Lecture 9
SQL part II

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

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

Outline

- ⌛ More on data types
- Null values
- Joins
- Subqueries
- Views

More on data types

- Data types
  - Strings: char(*n*), varchar(*n*)
  - Numbers: int, numeric(*p*,*d*), real
  - Date/time: date, time, timestamp
  - Large object types: clob, blob

- Some data types support special operations

String

- Example of string value: `'123 ABC Road'`
- The keyword like is used for string pattern matching
- Special characters
  - _: it can match any single character
  - %: it can match any substring
- Conversions
  - upper( ), lower( )

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

```
select upper(name)
from Product
where prod_id=2
```

name
PEPSI

```
select name
from Product
where name like '%la'
```

name
Coca Cola

Date/time

- Examples of values
  - date: `'2018-09-27'`
  - time: `'18:30:00'`
  - timestamp: `'2018-09-27 18:30:00'`
- Extract values from a date attribute *d*
  - year(*d*), month(*d*), day(*d*)
- Extract values from a time attribute *t*
  - hour(*t*), minute(*t*), second(*t*)
- Example SQL:
  - Assume the table *Sold* has an attribute *ts* of type timestamp

```
select *
from Sold
where year(ts)=2018
```

Large object types

- "lob" means large object
  - Typical size: kilobytes, megabytes
- blob type
  - It stores binary data
  - E.g., image, video
- clob type
  - It stores character data
  - E.g., text comment

Outline

- More on data types
- ⌛ Null values
- Joins
- Subqueries
- Views

Null value

- Keyword null
  - Represents a missing value

- Example: find products that have missing price

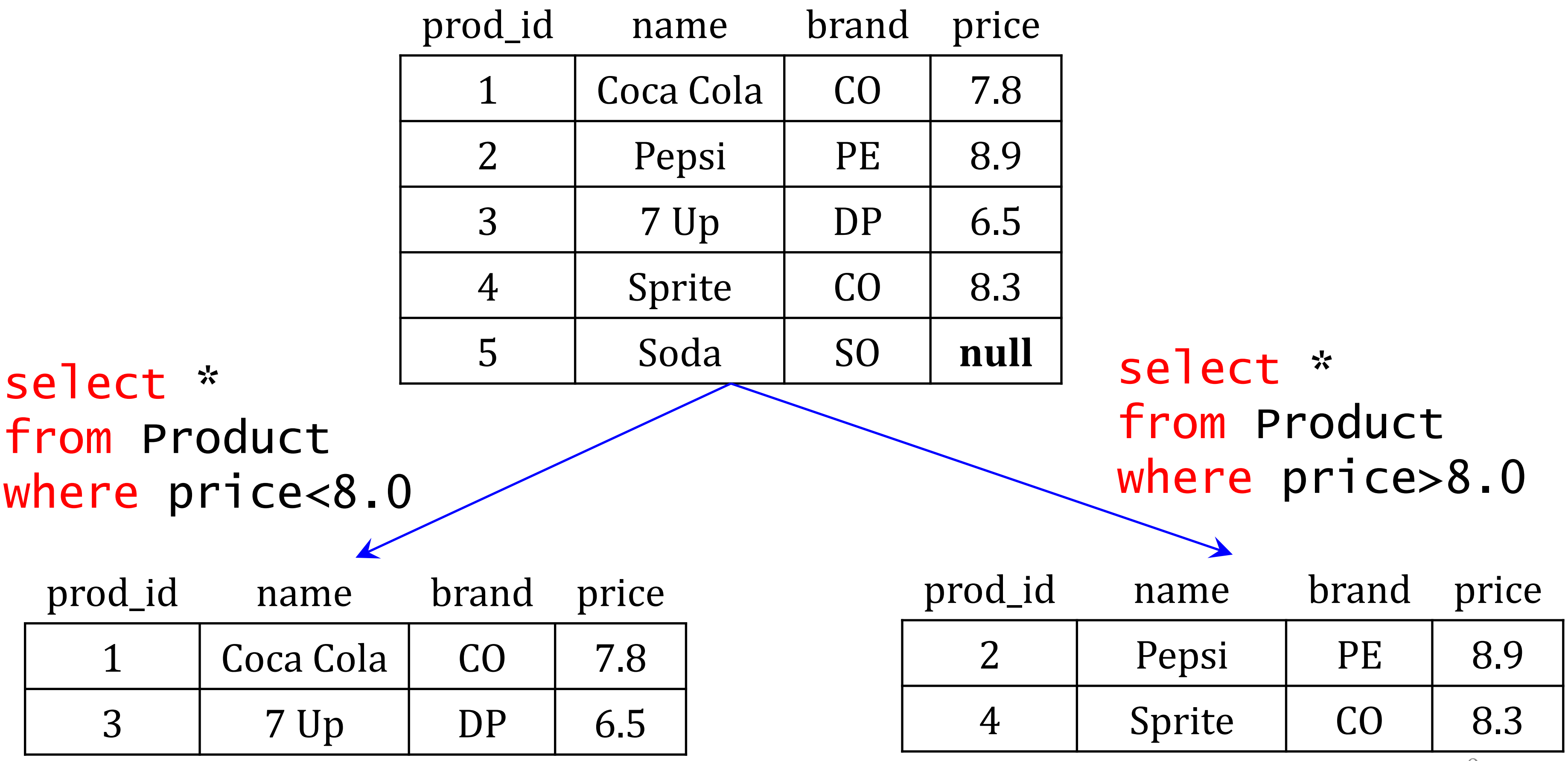
prod_id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
3	7 Up	DP	6.5
4	Sprite	CO	8.3
5	Soda	SO	**null**

```
select *
from Product
where price is null
```

prod_id	name	brand	price
5	Soda	SO	**null**

Null value

- In select-from-where statement, a tuple belongs to the result if it is true



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
5	Soda	SO	**null**

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

prod_id	name	brand	price
1	Coca Cola	CO	7.8
3	7 Up	DP	6.5

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

prod_id	name	brand	price
2	Pepsi	PE	8.9
4	Sprite	CO	8.3

Null value

- Aggregate function on a column
  - Null values are ignored

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

prod_id	name	brand	price
1	Coca Cola	CO	7.8
2	Pepsi	PE	8.9
5	Soda	SO	**null**

max(price)
8.9

prod_id	name	brand	price
4	Sprite	CO	**null**
5	Soda	SO	**null**

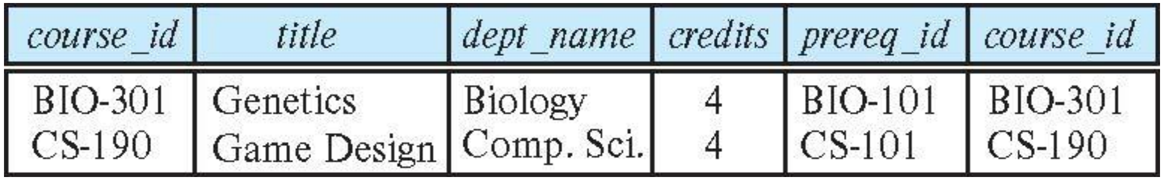
max(price)
null

Outline

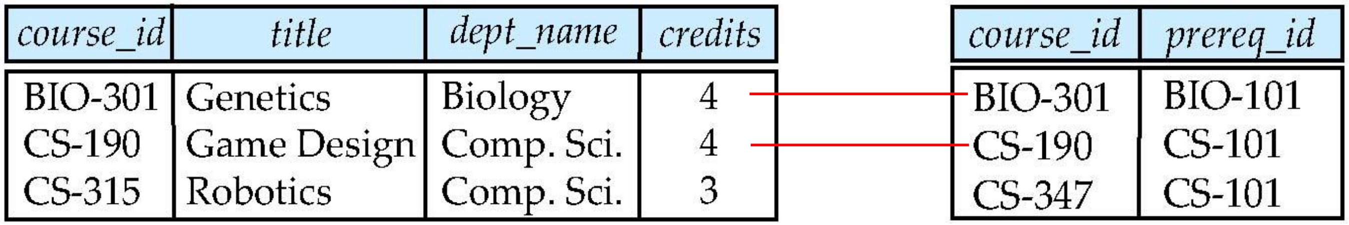
- More on data types
- Null values
- ⌛ Joins
- Subqueries
- Views

Inner join

- Inner join
  - The join attribute value must appear in both tables
- **select** * **from**
  course **inner join** *prereq* **on**
  course.course_id = prereq.course_id



course_id	*title*	*dept_name*	*credits*	*prereq_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190



Relation *course*

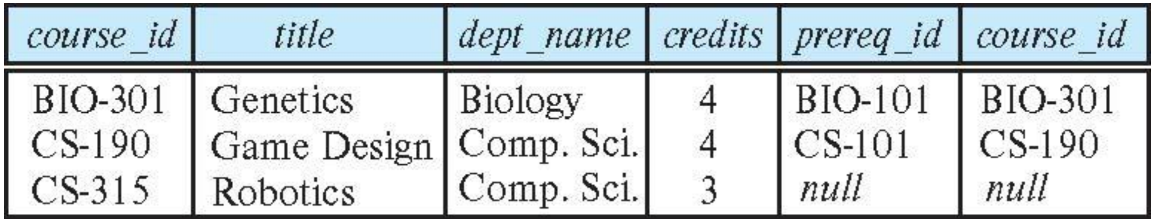
course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Relation *prereq*

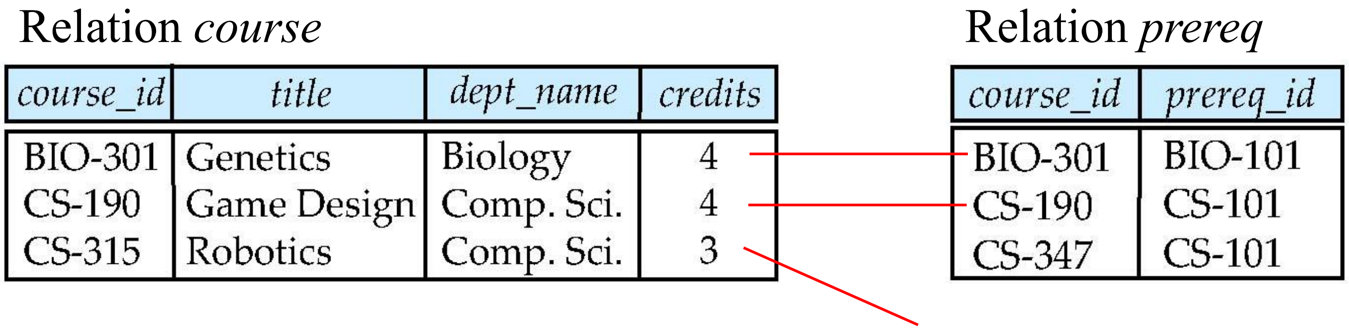
course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Left outer join

- Left outer join
  - All join attribute values in the left table appear in the result
  - If the join attribute value is missing in the right table, mark the missing attributes as null
- **select** * **from**
  course **left outer join** *prereq* **on**
  course.course_id = prereq.course_id



course_id	*title*	*dept_name*	*credits*	*prereq_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	*null*	*null*



Relation *course*

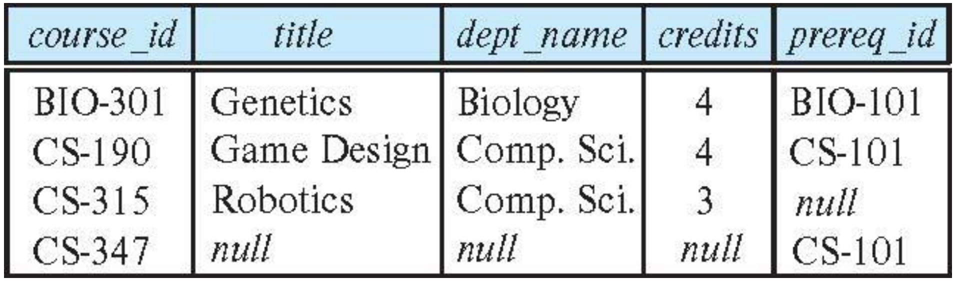
course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Relation *prereq*

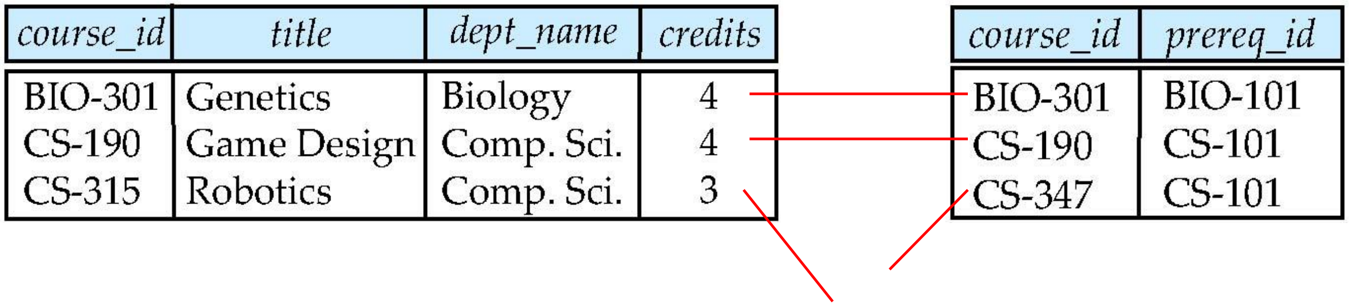
course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Full outer join

- Full outer join
  - If the join attribute value is missing in the left table, mark the missing attributes as null
  - If the join attribute value is missing in the right table, mark the missing attributes as null
- **select** * **from**
  course **full outer join** *prereq* **using** (*course_id*)



course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101



Relation *course*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Outline

- More on data types
- Null values
- Joins
- ⌛ Subqueries
- Views

Subqueries

- What is a subquery?
  - A “select-from-where” statement within a larger query statement
  - It should be enclosed in (…)

- A subquery may produce
  - A single value
  - A single-column table
    - Compare with it using: in, some, all, exists
  - A multi-column table
    - Compare with it using: exists

Subqueries

- Let’s use subqueries to express the following tasks
  - Find the products with price below the average price of products
  - Find the products with the same brand as the product with prod_id=4
  - Find the products that are cheaper than some ‘CO’ products
  - Find the products that are cheaper than all ‘CO’ products
  - Find the email of customers who have not purchased anything

Subquery output as single value

- Example: find the products with price below the average price of products
  - Get the average price by subquery:

```sql
select avg(price)
from 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

```sql
select *
from Product
where price <
    ( select avg(price)
      from Product )
```

prod_id	name	brand	price
1	Coca Cola	CO	7.8
3	7 Up	DP	6.5

Subquery output as single value

- Example: find the products with the same brand as the product with prod_id=4
  - Get the brand by subquery:

```
select brand
from Product
where prod_id=4
```

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

```
select *
from Product
where brand =
    ( select brand
      from Product
      where prod_id=4 )
```

prod_id	name	brand	price
1	Coca Cola	CO	7.8
4	Sprite	CO	8.3

Subquery output as single column

- Notations
  - Let *X* be a value, and *SC* be the result of a subquery
  - Let <compare> be a comparison operator (e.g., =, <, >)

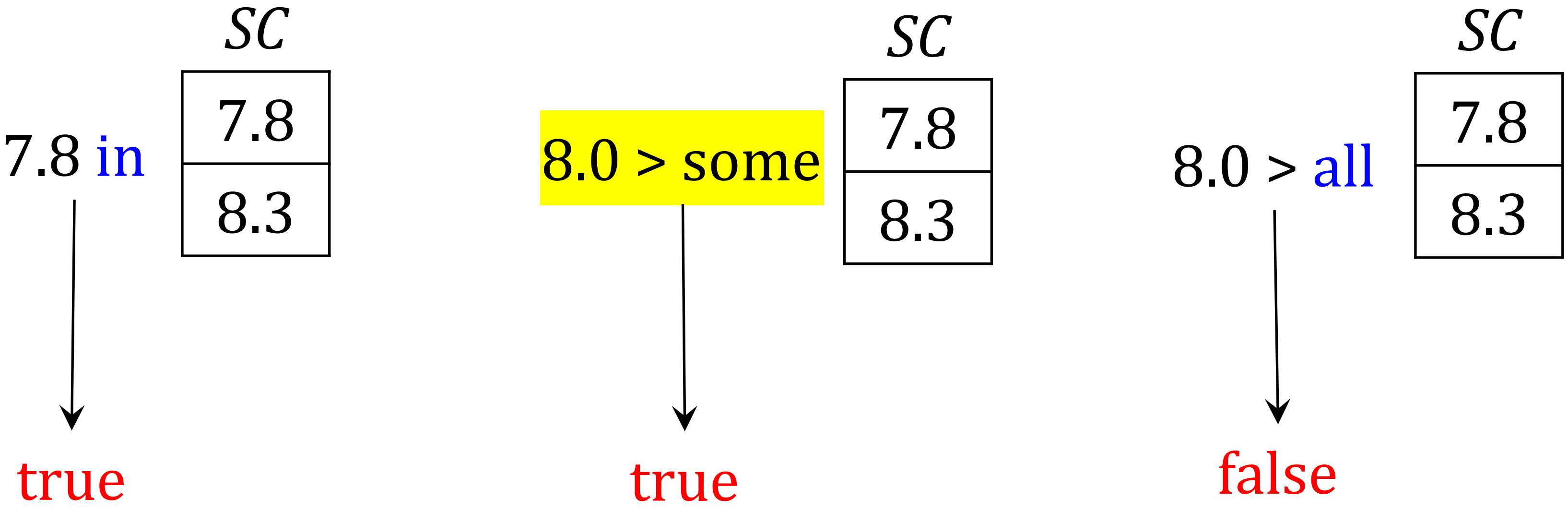
- *X* in *SC*
  - Returns true if SC contains X

- *X* <compare> some *SC*
  - Returns true for comparison with at least one value in *SC*

- *X* <compare> all *SC*
  - Returns true for comparison with all values in *SC*

Subquery output as single column

- Let *SC* be the result of a subquery
- Examples for comparison:



7.8 in

SC
7.8
8.3

true

8.0 > some

SC
7.8
8.3

true

8.0 > all

SC
7.8
8.3

false

Subquery output as single column

- Example: find the products that are cheaper than some ‘CO’ products

```
select *
from Product
where price < some
    ( select price
      from Product
      where brand='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
5	Soda	SO	8.0

prod_id	name	brand	price
1	Coca Cola	CO	7.8
3	7 Up	DP	6.5
5	Soda	SO	8.0

Subquery output as single column

- Example: find the products that are cheaper than all ‘CO’ products

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
5	Soda	SO	8.0

```
select *
from Product
where price < all
   ( select price
     from Product
     where brand='CO' )
```

prod_id	name	brand	price
3	7 Up	DP	6.5

Subquery output as single column

- Correlated subquery
  - A subquery that refers to the outer statement
    - E.g., `P.brand` refers to the outer statement

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
5	Soda	SO	8.0

```
select *
from Product P
where P.price > some
     ( select R.price
       from Product R
       where R.brand=P.brand)
```

prod_id	name	brand	price
4	Sprite	CO	8.3

[*Question*] Is this result correct? Why?

Subquery output as table

- Let *ST* be the result of a subquery

- exists (ST)
  - Returns true if ST is not empty

- unique (ST)
  - Returns true if ST does not have duplicates
  - *Note*: this keyword is not supported in MySQL

Subquery output as table

- Find the email of customers who have not purchased anything

```
select email
from Customer C
where not exists
   ( select *
     from Invoice I
     where I.cust_id=C.cust_id)
```

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**

Invoice:

inv_id	timestamp	cust_id	amount
1	101	3	8.9
2	102	2	7.8

email
james@yahoo.com
peter@gmail.com

Subquery: use it or not?

- Different SQL statements may be used to express the same task
  - E.g., SQL with subquery vs. SQL without subquery
  - Just use a SQL statement convenient to you

- [*Question*] Rewrite the following SQL so that no subqueries are used

```
select prod_id, price
from Product
where price < some
    ( select price
      from Product
      where brand='CO' )
```

- [*Question*] Rewrite the following SQL so that no subqueries are used
  - Two alternative answers are provided

```
select prod_id, price
from Product
where price < some
   ( select price
     from Product
     where brand='CO' )
```

```
select PA.prod_id, PA.price
from Product as PA,
     Product as PB
where PB.brand='CO'
  and PA.price<PB.price
group by PA.prod_id, PA.price
```

```
select distinct
     PA.prod_id, PA.price
from Product as PA,
     Product as PB
where PB.brand='CO'
  and PA.price<PB.price
```

Derived relation

- *Derived relation*: a relation obtained by a subquery in the from clause
- The following two queries are equivalent
  - The second one uses a derived relation *TX*

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

```
select * from
  ( select cust_id, sum(amount) as B
    from Invoice
    group by cust_id ) as TX
where B>15.0
```

Outline

- More on data types
- Null values
- Joins
- Subqueries
- Views

View

- What is a **view**?
  - A virtual table defined by a SQL statement
    - Its content not physically stored in DBMS
- Advantages of using a view
  - Improve the readability of SQL statements
  - Hide unnecessary data from users
  - Provide fine-grained access control to users

How to create a view?

- Consider the table *Customer*

Customer

cust_id
name
email
address

- The marketing team wishes to get the contact information of customers only, but not their names

- Let's create a view called *Contact*

```
create view Contact as
select email, address
from Customer
```

Contact

email
address

How to use a view?

- Just use a view like a table

Contact

email
address

- Example:

```
select *
from Contact
where email like '%yahoo.com'
```

[*Question*]

- Try to define a view called *CustomerLoyalty* so that
  - It has two attributes: cust_id, amount
  - The attribute amount stores the total amount paid by the corresponding customer in *Invoice*

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**

Invoice:

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

SQL Authorization

- **grant** statement: confer authorization

  grant <privilege list> **on** <relation or view > **to** <user list>

- **revoke** statement: revoke authorization

  revoke <privilege list> **on** <relation or view> **from** <user list>

- <privilege list> can contain:
  - **select**: able to query views or relations
  - **insert**: able to insert tuples
  - **update**: able to use the SQL update statement
  - **delete**: able to delete tuples
  - **all**: all the above

Roles

- The users with the same privileges should be assigned the same **role**
- How to create a role?
  - **create role** instructor;
- How to grant a role to users?
  - **grant** *instructor* **to** John**;**
  - **grant** *instructor* **to** Peter**;**
- How to grant a privilege to a role?
  - **grant select on** *takes* **to** *instructor*;
- Then, John and Peter can execute this SQL query:
  - **select** * **from** *takes;*

Summary

- After this lecture, you should be able to:
  1) Understand null values and more data types in SQL
  2) Use subqueries to solve problems
  3) Understand the concept of view

- Please read Chapter 4 in the book
  "*Database System Concepts*", 7th Edition

- Next lecture: How to design database tables?