

Lecture 7

Relational Model

Subject Lecturer: Kevin K.F. YUEN, PhD.

Acknowledgement: Slides were offered from Prof. Ken Yiu.
Some parts have been revised and indicated.

Week	Tue Class	Wed Class	Lecture Sequence	Lab /Tutorial Sequence	Venue (Tue Class)	Venue (Wed Class)
7	Oct. 15	Oct. 16	7) Relational model	Lab 7: Relational Model	N001	PQ306
8	Oct. 22	Oct. 23	8) SQL I		N001	PQ306
9	Oct. 29	Oct. 30	9) SQL II	Labs 6, 8 & 9: SQL	6:30-7:30: N001; 7:30-9:30: PQ 604ABC	PQ 604ABC
10	Nov. 5	Nov. 6	10) Entity-relationship model	Lab 10: E-R Diagrams	N001	PQ306
11	Nov. 12	Nov. 13	11) Database normalization	Lab 11: Database normalization	N001	PQ306
12	Nov. 19	Nov. 20	12) Data storage and indexing	Lab 12: Storage and indexing	N001	PQ306
13	Nov. 26	Nov. 27	13) Query processing [Quiz 2]	Lab 13: Query processing	N001	PQ306

Outline



- ◆ Relational model

- ◆ Relational algebra: basic operations

- ◆ Relational algebra: other operations

- ◆ Null value

Scenario: online shopping

◇ Customer



◇ Product



◇ Invoice



◇ Others: Payment, Delivery,

cust-id	name	email	address
1	James	james@yahoo.com	AB
2	Mary	mary@gmail.com	CD
3	Peter	peter@yahoo.com	EF
4	Peter	peter@gmail.com	null
...

Basic Concepts

- ◆ Attribute / column

- ◆ E.g., name

- ◆ Schema

- ◆ Defines a table by a list of attributes (with types)

- ◆ E.g., Customer-schema=(cust-id, name, email, address)

cust-id	name	email	address
...
...
...
...
...

Basic Concepts

- ◆ **Tuple** (also called as **row**)

- ◆ E.g., (1, James, james@yahoo.com, AB, 159)

- ◆ **Relation** (also called as **table**)

- ◆ A **set** of tuples (**no duplicates**); tuples are *unordered*

- ◆ E.g., Customer table

cust-id	name	email	address
1	James	james@yahoo.com	AB
2	Mary	mary@gmail.com	CD
3	Peter	peter@yahoo.com	EF
4	Peter	peter@gmail.com	null
...

indicates a missing value

Keys



- ◆ Superkey
 - ◆ A set of attributes that **uniquely identify** a tuple

- ◆ Candidate key
 - ◆ A **minimal** superkey (such that none of its subsets can be superkey)

- ◆ Primary key
 - ◆ A candidate key chosen by database designer (if there are multiple choices)

Exercises on Keys

Relation **Customer**:

cust-id	name	email	address
1	James	james@yahoo.com	AB
2	Mary	mary@gmail.com	CD
3	Peter	peter@yahoo.com	EF
4	Peter	peter@gmail.com	null
...

Which one is a superkey?

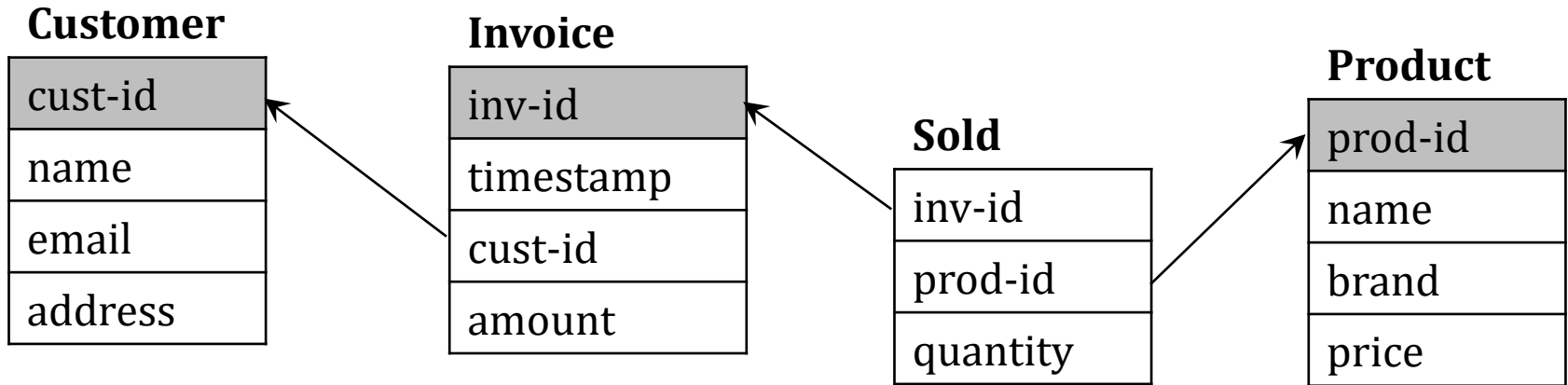
- {cust-id}
- {name}
- {cust-id, name}

Which one is a candidate key?

- {cust-id}
- {cust-id, name}

We choose {cust-id} as the primary key.

Schema Diagram



- ◆ Schema diagram
 - ◆ Express the schemas of tables and their relationships
- ◆ Primary key (highlighted in gray)
- ◆ Foreign key
 - ◆ An attribute that uniquely identifies a tuple in another table
 - ◆ E.g., The attribute *cust-id* in *Invoice* is a foreign key to *Customer*
- ◆ Why is the table ***Sold*** useful? What happens if we remove it?

Outline

- ◆ Relational model



- ◆ Relational algebra: basic operations

- ◆ Relational algebra: other operations

- ◆ Null value

Relational Algebra

- ◆ A language for expressing queries
 - ◆ E.g., “find products that have price > 8.0”



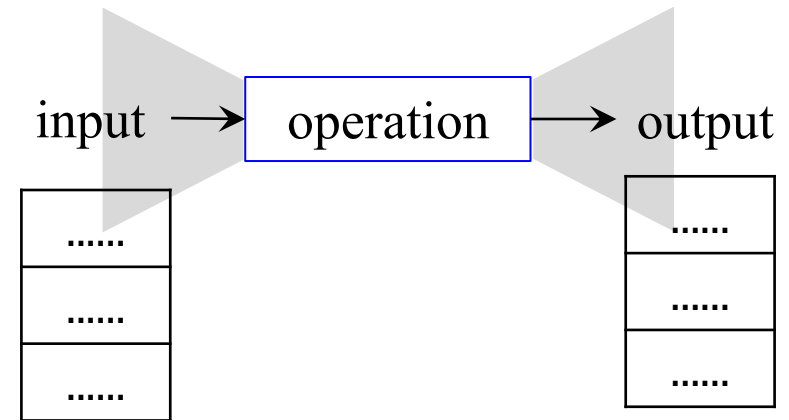
$$\sigma_{price > 8.0} (Product)$$

- ◆ It forms the foundation of SQL
- ◆ It is used within DBMS (for query processing & optimization)

Relational algebra: Basic Operations

- ◆ Basic operations

- ◆ select: σ
- ◆ project: Π
- ◆ Cartesian product: \times
- ◆ union: \cup
- ◆ set difference: $-$
- ◆ rename: ρ



- ◆ Each operation takes table(s) as input, then returns a table as output

Relational algebra: expression vs. tree

◆ Relational algebra **expression**

- ◆ The output of a sub-expression can be used as the input of a larger expression

$$\Pi_{name} (\sigma_{price > 8.0} (Product))$$

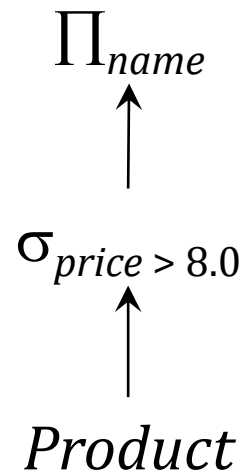
②

①

- ◆ The innermost operation is applied first

◆ Relational algebra **tree**

- ◆ A graphical representation of relational algebra expression (easy to understand)



How to express the following queries in relational algebra?

- ◆ (Q1) Find **products** that have **price** > 8.0
- ◆ (Q2) Find the **names** of **products**
- ◆ (Q3) Find the **customer name** and **inv-id** (invoice id) of each **invoice**
- ◆ (Q4) Find the highest **price** in the table *Product*
- ◆ (Q5) Find the total **amount** in the table *Invoice*

Relational algebra: Select

◆ **Select** σ : retrieve tuples that satisfy a condition

◆ Example: Find products that have price > 8.0

$$\sigma_{price > 8.0} (Product)$$

Relation **Product**:

condition

input

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3



prod-id	name	brand	price
2	Pepsi	PE	8.9
4	Sprite	CO	8.3

Relational algebra: Select

- ◆ In the condition, we may use
 - ◆ Comparisons: $=, \neq, <, >, \leq, \geq$
 - ◆ Connectives: and (\wedge), or (\vee), not (\neg)

- ◆ Examples:

$\sigma_{price > 8.5 \vee brand = \text{"DP"}} (Product)$

prod-id	name	brand	price
2	Pepsi	PE	8.9
3	7 Up	DP	6.5

$\sigma_{price > 8.0 \wedge brand = \text{"CO"}} (Product)$

prod-id	name	brand	price
4	Sprite	CO	8.3

Relation **Product**:

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3

Relational algebra: Project

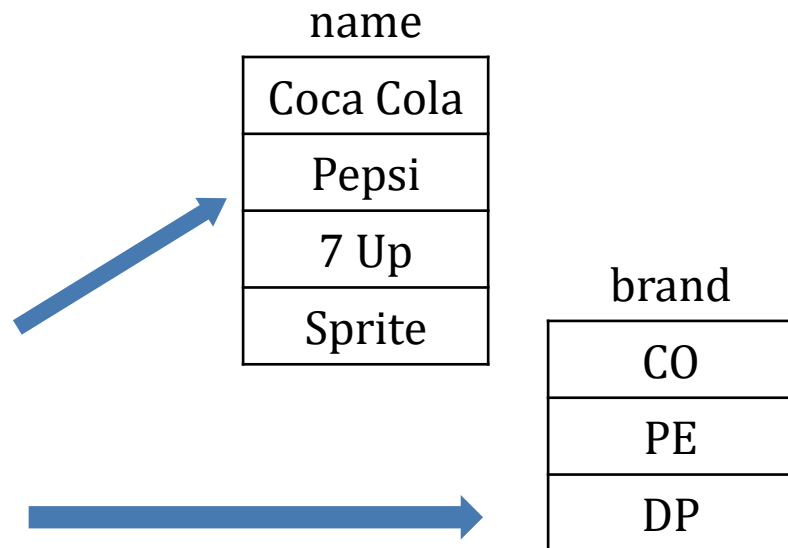
- ◆ **Project Π** : choose attributes in the output

Examples *attribute(s)* *input*

- ◆ 1) Find the names of products: $\Pi_{name}(Product)$
- ◆ 2) Find the brands of products: $\Pi_{brand}(Product)$

Relation **Product**:

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3



Why only 1 “CO”?

Composite Operations

- ◆ Apply operations one-by-one
 - ◆ The innermost operation is applied first
 - ◆ E.g., Find the names of products that have price > 8.0

$$\Pi_{name} (\sigma_{price > 8.0} (Product))$$

② ①

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3



name
Pepsi
Sprite

- ◆ Does the following expression make sense? Why?

$$\sigma_{price > 8.0} (\Pi_{name} (Product))$$

Query (Q3)

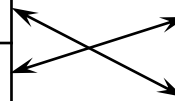
- ◆ “Find the customer name and inv-id (invoice id) of each invoice”
- ◆ How to express this in relational algebra?
 - ◆ *Invoice* contains inv-id but not customer name
 - ◆ *Customer* contains customer name but not inv-id
 - ◆ How to combine these two tables (correctly)?

Relation **Customer**:

cust-id	name	email	address
1	James	james@yahoo.com	AB
2	Mary	mary@gmail.com	CD
3	Peter	peter@yahoo.com	EF
4	Peter	peter@gmail.com	null

Relation **Invoice**:

inv-id	timestamp	cust-id	amount
1	101	3	8.9
2	102	2	7.8



Relational algebra: Cartesian Product

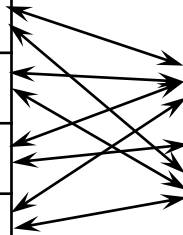
- ◆ **Cartesian product** \times : given two relations R and S , this operation returns a relation T containing every possible pair of tuples from R and S
 - ◆ Example: $Customer \times Invoice$

Relation **Customer**:

cust-id	name	email	address
1	James	james@yahoo.com	AB
2	Mary	mary@gmail.com	CD
3	Peter	peter@yahoo.com	EF
4	Peter	peter@gmail.com	null

Relation **Invoice**:


inv-id	timestamp	cust-id	amount
1	101	3	8.9
2	102	2	7.8




Relational algebra: Cartesian Product

◇ Result of $Customer \times Invoice$

- ◇ When there are different “cust-id” attributes, we distinguish them by using the table name as **prefix**



Customer.cust-id	name	email	address	inv-id	timestamp	Invoice.cust-id	amount
1	James	james@yahoo.com	AB	1	101	3	8.9
2	Mary	mary@gmail.com	CD	1	101	3	8.9
3	Peter	peter@yahoo.com	EF	1	101	3	8.9
4	Peter	peter@gmail.com	null	1	101	3	8.9
1	James	james@yahoo.com	AB	2	102	2	7.8
2	Mary	mary@gmail.com	CD	2	102	2	7.8
3	Peter	peter@yahoo.com	EF	2	102	2	7.8
4	Peter	peter@gmail.com	null	2	102	2	7.8



- ◇ But some result tuples are not meaningful

Back to query (Q3)

- ◆ (Q3) “find the customer name and inv-id of each invoice”
- ◆ Step 1: apply selection to match keys
 - ◆ $\sigma_{Customer.cust-id=Invoice.cust-id} (Customer \times Invoice)$

Customer. cust-id	name	email	address	inv-id	timestamp	Invoice. cust-id	amount
3	Peter	peter@yahoo.com	EF	1	101	3	8.9
2	Mary	mary@gmail.com	CD	2	102	2	7.8

- ◆ Step 2: apply projection to keep the required attributes
 - ◆ $\Pi_{inv-id, name} (\sigma_{Customer.cust-id=Invoice.cust-id} (Customer \times Invoice))$

inv-id	name
1	Peter
2	Mary

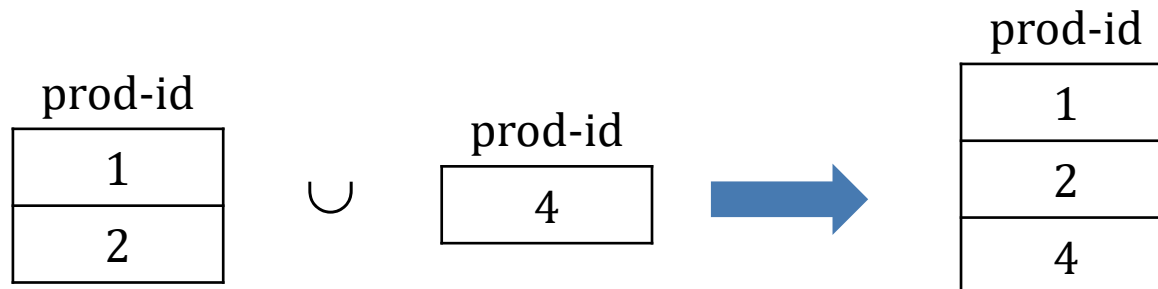
Relational algebra: Union

- ◆ **Union** \cup : given two relations R and S with **compatible schema**, it returns a relation containing tuples in either R or S

Example:

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3

$$\Pi_{prod-id}(\sigma_{prod-id \leq 2}(Product)) \cup \Pi_{prod-id}(\sigma_{prod-id=4}(Product))$$



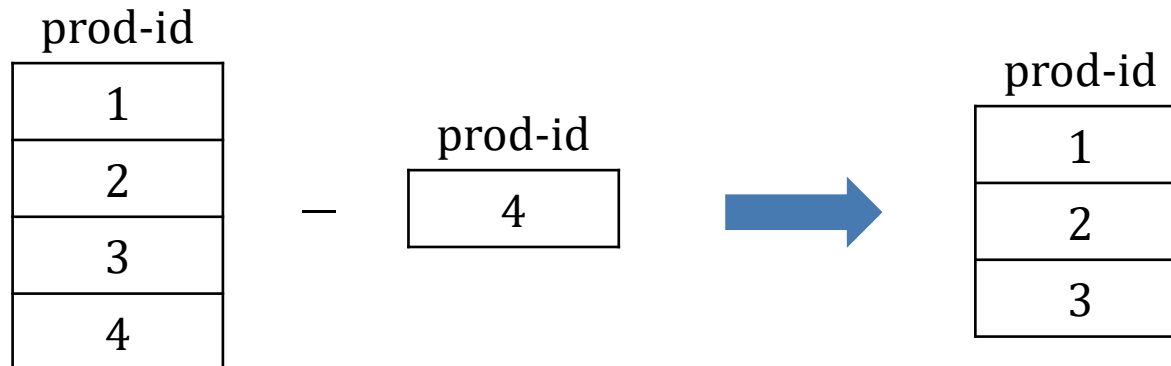
Relational algebra: Set Difference

- ◆ **Set difference** $-$: given two relations R and S with **compatible schema**, it returns a relation containing tuples in R but not in S

- ◆ **Example:**

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3

$$\Pi_{prod-id}(Product) - \Pi_{prod-id}(\sigma_{prod-id=4}(Product))$$



Relational algebra: Rename

- ◆ **Rename** ρ : assign a name (e.g., *Temp*) to the result of relational-algebra expression E


Example:

$$\rho_{Temp} (\Pi_{name} (\sigma_{price > 8.0} (Product)))$$

Relation **Product**:

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3

Relation **Temp**:



name
Pepsi
Sprite

Exercises on relational algebra tree

Draw the relational algebra tree for each expression below

$$(1) \quad \rho_{Temp} (\Pi_{name} (\sigma_{price > 8.0}(Product)))$$

$$(2) \quad \sigma_{Customer.cust-id=Invoice.cust-id} (Customer \times Invoice)$$

$$(3) \quad \Pi_{prod-id}(Product) - \Pi_{prod-id}(\sigma_{prod-id=4}(Product))$$

Exercise: Find the highest price in Product

♦ Idea:

- ♦ Step (1) finds prices that are not the highest
- ♦ Step (2) computes the set difference between $\Pi_{price}(Product)$ and the above intermediate result

♦ Expression for step (1):

$$\Pi_{Temp.price} (\sigma_{Temp.price < Product.price} (Product \times \rho_{Temp}(Product)))$$


Relation **Product**:

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3



price
7.8
6.5
8.3

Outline

- ◆ Relational model
- ◆ Relational algebra: basic operations
-  ◆ Relational algebra: other operations
- ◆ Null value

Other Relational Algebra Operations

- ◆ **Convenient** operations for simplifying expressions; have the same expressive power as basic operations
 - ◆ Natural join
 - ◆ Set intersection
 - ◆ Assignment
 - ◆ Division (we skip this)
- ◆ **Extended** operations that have more expressive power than basic operations
 - ◆ Generalized projection
 - ◆ Aggregation functions
 - ◆ Outer join (we skip this)

Natural join

⋈:\bowtie , join

◆ Natural join ⋈

- ◆ Given two relations R and S , let the union schema be A , let the common attributes be a_1, a_2, \dots, a_k
- ◆ The natural join $R \bowtie S$ returns:

$$\Pi_A \left(\sigma_{R.a_1=S.a_1, R.a_2=S.a_2, \dots, R.a_k=S.a_k} (R \times S) \right)$$

◆ Example: *Customer* ⋈ *Invoice*

- ◆ Common attribute: cust-id
- ◆ Schema: (cust-id, name, email, address, inv-id, timestamp, amount)

cust-id	name	email	address	inv-id	timestamp	amount
3	Peter	peter@yahoo.com	EF	1	101	8.9
2	Mary	mary@gmail.com	CD	2	102	7.8

cust-id	name	email	address
1	James	james@yahoo.com	AB
2	Mary	mary@gmail.com	CD
3	Peter	peter@yahoo.com	EF
4	Peter	peter@gmail.com	null

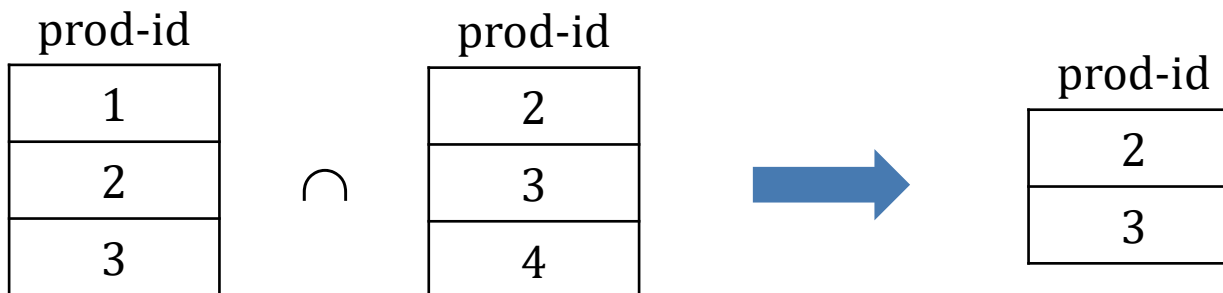
inv-id	timestamp	cust-id	amount
1	101	3	8.9
2	102	2	7.8

Set intersection

- ◆ **Set intersection** \cap : given two relations R and S with **compatible schema**, it returns a relation containing tuples that are in both R and S

Example:

$$\Pi_{prod-id}(\sigma_{prod-id \leq 3}(Product)) \cap \Pi_{prod-id}(\sigma_{prod-id \geq 2}(Product))$$



Assignment

- ◆ **Assignment** \leftarrow : assign temporary result to a variable
 - ◆ Improve the readability of relational algebra
 - ◆ For example, the following expression

$$\Pi_{prod-id}(\sigma_{prod-id \leq 2}(Product)) \cup \Pi_{prod-id}(\sigma_{prod-id=4}(Product))$$

can be rewritten as:

$$Temp1 \leftarrow \Pi_{prod-id}(\sigma_{prod-id \leq 2}(Product))$$

$$Temp2 \leftarrow \Pi_{prod-id}(\sigma_{prod-id=4}(Product))$$

$$Temp1 \cup Temp2$$

Assignment: Database Modification

◆ Insertion

- ◆ Insert tuples into a relation
- ◆ E.g., insert a new product “Soda”

$$Product \leftarrow Product \cup \{ (5, \text{“Soda”}, \text{“AB”}, 7.5) \}$$

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
4	Sprite	CO	8.3
5	Soda	AB	7.5

Assignment: Database Modification

◆ Deletion

- ◆ Delete tuples from a relation
- ◆ E.g., remove products that have price < 7.0

$Product \leftarrow Product - \sigma_{price < 7.0}(Product)$

Original

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3

After deletion

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3

Generalized projection

◆ Generalized projection

- ◆ Functions may be used to compute derived attributes
- ◆ Example: compute the price of each product at 20% discount
 - ◆ “special-price” is a derived attribute

$\Pi_{prod-id, price*80\% \text{ as } special-price} (Product)$

Relation **Product**:

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3



prod-id	special-price
1	6.24
2	7.12
3	5.2
4	6.64

Aggregate function

◆ Aggregate function

- ◆ Apply on a collection of values to obtain a single result
- ◆ Functions: sum, avg, count, min, max

◆ Examples

- ◆ Find the total amount of *Invoice*: $\mathcal{G}_{\text{sum}(\text{amount})}(\text{Invoice})$
- ◆ Find the highest price of *Product*: $\mathcal{G}_{\text{max}(\text{price})}(\text{Product})$

Relation **Invoice**:

inv-id	timestamp	cust-id	amount
1	101	3	8.9
2	102	2	7.8
3	103	2	6.5
4	104	3	8.3



sum(amount)
31.5

Aggregate function + grouping

- ◆ In some scenario, we wish to partition the table into **groups**, then use aggregate function on each **group**
 - ◆ E.g., find the total amount spent by each customer

cust-id \mathcal{G} $\text{sum}(\text{amount})$ (*Invoice*)

inv-id	timestamp	cust-id	amount
1	101	3	8.9
2	102	2	7.8
3	103	2	6.5
4	104	3	8.3

grouping



inv-id	timestamp	cust-id	amount
1	101	3	8.9
4	104	3	8.3
2	102	2	7.8
3	103	2	6.5

aggregation



cust-id	sum(amount)
3	17.2
2	14.3

Outline

- ◆ Relational model
- ◆ Relational algebra: basic operations
- ◆ Relational algebra: other operations



- ◆ Null value

Null Value

- ◆ **Null** represents a missing value
- ◆ When we compare **null** and a known value (e.g., **null** > 8.0)
 - ◆ The result is **unknown**

Relation **Product**:

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	null	6.5
4	Sprite	CO	null

A and B

input output

True	Unknown	Unknown
False	Unknown	False
Unknown	Unknown	Unknown

A or B

input output

True	Unknown	True
False	Unknown	Unknown
Unknown	Unknown	Unknown

not A

input output

Unknown	Unknown
---------	---------

Null Value

Relation **Product**:

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	null	6.5
4	Sprite	CO	null

◆ Select operation

- ◆ A tuple belongs to the result if it is *true* for the condition in select

◆ Find the results of

$$\sigma_{price > 8.0} (Product)$$

prod-id	name	brand	price
2	Pepsi	PE	8.9

$$\sigma_{price > 8.0 \vee brand = "CO"} (Product)$$

prod-id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
4	Sprite	CO	null

$$\sigma_{price > 7.0 \wedge brand = "CO"} (Product)$$

prod-id	name	brand	price
1	Coca Cola	CO	7.8

Summary

- ◆ After this lecture, you should be able to:
 - 1) Understand concepts in relational model
(e.g., schema, relation, keys, schema diagram)
 - 2) Apply relational algebra to express queries
- ◆ Please read Chapter 2 in the book
“*Database System Concepts*”, 7th Edition
- ◆ Next lecture: SQL