

COMP 3311

DATABASE MANAGEMENT

SYSTEMS

LECTURE 9 EXERCISES

STRUCTURED QUERY LANGUAGE (SQL)

BOOK STORE RELATIONAL SCHEMA

Book(bookId, title, subject, quantityInStock, price, *authorId*)

Author(authorId, firstName, lastName)

Customer(customerId, firstName, lastName)

BookOrder(orderId, *customerId*, orderYear)

OrderDetails(*orderId*, *bookId*, quantity)

Attribute names in
italics are foreign
key attributes.

Assumptions

- Each author has authored at least one book in the store.
- Each book has exactly one Author.
- Each order is made by exactly one customer and has one or more associated tuples in OrderDetails (e.g., one order may contain several different books).

EXERCISE I

Given the foreign keys of the Book Store relations and assuming the referential integrity constraints are included in the SQL create statements, what should be the create order?

Book(bookId, title, subject, quantityInStock, price, *authorId*)

Author(authorId, firstName, lastName)

Customer(customerId, firstName, lastName)

BookOrder(orderId, *customerId*, orderYear)

OrderDetails(orderId, bookId, quantity)

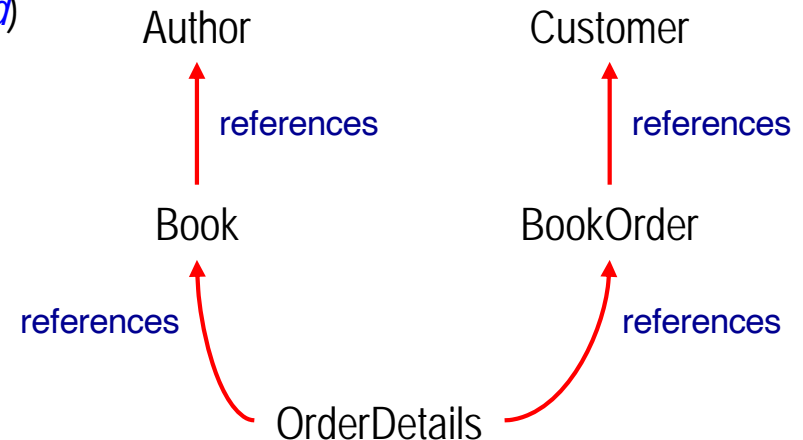


Table	Possible create order					
Author	1	1	2	2	1	3
Customer	2	2	1	1	3	1
Book	3	4	3	4	2	4
BookOrder	4	3	4	3	4	2
OrderDetails	5	5	5	5	5	5

Create Order		
Author	<i>before</i>	Book
Customer	<i>before</i>	BookOrder
Book, BookOrder	<i>before</i>	OrderDetails



EXERCISE 2

For all authors who wrote books on at least two subjects, increase the price of all their books by 5%.

```
update Book
set price=1.05*price
where authorId in (select B1.authorId
from Book B1, Book B2
where B1.authorId=B2.authorId
and B1.subject<>B2.subject);
```

Update the price by 5%.

Select only those tuples where the subject is different (i.e., the result contains only those authors who wrote books on more than one subject).

Join Book with itself on authorId.

Note: Natural join cannot be used if self join is required. **Why?**



EXERCISE 2 (CONTD)

Do not
use join

For all authors who wrote books on at least two subjects, increase the price of all their books by 5%.

```
update Book
set price=1.05*price
where authorId in (select authorId
from Book
group by authorId
having count(distinct subject)>=2);
```

Update the
price by 5%.

Authors who wrote books
on more than one subject.



EXERCISE 3

Find the last name and first name of all authors who wrote books on both the subjects of Art and Business.

Can we say \Rightarrow `where subject='Art' and subject='Business'`? **No. Why?**

 **Selects nothing.**

Can we say \Rightarrow `where subject='Art' or subject='Business'`? **No. Why?**

 **Selects authors who wrote either Art or Business books, but not necessarily on both subjects.**

```
select lastName, firstName
from (select authorId, lastName, firstName
      from Author natural join Book
      where subject='Art'
     intersect
     select authorId, lastName, firstName
      from Author natural join Book
      where subject='Business');
```

Authors who wrote books on Art.

Select only those authors in the Art set who are also in the Business set.

Authors who wrote books on Business.

EXERCISE 3 (CONT'D)

Use *only* set membership

Find the last name and first name of all authors who wrote books in both the subjects of Art and Business.

```
select lastName, firstName
from Author natural join Book
where subject='Art'
and authorId in (select authorId
from Author natural join Book
where subject='Business');
```

Authors who wrote books on Art (**Art set**).

Select only those authors in the **Art set** who are also in the **Business set**.

Authors who wrote books on Business (**Business set**).



Do not use
subqueries.

EXERCISE 4

Do not create any
derived relations.

Find the last name and first name of all authors who wrote books on exactly ten different subjects.

Is this a
correct
solution?
NO! Why?

```
select lastName, firstName, count(distinct subject) as numSubjects
from Author natural join Book
group by authorId, lastName, firstName
having numSubjects=10;
```

Select only those groups
having exactly ten subjects.

Group the result by authorId,
lastName and firstName.

Join Customer and
Book on authorId.

❗ **Cannot use an alias defined in the select clause in the having clause.**

**Do not use
subqueries.**

EXERCISE 4 (CONT'D)

**Do not create any
derived relations.**

Find the last name and first name of all authors who wrote books on exactly ten different subjects.

Is this a
correct
solution?
NO! Why?

```
select lastName, firstName
from Author A
where 10 = (select count(*)
           from Book
           where A.authorId=Book.authorId
           group by A.authorId);
```

Selects last name of author
if the `where` clause is true.

Counts the number of books written by each author and
returns true if the author wrote exactly ten books.

✎ **Selects authors who wrote exactly ten books.**
(But the subject could be the same!)

✎ **How to fix this?**

Change `select count(*)` **to** `select count(distinct subject)`.



**Do not use
subqueries.**

EXERCISE 4 (CONTD)

**Do not create any
derived relations.**

Find the last name and first name of all authors who wrote books on **exactly ten different subjects.**

Is this a
correct
solution?
YES!

```
select lastName, firstName
from Author natural join Book
group by authorId, lastName, firstName
having count(distinct subject)=10;
```

Join Author and
Book on authorId.

Group the result by
authorId and lastName.

Select only those groups having
exactly ten different subjects.

Is authorId **needed in the group by clause?**

 **YES**, otherwise the count for two different authors with the same last and first name will be incorrect resulting in an incorrect result.

Do not use
subqueries.

EXERCISE 4 (CONT'D)

Do not create any
derived relations.

Find the last name and first name of all authors who wrote books on exactly ten different subjects.

Is this a
correct
solution?

YES!

(But should
not use
subquery!)

```
select lastName, firstName
from Author
where authorId in (select authorId
from Book
group by authorId
having count(distinct subject)=10);
```

Last and first names of authors who
are in the result of the inner select.

Ids of authors who have written
books on ten different subjects.



Do not use
subqueries.

EXERCISE 5

Do not create any
derived relations.

For each customer who made more than 10 orders in 2019, find the customer id, last name and the number of orders in 2019.

Is this a
correct
solution?

Yes!

```
select customerId, lastName, count(*)  
from Customer natural join BookOrder  
where orderYear='2019'  
group by customerId, lastName  
having count(*)>10
```

Select only those groups having
more than 10 orders.

Group the result by
customerId and lastName.

Join Customer and Order on
customerId for orders in 2019.

Are both customerId **and** lastName **needed in the group by clause?**

 **YES**, since they are both present in the select clause.



**Do not use
subqueries.**

EXERCISE 5 (CONTD)

**Do not create any
derived relations.**

For each customer who made more than 10 orders in 2019, find the customer id, last name and the number of orders in 2019.

**Is this a
correct
solution?
NO! Why?**

```
select customerId, lastName, count(*)  
from Customer natural join BookOrder  
group by customerId, lastName  
having count(*) > 10 and orderYear = '2019';
```

✋ **Any attribute present in the *having* clause *that is not being aggregated* must appear in the *group by* clause.**

**Correct
solution.**

```
select customerId, lastName, count(*)  
from Customer natural join BookOrder  
group by customerId, lastName, orderYear  
having count(*) > 10 and orderYear = '2019';
```



EXERCISE 6

Find the customer id, last name and total quantity ordered for those customers who ordered the **largest total quantity** of books.

```
select customerId, lastName, sum(quantity) as totalQuantity
from Customer natural join BookOrder natural join OrderDetails
group by customerId, lastName
having sum(quantity) = (select max(sum(quantity))
                       from BookOrder natural join OrderDetails
                       group by customerId);
```

The customers who ordered the largest total quantity of books.

Select those customers whose total quantity is the largest.

The total quantity of books ordered by each customer.

Customer(customerId, firstName, lastName)

BookOrder(orderId, customerId, orderYear)

OrderDetails(orderId, bookId, quantity)

EXERCISE 6 (CONT'D)

Find the customer id, last name and total quantity ordered for those customers who ordered the **largest total quantity** of books.

```
select customerId, lastName, sum(quantity) as totalQuantity
from Customer natural join BookOrder natural join OrderDetails
group by customerId, lastName
having sum(quantity) >= all (select sum(quantity)
from BookOrder natural join OrderDetails
group by customerId);
```

The customers who ordered the largest total quantity of books.

Select those customers whose total quantity is the largest.

The total quantity of books ordered by each customer.

Customer(customerId, firstName, lastName)

BookOrder(orderId, customerId, orderYear)

OrderDetails(orderId, bookId, quantity)

EXERCISE 6 (CONT'D)

Find the customer id, last name and total quantity ordered for those customers who ordered the **largest total quantity** of books.

```
with TotalBooksOrdered as
  (select customerId, lastName, sum(quantity) as totalQuantity
   from Customer natural join BookOrder natural join OrderDetails
   group by customerId, lastName)
select customerId, lastName, totalQuantity
from TotalBooksOrdered
where totalQuantity = (select max(totalQuantity)
                     from TotalBooksOrdered);
```

The customers who ordered the largest total quantity of books.

The largest total quantity of books ordered (from the *TotalBooksOrdered* derived relation).

The total quantity of books ordered by each customer.

Customer(customerId, firstName, lastName)

BookOrder(orderId, customerId, orderYear)

OrderDetails(orderId, bookId, quantity)

EXERCISE 6 (CONT'D)

Find the customer id, last name and total quantity ordered for those customers who ordered the **largest total quantity** of books.

Is this a
correct
solution?
NO! Why?

```
with TotalBooksOrdered as
  (select customerId, lastName, sum(quantity) as totalQuantity
   from Customer natural join BookOrder natural join OrderDetails
   group by customerId, lastName)
select customerId, lastName, max(totalQuantity)
from TotalBooksOrdered;
```

 **Cannot use an aggregate function in the select clause unless only one tuple is returned.**

Customer(customerId, firstName, lastName)

BookOrder(orderId, customerId, orderYear)

OrderDetails(orderId, bookId, quantity)



EXERCISE 7

The following PL/SQL procedure is used to calculate the interest payable to an account and to update the account balance with the interest payable according to the following schedule.

0% if balance < \$10,000

2% if $\$10,000 \leq \text{balance} < \$100,000$

4% if balance $\geq \$100,000$

Additionally, if the account balance is greater than or equal to \$100,000 and the client holding the account has a loan, then an additional 1% interest is given.

Complete the accountCursor and borrowerCursor definitions so that the PL/SQL procedure executes correctly .

Branch(branchName, district, assets)

Client(clientId, name, address, district)

Loan(loanNo, amount, *branchName*)

Account(accountNo, balance, *branchName*)

Borrower(clientId, loanNo)

Depositor(clientId, accountNo)

EXERCISE 7 (CONT'D)

```
create or replace procedure CalculateInterest as
  currentAccountNo Account.accountNo%type;
  interestPayable   Account.balance%type;
  percentInterest   number;
  -- The cursor for the Account table
  cursor accountCursor is select accountNo, balance from Account;
  -- The cursor for the join of the Borrower and Depositor tables for the current account
  cursor borrowerCursor is select count(loanNo) numLoans from Borrower natural join
    Depositor where accountNo=currentAccountNo;
begin
  for accountRecord in accountCursor loop
    currentAccountNo := accountRecord.accountNo;
    -- Determine the percent interest to pay
    percentInterest := 0;
    if (accountRecord.balance >= 10000 and accountRecord.balance < 100000) then
      percentInterest := 0.02;
```

EXERCISE 7 (CONT'D)

```
elsif (accountRecord.balance >= 100000) then percentInterest := 0.04;  
    -- Give an additional 1% interest if the client has a loan  
    for borrowerRecord in borrowerCursor loop  
        if (borrowerRecord.numLoans <> 0) then  
            percentInterest := percentInterest + 0.01;  
        end if;  
    end loop;  
end if;  
-- Calculate the interest payable  
interestPayable := accountRecord.balance * percentInterest;  
-- Update the client's account balance  
update Account set balance = balance + interestPayable where  
    accountNo=currentAccountNo;  
end loop;  
end CalculateInterest;
```