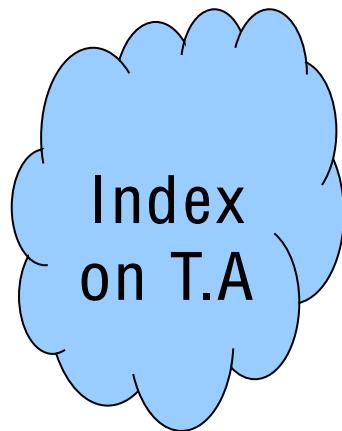
The “needed” attributes are the union of those in the SELECT and WHERE clauses...

Example:

```
SELECT Published, BID  
FROM Billion_Books  
WHERE Published > 1867
```

# Functionality

- Used by query processor to speed up data access

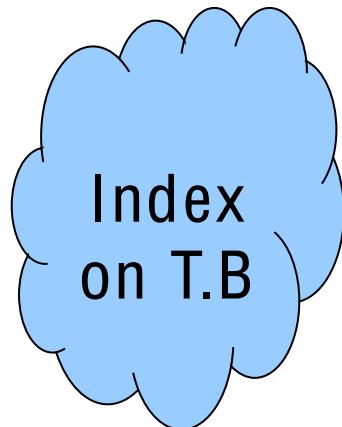


- $T.A = \text{'cow'}$
- $T.A = \text{'cat'}$

|   | A   | B   | C   |
|---|-----|-----|-----|
| 1 | cat | 2   | ... |
| 2 | dog | 5   | ... |
| 3 | cow | 1   | ... |
| 4 | dog | 9   | ... |
| 5 | cat | 2   | ... |
| 6 | cat | 8   | ... |
| 7 | cow | 6   | ... |
|   | ... | ... | ... |

# Functionality

- Used by query processor to speed up data access

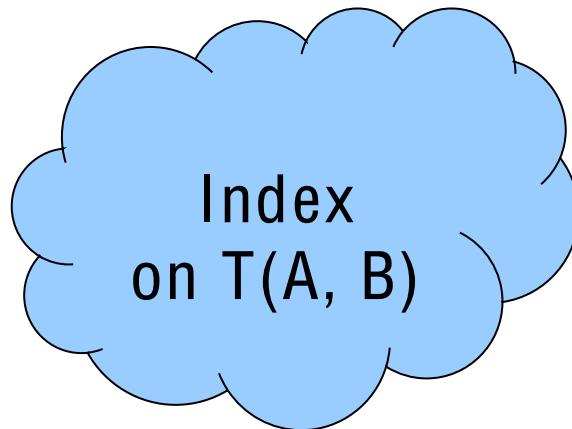


- $T.B = 2$
- $T.B < 4$
- $3 \leq T.B < 5$

|   | A   | B   | C   |
|---|-----|-----|-----|
| 1 | cat | 2   | ... |
| 2 | dog | 5   | ... |
| 3 | cow | 1   | ... |
| 4 | dog | 9   | ... |
| 5 | cat | 2   | ... |
| 6 | cat | 8   | ... |
| 7 | cow | 6   | ... |
|   | ... | ... | ... |

# Functionality

- Used by query processor to speed up data access



- $T.A = \text{'cat'}$  and  $T.B = 2$
- $T.A < \text{'d'}$  and  $T.B < 4$
- $3 \leq T.B < 5$

| T |     |     |     |
|---|-----|-----|-----|
|   | A   | B   | C   |
| 1 | cat | 2   | ... |
| 2 | dog | 5   | ... |
| 3 | cow | 1   | ... |
| 4 | dog | 9   | ... |
| 5 | cat | 2   | ... |
| 6 | cat | 8   | ... |
| 7 | cow | 6   | ... |
|   | ... | ... | ... |

# Answering Queries using Indexes

```
Select sName, cName  
From Student, Apply  
Where Student.sID = Apply.sID
```

- Scan Student, use an Index on Apply
- Scan Apply, use an Index on Student
- Use Indexes on both Apply and Student



# Indexes (definition)

An index is a **data structure** mapping search keys to sets of rows in table

- Provides efficient lookup & retrieval by search key value (usually much faster than scanning all rows and searching)

An index can store

- full rows it points to, OR
- pointers to rows



# Operations on an Index

- Search: Quickly find all records which meet some *condition on the search key attributes*
  - (Advanced: across rows, across tables)
- Insert / Remove entries
  - Bulk Load / Delete. Why?

Indexing is one of the most important features provided by a database for performance

# Why Not Store Everything in Main Memory (RAM)?

- **Main memory is volatile. But** We want data to be saved.
- **Cost too much:** Main memory is much more expensive!
- **Answer is Disk**
  - Many DB related issues involve hard disk I/O!
  - Thus we will now study how a hard disk works.

# Storing a Relation

## Recall

- Tuples are unordered
- Focus (in SQL) is on the tuples individually

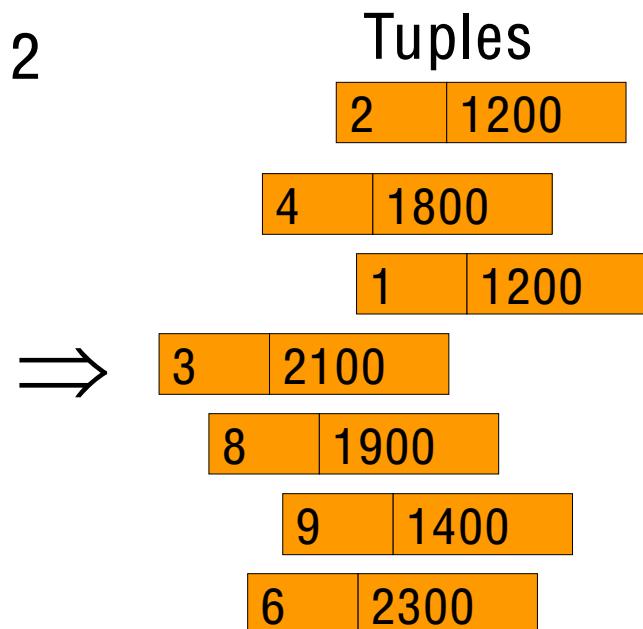
Relation table 1

| <u>E#</u> | Salary |
|-----------|--------|
| 3         | 2100   |
| 1         | 1200   |
| 8         | 1900   |
| 9         | 1400   |
| 2         | 1200   |
| 4         | 1800   |
| 6         | 2300   |

Relation table 2

| <u>E#</u> | Salary |
|-----------|--------|
| 1         | 1200   |
| 3         | 2100   |
| 4         | 1800   |
| 2         | 1200   |
| 6         | 2300   |
| 9         | 1400   |
| 8         | 1900   |

Identical!



# Indexing Definition in SQL

## Syntax

CREATE INDEX name ON rel (attr)

CREATE UNIQUE INDEX name ON rel (attr)

Duplicate values are not allowed

DROP INDEX name;

**Note:** The syntax for creating indexes varies amongst different databases.

## In practice

- **PRIMARY KEY declaration:** Automatically creates a primary/clustered index
- **UNIQUE declaration:** Automatically creates a secondary/nonclustered index

# Indexing Definition in SQL

- ❑ You can always specify which sets of attributes you want to build indexes
  - ❑ **Good:** Index on an attribute may speed up the execution of queries in which a value/a range of values are specified for the attribute, and may also help joins involving that attribute
  - ❑ **Bad:** it makes insertions, deletions, and updates slower

# Build index on attribute list

You can build an index on multiple attributes, also called **Composite index**

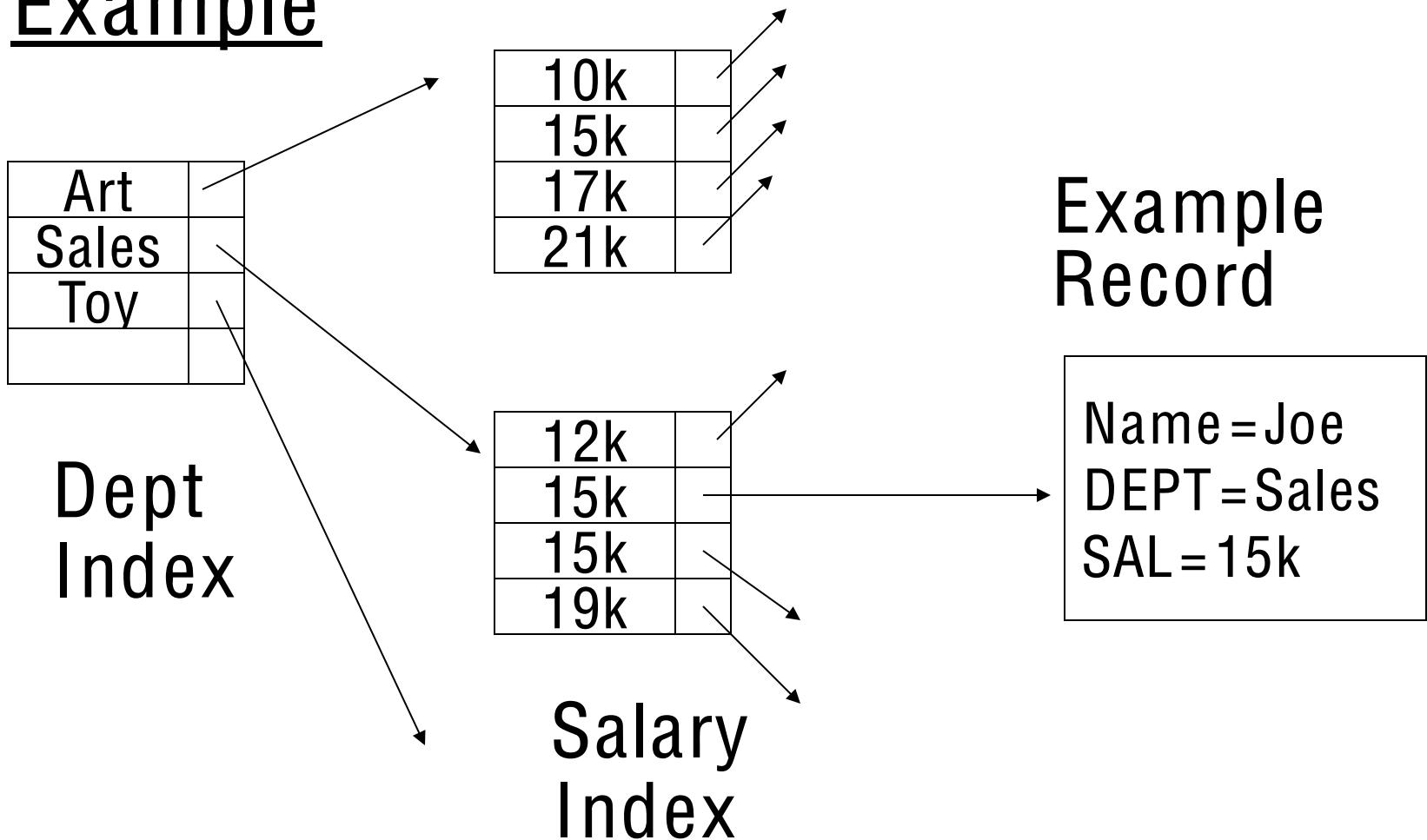
- ❑ Syntax: CREATE INDEX foo ON R(A,B,C)
- ❑ Example 1:
  - CREATE INDEX PnameIndex ON FacebookUser (firstname, lastname)
- ❑ Why?

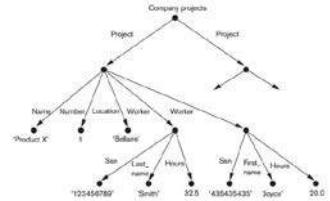
Motivation: Find records where  
DEPT = “Art” AND SAL > 50k

# Motivation

- ❑ Strategy 1: index on single attribute
  - ❑ Use one index on Dept: Get all Dept = "Art" records and check their salary
  - ❑ Use one index on Salary: Get all Salary > 50k records and check their Dept
- ❑ Strategy 3 Composite index:
  - ❑ Create index DeptSalaryIndex on EMP (Dept, Salary)
    - ❑ See next slide
  - ❑ Create index SalaryDeptIndex on EMP (Salary, Dept)

# Example





# CZ2007

# Introduction to Databases

## Semi-Structured Data

Cong Gao

Professor

School of Computer Science and Engineering  
Nanyang Technological University, Singapore

# Schedule after Recess Week

## SQL

- Week 8
- Week 9
- Week 10
- Week 11

## Semi-Structured Data

- Week 12 (No Lecture on 11 April)
- Week 13

## Quiz-2

- Week 12
- Quiz during **Tutorial** session
- Quiz syllabus: The first 4 weeks for Quiz

# Roadmap (Semi-Structured Data)

- **Semi-structured Data**
- XML
- XML DTD
- JSON

# The More Data, The Merrier

## Power of Data

- the more data the merrier (GB -> TB -> PB)
- data comes from everywhere in all shapes
- value of data often discovered later

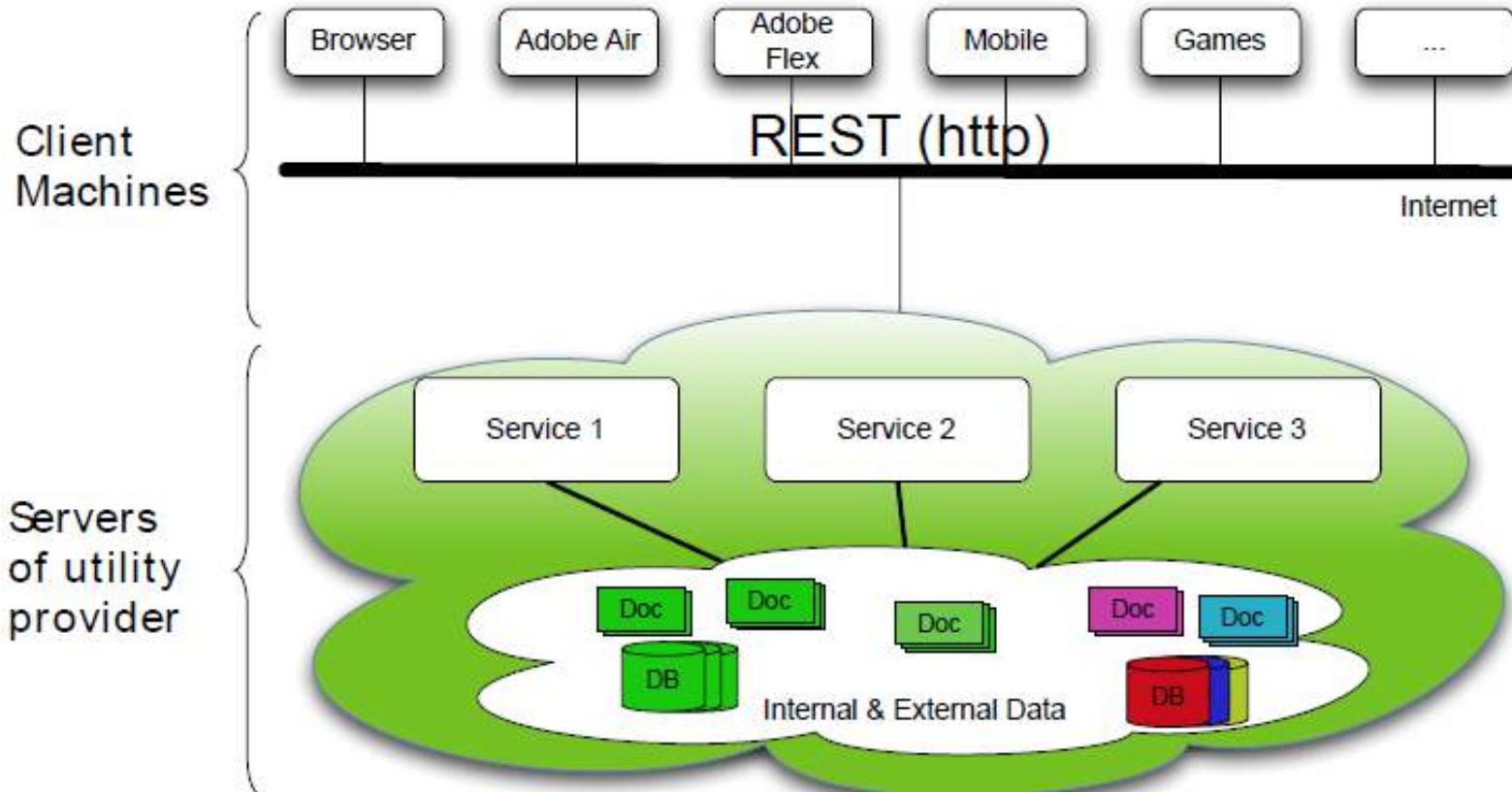
## Services turn data into \$\$\$

- the more services the merrier (10 -> 1000 -> 1M -> 1B)
- need to adapt quickly

## Goal: Platforms for data and services

- any data, any service, anywhere and anytime

# Data Arrive in Many Shapes



# Structured vs. Unstructured Data

| Patient No. | Last name | First name  | Sex | Date of birth | Ward No. |
|-------------|-----------|-------------|-----|---------------|----------|
| 454         | Smith     | John        | M   | 14.08.58      | 6        |
| 223         | Jones     | Peter       | M   | 07.12.65      | 8        |
| 597         | Brown     | Brenda      | F   | 17.06.61      | 3        |
| 234         | Jenkins   | Alan        | M   | 29.01.67      | 7        |
| 244         | Wells     | Christopher | M   | 25.02.55      | 6        |

## Relational databases are highly structured

- All data resides in tables
- Must define schema before entering data
- Every row conforms to the table schema
- Changing the schema is hard and may break many things

| Ward No. | Ward name | Type     | No. of Beds |
|----------|-----------|----------|-------------|
| 3        | Carey     | Medical  | 8           |
| 6        | Bracken   | Medical  | 16          |
| 7        | Brent     | Surgical | 12          |
| 8        | Meavy     | Surgical | 10          |

## Texts are highly unstructured

- Data is free-form
- No schema and it's hard to define one
- Readers need to infer structures & meanings

signal. binary code with which the present is may take various forms, all of e property that the symbol (or representing each number (or sign) differs from the ones representi er and the next higher number ( oritude) in only one digit (or puls Because this code in its primary built up from the conventional a sort of reflection process and rums may in turn be built up free form in similar fashion, the c , which has as yet no recognized nated in this specification and s the "reflected binary code." a receiver station, reflected bina

## What's in between these two extremes?

# Semi-Structured Data

## Observation: most data have “some” structure, e.g.

- Book: chapters, sections, titles, paragraphs, references, index, etc.
- Item for sale: name, picture, price, ratings, promotion, etc.
- Web page: HTML

## Ideas

- Ensure data is “well-formatted”
- If needed, ensure data is also “well-structured”
  - But make it easy to define and extend this structure
- Make data “self-describing”

# A Little Bit of History ...

## *Database world*

- 1970 relational databases
- 1990 nested relational model and object oriented databases
- 1995 semi-structured databases

## *Documents world*

- 1974 **SGML** (Structured Generalized Markup Language)
- 1990 **HTML** (Hypertext Markup Language)
- 1992 **URL** (Universal Resource Locator)

*Data + documents = information*

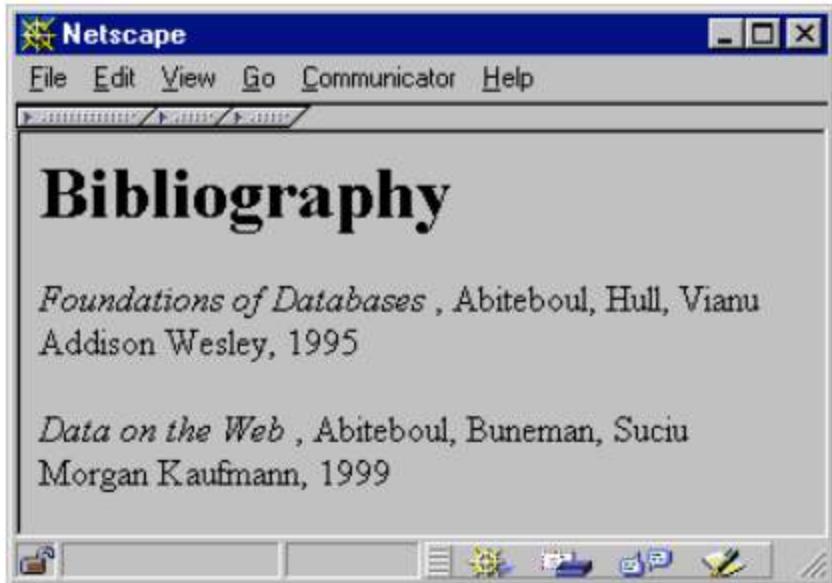
1996 **XML** (Extended Markup Language)

**URI** (Universal Resource Identifier)

# XML as Semi-Structured Data

- XML - The EXtensible Markup Language
- A flexible syntax for data: semi-structured data
- Used in:
  - Configuration files, e.g. Web.Config
  - Replacement for binary formats (MS Word)
  - Document markup: e.g. XHTML
  - Data: data exchange, semistructured data (sensor data, logs, blogs)
- Warning: not normal form! Not even 1NF
- XML is about half as popular as SQL

# From HTML to XML



HTML

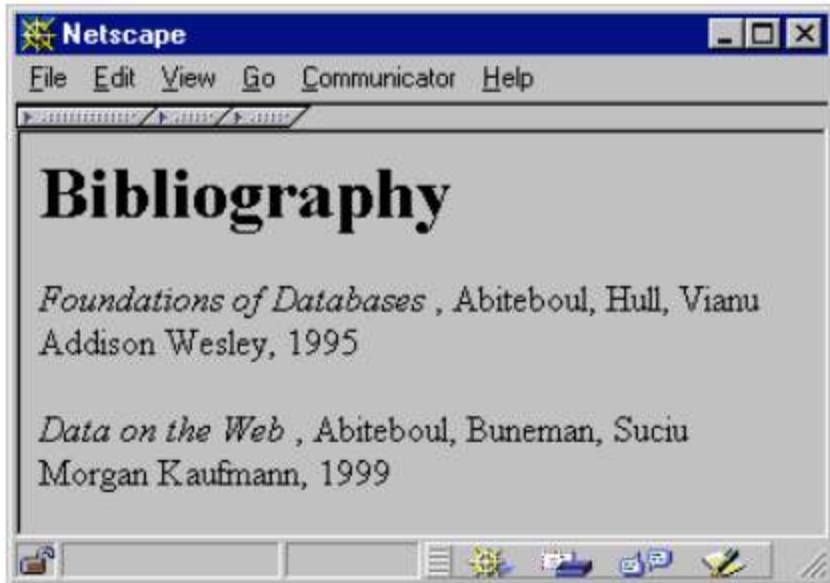
- The HyperText Markup Language

HTML describes the presentation

HTML

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
    Abiteboul, Hull, Vianu
    <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
    Abiteboul, Buneman, Suciu
    <br> Morgan Kaufmann, 1999
```

# From HTML to XML



HTML

- The HyperText Markup Language

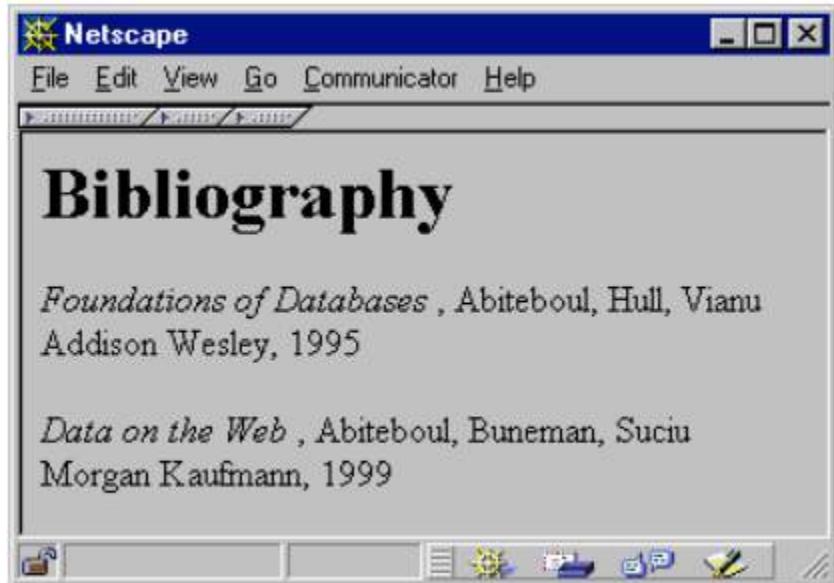
HTML describes the presentation

- It's mostly a “formatting” language
- It mixes presentation and content

HTML

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
    Abiteboul, Hull, Vianu
    <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
    Abiteboul, Buneman, Suciu
    <br> Morgan Kaufmann, 1999
```

# From HTML to XML



XML describes the content

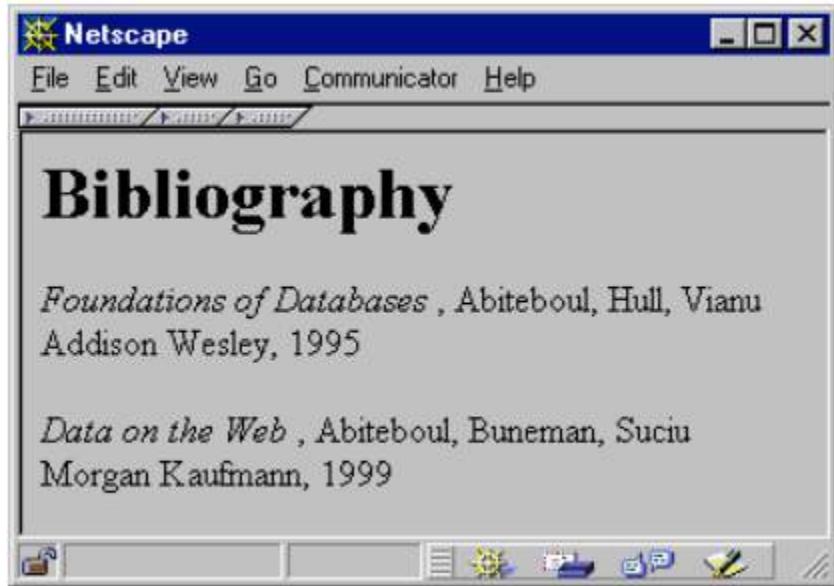
## XML

- The EXtensible Markup Language

## XML Syntax

```
<bibliography>
  <book>    <title> Foundations... </title>
            <author> Abiteboul </author>
            <author> Hull </author>
            <author> Vianu </author>
            <publisher> Addison Wesley </publisher>
            <year> 1995 </year>
  </book>
  ...
</bibliography>
```

# From HTML to XML



XML describes the content

- Text-based
- Capture data (content),  
not presentation
- Data self-describes its  
structure
  - Names and nesting of tags  
have meanings!

## XML

- The EXtensible Markup Language

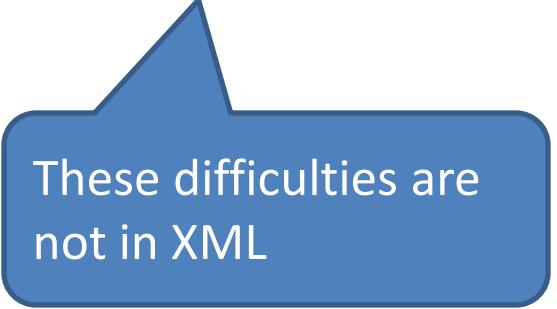
## XML Syntax

```
<bibliography>
  <book>    <title> Foundations... </title>
            <author> Abiteboul </author>
            <author> Hull </author>
            <author> Vianu </author>
            <publisher> Addison Wesley </publisher>
            <year> 1995 </year>
  </book>
  ...
</bibliography>
```

# HTML vs. XML

## Difficulties with HTML ?

- Fixed set of tags
- Elements have document structuring semantics
- For presentation to human readers
- Applications cannot consume and process HTML easily



These difficulties are  
not in XML

# XML Terminology

- Tag names: book, title, ...
- Start tags: <book>, <title>, ...
- End tags: </book>, </title>, ...
- An element is enclosed by a pair of start and end tags:  
`<book>...</book>`
- Elements can be nested:  
`<book>...<title>...</title>...</book>`
- Empty elements:  
`<is_textbook></is_textbook>`  
- Can be abbreviated:  
`<is_textbook/>`
- Elements can also have attributes: `<book ISBN="..." price="80.00">`

```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>...
</bibliography>
```

Ordering generally matters, except for attributes

# Well-formed XML documents

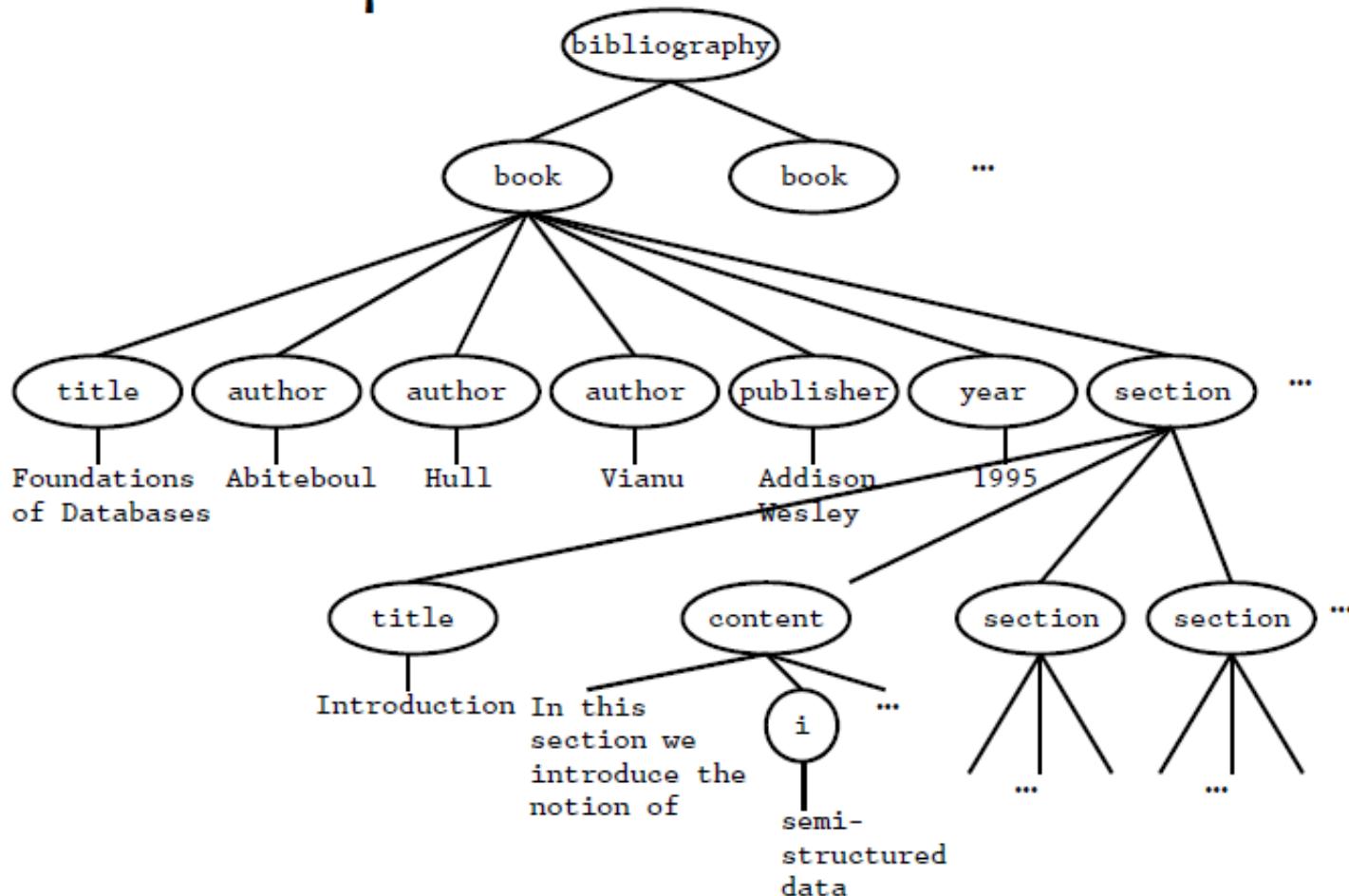
## A well-formed XML document

- Follows XML lexical conventions
  - Wrong: <section>We show that x < 0...</section>
  - Right: <section>We show that x &lt; 0...</section>
  - Other special entities: > becomes &gt; and & becomes &amp;
- Contains a single root element
- Has properly matched tags and properly nested elements
  - Right: <section>...<subsection>...</subsection>...</section>
  - Wrong: <section>...<subsection>...</section>...</subsection>

# Tree Representation of XML Documents

11

## A tree representation



# More XML Example: Attributes

```
<book price = "55" currency = "USD">
    <title> Foundations of Databases </title>
    <author> Abiteboul </author>
    ...
    <year> 1995 </year>
</book>
```

# Attributes vs. Elements

```
<book price = "55" currency = "USD">
    <title> Foundations of DBs </title>
    <author> Abiteboul </author>
    ...
    <year> 1995 </year>
</book>
```

```
<book>
    <title> Foundations of DBs </title>
    <author> Abiteboul </author>
    ...
    <year> 1995 </year>
    <price> 55 </price>
    <currency> USD </currency>
</book>
```

attributes are alternative ways to represent data

# Attributes vs. Elements

| Elements        | Attributes     |
|-----------------|----------------|
| Ordered         | Unordered      |
| May be repeated | Must be unique |
| May be nested   | Must be atomic |

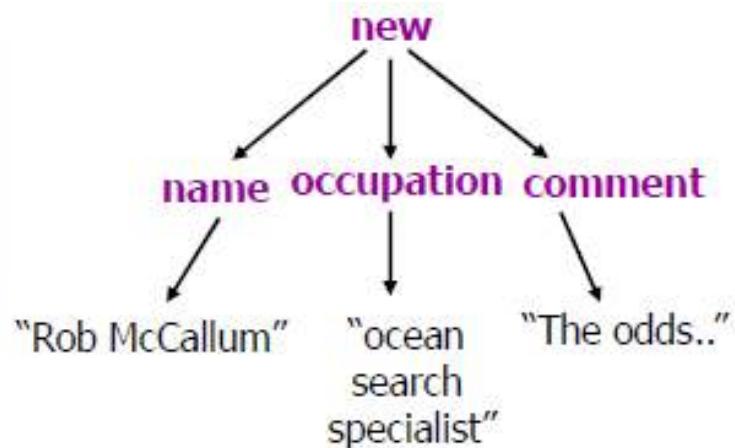
Attribute names must be unique! (No Multisets)  
`<person name = "Wilde" name = "Wutz"/>` is illegal!

# Documents to XML

Documents are a quite natural way to represent “objects”

- A great deal of text and semi-structured info

```
... <comment> "The odds of finding the pinger are  
very slim," </comment> said <name>Rob  
McCallum</name>, an <occupation> ocean  
search specialist </occupation>. "Even when  
you know roughly where the target is, it can be very  
tricky to find the pinger. They have a very limited  
range." ...
```



```
<news>  
  <name>Rob McCallum</name>  
  <occupation>ocean search specialist</occupation>  
  <comment> The odds of finding the pinger are very slim </comment>  
</news>
```

# Benefits of XML over Relational Data

- **Portability**: Just like HTML, you can ship XML data across platforms
  - Relational data requires heavy-weight API's
- 
- **Flexibility**: You can represent any information (structured, semi-structured, documents, ...)
  - Relational data is best suited for structured data
- 
- **Extensibility**: Since data describes itself, you can change the schema easily
  - Relational schema is rigid and difficult to change

# XML vs. Relational Data

- Relational data
  - *Killer application:* Banking
  - Invented as a mathematically clean *abstract data model*
  - *Philosophy:* schema first, then data
- XML
  - First *killer application:* publishing industry
  - Invented as a *syntax for data*, only later an abstract data model
  - *Philosophy:* data and schemas should not be correlated, data can exist with or without schema, or with multiple schemas

# XML vs. Relational Data

## ■ Relational data

- Never had a *standard syntax* for data
- Strict rules for data *normalization*, flat tables
- *Order* is irrelevant, textual data supported but not primary goal

## ■ XML

- *Standard syntax* existed before the data model
- No data *normalization*, flexibility is a must, nesting is good
- *Order may be very important*, textual data support a primary goal

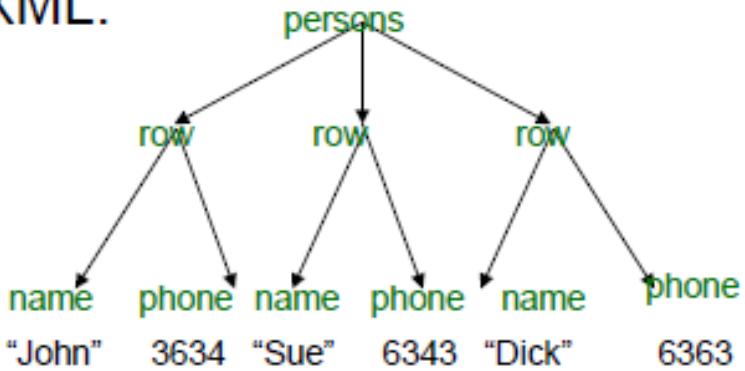
# Mapping Relational Data to XML

XML view of relational data

Persons

| Name | Phone |
|------|-------|
| John | 3634  |
| Sue  | 6343  |
| Dick | 6363  |

XML:



```

<persons>
  <row> <name>John</name>
        <phone> 3634</phone></row>
  <row> <name>Sue</name>
        <phone> 6343</phone>
  <row> <name>Dick</name>
        <phone> 6363</phone></row>
</persons>
  
```

# Mapping Relational Data to XML

XML view of relational data

Persons

| Name | Phone |
|------|-------|
| John | 3634  |
| Sue  | 6343  |

Orders

| PersonName | Date | Product |
|------------|------|---------|
| John       | 2002 | Gizmo   |
| John       | 2004 | Gadget  |
| Sue        | 2002 | Gadget  |

XML

```

<persons>
  <person>
    <name> John </name>
    <phone> 3634 </phone>
    <order> <date> 2002 </date>
              <product> Gizmo </product>
    </order>
    <order> <date> 2004 </date>
              <product> Gadget </product>
    </order>
  </person>
  <person>
    <name> Sue </name>
    <phone> 6343 </phone>
    <order> <date> 2004 </date>
              <product> Gadget </product>
    </order>
  </person>
</persons>

```

# XML is Semi-Structured

- Missing attributes:

```
<person> <name> John</name>
          <phone>1234</phone>
</person>
```

```
<person> <name> Joe</name>
</person>
```

no phone !

- Could represent in a table with nulls

| name | phone |
|------|-------|
| John | 1234  |
| Joe  | -     |

# XML is Semi-Structured

- Repeated attributes

```
<person> <name> Mary</name>
    <phone>2345</phone>
    <phone>3456</phone>
</person>
```

Two phones !

- Impossible in tables:

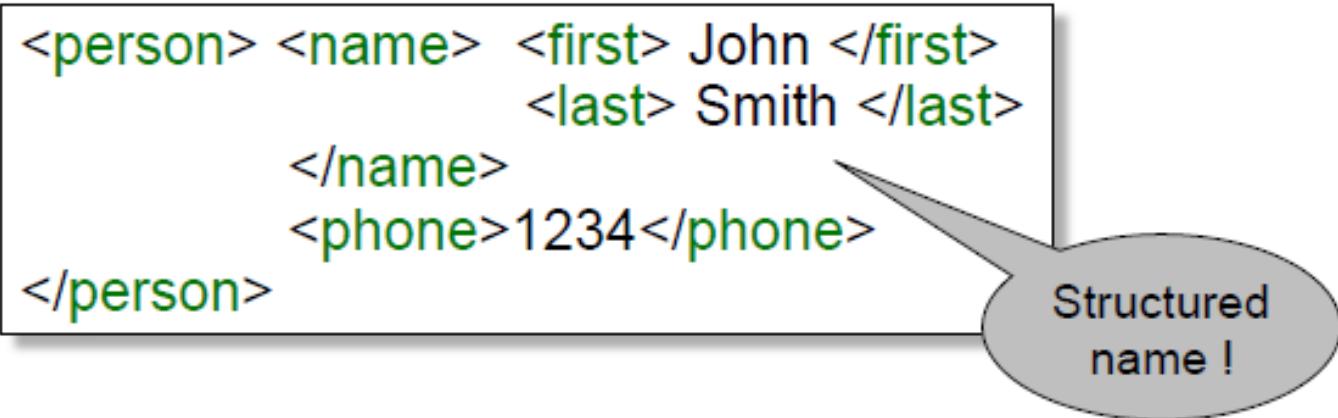
| name | phone |      | ??? |
|------|-------|------|-----|
| Mary | 2345  | 3456 |     |
|      |       |      |     |

# XML is Semi-Structured

## XML is Semi-structured Data

- Attributes with different types in different objects

```
<person> <name> <first> John </first>
          <last> Smith </last>
        </name>
        <phone>1234</phone>
</person>
```



Structured  
name !

- Nested collections (no 1NF)

# Questions?

- Semi-structured Data ✓
- XML ✓
- XML DTD
- JSON

# XML Format Descriptions

- Easy to start with, use your own tags
  - Contrast to relational DB, OO languages
- Only restriction: XML needs to be well-formed
- At some point, this is too much freedom
  - Use same syntax for different documents
  - Facilitate the writing of applications that process data
  - Exchange data with other parties
- Need to restrict the amount of freedom
  - Document Description Methods

# Overview of XML Schema Languages

- Several standard Schema Languages
  - **DTDs**, XML Schema, RelaxNG, Schematron
- Schema languages have been designed after, and in an orthogonal fashion, to XML itself
- **Schemas and data are decoupled in XML**
  - Data can exist with or without schemas
  - Or with multiple schemas
  - Schema evolutions rarely impose evolving the data
  - Schemas can be designed before the data, or extracted from the data
- Makes XML the right choice for manipulating semi-structured data, or rapidly evolving data, or highly customizable data

# Document Type Definition (DTD)

## Goals:

- Define what tags and attributes are allowed
- Define how they are nested
- Define how they are ordered

Superseded by XML Schema

- Very complex: DTDs still used widely

# Element Type Declaration

- Element Types are composed of:
  - Subelements (identified by Name)
  - Attribute lists (identified by Name)
  - Selection of Subelements (choice)
  - PCDATA text that WILL be parsed by a parser
- Quantifier for Subelements and Choice
  - "+" for at least 1
  - "\*" for 0 or more
  - "?" for 0 or 1
  - Default: exactly 1
- EMPTY and ANY are special predefined Types

```
<!ELEMENT element-name category>
```

or

```
<!ELEMENT element-name (element-content)>
```

Example:

```
<!ELEMENT br EMPTY>
```

XML example:

```
<br />
```

# Element Type Declaration

- Structure: <!ELEMENT *name* *content*>
- Example

```
<!ELEMENT book (title, (author+ | editor), publisher?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author EMPTY>
<!ELEMENT publisher ANY>
```

- Valid document according to this DTD

```
<book>
  <title>Die wilde Wutz</title>
  <author/> <author></author>
  <publisher><anything>...</anything></publisher>
</book>
```

# Declaring Attributes

- An attribute declaration has the following syntax:
- `<!ATTLIST element-name attribute-name attribute-type attribute-value>`

DTD example:

```
<!ATTLIST payment type CDATA "check">
```

XML example:

```
<payment type="check" />
```

# attribute-type

The **attribute-type** can be one of the following:

| Type         | Description                                   |
|--------------|---|
| CDATA        | The value is character data                   |
| (en1 en2 ..) | The value must be one from an enumerated list |
| ID           | The value is a unique id                      |
| IDREF        | The value is the id of another element        |
| IDREFS       | The value is a list of other ids              |
| NMTOKEN      | The value is a valid XML name                 |
| NMTOKENS     | The value is a list of valid XML names        |
| ENTITY       | The value is an entity                        |
| ENTITIES     | The value is a list of entities               |
| NOTATION     | The value is a name of a notation             |
| xml:         | The value is a predefined xml value           |

# attribute-value

The **attribute-value** can be one of the following:

| Value               | Explanation                        |
|---------------------|------------------------------------|
| <i>value</i>        | The default value of the attribute |
| #REQUIRED           | The attribute is required          |
| #IMPLIED            | The attribute is optional          |
| #FIXED <i>value</i> | The attribute value is fixed       |

Default Attribute Value

DTD:

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

Valid XML:

```
<square width="100" />
```

In the example above, the "square" element is defined to be an empty element with a "width" attribute of type CDATA. If no width is specified, it has a default value of 0.

# Attribute type--#REQUIRED

- Syntax: <!ATTLIST element-name attribute-name attribute-type #REQUIRED>
- Example
- DTD:  
<!ATTLIST person number CDATA #REQUIRED>

Valid XML:

```
<person number="5677" />
```

Invalid XML:

```
<person />
```

- Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

# Attribute type-- #IMPLIED

- Syntax: <!ATTLIST element-name attribute-name attribute-type #IMPLIED>
- Example
- DTD:  
<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:

```
<contact fax="555-667788" />
```

Valid XML:

```
<contact />
```

- Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

# Attribute type (#fixed)

- Syntax
- ```
<!ATTLIST element-name attribute-name attribute-type #FIXED  
"value">
```
- Example
- DTD:  

```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

Valid XML:

```
<sender company="Microsoft" />
```

Invalid XML:

```
<sender company="W3Schools" />
```

- Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

# Attribute Lists

- Structure: <!ATTLIST *ElementName definition*>

- <!ATTLIST book  
    isbn ID #REQUIRED  
    price CDATA #IMPLIED  
    curr CDATA #FIXED "EUR"  
    index IDREFS "" >

- Valid and Not-valid Books

- ```
<book isbn="abc" curr="EUR"/> !! no price
```
- ```
<book isbn="abc" price="30"/> !! Curr, index default
```
- ```
<book index="DE" isbn="abc" curr="EUR"/>
```
- ```
<book/> !! Missing isbn Attribute
```
- ```
<book isbn="abc" curr="USD"/> !! wrong currency
```

# Entity

| Entity References | Character |
|-------------------|-----------|
| &lt;              | <         |
| &gt;              | >         |
| &amp;             | &         |
| &quot;            | "         |
| &apos;            | '         |

- Syntax: <!ENTITY entity-name "entity-value">
- DTD Example:  
    <!ENTITY writer "Donald Duck.">  
    <!ENTITY copyright "Copyright W3Schools.">
- XML example:  
    <author>&writer;&copyright;</author>

**Note:** An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

# Entity

- An External Entity Declaration
- Syntax `<!ENTITY entity-name SYSTEM "URI/URL">`
- Example
- DTD Example:

```
<!ENTITY writer SYSTEM  
"https://www.w3schools.com/entities.dtd">  
<!ENTITY copyright SYSTEM  
"https://www.w3schools.com/entities.dtd">
```

- XML example:
- ```
<author>&writer;&copyright;</author>
```

# DTD Example

```
<!ELEMENT book (title, (author+ | editor), publisher?)>
<!ATTLIST book
    year CDATA #REQUIRED
    isbn   ID      #REQUIRED
    price  CDATA  #IMPLIED
    curr   CDATA  #FIXED "EUR"
    index  IDREFS  "" >
<!ELEMENT author (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

# SUMMARY

- Semi-structured Data ✓
- XML ✓
- XML DTD ✓
- JSON

# Difficulties with XML

- “Tree, and not a graph.”
  - Difficulty in modeling N:M relationships
  - The notion of reference (e.g. XLink, XPointer) not well integrated in the XML stack
- “Duplication of concepts”
  - Many ways to do the same thing
  - Justification for a “simpler” data model like RDF
- “Concepts that seem logically unnecessary”
  - PIs, comments, documents, etc
- Additional complexity factors
  - xsi:nil, QName in content, etc
- “Boring”
  - so is the (enterprise) world where XML lives

# Other Semi-Structured Data

- JSON
- CSV
- Avro
- Protocol Buffers
- RDF
- Property Graphs
- ...

# Why do we still talk about XML ?

- It is a standard (not owned by anybody)
- Very well documented
- Many tools available
- Mother of all semi-structured data
- has the most features
- XML is here to stay
- It actually works!

# JSON

## JSON

- **JavaScript Object Notation**
  - lightweight text-based open standard designed for human-readable data interchange.
- Interfaces in C, C++, Java, Python, Perl, etc.
- The filename extension is **.json**.

## Semistructured data model

- Flexible, nested structure (trees)
- Does not require predefined schema ("self describing")
- Text representation: good for exchange, bad for performance
- Most common use: Language API

# JSON - Syntax

```
{ "book": [  
    {"id": "01",  
        "language": "Java",  
        "author": "H. Javeson",  
        "year": 2015  
    },  
    {"id": "07",  
        "language": "C++",  
        "edition": "second"  
        "author": "E. Sepp",  
        "price": 22.25  
    }  
]
```

# JSON - Terminology

## Curly braces

- Hold objects
- Each object is a list of name/value pairs separated by , (comma)
- Each pair is a name followed by ':' (colon) followed by the value

## Square brackets

- Hold arrays and values are separated by , (comma).

## What is the data made up of?

- Objects, lists, and atomic values (integers, floats, strings, booleans).

# JSON – Data Structure

## Collection

- Collections of name-value pairs:
  - {"name1": value1, "name2": value2, ...}
- The “name” is also called a “key”
- Ordered lists of values: [obj1, obj2, obj3, ...]

# XML vs. JSON

## XML

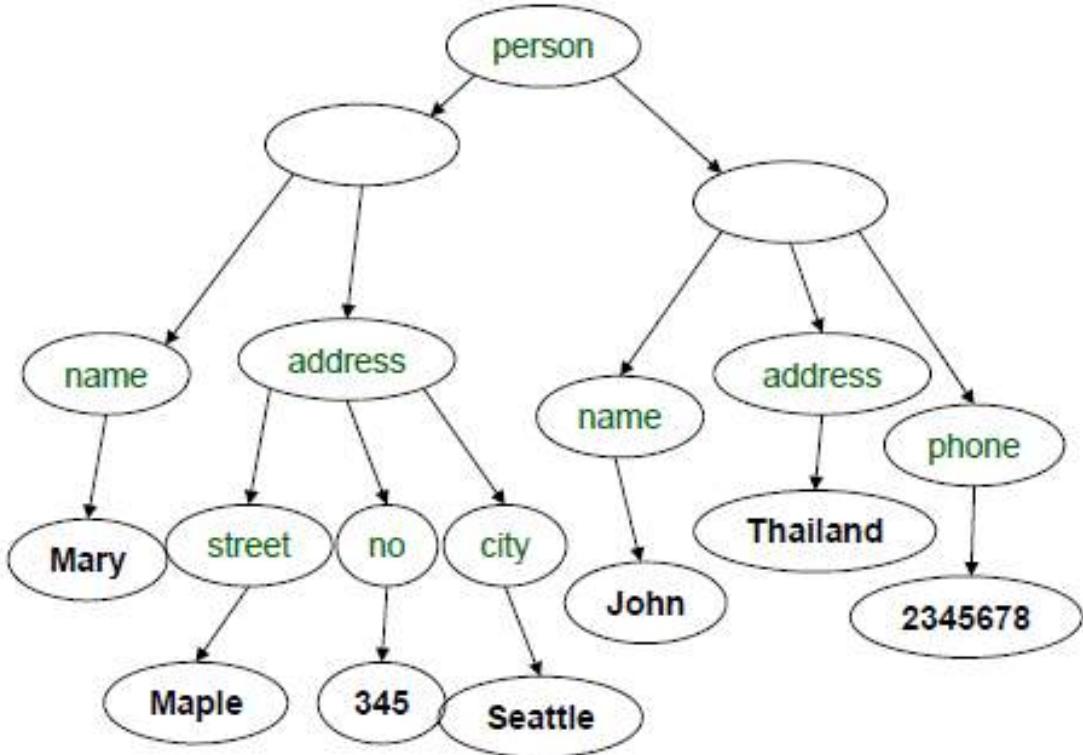
```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

## JSON

```
{ "empinfo" :
  {
    "employees": [
      {
        "name": "James Kirk",
        "age": 40,
      },
      {
        "name": "Jean-Luc Picard",
        "age": 45,
      },
      {
        "name": "Wesley Crusher",
        "age": 27,
      }
    ]
  }
}
```

# Tree View of JSON Data

```
{
  "person": [
    { "name": "Mary",
      "address": {
        "street": "Maple",
        "no": 345,
        "city": "Seattle" },
      {"name": "John",
        "address": "Thailand",
        "phone": 2345678}
    ]
  }
}
```

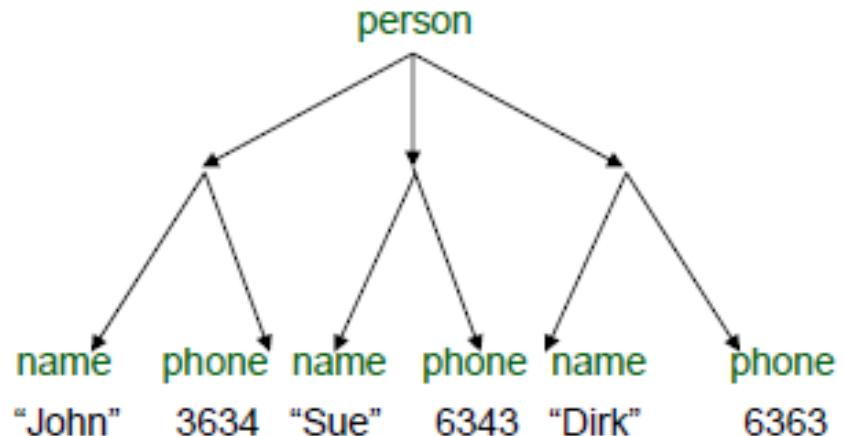


## Self-describing

# Mapping Relational Data to JSON

Person

| name | phone |
|------|-------|
| John | 3634  |
| Sue  | 6343  |
| Dirk | 6363  |



```
{
  "person": [
    {"name": "John", "phone": 3634},
    {"name": "Sue", "phone": 6343},
    {"name": "Dirk", "phone": 6383}
  ]
}
```

# Mapping Relational Data to JSON

Person

| name | phone |
|------|-------|
| John | 3634  |
| Sue  | 6343  |

Orders

| personName | date | product |
|------------|------|---------|
| John       | 2002 | Gizmo   |
| John       | 2004 | Gadget  |
| Sue        | 2002 | Gadget  |

```
{
  "Person": [
    {"name": "John",
     "phone": 3646,
     "Orders": [{"date": 2002,
                 "product": "Gizmo"}, {"date": 2004,
                 "product": "Gadget"}]},
    {"name": "Sue",
     "phone": 6343,
     "Orders": [{"date": 2002,
                 "product": "Gadget"}]}
  ]
}
```

# Handling NULL and Repeated Values

| name | phone |
|------|-------|
| John | 1234  |
| Joe  | -     |

```
{"person":  
  [{"name": "John", "phone": 1234},  
   {"name": "Joe"}]  
}
```

no phone !

```
{"person":  
  [{"name": "John", "phone": 1234},  
   {"name": "Mary", "phone": [1234, 5678]}]  
}
```

Two phones !

# Handling Heterogeneous Objects

```
{"person":  
  [{"name": "Sue", "phone": 3456},  
   {"name": {"first": "John", "last": "Smith"}, "phone": 2345}  
  ]  
}
```

Structured  
name !

- Nested collections
- Heterogeneous collections

# Summary

## Data Exchange Format

- Well suited for exchanging data between applications
- XML, JSON

## Data Models

- Some systems use them as data models
- SQL Server – supports XML-valued relations
- CouchBase, Mongodb – JSON as data model

## Query Languages

- Xpath, Xquery
- CouchBase – N1QL
- JSONiq

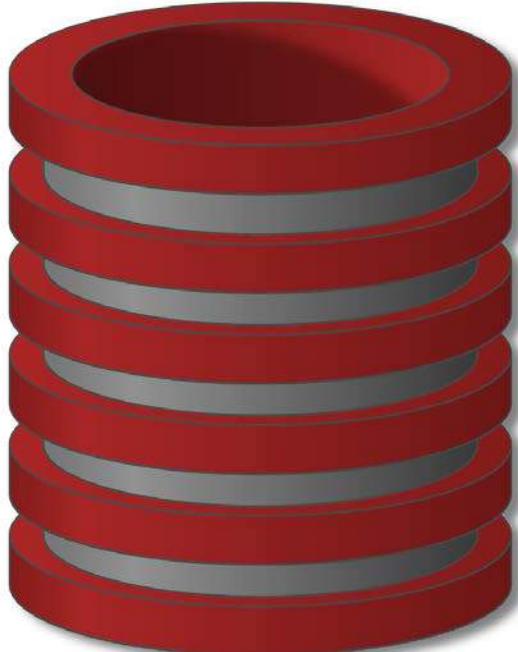
Will NOT discuss in this lecture!

# Questions ??

---



Thank You !



# NoSQL Systems

---

## Motivation

# NoSQL: The Name

- “SQL” = Traditional relational DBMS
- Recognition over past decade or so:  
Not every data management/analysis problem is best solved using a traditional relational DBMS
- “NoSQL” = “No SQL” =  
Not using traditional relational DBMS
- “No SQL” ≠ Don’t use SQL language

# NoSQL: The Name

- “SQL” = Traditional relational DBMS
- Recognition over past decade or so:  
**Not every data management/analysis problem is best solved using a traditional relational DBMS**
- “NoSQL” = “No SQL” =  
    Not using traditional relational DBMS
- “No SQL” ≠ Don’t use SQL language
- \* “NoSQL” = “Not Only SQL”

# Not every data management/analysis problem is best solved using a traditional DBMS

Database Management System (DBMS) provides....

... efficient, reliable, convenient, and safe multi-user storage of and access to massive amounts of persistent data.

# NoSQL Systems

## Alternative to traditional relational DBMS

- + Flexible schema
- + Quicker/cheaper to set up
- + Massive scalability
- + Relaxed consistency → higher performance & availability
  
- No declarative query language → more programming
- Relaxed consistency → fewer guarantees

## Example #1: Web log analysis

Each record: UserID, URL, timestamp, additional-info

Task: Load into database system

## Example #1: Web log analysis

Each record: UserID, URL, timestamp, additional-info

Task: Find all records for...

- Given UserID
- Given URL
- Given timestamp
- Certain construct appearing in additional-info

## Example #1: Web log analysis

Each record: UserID, URL, timestamp, additional-info

Separate records: UserID, name, age, gender, ...

Task: Find average age of user accessing given URL

## Example #2: Wikipedia pages

Large collection of documents

Combination of structured and unstructured data

Task: Retrieve introductory paragraph of all pages about U.S. presidents before 1900

# NoSQL Systems

## Alternative to traditional relational DBMS

- + Flexible schema
- + Quicker/cheaper to set up
- + Massive scalability
- + Relaxed consistency → higher performance & availability
  
- No declarative query language → more programming
- Relaxed consistency → fewer guarantees



# NoSQL Systems

---

## Overview

# NoSQL Systems

- Not every data management/analysis problem is best solved *exclusively* using a traditional DBMS
- “NoSQL” = “Not Only SQL”

# NoSQL Systems

## Alternative to traditional relational DBMS

- + Flexible schema
- + Quicker/cheaper to set up
- + Massive scalability
- + Relaxed consistency → higher performance & availability
  
- No declarative query language → more programming
- Relaxed consistency → fewer guarantees

# NoSQL Systems

## Several incarnations

- MapReduce framework
- Key-value stores
- Document stores
- Graph database systems

# MapReduce Framework

Originally from Google, open source Hadoop

- No data model, data stored in files
- User provides specific functions  
map() reduce()
- System provides data processing “glue”, fault-tolerance, scalability

# Map and Reduce Functions

**Map:** Divide problem into subproblems

**Reduce:** Do work on subproblems, combine results

# MapReduce Architecture

# MapReduce Example: Web log analysis

Each record: UserID, URL, timestamp, additional-info

Task: Count number of accesses for each domain (inside URL)

# MapReduce Example (modified #1)

Each record: UserID, URL, timestamp, additional-info

Task: Total “value” of accesses for each domain based on additional-info

# MapReduce Framework

- No data model, data stored in files
- User provides specific functions
- System provides data processing “glue”, fault-tolerance, scalability

# MapReduce Framework

Schemas and declarative queries are missed

**Hive** – schemas, SQL-like query language

**Pig** – more imperative but with relational operators

- Both compile to “workflow” of Hadoop (MapReduce) jobs

# Key-Value Stores

Extremely simple interface

- Data model: (key, value) pairs
- Operations: Insert(key,value), Fetch(key),  
Update(key), Delete(key)

Implementation: efficiency, scalability, fault-tolerance

- Records distributed to nodes based on key
- Replication
- Single-record transactions, “eventual consistency”

# Key-Value Stores

## Extremely simple interface

- Data model: (key, value) pairs
- Operations: Insert(key,value), Fetch(key),  
Update(key), Delete(key)
- Some allow (non-uniform) columns within value
- Some allow Fetch on range of keys

## Example systems

- Google BigTable, Amazon Dynamo, Cassandra,  
Voldemort, HBase, ...

# Document Stores

Like Key-Value Stores except value is document

- Data model: (key, document) pairs
- Document: JSON, XML, other semistructured formats
- Basic operations: Insert(key,document), Fetch(key),  
Update(key), Delete(key)
- Also Fetch based on document contents

Example systems

- CouchDB, MongoDB, SimpleDB, ...

# NoSQL Systems

- “NoSQL” = “Not Only SQL”
  - Not every data management/analysis problem is best solved *exclusively* using a traditional DBMS
- Current incarnations
  - MapReduce framework
  - Key-value stores
  - Document stores
  - Graph database systems