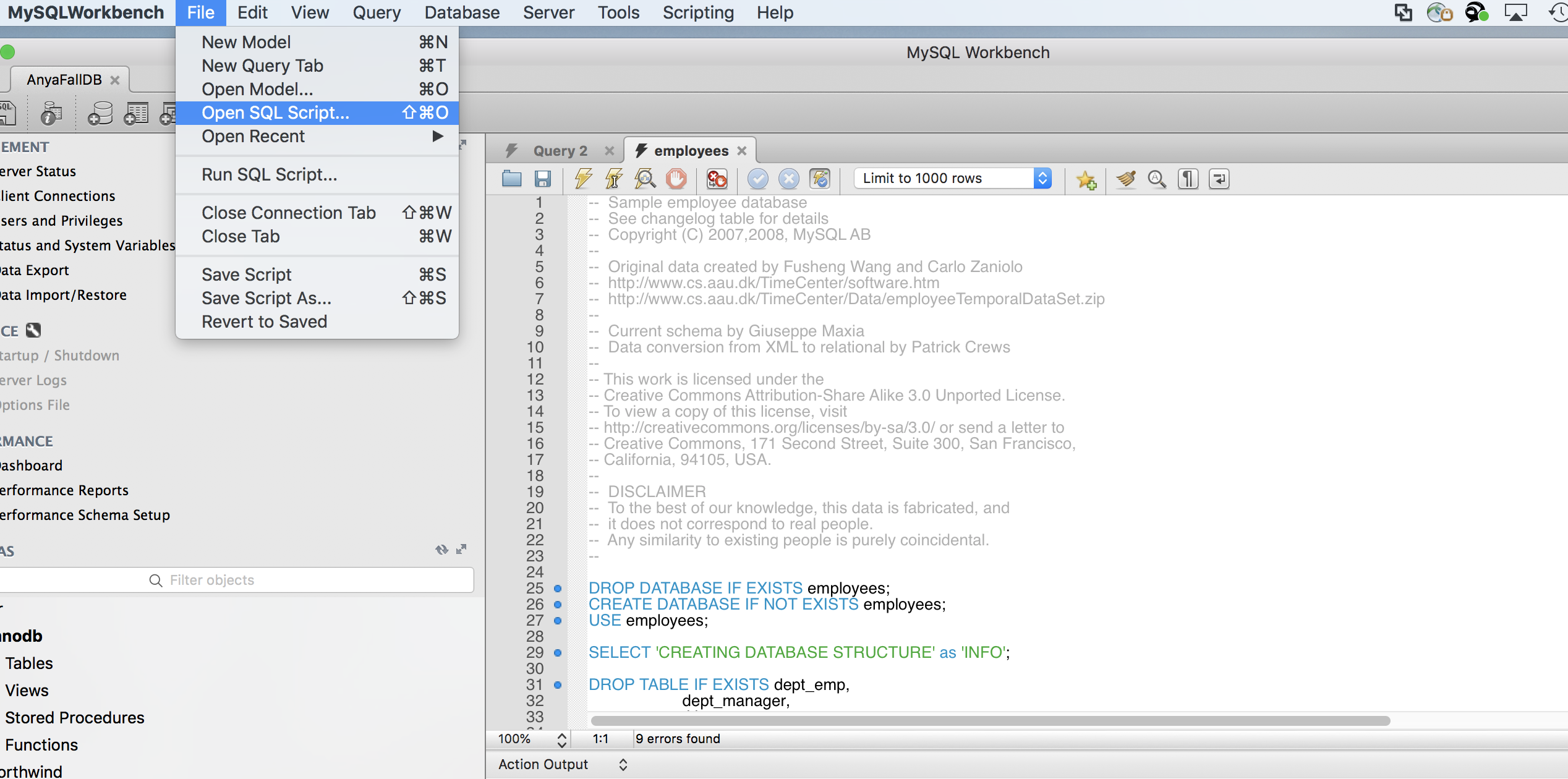
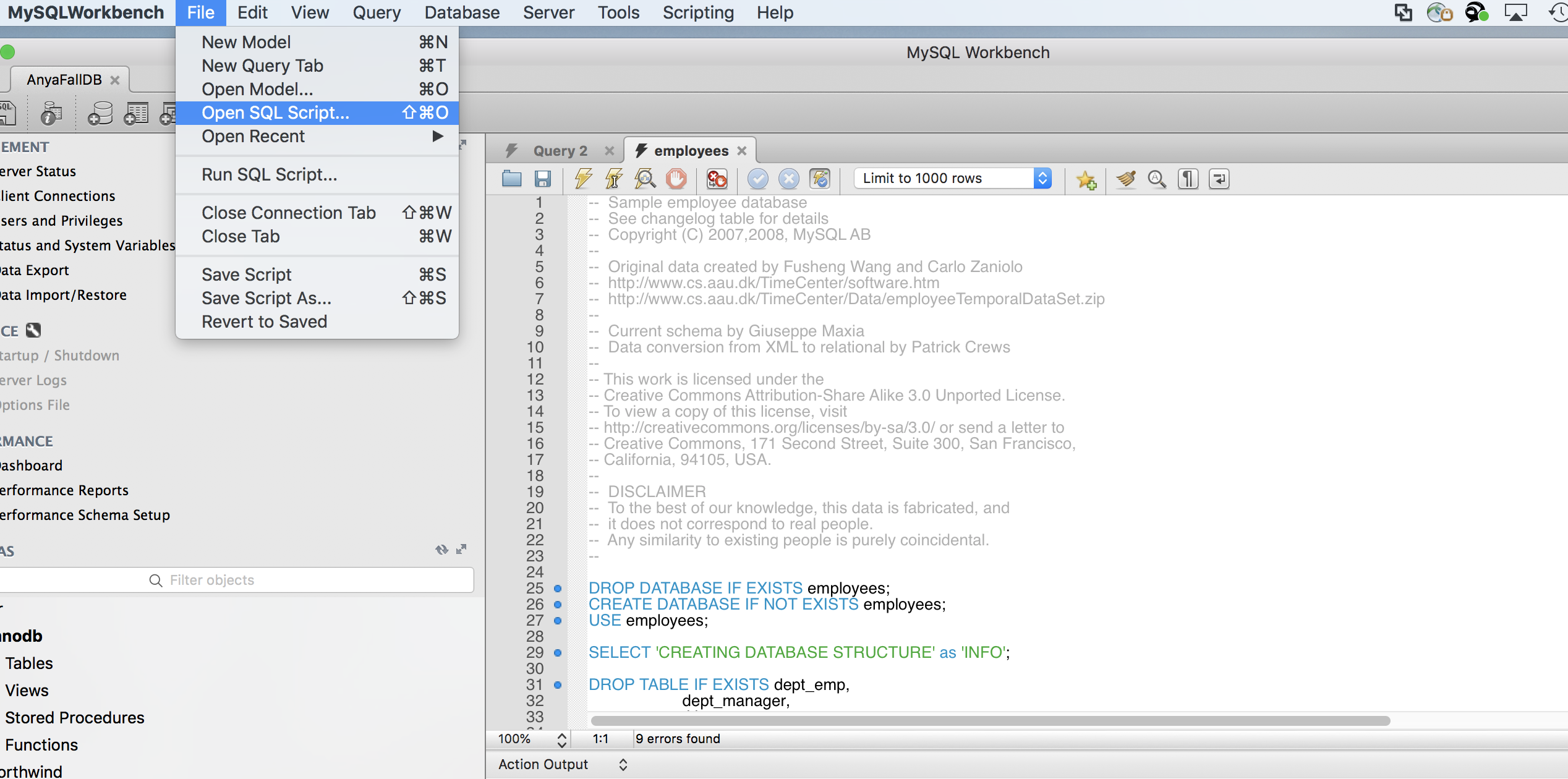
**Lab 2 (MySQL Week 2)**

**Step 1**

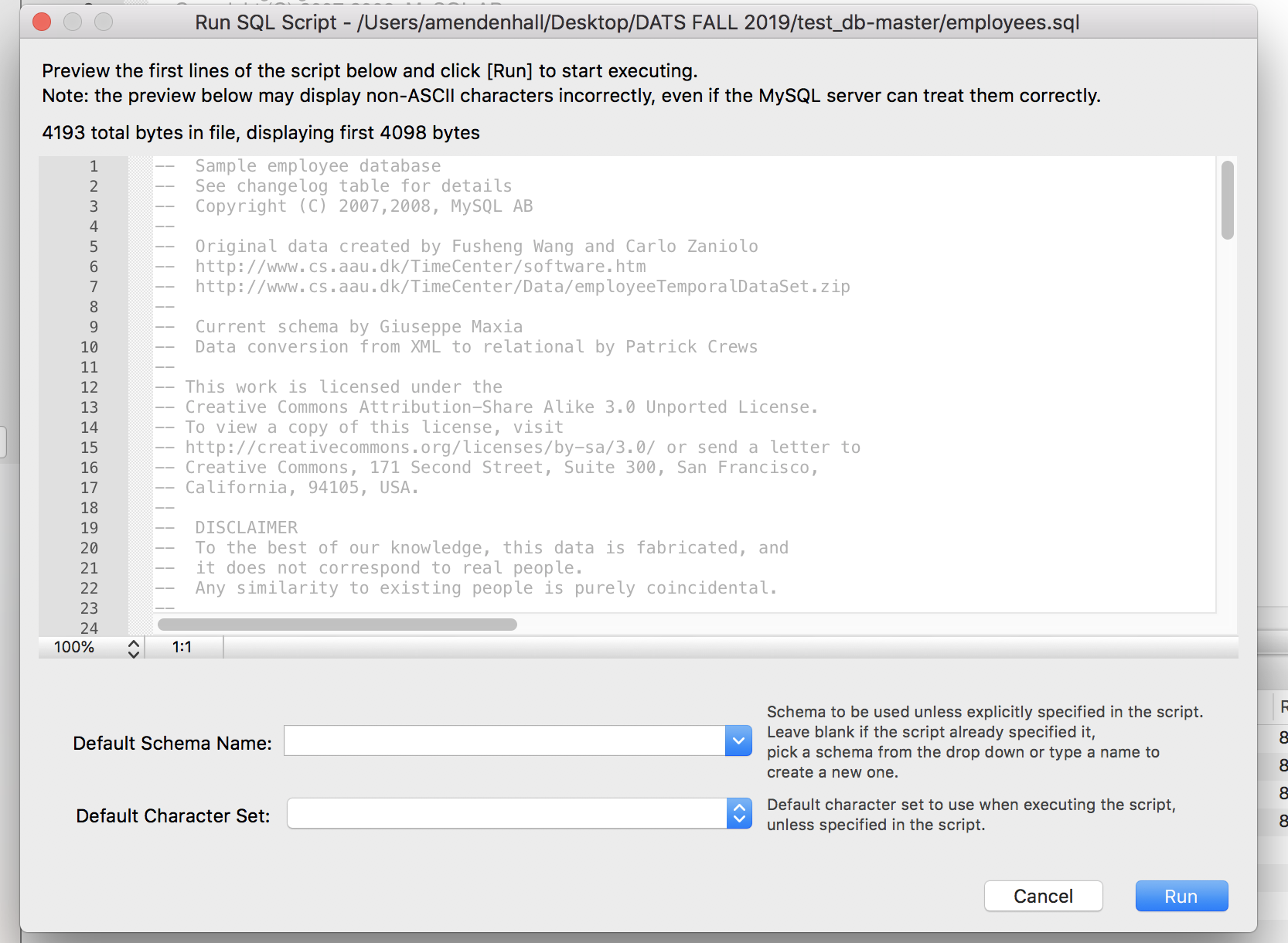
1. Import HR (will download as db.sql file, name the schema **hr)** and Northwind (will download as wrpracti\_northwind file, name the schema northwind) databases into MySQL Workbench in the same way we did in last class
2. **Learn how to load a database into MySQL Workbench as a script**: Download zip file test-db-master to your local Desktop and unzip it, you should get folder test-db-master (contains employee database)
3. Open up MySQL Workbench and Open employees.sql via File 🡪 Open SQL Script 🡪 Open



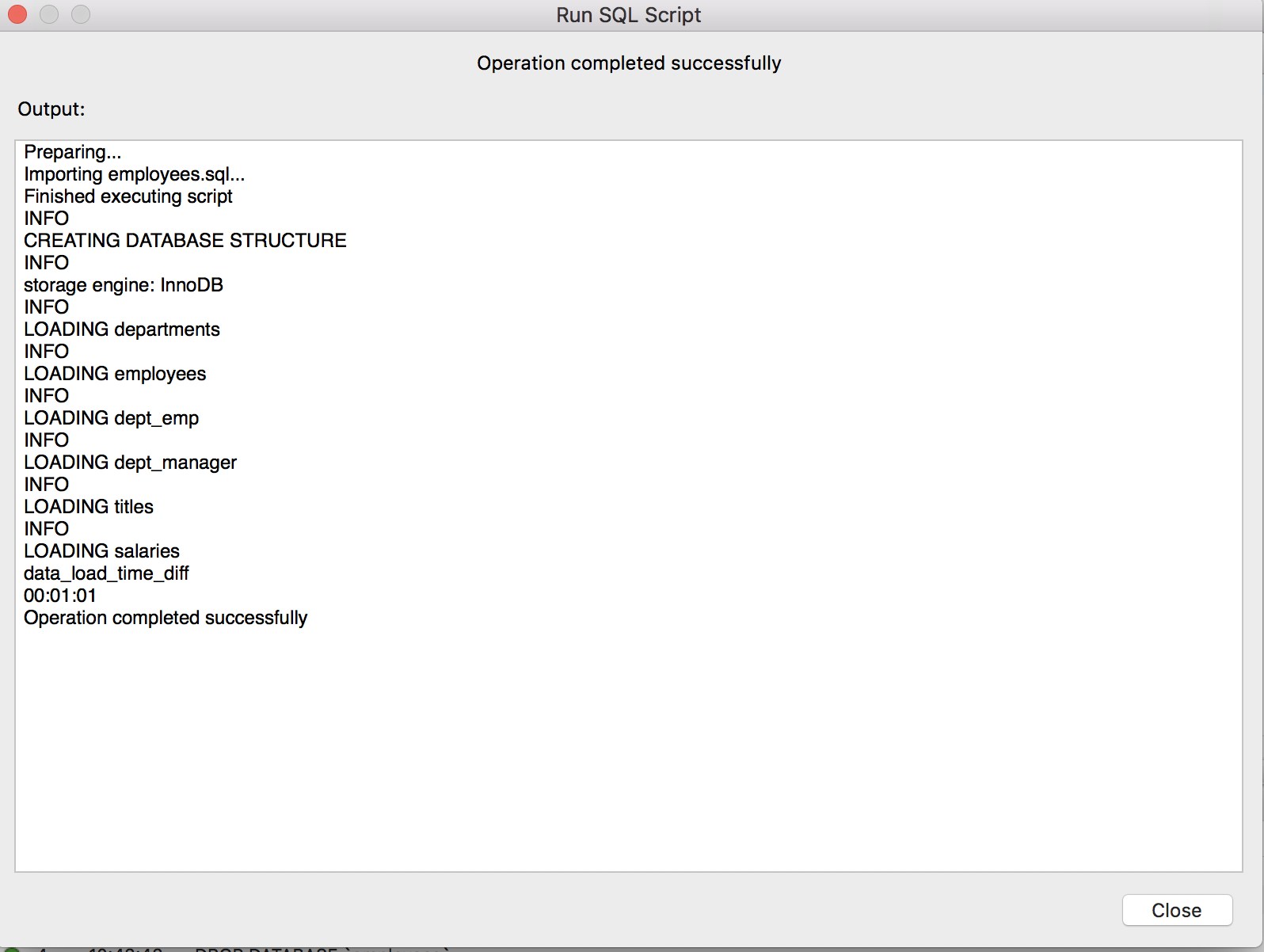
1. Click on Run SQL Script



Leave everything default and click Run:



It will take about 45 seconds or so to run, you will get a message indicating that the script run successfully:



5.

**Functions**

--Write a query to show the total salaries paid to employees (EMPLOYEES table from HR schema)

**Use hr;**

select SUM(salary) from employees;

--Write a query to show minimum salary paid to employees (EMPLOYEES table from HR schema)

select MIN(salary) from employees;

--Write a query to show maximum salary paid to employee working as Programmer (IT\_PROG)

SELECT MAX(salary)

FROM employees

WHERE job\_id = 'IT\_PROG';

--Write a query to show average salary and number of employees working in department 90 (EMPLOYEES table from HR schema)

SELECT AVG(salary),count(\*)

FROM employees

WHERE department\_id = 90;

--Write a query to show highest, lowest, sum and average salary of all employees (EMPLOYEES table from HR schema)

SELECT ROUND(MAX(salary),0) 'Maximum',

ROUND(MIN(salary),0) 'Minimum',

ROUND(SUM(salary),0) 'Sum',

ROUND(AVG(salary),0) 'Average'

FROM employees;

--Write a query to show the difference between the highest and lowest salaries paid to employees (EMPLOYEES table from HR schema)

SELECT MAX(salary) - MIN(salary) DIFFERENCE

FROM employees;

--Write a query to list details of employees where the length of the first name greater than or equal to 8 (EMPLOYEES table from HR schema)

SELECT \*

FROM employees

WHERE LENGTH(first\_name) >= 8;

--Write a query to list current date in the following format

--Sample date: 2018-09-14

--Output date: September 17, 2019

SELECT DATE\_FORMAT(CURDATE(),'%M %e, %Y')

AS 'Current\_date';

--Write a query to list current date in the following format

--Sample date: 2018-09-14

--Output date: Tuesday September , 2019

SELECT DATE\_FORMAT(NOW(), '%W %M %Y');

--Write a query to extract a year from a current date:

SELECT EXTRACT(YEAR FROM NOW());

--Using LEFT/RIGHT functions

SELECT RIGHT('asdf', 1);

SELECT LEFT('asdf', 2);

SELECT \* from employees;

SELECT FIRST\_NAME, LAST\_NAME,

concat(LEFT(FIRST\_NAME, 1), LEFT(LAST\_NAME, 1)) as Initials from employees;

**Aggregate Functions**

--Write a query to list job id and maximum salary of employees where max salary is greater or equal to $4000 (EMPLOYEES table from HR schema). Notice how job\_id have been arranged in a group where each salary is >=4000

SELECT job\_id, MAX(salary)

FROM employees

GROUP BY job\_id

HAVING MAX(salary) >=4000;

--Write a query to list average salary for all departments employing more than 10 employees. (EMPLOYEES table from HR schema). Notice how department id have been arranged in a group with average salary and number of employees in each group.

SELECT department\_id, AVG(salary), COUNT(\*)

FROM employees

GROUP BY department\_id

HAVING COUNT(\*) > 10;

**Grouping**

--Write a query to list the number of employees who have the same job (EMPLOYEES table from HR schema)

SELECT job\_id, COUNT(\*)

FROM employees

GROUP BY job\_id;

--Write a query to list the department ID and the total salary paid in each department (EMPLOYEES table from HR schema)

SELECT department\_id, SUM(salary)

FROM employees

GROUP BY department\_id;

--Write a query to list the average salary for each job ID EXLUDING programmer (IT\_PROG) (EMPLOYEES table from HR schema)

SELECT job\_id, AVG(salary)

FROM employees

WHERE job\_id <> 'IT\_PROG'

GROUP BY job\_id;

--Write a query to list the total salary, maximum, minimum, average salary of employees for department 90 only (EMPLOYEES table from HR schema). Notice how we are arranging jobs in a group for department id 90:

SELECT job\_id, SUM(salary), AVG(salary), MAX(salary), MIN(salary)

FROM employees

WHERE department\_id = '90'

GROUP BY job\_id;

**Joins**

--Write a query to find the name (first\_name, last\_name), department id and name of all the employees (connect employees and departments table by department id.

SELECT first\_name, last\_name, department\_id, department\_name

FROM employees

JOIN departments USING (department\_id);

--Write a query to find the name (first\_name, last\_name), department id and name of all the employees who work in London (connect employees and departments and location tables):

SELECT e.first\_name, e.last\_name, e.job\_id, e.department\_id, d.department\_name

FROM employees e

JOIN departments d

ON (e.department\_id = d.department\_id)

JOIN locations l ON

(d.location\_id = l.location\_id)

WHERE LOWER(l.city) = 'London';

--Write a query to find the name (first\_name, last\_name) and hire date of the employees who were hired after ‘Jones’:

SELECT e.first\_name, e.last\_name, e.hire\_date

FROM employees e

JOIN employees davies

ON (davies.last\_name = 'Jones')

WHERE davies.hire\_date < e.hire\_date;

--Write a query to find employee id, the name (last\_name) along with their manager\_id and name (last\_name):

SELECT e.employee\_id 'Emp\_Id', e.last\_name 'Employee',

m.employee\_id 'Mgr\_Id', m.last\_name 'Manager'

FROM employees e

join employees m

ON (e.manager\_id = m.employee\_id);

**Subqueries**

--Write a query to find a name (first\_name, last\_name) and salary of the employees who have a higher salary than the employee whose last\_name = ‘Bull’ (EMPLOYEES table from HR schema).

--The subquery (inner query) will execute once before the main query (outer query) executes

--The main query (outer query) will use the subquery result.

SELECT FIRST\_NAME, LAST\_NAME, SALARY

FROM employees

WHERE SALARY >

(SELECT salary FROM employees WHERE last\_name = 'Bull');

--Write a query to find a name (first\_name, last\_name) of all employees who work in the IT department. (EMPLOYEES table from HR schema).

--Notice how IN operator checks if a value is within a set of values

SELECT first\_name, last\_name

FROM employees

WHERE department\_id

IN (SELECT department\_id FROM departments WHERE department\_name='IT');

--Write a query to find a name (first\_name, last\_name) of all employees who have a manager in USA based department. (EMPLOYEES table from HR schema).

SELECT first\_name, last\_name FROM employees

WHERE manager\_id in (select employee\_id

FROM employees WHERE department\_id

IN (SELECT department\_id FROM departments WHERE location\_id

IN (select location\_id from locations where country\_id='US')));

--Write a query to find a name (first\_name, last\_name) of all employees who are managers (EMPLOYEES table from HR schema).

SELECT first\_name, last\_name

FROM employees

WHERE (employee\_id IN (SELECT manager\_id FROM employees));

--Write a query to find a name (first\_name, last\_name) and salary of the employees whose salary is greater than the average salary (EMPLOYEES table from HR schema).

SELECT first\_name, last\_name, salary FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

**Special Functionality**

--Write a query to list the number of available jobs in EMPLOYEES table from HR schema

use hr;

select COUNT(DISTINCT job\_id) FROM employees;

**Demonstrating Transactions**

**use employees;**

-- create employee record

-- create title record

-- create employee department record

-- create employee salary record

--Start transaction

begin;

--Create new employee

INSERT INTO employees SELECT max(emp\_no) + 1, '1985-01-01', 'Jessie', 'Porter', 'M', '2018-01-01' FROM employees;

--Create title record for employee

INSERT INTO titles SELECT max(emp\_no), 'Staff', '2018-01-01', '9999-01-01'FROM employees;

--Assign employee to a department

INSERT INTO dept\_emp SELECT max(emp\_no), 'd008', '2018-01-01', '9999-01-01'FROM employees;

--Give employee salary

INSERT INTO salaries SELECT max(emp\_no), 120000, '2018-01-01', '9999-01-01' FROM employees;

--Commit transaction

commit;

SELECT \* from employees where first\_name = 'Jessie' and last\_name = 'Porter';

-- Remember that MySQL is using Autocomit within SQL session, but if you explicitly start and end the transaction, the autocommit does not take effect

--use begin to begin a transaction

-- insert into EMPLOYEES table (Michael Weston record)

-- select to check that this record is there

-- rollback (wipe out your insert)

-- select to check that record is no longer there

begin;

INSERT INTO employees SELECT max(emp\_no) + 1, '1976-02-02', 'Micheal', 'Weston', 'M', '2015-01-02' FROM employees;

SELECT \* from employees where last\_name = 'Weston';

Rollback;

SELECT \* from employees where last\_name = 'Weston';

--use begin to begin a transaction

-- insert into EMPLOYEES table (Michael Weston record)

-- select to check that this record is there

-- run update of this record

-- commit (saved both insert and update)

-- select to check that record is no longer there

-- now start another transaction (by using begin)

-- delete this record

-- select it to make sure its gone

-- rollback (to undue your delete)

-- run select again to see that your record is there

begin;

INSERT INTO employees SELECT max(emp\_no) + 1, '1976-02-02', 'Micheal', 'Weston', 'M', '2015-01-02' FROM employees;

SELECT \* from employees where last\_name = 'Weston';

UPDATE employees

SET birth\_date = '1976-03-02'

WHERE emp\_no = 500001;

commit;

SELECT \* from employees where last\_name = 'Weston';

Begin

DELETE FROM employees

WHERE emp\_no = 500001;

SELECT \* from employees where last\_name = 'Weston';

Rollback;

SELECT \* from employees where last\_name = 'Weston';

**ADVANCED QUERIES**

/\* Select name of product, UnitsInStock columns from Product table where UnitInStock > average

\*/

**use northwind;**

SELECT ProductName, UnitsInStock

FROM Products

WHERE UnitsInStock > (SELECT AVG(UnitsInStock) FROM Products);

/\*  
This query retrieves a list of customers that made   
purchases after the date 1998-05-01.  
   
The subquery returns a list of CustomerIDs which is  
used in outer query.  
\*/

select CustomerID, CompanyName

from Customers

where CustomerID in

(

select CustomerID

from Orders

where orderDate > '1998-05-01'

);

/\*  
This query returns the same result as query above  
because the list of CustomerIDs are used rather than  
the subquery.  
\*/

select CustomerID, CompanyName

from Customers

where CustomerID in

(

'BONAP',

'DRACD',

'ERNSH',

'LEHMS',

'LILAS',

'PERIC',

'QUEEN',

'RATTC',

'RICSU',

'SIMOB',

'TORTU'

);

/\*  
This query returns the same result as the one  
in Practice #1 but here no subquery is used.   
Instead, we used inner join.   
   
Often, a query that contains subqueries can be   
rewritten as a join.  
   
Using inner join allows the query optimizer to   
retrieve data in the most efficient way.  
\*/

select a.CustomerID, a.CompanyName

from Customers as a

inner join Orders as b on a.CustomerID = b.CustomerID

where b.orderDate > '1998-05-01'

/\*  
This query finds out all the employees who live  
in the same city and country as customers.  
   
The subquery returns a table of two columns and  
91 rows. It's returned to outer query and City  
and Country in employees table are compared with  
each row in the table.  
\*/

select EmployeeID, FirstName, LastName, City, Country

from Employees

where row(City, Country) in

(select City, Country from Customers);

/\*  
This query uses a subquery in the FROM clause.  
The subquery is given an alias x so that we can  
refer to it in the outer select statement.  
\*/

select x.ProductID,

y.ProductName,

x.max\_unit\_price

from

(

select ProductID, max(UnitPrice) as max\_unit\_price

from OrderDetails

group by ProductID

) as x

inner join Products as y on x.ProductID = y.ProductID

/\*  
This query uses EXISTS keyword in WHERE clause   
to return a list of customers whose products   
were shipped to UK.  
   
Note that the outer query only returns a row  
where the subquery returns TRUE.  
\*/

select CustomerID, CompanyName

from Customers as a

where exists

(

select \* from Orders as b

where a.CustomerID = b.CustomerID

and ShipCountry = 'UK'

);

/\*  
This query uses INNER JOIN and returns the same   
result set as the query above. It shows you that  
the correlated subquery can be rewritten as join  
operation.  
\*/

select CustomerID, CompanyName

from Customers as a

where exists

(

select \* from Orders as b

where a.CustomerID = b.CustomerID

and ShipCountry = 'UK'

);

/\*  
This query uses NOT EXISTS keyword in WHERE clause   
to return a list of customers whose products were   
NOT shipped to UK.  
   
Note that this query returns two more rows than  
the query in Practice #1. This is because Customer  
PARIS and FISSA do not have records in orders table.  
\*/

select CustomerID, CompanyName

from Customers as a

where not exists

(

select \* from Orders as b

where a.CustomerID = b.CustomerID

and ShipCountry <> 'UK'

);

use northwind

-- Display distinct Quantity values

-- by using DISTINCT keyword

SELECT DISTINCT Quantity

FROM OrderDetails

ORDER BY Quantity;

-- Retrieve distinct rows based on

-- two columns (Quantity and Discount)

SELECT DISTINCT Quantity, Discount

FROM OrderDetails

ORDER by Quantity, Discount;

/\*

Get the supplier's continent based on its country. For

example, if country is USA, then continent is North America.

If the country is not USA, Canada, Brazil, Japan, Singapore,

Australia, the continent is Europe which is covered in the

ELSE part of the CASE.

\*/

SELECT CompanyName,

CASE Country WHEN 'USA' THEN 'North America'

WHEN 'Canada' THEN 'North America'

WHEN 'Brazil' THEN 'South America'

WHEN 'Japan' THEN 'Asia'

WHEN 'Singapore' THEN 'Asia'

WHEN 'Australia' THEN 'Australia'

ELSE 'Europe' END AS Continent

FROM Suppliers

ORDER BY CompanyName;

/\*

Get the supplier's continent based on its country.

The result is ordered by using column alias Continent.

\*/

SELECT CompanyName,

CASE Country WHEN 'USA' THEN 'North America'

WHEN 'Canada' THEN 'North America'

WHEN 'Brazil' THEN 'South America'

WHEN 'Japan' THEN 'Asia'

WHEN 'Singapore' THEN 'Asia'

WHEN 'Australia' THEN 'Australia'

ELSE 'Europe' END AS Continent

FROM Suppliers

ORDER BY Continent;

/\*

This query calculates the average freight in orders

table for each customer. GROUP BY clause is used

to group all orders for each individual customer.

\*/

select CustomerID, avg(Freight) as average\_freight

from Orders

/\*

This query returns how many distinct countries

the suppliers are from.

\*/

select count(distinct Country) as num\_of\_countries

from Suppliers;

/\*

Query 1:

This query returns 2 and shows you that

NULL values are not counted when using DISTINCT

keyword in COUNT function.

\*/

select count(distinct ReportsTo) as num\_of\_managers

from Employees;

/\*

Query 2:

This query shows how many distinct values are

in ReportsTo column. Note that NULL is in the

result but not counted as a distinct value in

Query 1 above.

\*/

select distinct ReportsTo

from Employees;

/\*

This query only counts where ReportsTo column is null.

It shows that count(\*) also returns the row that

contains NULL value.

\*/

select count(\*)

from Employees;

where ReportsTo is null;