Lecture 8
SQL part I

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

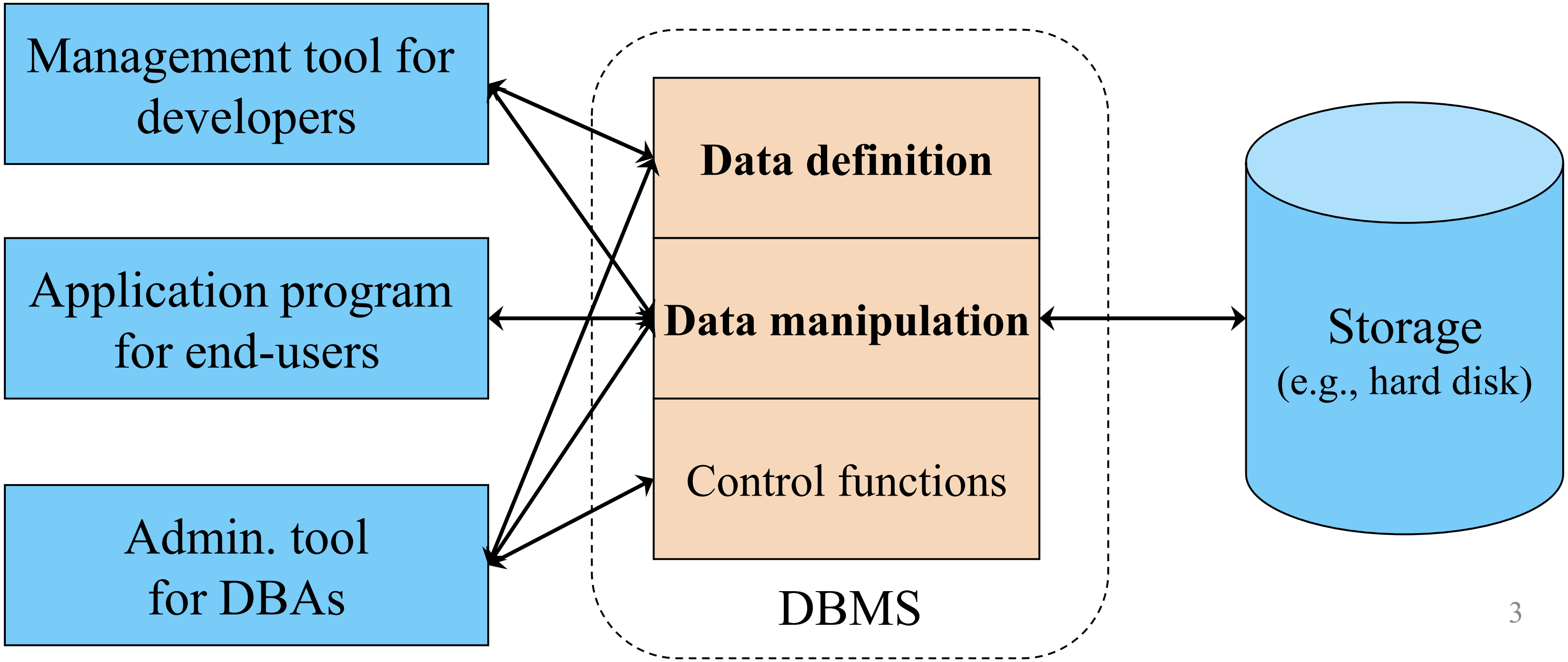
Acknowledgement: Slides were offered from Prof. Ken Yiu.
Some parts might be revised and indicated.

Outline

- ⌛ Background
- How to define a table?
- Basic SQL: select, from, where
- Order by, aggregation, group by
- Set operations
- How to insert, delete, update data?

How to use DBMS?

- *Data Definition Language* (DDL):
  - define the database schema, i.e., the record type
- *Data Manipulation Language* (DML):
  - access / update the database content
- SQL consists of both DDL and DML



History of SQL

- Early days (1970s)
  - Known as SEQUEL (Structured English Query Language)
  - Developed for IBM’s system R
- Now called SQL (Structured Query Language)
- Standardization by ISO
  - SQL-86
  - SQL-89
    - + integrity constraints, 100 pages
  - SQL-92
    - + new DDL,DML features, 500 pages
  - SQL:1999, 2003, 2006, 2008, 2011, 2016
    - + many features, many pages

DBMS Implementations

- DBMS implementations
  - Microsoft SQL server, Oracle, IBM DB2, ……
- Many implementations fully support "intermediate SQL" --- half of new features in SQL-92
- Some DBMS vendors have proprietary extensions to the SQL standard
  - E.g., a SQL statement used in DBMS "A" may not be directly supported in DBMS "B"

SQL vs. relational algebra

- In SQL, **duplicates** are allowed in a table
  - In relational algebra, duplicates are not allowed
- Some functions are supported in SQL but not in relational algebra
  - Data definition language (e.g., create table, drop table)
  - Nested queries
  - Sorting a table
  - Keywords for data types
  - Conditional expressions (CASE)
  - String functions

Outline

- Background
- ⌛ How to define a table?
- Basic SQL: select, from, where
- Order by, aggregation, group by
- Set operations
- How to insert, delete, update data?

How to create tables?

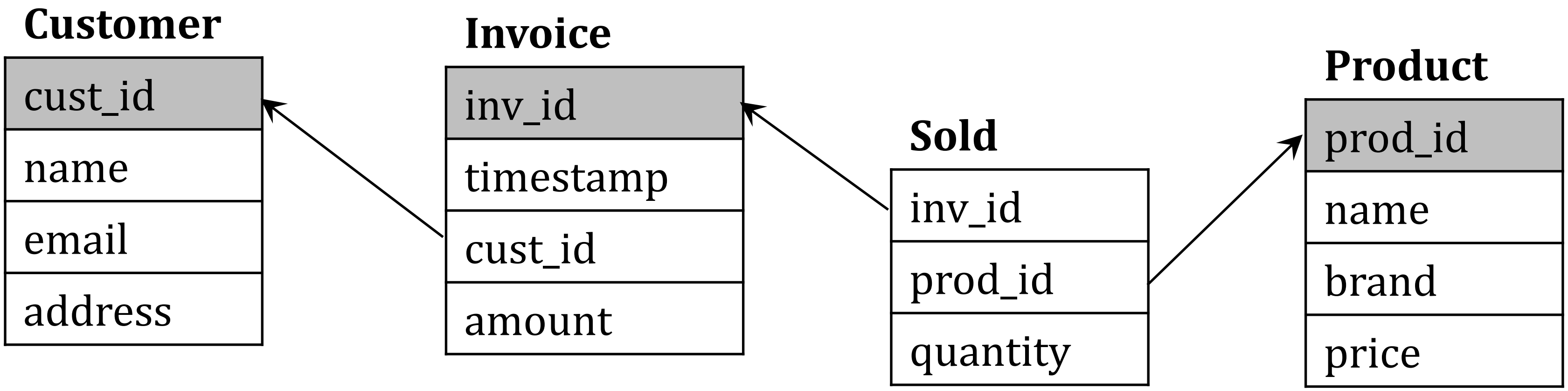
- When we create a table, we provide:
  - The table name
  - The name and type of each attribute name
    - A name may contain `'_'` but not `'-'`
  - Integrity constraints
    - E.g., primary key, foreign key, NOT NULL, unique key, CHECK

Customer

cust_id
name
email
address

- Basic types
  - char(*n*): a string with fixed length *n*
  - int: an integer
  - numeric(*p*,*d*): a fixed point number with total *p* digits and *d* digits after the decimal point
  - real: floating-point number

How to create tables?



Customer

- cust_id
- name
- email
- address

Invoice

- inv_id
- timestamp
- cust_id
- amount

Sold

- inv_id
- prod_id
- quantity

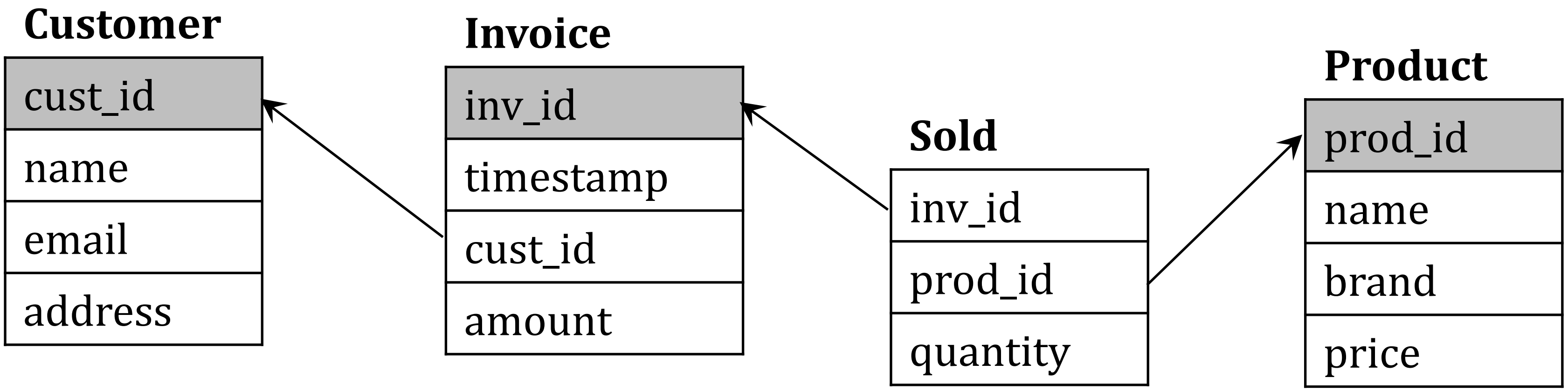
Product

- prod_id
- name
- brand
- price

- Create the Invoice table by:

```
create table Invoice
(inv_id int,
timestamp int,
cust_id int,
amount numeric(10,2),
primary key (inv_id))
```

How to create tables?



Customer

cust_id

name

email

address

Invoice

inv_id

timestamp

cust_id

amount

Sold

inv_id

prod_id

quantity

Product

prod_id

name

brand

price

- Create the Sold table by:

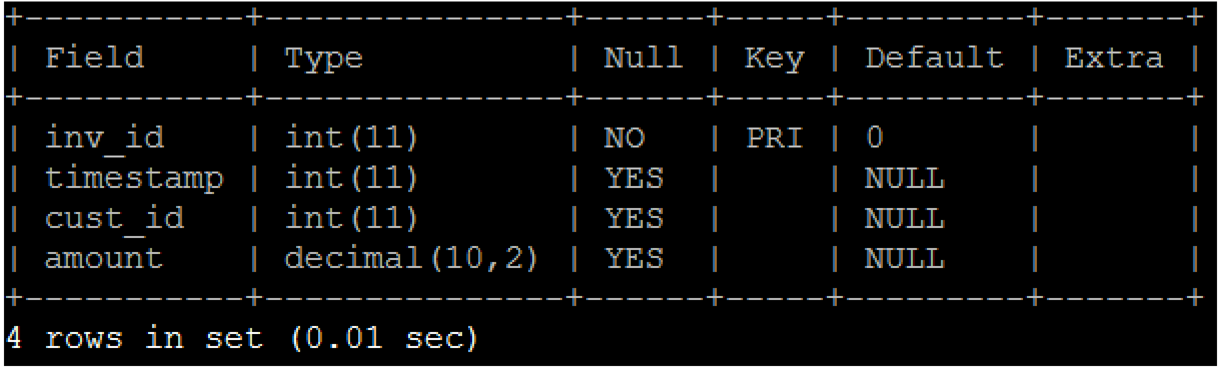
```
create table Sold
(inv_id int,
prod_id int,
quantity int,
primary key (inv_id,prod_id))
```

Other SQL statements for data definition

Purpose	Example statement
Show tables in a database	`show tables`
Describe the schema of a table	`describe Invoice`
Remove a table	`drop table Invoice`

Add a column in a table	`alter table Invoice`
	`add staff_id int`

Drop a column in a table	`alter table Invoice`
	`drop staff_id`



```
+-----------+---------------+------+-----+---------+-------+
| Field     | Type          | Null | Key | Default | Extra |
+-----------+---------------+------+-----+---------+-------+
| inv_id    | int(11)       | NO   | PRI | 0       |       |
| timestamp | int(11)       | YES  |     | NULL    |       |
| cust_id   | int(11)       | YES  |     | NULL    |       |
| amount    | decimal(10,2) | YES  |     | NULL    |       |
+-----------+---------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Outline

- Background
- How to define a table?
- ⌛ Basic SQL: select, from, where
- Order by, aggregation, group by
- Set operations
- How to insert, delete, update data?

SQL and Relational Algebra

- A simple SQL query has the form:

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

  - A_i represents an attribute
  - r_i represents a relation
  - P is a predicate
- Equivalent to the relational algebra expression

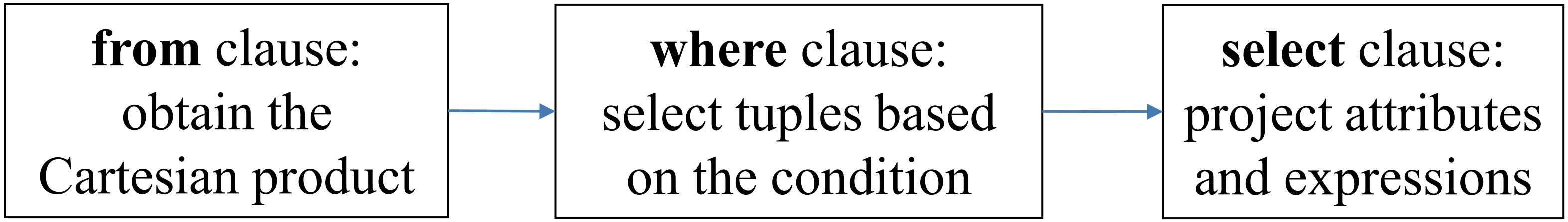
  Projection Selection Cartesian product

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

- The result of an SQL query is a relation
  - Note: the **select** keyword in SQL means projection (but not selection) in relational algebra!

Flow of Clauses

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



from clause: obtain the Cartesian product

where clause: select tuples based on the condition

select clause: project attributes and expressions

Cartesian product

Selection

Projection

How to express the following queries in SQL?

- (Q1) Find products that have price > 8.0
- (Q2) Find the brands 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*

SQL: where clause

- (Q1) Find products that have price > 8.0

```
select *
from Product
where price > 8.0
```

* means all attributes

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	name	brand	price
2	Pepsi	PE	8.9
4	Sprite	CO	8.3

SQL: where clause

- In the condition, we may use
  - Comparisons: =, !=, <, >, <=, >=
  - Connectives: and, or, not, between…and…

- Examples:

```
select *
from Product
where price>=8.0 and price<=9.0

select *
from Product
where price between 8.0 and 9.0
```

SQL: select clause

- (Q2) Find the brands of products
  - By default, SQL allows duplicates
  - To remove duplicates, use the keyword distinct

```
select brand
from Product
```

```
select distinct brand
from Product
```

brand
CO
PE
DP
CO

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

brand
CO
PE
DP

SQL: from clause, rename

- (Q3) "find the customer name and inv_id of each invoice"
  - Place both tables Customer and Invoice in the from clause
  - *Optional*: use the keyword as to rename a table and make the SQL statement easier to read

Remove 'as' in oracle

```
select I.inv_id, C.name
from Customer as C, Invoice as I
where C.cust_id=I.cust_id
```

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

inv_id	name
1	Peter
2	Mary

Outline

- Background
- How to define a table?
- Basic SQL: select, from, where
- ⌛ Order by, aggregation, group by
- Set operations
- How to insert, delete, update data?

SQL: order by

- The order by clause allows us to sort the results
  - asc: ascending order; desc: descending order

```
select *
from Product
order by price asc
```

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

```
select *
from Product
order by price desc
```

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

SQL: aggregation

- Aggregate functions
  - sum, count, avg, min, max
- (Q4) Find the highest price in the table *Product*
- (Q5) Find the total amount in the table *Invoice*

```
select max(price)
from Product
```

```
select sum(amount)
from Invoice
```

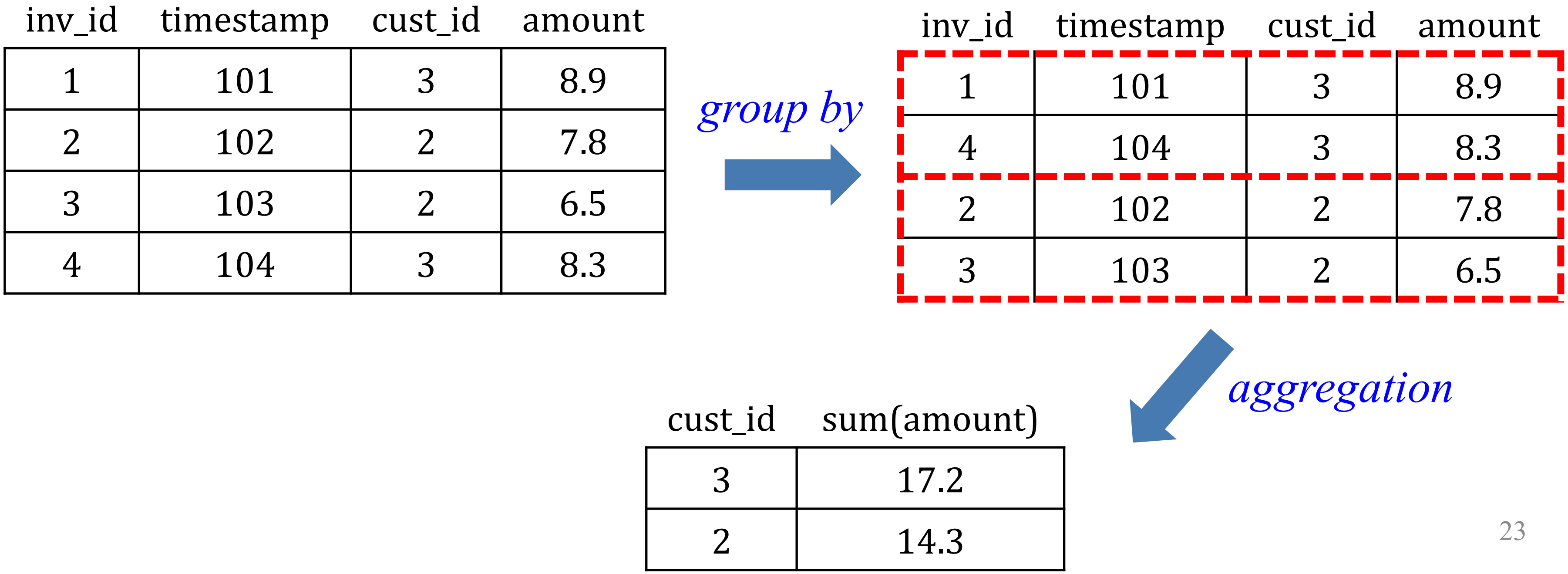
inv_id	timestamp	cust_id	amount
1	101	3	8.9
2	102	2	7.8
3	103	2	6.5
4	104	3	8.3

sum(amount)
31.5

SQL: group by

- E.g., find the total amount spent by each customer
  - First partition the table into groups (by cust_id), then use aggregate function on each group

```
select cust_id, sum(amount)
from Invoice
group by cust_id
```



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

group by

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

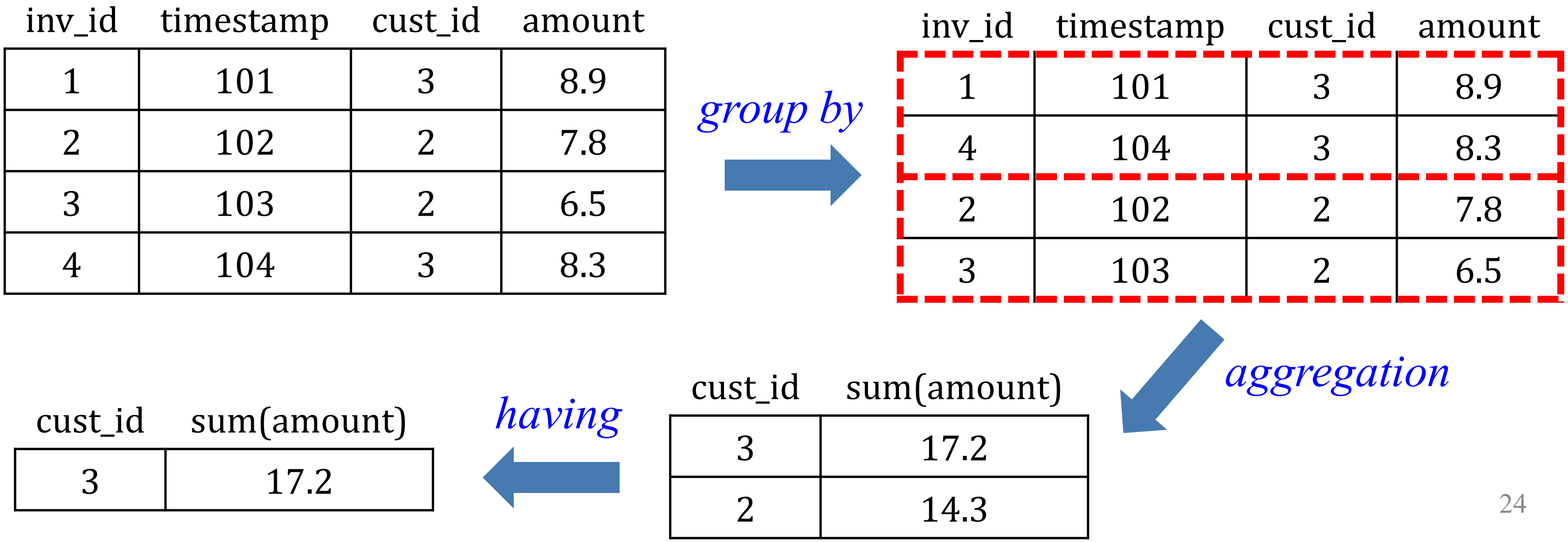
aggregation

cust_id	sum(amount)
3	17.2
2	14.3

SQL: group by + having

- Keyword having: specify filter condition for groups
- E.g., find the total amount spent by each customer; display those with total amount greater than 15.0

```
select cust_id, sum(amount)
from Invoice
group by cust_id
having sum(amount)>15.0
```



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

group by

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

having

cust_id	sum(amount)
3	17.2

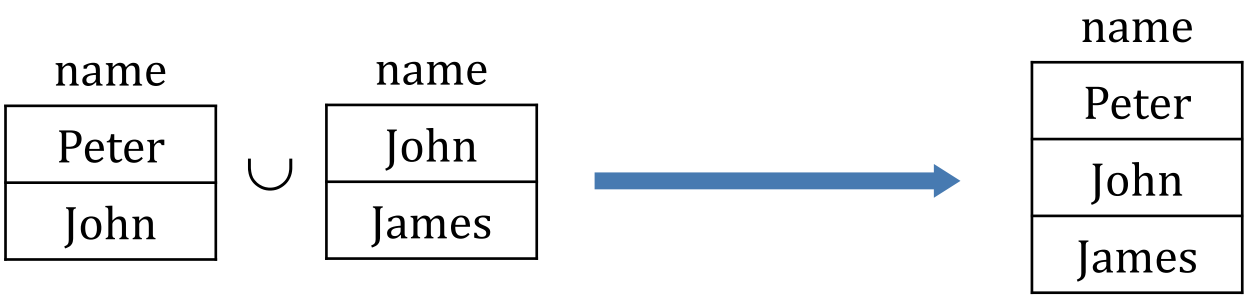
Outline

- Background
- How to define a table?
- Basic SQL: select, from, where
- Order by, aggregation, group by
- ⌛ Set operations
- How to insert, delete, update data?

SQL: set operations

- Set operations: union, intersect, except
  - By default, these operations **remove duplicates**

```
(select name from A)
union
(select name from B)
```



name
Peter
John

∪

name
John
James

→

name
Peter
John
James

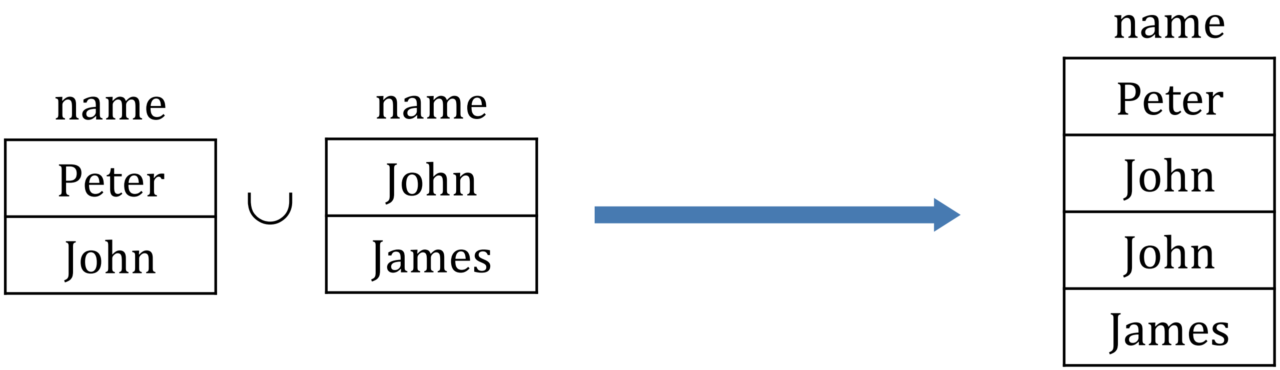
SQL: set operations

Minus in Oracle

MS SQL

- Set operations: union, intersect, except
  - To allow duplicates, use the keyword **all**

```
(select name from A)
union all
(select name from B)
```



name
Peter
John

∪

name
John
James

→

name
Peter
John
John
James

Oracle MINUS is an operator; it's equivalent to EXCEPT in SQL Server.
https://stackoverflow.com/questions/5557991/minus-vs-except-difference-in-oracle-sql-server

Outline

- Background
- How to define a table?
- Basic SQL: select, from, where
- Order by, aggregation, group by
- Set operations
- How to insert, delete, update data?

Insertion

- **Insertion**: insert tuple(s) into a relation
  - Example 1: insert a tuple

```
insert into Product
values (5,'Soda','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

  - Example 2: insert tuples obtained by select statement

```
insert into Product
   select ...
   from ...
   where ...
```

Deletion

- **Deletion**: delete tuple(s) from a relation
  - Example: remove the product that has id=3

```
delete from Product
where prod_id=3
```

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

Update

- Update tuple(s) based on a condition
  - Example: double the price of each product of the brand 'AB'

```
update Product
set price=price*2
where brand='AB'
```

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~~ 15

Summary

- After this lecture, you should be able to:
  1) Apply SQL to define tables
  2) Apply SQL to express simple queries

- Please read Chapter 3 in the book "*Database System Concepts*", 7th Edition

- Next lecture: advanced features of SQL