COP 4710: Database Systems
Fall 2013

Chapter 5 – Introduction To SQL – Part 2

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop4710/fall2013

Department Of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

SQL Join Operators

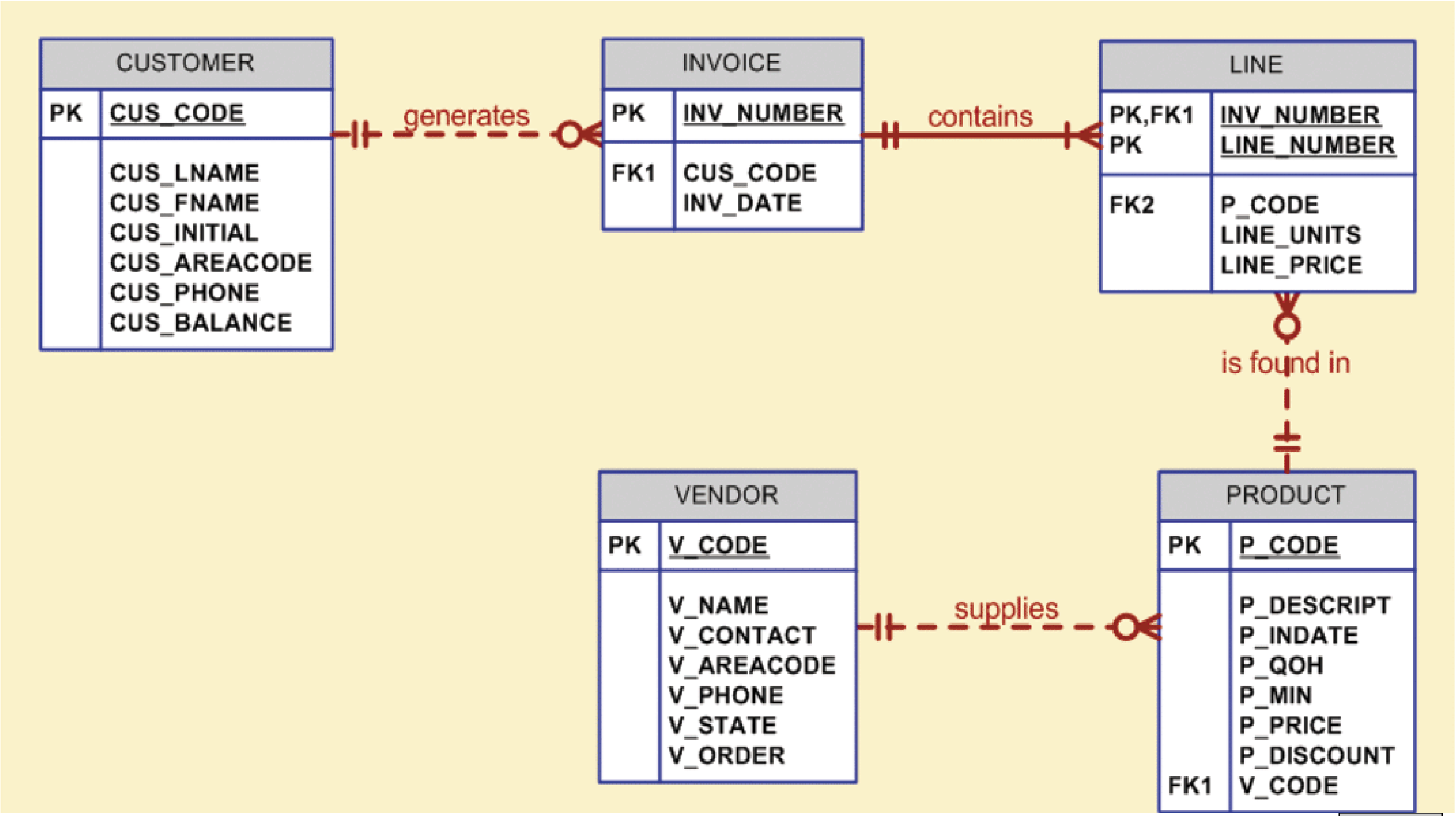
- In the first part of the SQL notes, we covered the major DDL and DML commands available in ANSI-standard SQL.

- In this section of notes, we'll focus only on the SELECT command and in particular its use and forms when multiple tables are involved in the query.

- There are many different ways to join tables together in SQL and this set of notes will examine all of them.

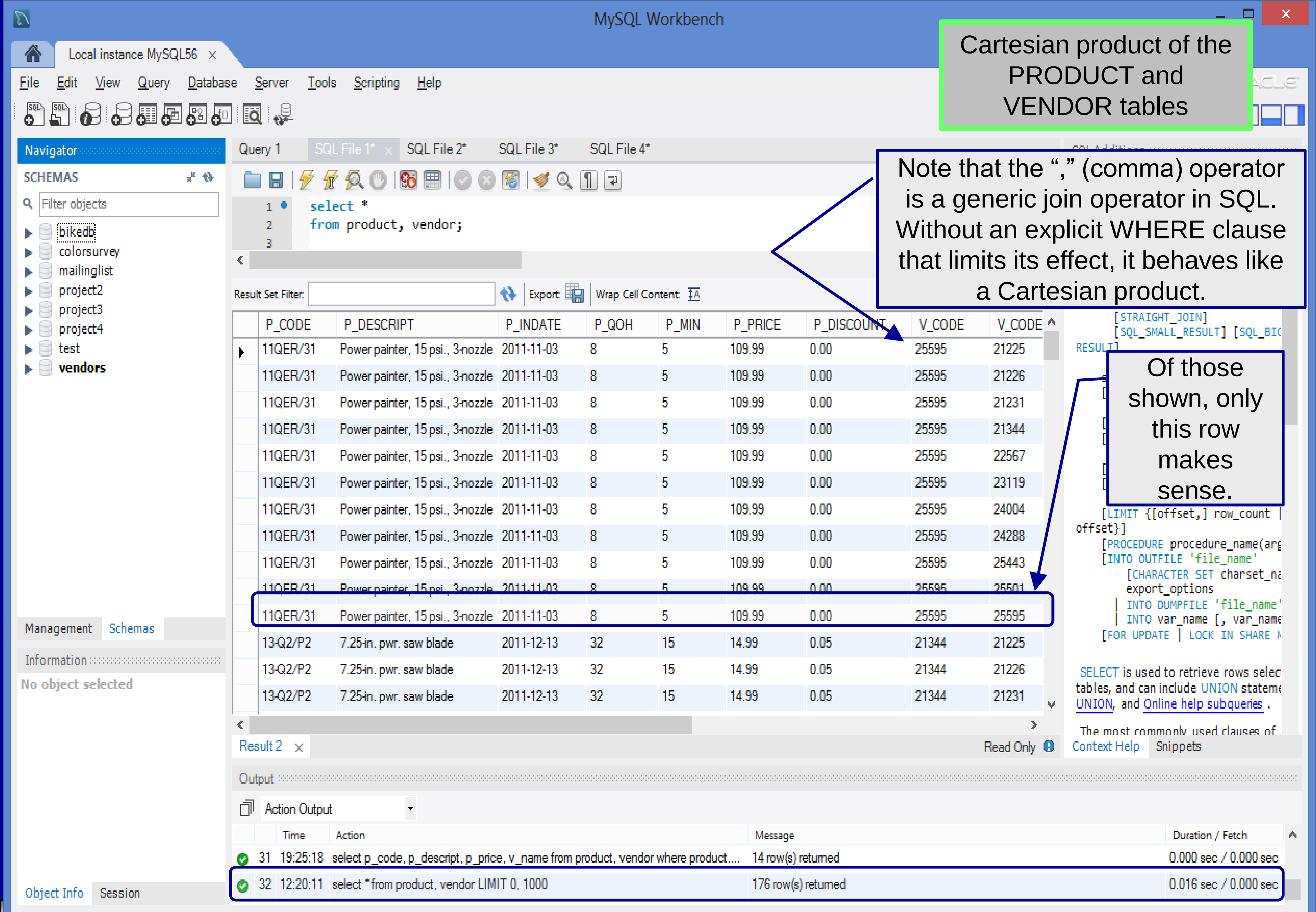
- We'll again use the sample database shown on the next page for the examples in this set of notes.

The ERD for the database used in the examples in the following slides



SQL Join Operators

- The simplest technique for joining tables in SQL is to simply list the tables in the FROM clause of a SELECT command. The table names are separated by commas.
- There is no theoretical limit to the number of tables that can be joined in this fashion.
- Simply listing the tables in the FROM clause will cause SQL to form the Cartesian product of all the tables listed in the FROM clause.
- The following slide illustrates this technique of joining the VENDOR table with the PRODUCT table. Note that the current instance of the VENDOR table contains 11 rows, and the current instance of the PRODUCT table contains 16 rows, so the resulting Cartesian product will contain $11 \times 16 = 176$ rows.



Cartesian product of the PRODUCT and VENDOR tables

Note that the "," (comma) operator is a generic join operator in SQL. Without an explicit WHERE clause that limits its effect, it behaves like a Cartesian product.

Of those shown, only this row makes sense.

SQL Join Operators

- Using the Cartesian product type of join operation is often not the effect that you want to achieve, as this introduces many rows of unrelated data. Recall the problems we had when using this operator in relational algebra.
- In order to accomplish the effect of a natural join operation, explicit join conditions must be added to the WHERE clause of the SELECT command where equality is established across the foreign and primary keys of the joined tables.
- Returning to the previous example, in order to achieve the effect of a natural join, we need to ensure that the PRODUCT.v_code = VENDOR.v_code in every row of the joined tables. The next slide illustrates this case.

Natural Join of the PRODUCT and VENDOR tables

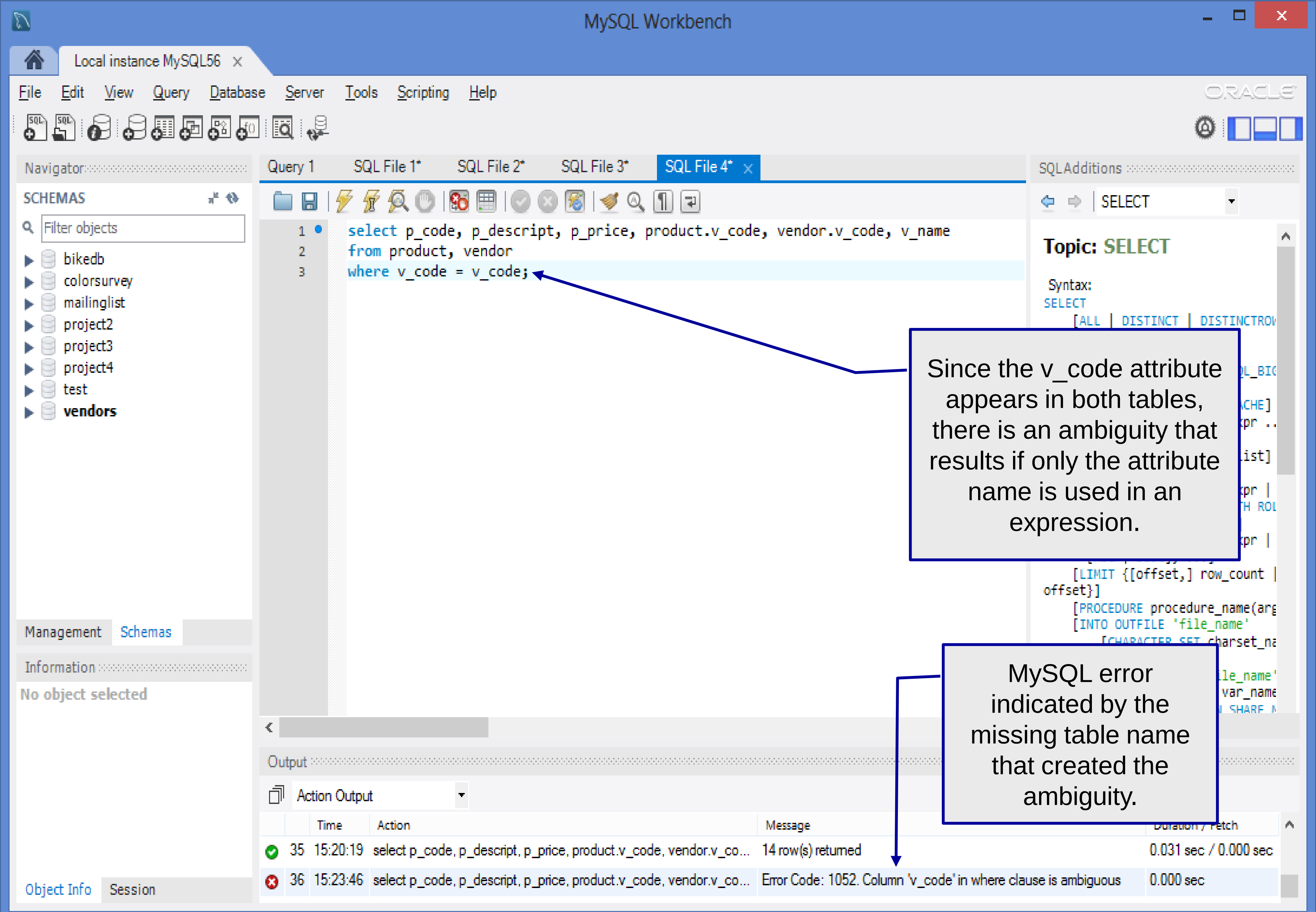
```sql
select p_code, p_descript, p_price, product.v_code, vendor.v_code, v_name
from product, vendor
where product.v_code = vendor.v_code;
```

Note that the WHERE condition contains the join condition that the foreign key v_code in PRODUCT matches the primary key v_code in the VENDOR table.

p_code	p_descript	p_price	v_code	v_code	v_name
11QER/31	Power painter, 15 psi., 3-nozzle	109.99	25595	25595	Rubicon Systems
13-Q2/P2	7.25-in. pwr. saw blade	14.99	21344	21344	Gomez Bros.
14-Q1/L3	9.00-in. pwr. saw blade	17.49	21344	21344	Gomez Bros.
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	39.95	23119	23119	Randsets Ltd.
1558-QW1	Hrd. cloth, 1/2-in., 3x50	43.99	23119	23119	Randsets Ltd.
2232/QTY	B&D jigsaw, 12-in. blade	109.92	24288	24288	ORDVA, Inc.
2232/QWE	B&D jigsaw, 8-in. blade	99.87	24288	24288	ORDVA, Inc.
2238/QPD	B&D cordless drill, 1/2-in.	38.95	25595	25595	Rubicon Systems
23109-HB	Claw hammer	9.95	21225	21225	Bryson, Inc.
54778-2T	Rat-tail file, 1/8-in. fine	4.99	21344	21344	Gomez Bros.
89-WRE-Q	Hicut chain saw, 16 in.	256.99	24288	24288	ORDVA, Inc.
SM-18277	1.25-in. metal screw, 25	6.99	21225	21225	Bryson, Inc.
SW-23116	2.5-in. wd. screw, 50	8.45	21231	21231	D&E Supply
WR3/TT3	Steel matting, 4'x8'x1/6", .5" ...	119.95	25595	25595	Rubicon Systems

All vendor codes match in each row.

	Time	Action	Message	Duration / Fetch
34	15:12:35	select * from product LIMIT 0, 1000	16 row(s) returned	0.000 sec / 0.000 sec
35	15:20:19	select p_code, p_descript, p_price, product.v_code, vendor.v_code, v_name fro...	14 row(s) returned	0.031 sec / 0.000 sec



Since the v_code attribute appears in both tables, there is an ambiguity that results if only the attribute name is used in an expression.

MySQL error indicated by the missing table name that created the ambiguity.

Query: List details of the products along with the v_code and v_name of the vendor who supplies that product. Order the results in ascending order of price.

Natural Join of the PRODUCT and VENDOR tables with ordering of the results

```sql
select p_code, p_descript, p_price, product.v_code, vendor.v_code, v_name
from product, vendor
where product.v_code = vendor.v_code
order by p_price;
```

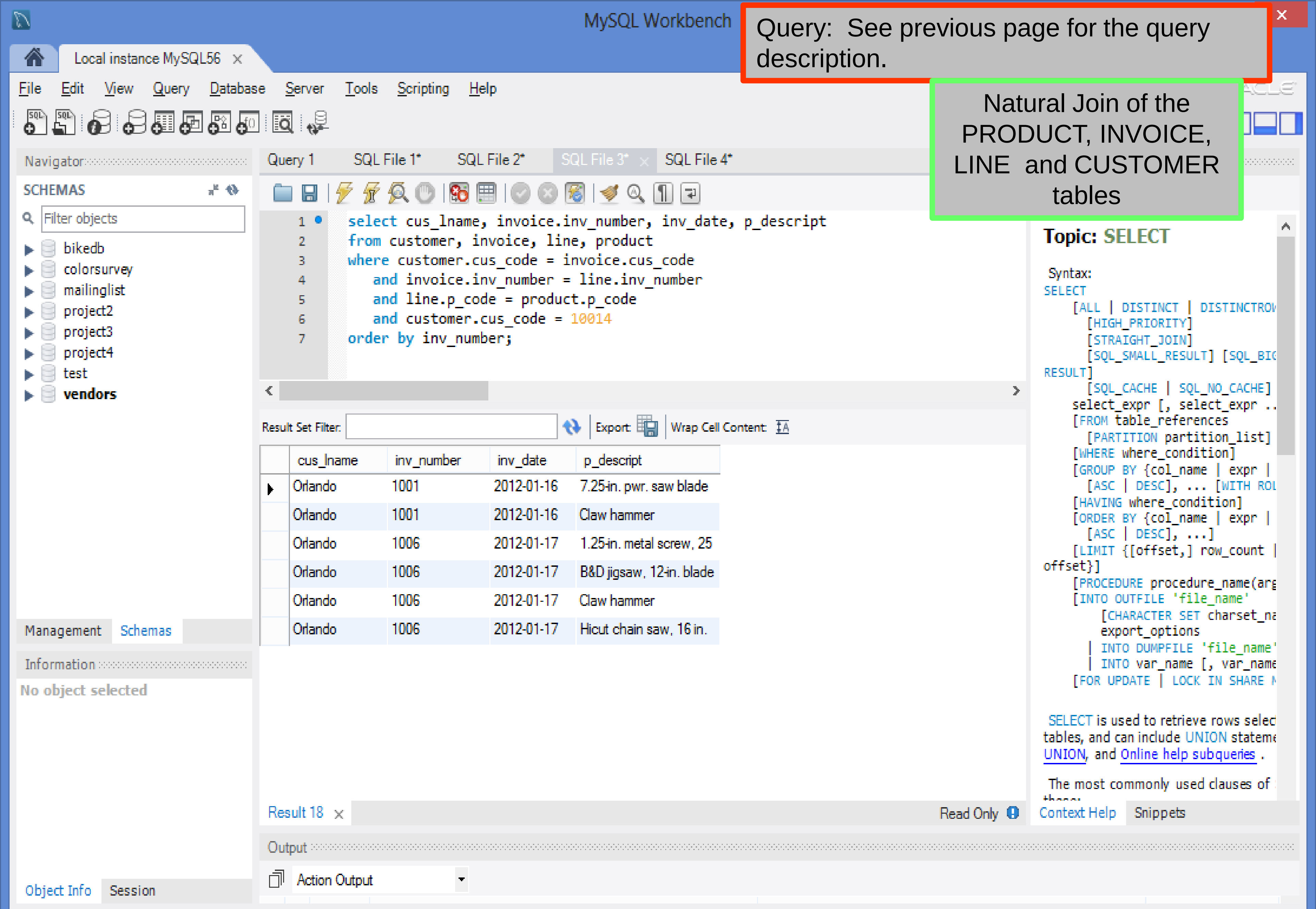
p_code	p_descript	p_price	v_code	v_code	v_name
54778-2T	Rat-tail file, 1/8-in. fine	4.99	21344	21344	Gomez Bros.
SM-18277	1.25-in. metal screw, 25	6.99	21225	21225	Bryson, Inc.
SW-23116	2.5-in. wd. screw, 50	8.45	21231	21231	D&E Supply
23109-HB	Claw hammer	9.95	21225	21225	Bryson, Inc.
13-Q2/P2	7.25-in. pwr. saw blade	14.99	21344	21344	Gomez Bros.
14-Q1/L3	9.00-in. pwr. saw blade	17.49	21344	21344	Gomez Bros.
2238/QPD	B&D cordless drill, 1/2-in.	38.95	25595	25595	Rubicon Systems
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	39.95	23119	23119	Randsets Ltd.
1558-QW1	Hrd. cloth, 1/2-in., 3x50	43.99	23119	23119	Randsets Ltd.
2232/QWE	B&D jigsaw, 8-in. blade	99.87	24288	24288	ORDVA, Inc.
2232/QTY	B&D jigsaw, 12-in. blade	109.92	24288	24288	ORDVA, Inc.
11QER/31	Power painter, 15 psi., ...	109.99	25595	25595	Rubicon Systems
WR3/TT3	Steel matting, 4'x8'x1/...	119.95	25595	25595	Rubicon Systems
89-WRE-Q	Hicut chain saw, 16 in.	256.99	24288	24288	ORDVA, Inc.

SQL Join Operators

- When using the Cartesian product type of join operation to effect a natural join operation, there will always need to be a join condition in the WHERE clause.
- In general, since there can be *n* tables listed in the FROM clause, there will be *n-1* join conditions in the WHERE clause if a natural join is the effect that is desired.
- The example on the following page illustrates answering the query: List the customer last name, invoice number, invoice date, and product description for all invoices for customer number 10014 and order the results in increasing order of invoice number.

Query: See previous page for the query description.

Natural Join of the PRODUCT, INVOICE, LINE and CUSTOMER tables



```sql
select cus_lname, invoice.inv_number, inv_date, p_descript
from customer, invoice, line, product
where customer.cus_code = invoice.cus_code
   and invoice.inv_number = line.inv_number
   and line.p_code = product.p_code
   and customer.cus_code = 10014
order by inv_number;
```

cus_lname	inv_number	inv_date	p_descript
Orlando	1001	2012-01-16	7.25-in. pwr. saw blade
Orlando	1001	2012-01-16	Claw hammer
Orlando	1006	2012-01-17	1.25-in. metal screw, 25
Orlando	1006	2012-01-17	B&D jigsaw, 12-in. blade
Orlando	1006	2012-01-17	Claw hammer
Orlando	1006	2012-01-17	Hicut chain saw, 16 in.

Query: Same query as previous page.

Natural Join of the PRODUCT, INVOICE, LINE and CUSTOMER tables using table aliases.

```sql
select cus_lname, i.inv_number, inv_date, p_descript
from customer as c, invoice as i, line as l, product as p
where c.cus_code = i.cus_code
   and i.inv_number = l.inv_number
   and l.p_code = p.p_code
   and c.cus_code = 10014
order by inv_number;
```

cus_lname	inv_number	inv_date	p_descript
Orlando	1001	2012-01-16	7.25-in. pwr. saw blade
Orlando	1001	2012-01-16	Claw hammer
Orlando	1006	2012-01-17	1.25-in. metal screw, 25
Orlando	1006	2012-01-17	B&D jigsaw, 12-in. blade
Orlando	1006	2012-01-17	Claw hammer
Orlando	1006	2012-01-17	Hicut chain saw, 16 in.

Query: Same query as previous page.

Natural Join of the PRODUCT, INVOICE, LINE and CUSTOMER tables using table aliases.

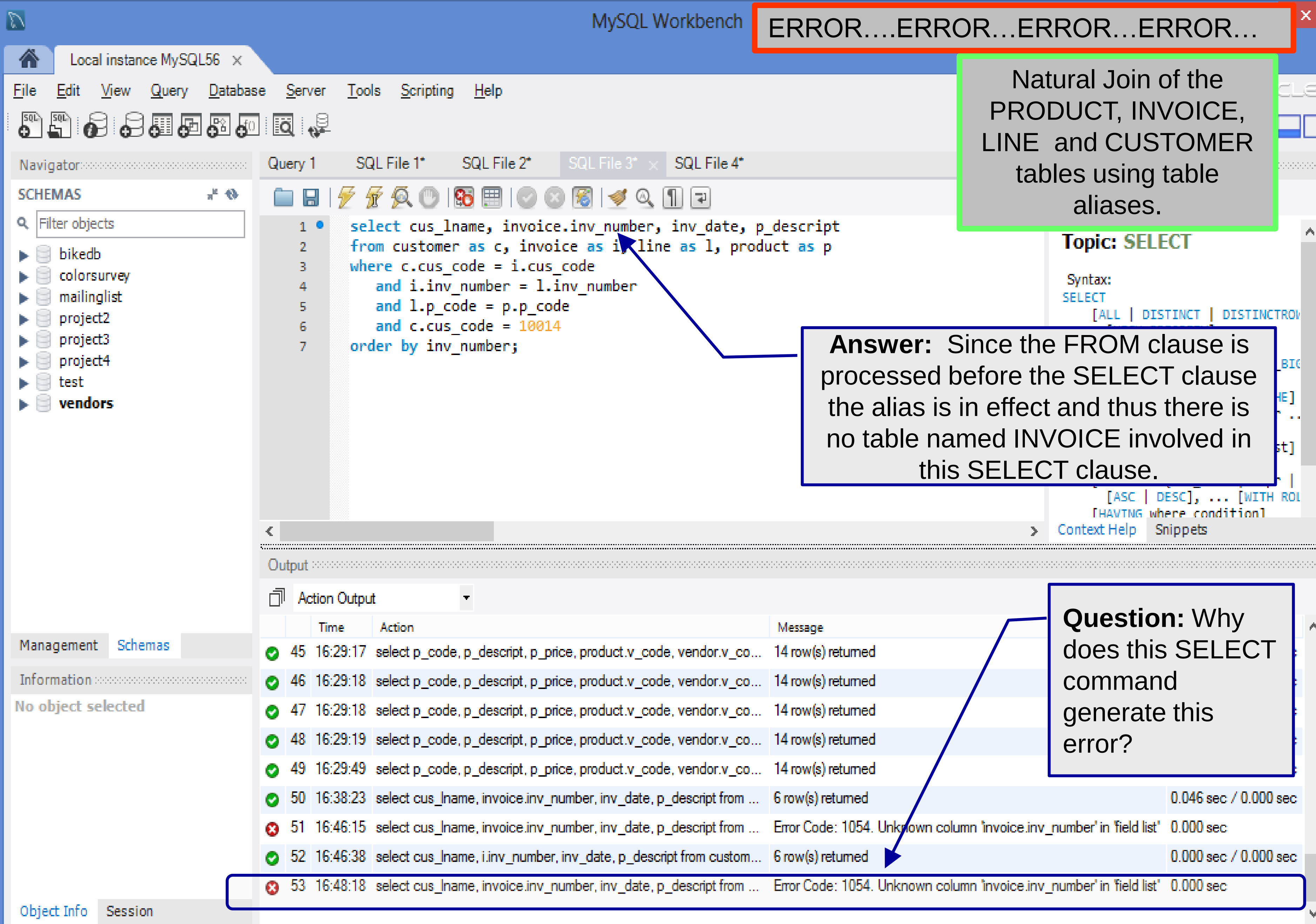
```
select cus_lname, i.inv_number, inv_date, p_descript
from customer c, invoice i, line l, product p
where c.cus_code = i.cus_code
   and i.inv_number = l.inv_number
   and l.p_code = p.p_code
   and c.cus_code = 10014
order by inv_number;
```

The keyword AS is optional when defining table aliases or column aliases in MySQL. Many SQL environments require it in both places, so I'll generally put it in.

cus_lname	inv_number	inv_date	p_descript
Orlando	1001	2012-01-16	7.25-in. pwr. saw blade
Orlando	1001	2012-01-16	Claw hammer
Orlando	1006	2012-01-17	1.25-in. metal screw, 25
Orlando	1006	2012-01-17	B&D jigsaw, 12-in. blade
Orlando	1006	2012-01-17	Claw hammer
Orlando	1006	2012-01-17	Hicut chain saw, 16 in.

ERROR….ERROR…ERROR…ERROR…

Natural Join of the PRODUCT, INVOICE, LINE and CUSTOMER tables using table aliases.

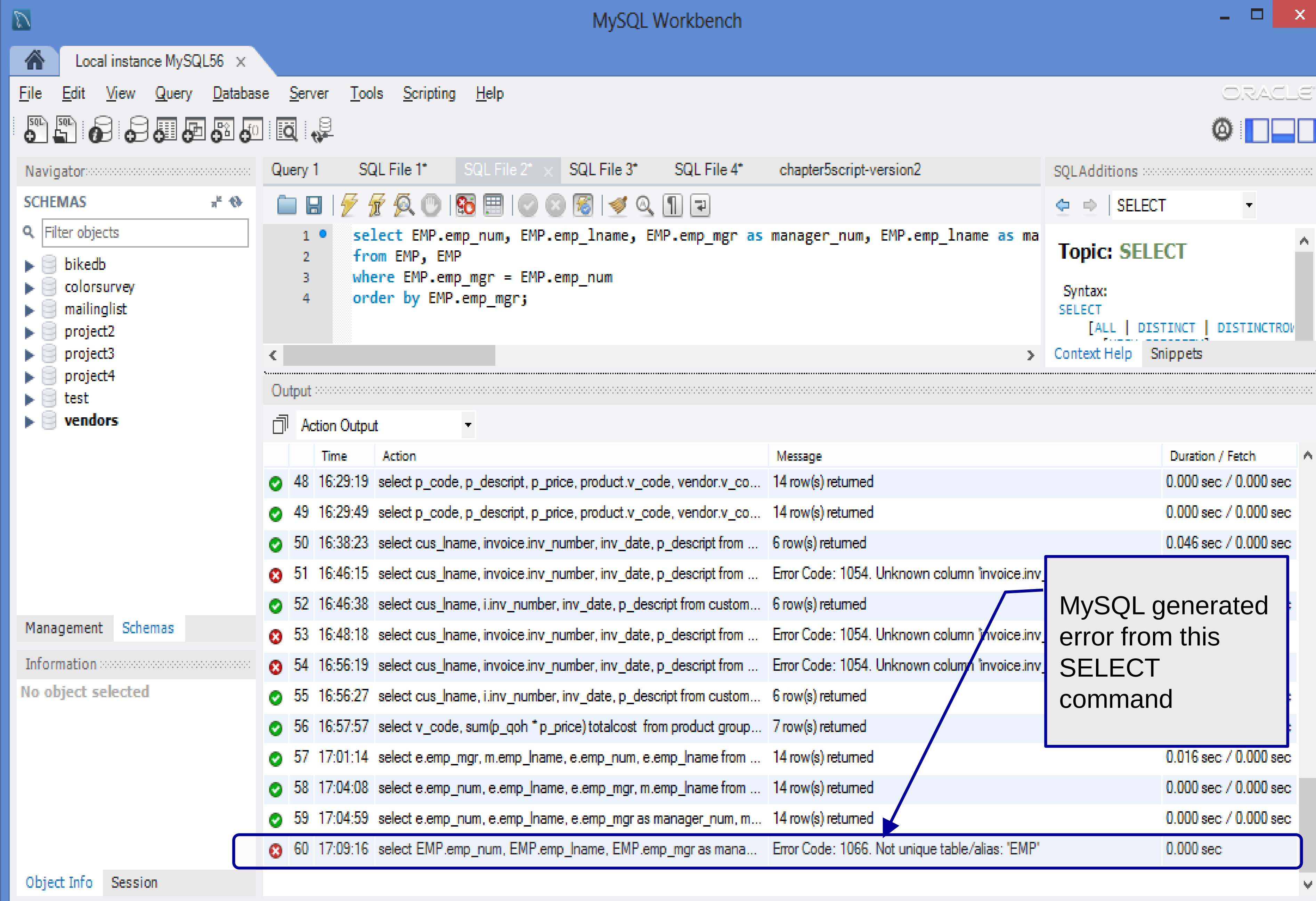


Answer: Since the FROM clause is processed before the SELECT clause the alias is in effect and thus there is no table named INVOICE involved in this SELECT clause.

Question: Why does this SELECT command generate this error?

SQL Join Operators – Recursive Joins

- A table can be joined with itself, a recursive join, and aliases are particularly useful in this case.
- Without an alias being defined at the table level, even fully qualified attribute names would still be ambiguous.
- Suppose that we would like to generate a list of employees with their manager's names. We need to join the EMP table with itself. The next slide illustrates the problem and the following slide illustrates the solution with table aliasing.


MySQL Workbench
Local instance MySQL56
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
bikedb
colorsurvey
mailinglist
project2
project3
project4
test
vendors
Query 1
SQL File 1*
SQL File 2*
SQL File 3*
SQL File 4*
chapter5script-version2
select EMP.emp_num, EMP.emp_lname, EMP.emp_mgr as manager_num, EMP.emp_lname as ma
from EMP, EMP
where EMP.emp_mgr = EMP.emp_num
order by EMP.emp_mgr;
SQL Additions
Topic: SELECT
Syntax:
SELECT
[ALL | DISTINCT | DISTINCTROW
Context Help
Snippets
Output
Action Output
Time
Action
Message
Duration / Fetch
48 16:29:19 select p_code, p_descript, p_price, product.v_code, vendor.v_co... 14 row(s) returned 0.000 sec / 0.000 sec
49 16:29:49 select p_code, p_descript, p_price, product.v_code, vendor.v_co... 14 row(s) returned 0.000 sec / 0.000 sec
50 16:38:23 select cus_lname, invoice.inv_number, inv_date, p_descript from ... 6 row(s) returned 0.046 sec / 0.000 sec
51 16:46:15 select cus_lname, invoice.inv_number, inv_date, p_descript from ... Error Code: 1054. Unknown column 'invoice.inv
52 16:46:38 select cus_lname, i.inv_number, inv_date, p_descript from custom... 6 row(s) returned
53 16:48:18 select cus_lname, invoice.inv_number, inv_date, p_descript from ... Error Code: 1054. Unknown column 'invoice.inv
54 16:56:19 select cus_lname, invoice.inv_number, inv_date, p_descript from ... Error Code: 1054. Unknown column 'invoice.inv
55 16:56:27 select cus_lname, i.inv_number, inv_date, p_descript from custom... 6 row(s) returned
56 16:57:57 select v_code, sum(p_qoh * p_price) totalcost from product group... 7 row(s) returned
57 17:01:14 select e.emp_mgr, m.emp_lname, e.emp_num, e.emp_lname from ... 14 row(s) returned 0.016 sec / 0.000 sec
58 17:04:08 select e.emp_num, e.emp_lname, e.emp_mgr, m.emp_lname from ... 14 row(s) returned 0.000 sec / 0.000 sec
59 17:04:59 select e.emp_num, e.emp_lname, e.emp_mgr as manager_num, m... 14 row(s) returned 0.000 sec / 0.000 sec
60 17:09:16 select EMP.emp_num, EMP.emp_lname, EMP.emp_mgr as mana... Error Code: 1066. Not unique table/alias: 'EMP' 0.000 sec
Management
Schemas
Information
No object selected
Object Info
Session
MySQL generated error from this SELECT command

```sql
select e.emp_num, e.emp_lname, e.emp_mgr as manager_num, m.emp_lname as manager_lname
from emp as e, emp as m
where e.emp_mgr = m.emp_num
order by e.emp_mgr;
```

emp_num	emp_lname	manager_num	manager_lname
101	Lewis	100	Kolmycz
102	Vandam	100	Kolmycz
103	Jones	100	Kolmycz
112	Johnson	100	Kolmycz
111	Washington	105	Williams
115	Saranda	105	Williams
107	Diante	105	Williams
106	Smith	105	Williams
104	Lange	105	Williams
113	Smythe	105	Williams
114	Brandon	108	Wiesenbach
110	Genkazi	108	Wiesenbach
109	Smith	108	Wiesenbach
116	Smith	108	Wiesenbach

Table aliases created for the EMP table. Note that I only actually needed to create one alias in this case.

SQL Join Operations

- In general, relational join operations merge rows from two tables and return the rows with one of the following conditions:
  - Have common values in common columns (natural join).
  - Meet a given join condition (theta-join, or equi-join).
  - Have common values in common columns or have no matching values (outer joins – variants are left, right, and full).
- The join syntax that we've seen so far is sometimes referred to as "old-style" SQL joins.
- Join operations can be classified as inner joins and outer joins. The inner join is the traditional join in which only rows that meet a specified criterion are selected. An outer join returns not only the matching rows but the rows with unmatched attribute values for one or both tables to be joined.
- The table on the next two pages summarizes the joins in SQL.

Join Classification	Join Type	SQL Syntax	Description
Cross	Cross Join	`SELECT * FROM T1, T2;`	Returns the Cartesian product of T1 and T2 (old style)
		`SELECT * FROM T1 CROSS JOIN T2;`	Returns the Cartesian product of T1 and T2 (new style)
Inner	Old-style JOIN	`SELECT * FROM T1, T2 WHERE T1.C1 = T2.C1;`	Returns only the rows that meet the join condition in the WHERE clause (old style); only rows with matching values are selected
	NATURAL JOIN	`SELECT * FROM T1 NATURAL JOIN T2;`	Returns only the rows with matching values in the matching columns; the matching columns must have the same names and similar data types.
	JOIN USING	`SELECT * FROM T1 JOIN T2 USING (C1);`	Returns only the rows with matching values in the columns indicated in the USING clause.
	JOIN ON	`SELECT * FROM T1 JOIN T2 ON T1.C1 = T2.C1;`	Returns only the rows that meet the join condition specified in the ON clause.

SQL Join Expression Styles

Join Classification	Join Type	SQL Syntax	Description

Outer	LEFT JOIN	`SELECT *`
		`FROM T1 LEFT OUTER JOIN T2`
		`ON T1.C1 = T2.C1;`
			Returns rows with matching values and includes all rows from the left table (T1) with unmatched values.

	RIGHT JOIN	`SELECT *`
		`FROM T1 RIGHT OUTER JOIN T2`
		`ON T1.C1 = T2.C1;`
			Returns rows with matching values and includes all rows from the right table (T2) with unmatched values.

	FULL JOIN	`SELECT *`
		`FROM T1 FULL OUTER JOIN T2`
		`ON T1.C1 = T2. C1`
			Returns rows with matching values and includes all rows from both tables (T1 and T2) with unmatched values.

SQL Join Expression Styles (continued)

CROSS JOIN

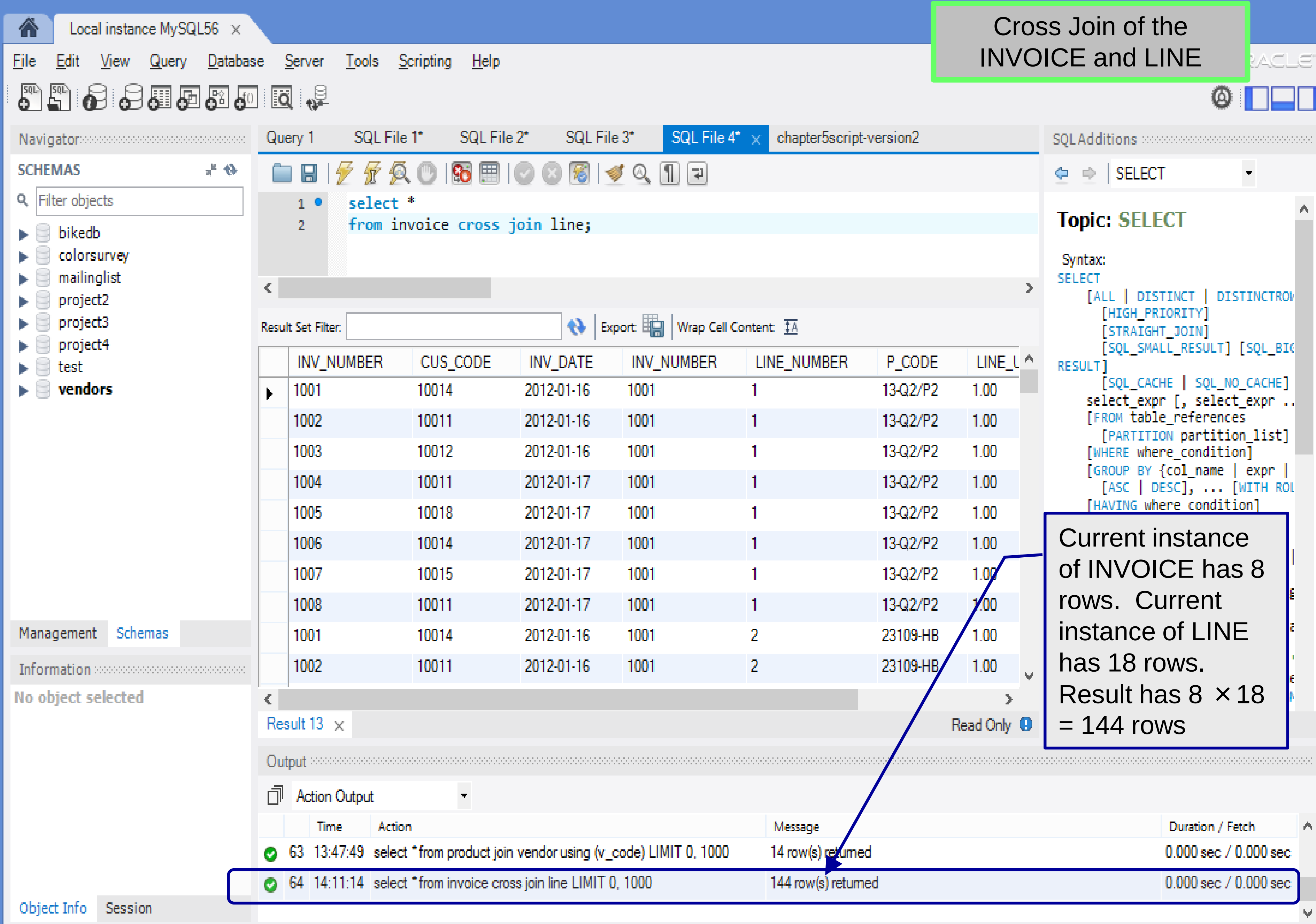
- A cross join performs a relational product (the Cartesian product) of two tables.

- The syntax is:

```
SELECT column-list
FROM table1 CROSS JOIN table2;
```

- The next couple of slides provide examples of the cross join operation in MySQL Workbench.

Cross Join of the INVOICE and LINE



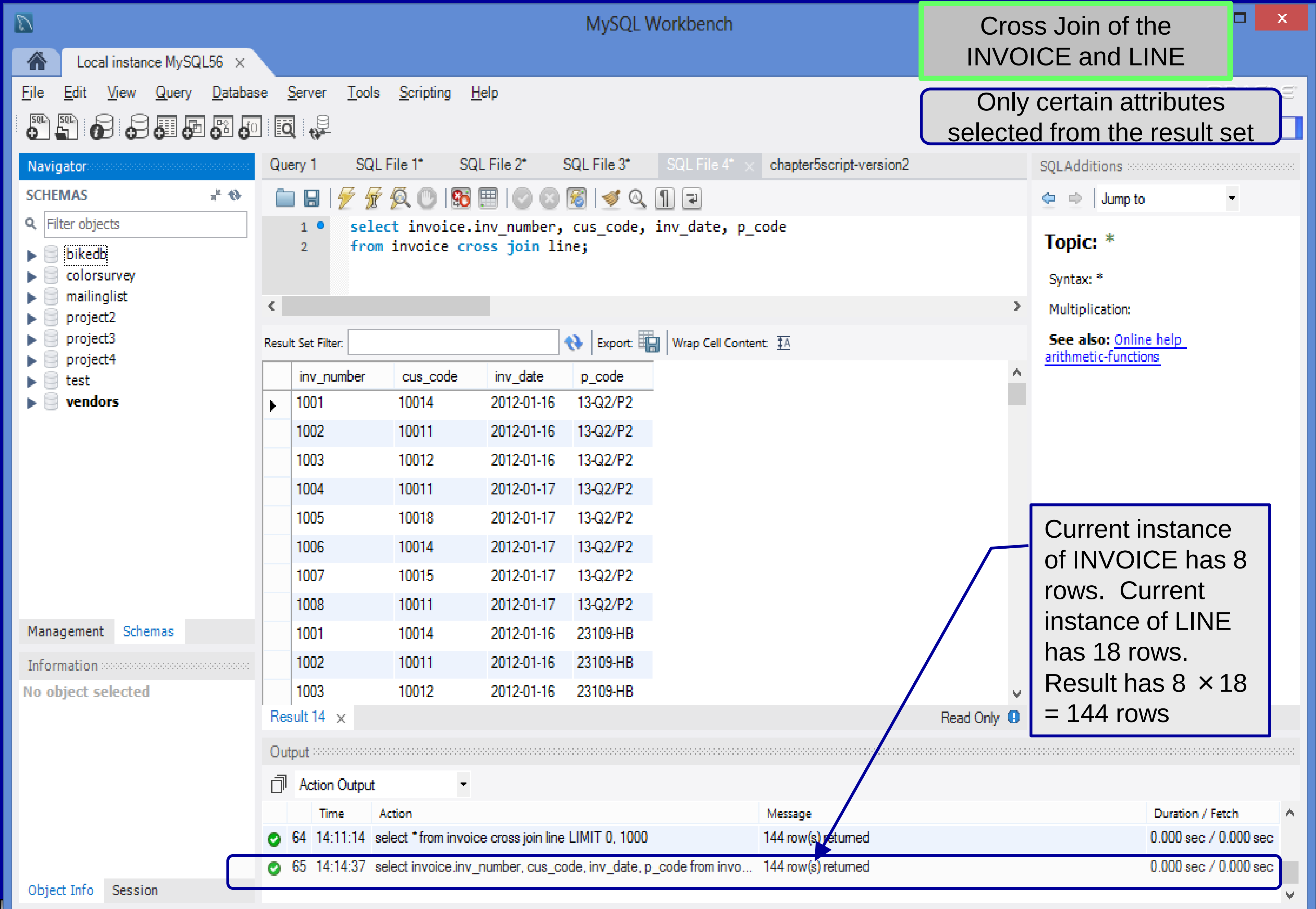
INV_NUMBER	CUS_CODE	INV_DATE	INV_NUMBER	LINE_NUMBER	P_CODE	LINE_U
1001	10014	2012-01-16	1001	1	13-Q2/P2	1.00
1002	10011	2012-01-16	1001	1	13-Q2/P2	1.00
1003	10012	2012-01-16	1001	1	13-Q2/P2	1.00
1004	10011	2012-01-17	1001	1	13-Q2/P2	1.00
1005	10018	2012-01-17	1001	1	13-Q2/P2	1.00
1006	10014	2012-01-17	1001	1	13-Q2/P2	1.00
1007	10015	2012-01-17	1001	1	13-Q2/P2	1.00
1008	10011	2012-01-17	1001	1	13-Q2/P2	1.00
1001	10014	2012-01-16	1001	2	23109-HB	1.00
1002	10011	2012-01-16	1001	2	23109-HB	1.00

Current instance of INVOICE has 8 rows. Current instance of LINE has 18 rows. Result has 8 × 18 = 144 rows

Cross Join of the INVOICE and LINE

Only certain attributes selected from the result set

Current instance of INVOICE has 8 rows. Current instance of LINE has 18 rows. Result has 8 × 18 = 144 rows



NATURAL JOIN

- A cross join performs a relational product (the Cartesian product) of two tables.
- The syntax is:

  SELECT *column-list*

  FROM *table1* NATURAL JOIN *table2*;

- The natural join will perform the following tasks:
  - Determine the common attribute(s) by looking for attributes with common names and compatible data types.
  - Select only the rows with the common values in the common attribute(s).
  - If there are no common attributes, return the Cartesian product of the two tables.
- The next couple of slides provide examples of the cross join operation in MySQL Workbench.

Natural Join of the CUSTOMER and INVOICE tables.

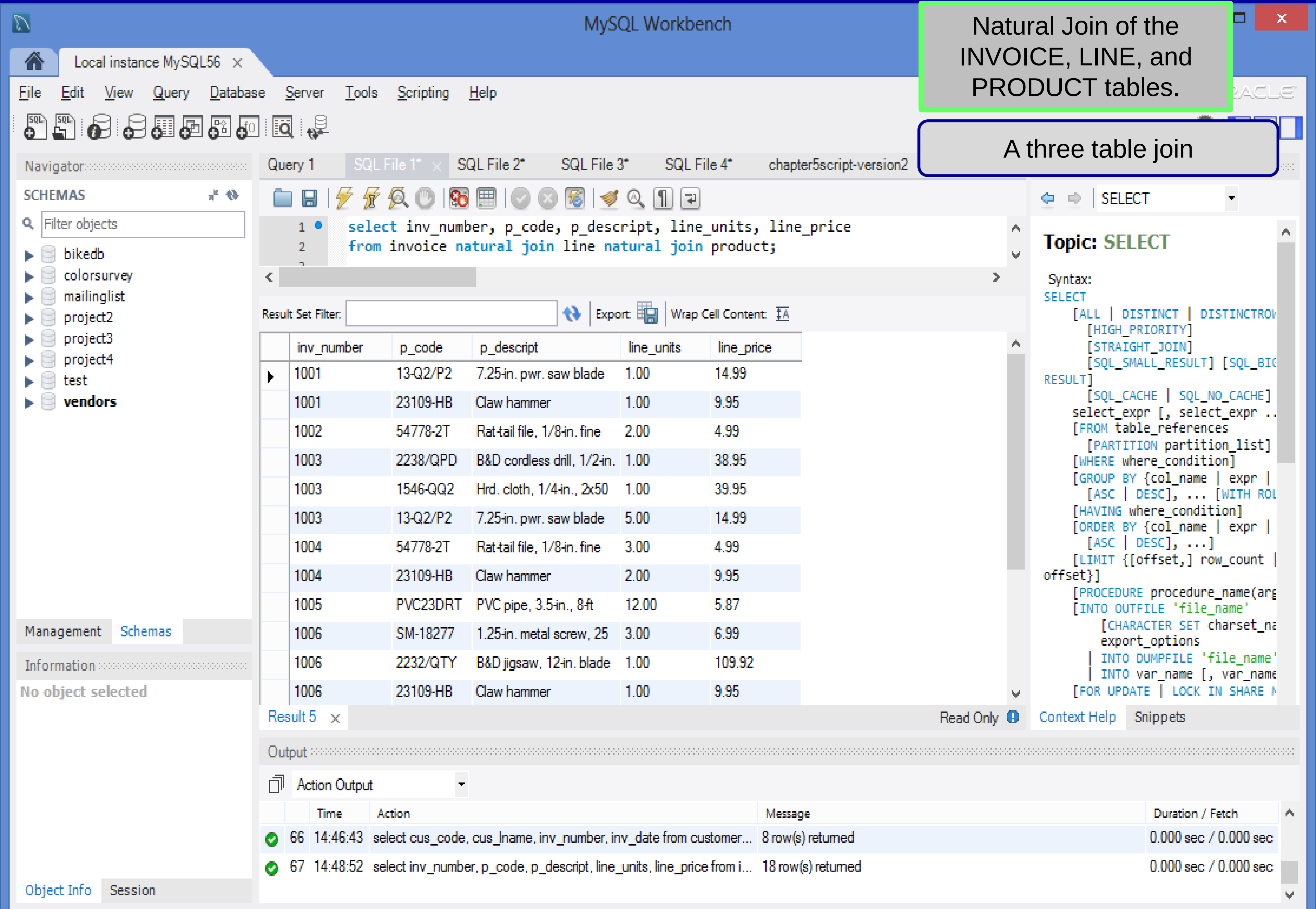
```sql
select cus_code, cus_lname, inv_number, inv_date
from customer natural join invoice;
```

cus_code	cus_lname	inv_number	inv_date
10014	Orlando	1001	2012-01-16
10011	Dunne	1002	2012-01-16
10012	Smith	1003	2012-01-16
10011	Dunne	1004	2012-01-17
10018	Farriss	1005	2012-01-17
10014	Orlando	1006	2012-01-17
10015	O'Brian	1007	2012-01-17
10011	Dunne	1008	2012-01-17

	Time	Action	Message	Duration / Fetch
65	14:14:37	select invoice.inv_number, cus_code, inv_date, p_code from invo...	144 row(s) returned	0.000 sec / 0.000 sec
66	14:46:43	select cus_code, cus_lname, inv_number, inv_date from customer...	8 row(s) returned	0.000 sec / 0.000 sec

Natural Join of the INVOICE, LINE, and PRODUCT tables.

A three table join



inv_number	p_code	p_descript	line_units	line_price
1001	13-Q2/P2	7.25-in. pwr. saw blade	1.00	14.99
1001	23109-HB	Claw hammer	1.00	9.95
1002	54778-2T	Rat-tail file, 1/8-in. fine	2.00	4.99
1003	2238/QPD	B&D cordless drill, 1/2-in.	1.00	38.95
1003	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	1.00	39.95
1003	13-Q2/P2	7.25-in. pwr. saw blade	5.00	14.99
1004	54778-2T	Rat-tail file, 1/8-in. fine	3.00	4.99
1004	23109-HB	Claw hammer	2.00	9.95
1005	PVC23DRT	PVC pipe, 3.5-in., 8-ft	12.00	5.87
1006	SM-18277	1.25-in. metal screw, 25	3.00	6.99
1006	2232/QTY	B&D jigsaw, 12-in. blade	1.00	109.92
1006	23109-HB	Claw hammer	1.00	9.95

JOIN USING Clause

- A second technique for expressing a join is via the USING clause. The query returns only the rows with matching values in the columns indicated in the USING clause – and that column must exist in both tables.
- The syntax is:

```
SELECT column-list
FROM table1 JOIN table2 USING (common-column);
```

- The next slide provide an example of a join operation that includes the USING clause.

Join USING clause of the INVOICE, LINE, and PRODUCT tables.

A three table join with the USING clause

```
select inv_number, p_code, p_descript, line_units, line_price
from invoice join line using (inv_number) join product using (p_code);
```

inv_number	p_code	p_descript	line_units	line_price
1001	13-Q2/P2	7.25-in. pwr. saw blade	1.00	14.99
1001	23109-HB	Claw hammer	1.00	9.95
1002	54778-2T	Rat-tail file, 1/8-in. fine	2.00	4.99
1003	2238/QPD	B&D cordless drill, 1/2-in.	1.00	38.95
1003	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	1.00	39.95
1003	13-Q2/P2	7.25-in. pwr. saw blade	5.00	14.99
1004	54778-2T	Rat-tail file, 1/8-in. fine	3.00	4.99
1004	23109-HB	Claw hammer	2.00	9.95
1005	PVC23DRT	PVC pipe, 3.5-in., 8-ft	12.00	5.87
1006	SM-18277	1.25-in. metal screw, 25	3.00	6.99
1006	2232/QTY	B&D jigsaw, 12-in. blade	1.00	109.92
1006	23109-HB	Claw hammer	1.00	9.95

	Time	Action	Message	Duration / Fetch
67	14:48:52	select inv_number, p_code, p_descript, line_units, line_price from i...	18 row(s) returned	0.000 sec / 0.000 sec
68	14:55:25	select inv_number, p_code, p_descript, line_units, line_price from i...	18 row(s) returned	0.000 sec / 0.000 sec

JOIN ON Clause

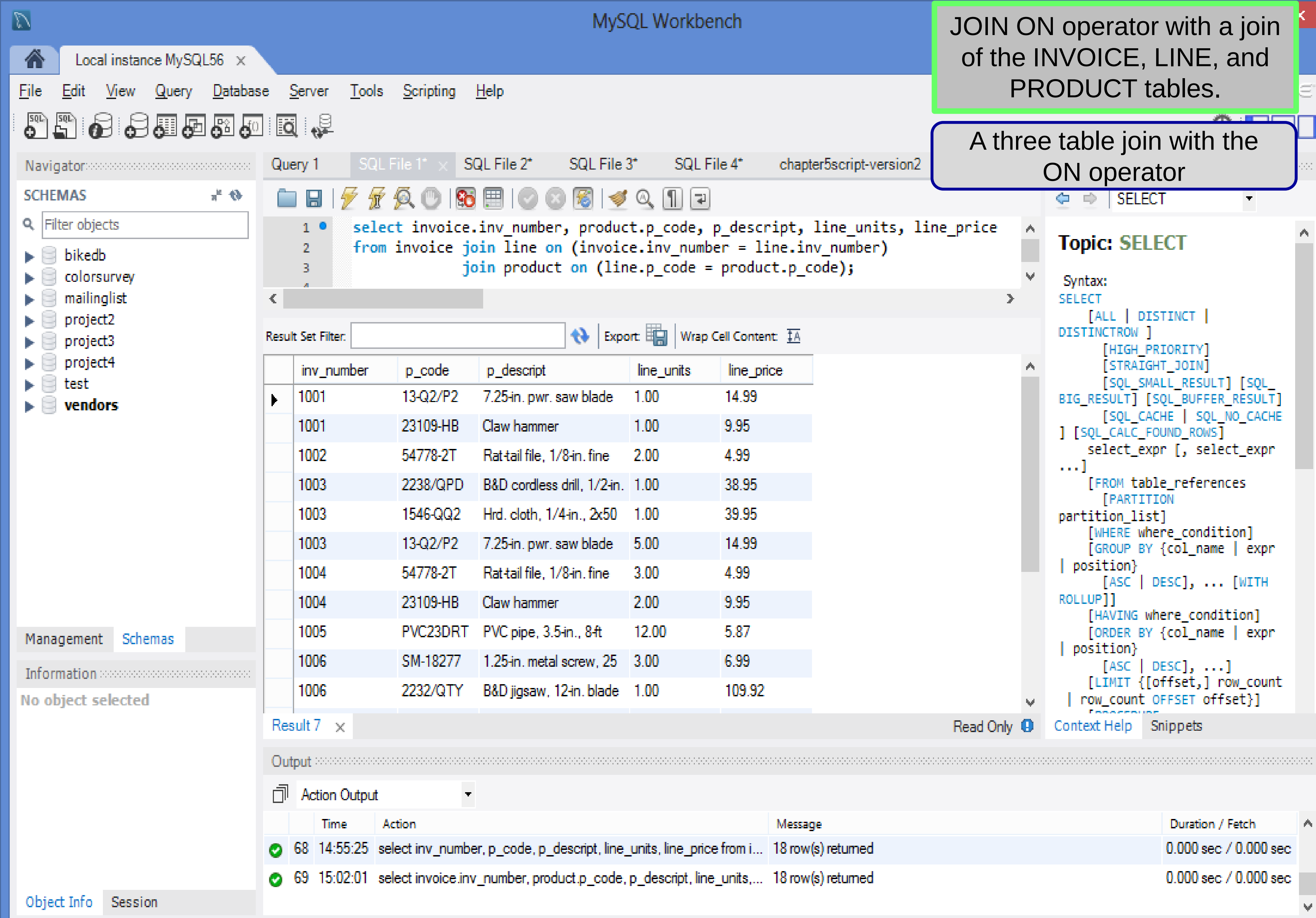
- The natural join and join USING join styles use common attribute names in the joining tables.
- Another way to express a join when the tables have no common attribute names is to use the JOIN ON operator.
- The query will return only the rows that meet the indicated join condition. The join condition will typically include an equality comparison expression of two columns. The two columns may or may not have the same name, but obviously must have comparable data types. The syntax is:

```
SELECT column-list
FROM table1 JOIN table2 ON (join-condition);
```

- The next slide provide an example of a join operation that includes the USING clause.

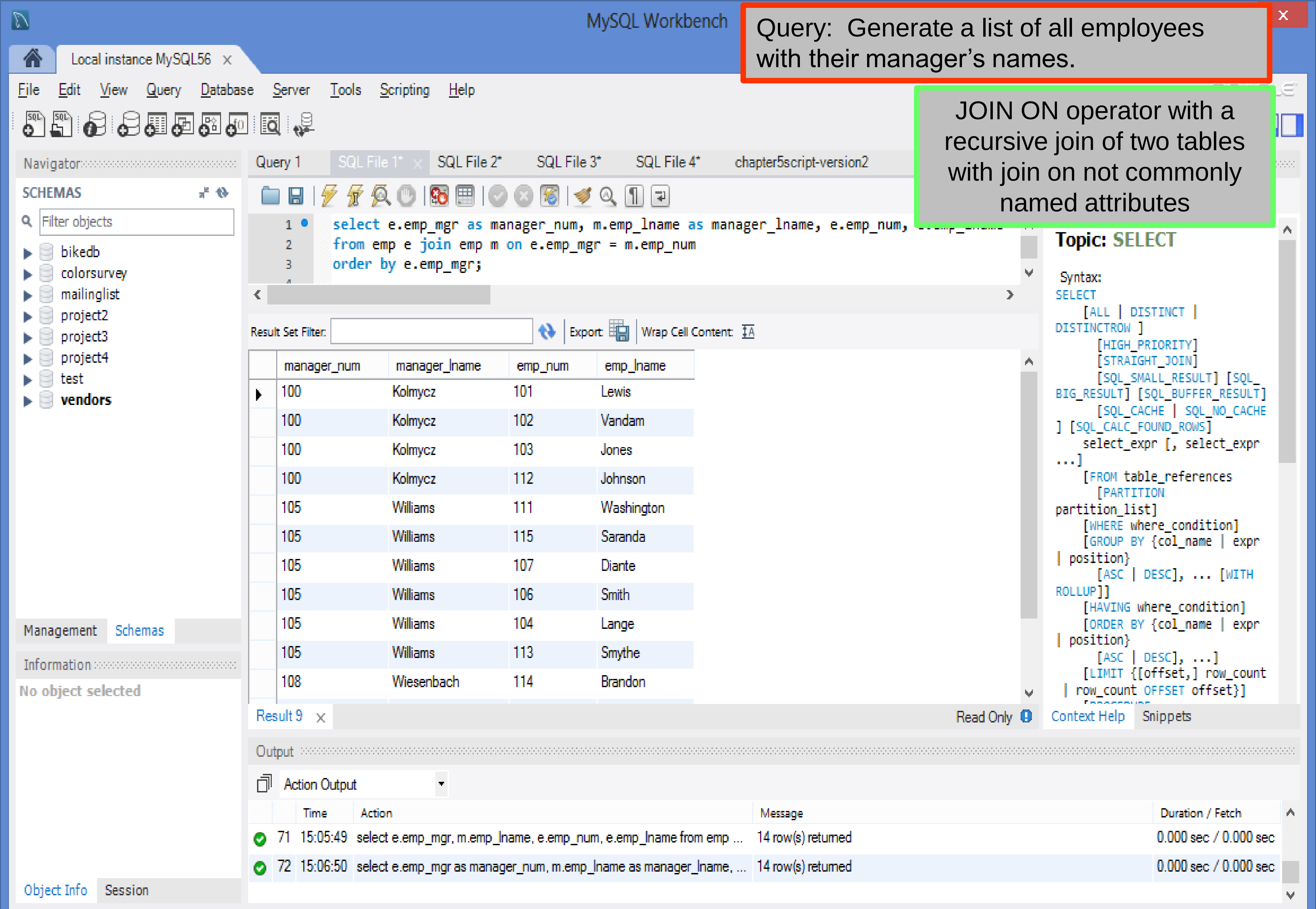
JOIN ON operator with a join of the INVOICE, LINE, and PRODUCT tables.

A three table join with the ON operator



Query: Generate a list of all employees with their manager's names.

JOIN ON operator with a recursive join of two tables with join on not commonly named attributes



Outer Joins

- An outer join returns not only the rows matching the join condition (that is, rows with matching values in the common columns), it also returns the rows with unmatched values.
- The ANSI standard defines three types of outer joins: left, right, and full.
- The left and right designations reflect the order in which the tables are processed by the DBMS.
- Remember that join operations take place two tables at a time. The first table named in the FROM clause will be the left side, and the second table named will be the right side.
- If three or more tables are being joined, the result of joining the first two tables becomes the left side, and the third table becomes the right side.

Left Outer Join

- The left outer join returns not only the rows matching the join condition (rows with matching values in the common column), it also returns rows in the left table with unmatched values in the right side.
- The syntax is:

```
SELECT column-list
FROM table1 LEFT [OUTER] JOIN table2 ON join-condition;
```

- The next slide provide an example of a left outer join operation.

Query: List details of product code, vendor code and vendor name for all products and include those vendors with no matching products.

LEFT OUTER JOIN where some of the vendors are not supplying products.

```sql
select p_code, vendor.v_code, v_name
from vendor left outer join product on vendor.v_code=product.v_code;
```

p_code	v_code	v_name
11QER/31	25595	Rubicon Systems
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
2238/QPD	25595	Rubicon Systems
23109-HB	21225	Bryson, Inc.
54778-2T	21344	Gomez Bros.
89-WRE-Q	24288	ORDVA, Inc.
SM-18277	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
WR3/TT3	25595	Rubicon Systems
NULL	21226	SuperLoo, Inc.
NULL	22567	Dome Supply
NULL	24004	Brackman Bros.
NULL	25443	B&K, Inc.
NULL	25501	Damal Supplies

Note the null values in the p_code (attribute of the right table) indicating that there are vendors who are currently not supplying any products.

In other words, these are the vendors without any matching products.

Query: List details of product code, vendor code and vendor name for all products and include those vendors with no matching products.

LEFT OUTER JOIN where some of the vendors are not supplying products.

```sql
select p_code, vendor.v_code, v_name
from vendor left join product on vendor.v_code=product.v_code;
```

Illustrates that use of keyword OUTER is optional

p_code	v_code	v_name
11QER/31	25595	Rubicon Systems
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
2238/QPD	25595	Rubicon Systems
23109-HB	21225	Bryson, Inc.
54778-2T	21344	Gomez Bros.
89-WRE-Q	24288	ORDVA, Inc.
SM-18277	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
WR3/TT3	25595	Rubicon Systems
NULL	21226	SuperLoo, Inc.
NULL	22567	Dome Supply
NULL	24004	Brackman Bros.
NULL	25443	B&K, Inc.
NULL	25501	Damal Supplies

Right Outer Join

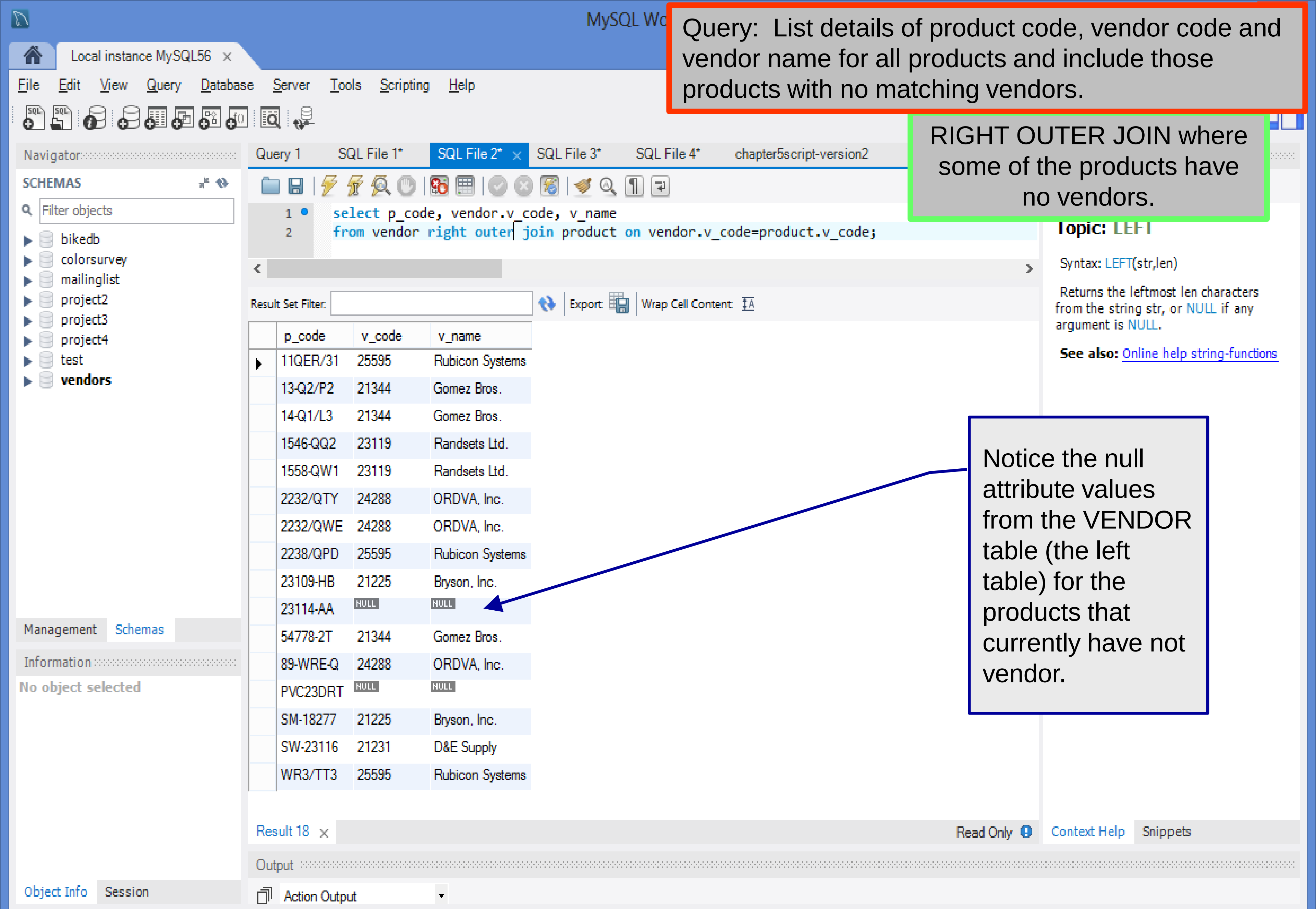
- The right outer join returns not only the rows matching the join condition (rows with matching values in the common column), it also returns rows in the right table with unmatched values in the left side.
- The syntax is:

```
SELECT column-list
FROM table1 RIGHT [OUTER] JOIN table2 ON join-condition;
```

- The next slide provide an example of a right outer join operation.

Query: List details of product code, vendor code and vendor name for all products and include those products with no matching vendors.

RIGHT OUTER JOIN where some of the products have no vendors.



Notice the null attribute values from the VENDOR table (the left table) for the products that currently have not vendor.

Full Outer Join

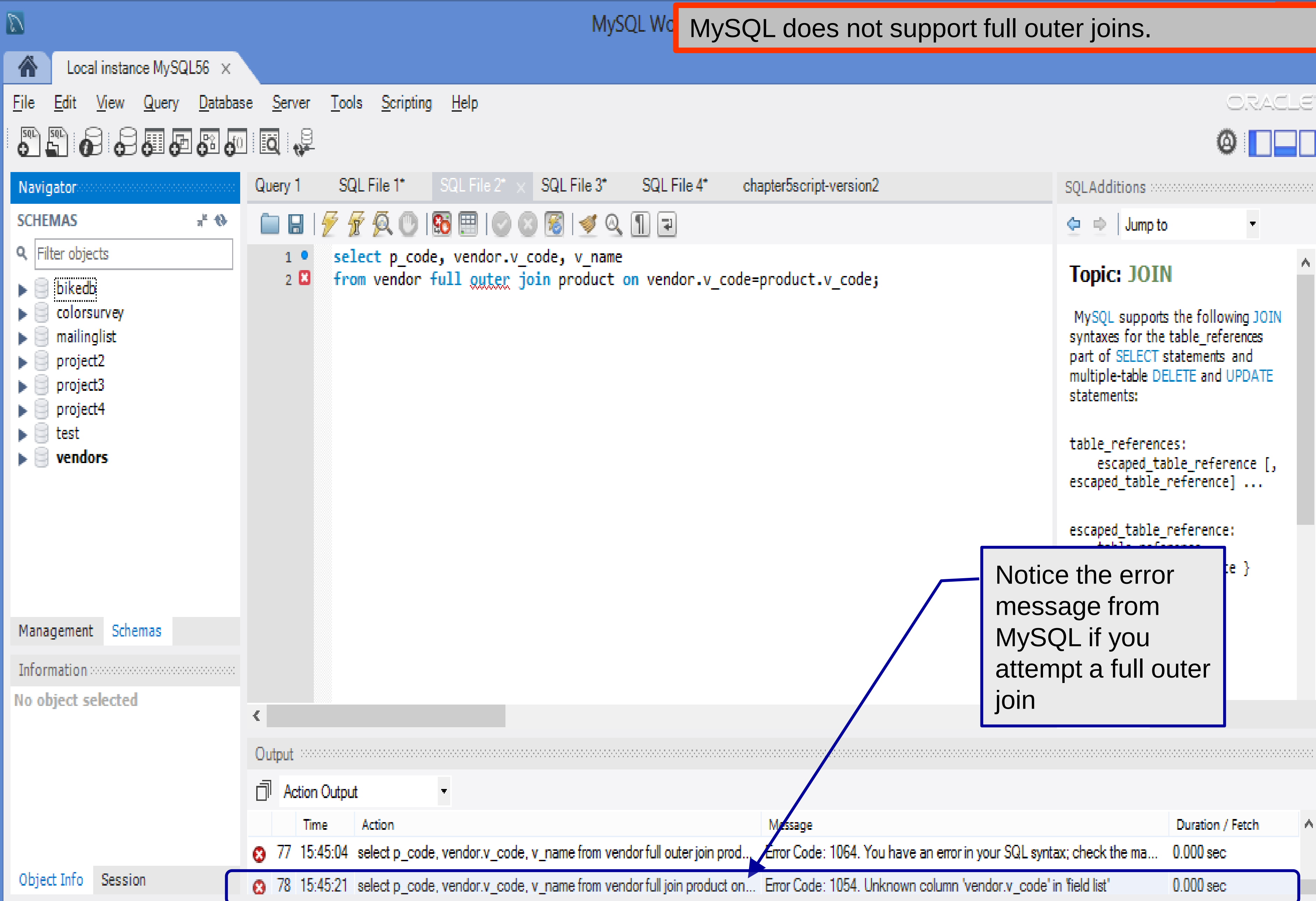
- The full outer join returns not only the rows matching the join condition (rows with matching values in the common column), it also returns rows in both the left and right tables with unmatched values in either side.
- The syntax is:

```
SELECT column-list
FROM table1 FULL [OUTER] JOIN table2 ON join-condition;
```

- The next slide provide an example of a full outer join operation.

NOTE: MySQL does not support full outer join. It can be simulated as shown on page 40.

MySQL does not support full outer joins.



Notice the error message from MySQL if you attempt a full outer join

Full Outer Join in MySQL

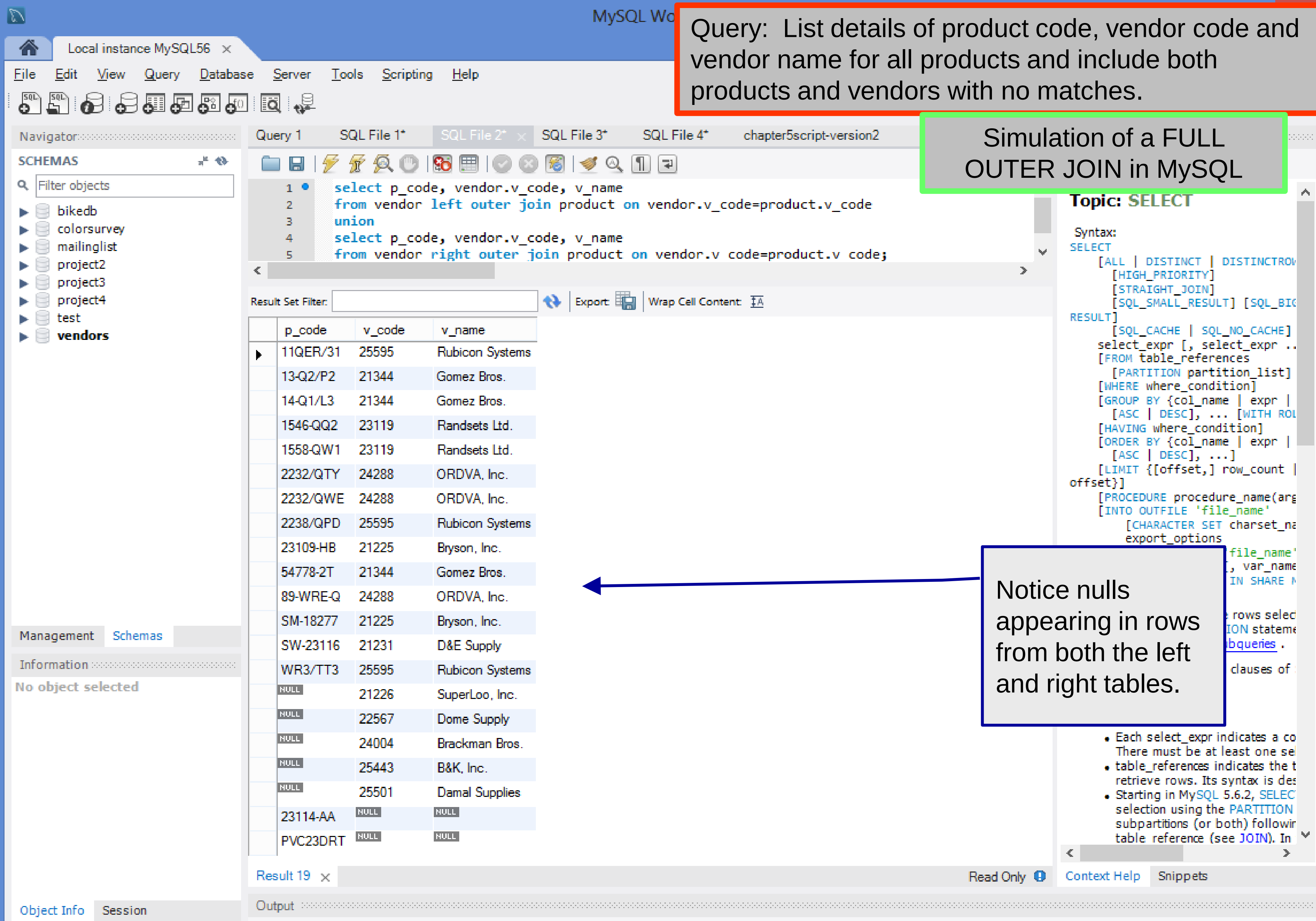
- To simulate a full outer join in MySQL use the following syntax:

```
SELECT column-list
FROM table1 LEFT [OUTER] JOIN table2 ON join-condition
UNION
SELECT column-list
FROM table1 RIGHT [OUTER] JOIN table2 ON join-condition;
```

- The next slide provide an example of a full outer join operation in MySQL.

Query: List details of product code, vendor code and vendor name for all products and include both products and vendors with no matches.

Simulation of a FULL OUTER JOIN in MySQL



```
select p_code, vendor.v_code, v_name
from vendor left outer join product on vendor.v_code=product.v_code
union
select p_code, vendor.v_code, v_name
from vendor right outer join product on vendor.v_code=product.v_code;
```

p_code	v_code	v_name
11QER/31	25595	Rubicon Systems
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
2238/QPD	25595	Rubicon Systems
23109-HB	21225	Bryson, Inc.
54778-2T	21344	Gomez Bros.
89-WRE-Q	24288	ORDVA, Inc.
SM-18277	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
WR3/TT3	25595	Rubicon Systems
NULL	21226	SuperLoo, Inc.
NULL	22567	Dome Supply
NULL	24004	Brackman Bros.
NULL	25443	B&K, Inc.
NULL	25501	Damal Supplies
23114-AA	NULL	NULL
PVC23DRT	NULL	NULL

Notice nulls appearing in rows from both the left and right tables.

Subqueries and Correlated Subqueries

- The use of joins in a relational database allows you to get information from two or more tables.
- However, it is often necessary to process data based on other processed data.
- For example, suppose that you want to generate a list of vendors who do not provide any products:

```
SELECT v_code, v_name
FROM vendor
WHERE v_code NOT IN (SELECT v_code
                     FROM product);
```

- In order to list the vendor information in the outer query you needed information that was not previously known. A subquery is used to generate the necessary information.

Subqueries and Correlated Subqueries

- The basic characteristics of subqueries are:
  - A subquery is a query (SELECT statement) inside a query.
  - A subquery is normally expressed inside parentheses.
  - The first query in the SELECT statement is referred to as the outer query.
  - The query inside the SELECT statement is referred to as the inner query.
  - The inner query is executed first.
  - The output of an inner query is used as the input to the outer query.
  - The entire SELECT statement is referred to as a nested query.
- A subquery can actually appear in DML statements such as INSERT, UPDATE, and DELETE. We'll hold off looking at these types of subqueries until later and for now focus on subqueries inside the SELECT statement only.

WHERE Clause Subqueries

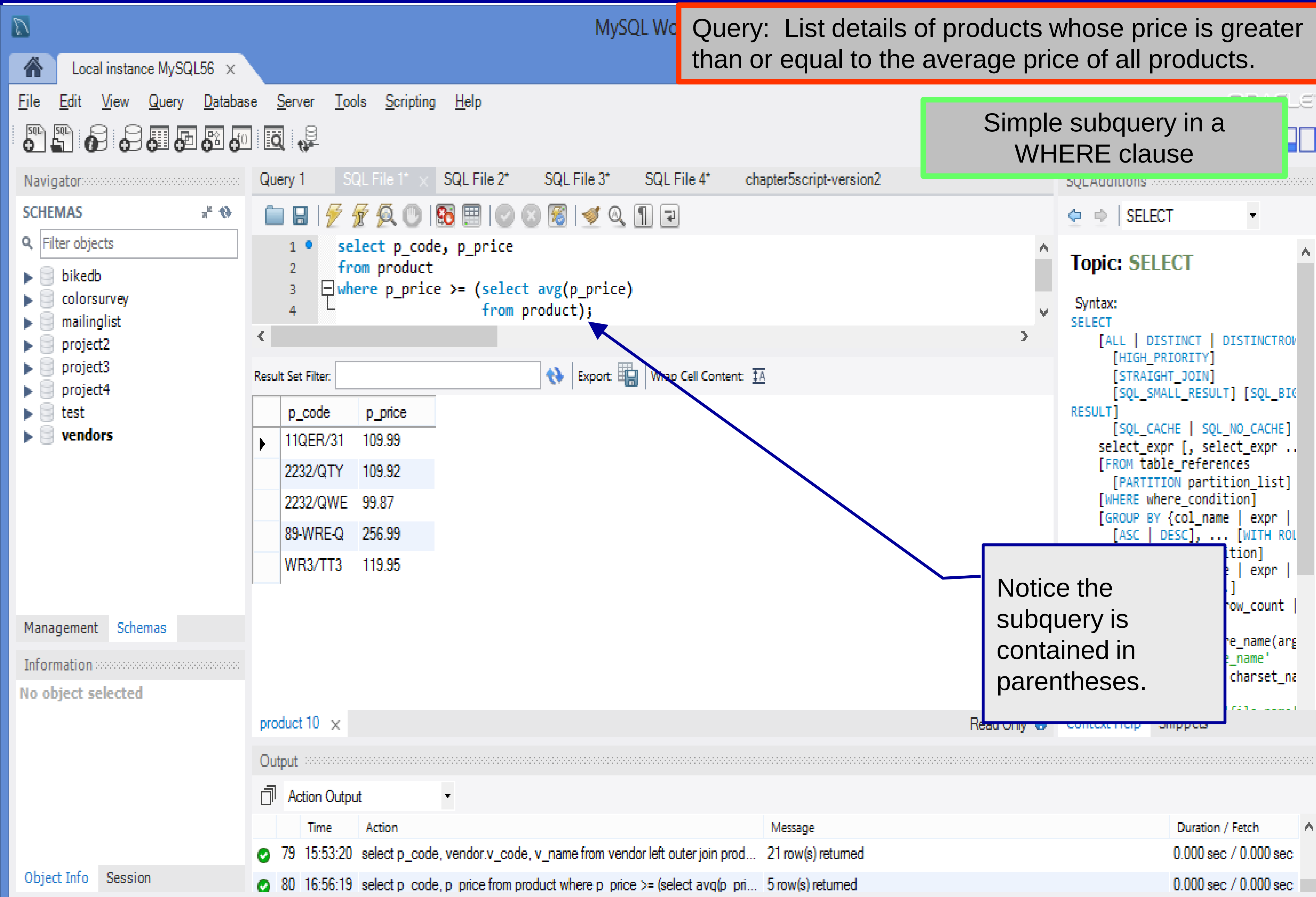
- The most common type of subquery uses an inner SELECT subquery on the right side of a WHERE comparison expression.
- For example, to find all products with a price greater than or equal to the average product price, you would construct the following query expression:

```
SELECT p-code, p_price
FROM product
WHERE p_price >= (SELECT AVG(p_price)
                  FROM product);
```

- Note that this type of subquery, when used in a >, <. =, >=, or <= conditional expression, requires that a subquery that returns only one value (one column, one row). The value generated by the subquery must be of a compatible data type.

Query: List details of products whose price is greater than or equal to the average price of all products.

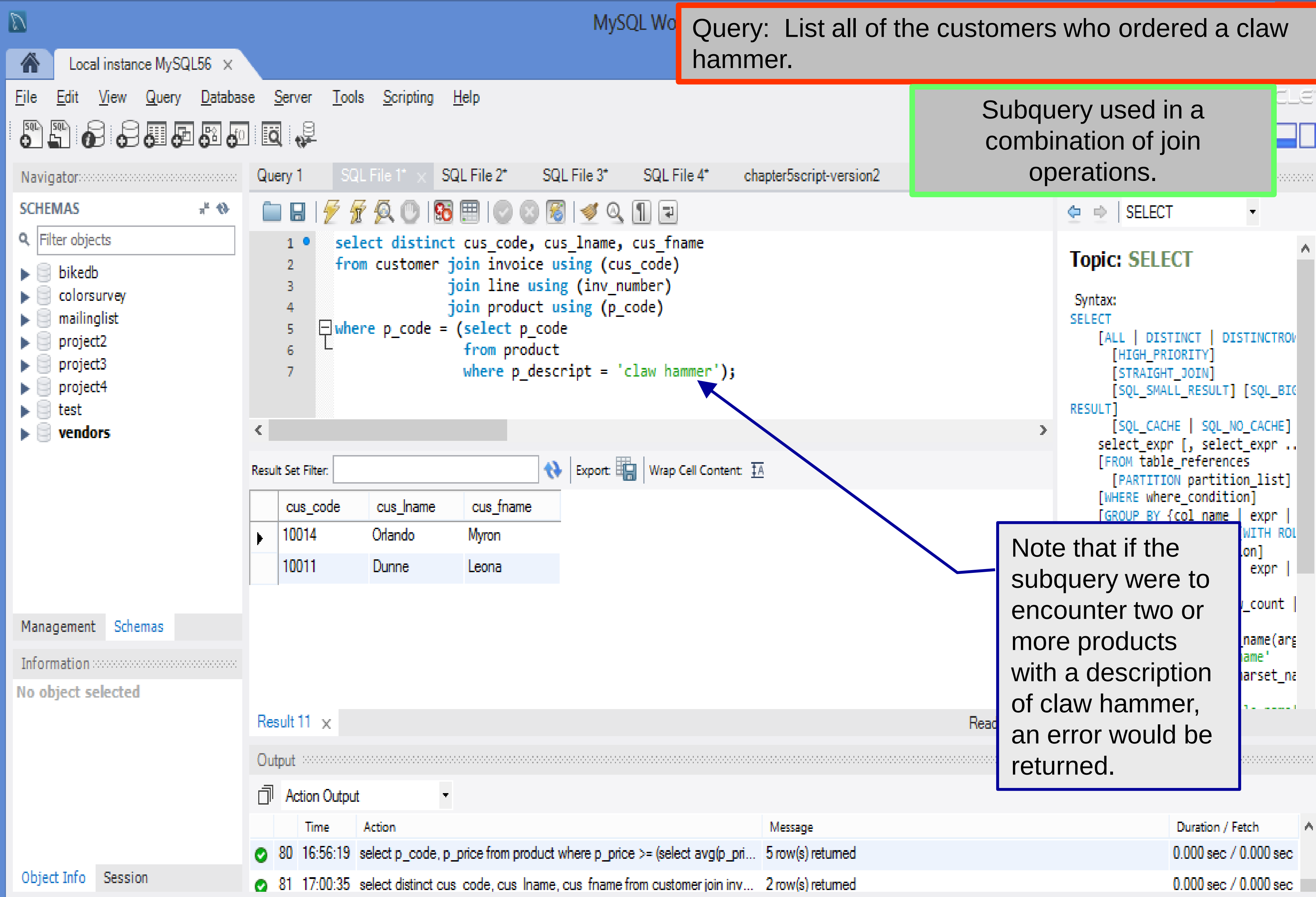
Simple subquery in a WHERE clause



Notice the subquery is contained in parentheses.

Query: List all of the customers who ordered a claw hammer.

Subquery used in a combination of join operations.




```sql
select distinct cus_code, cus_lname, cus_fname
from customer join invoice using (cus_code)
              join line using (inv_number)
              join product using (p_code)
where p_code = (select p_code
                from product
                where p_descript = 'claw hammer');
```

cus_code	cus_lname	cus_fname
10014	Orlando	Myron
10011	Dunne	Leona

	Time	Action	Message	Duration / Fetch
80	16:56:19	select p_code, p_price from product where p_price >= (select avg(p_pri...	5 row(s) returned	0.000 sec / 0.000 sec
81	17:00:35	select distinct cus_code, cus_lname, cus_fname from customer join inv...	2 row(s) returned	0.000 sec / 0.000 sec

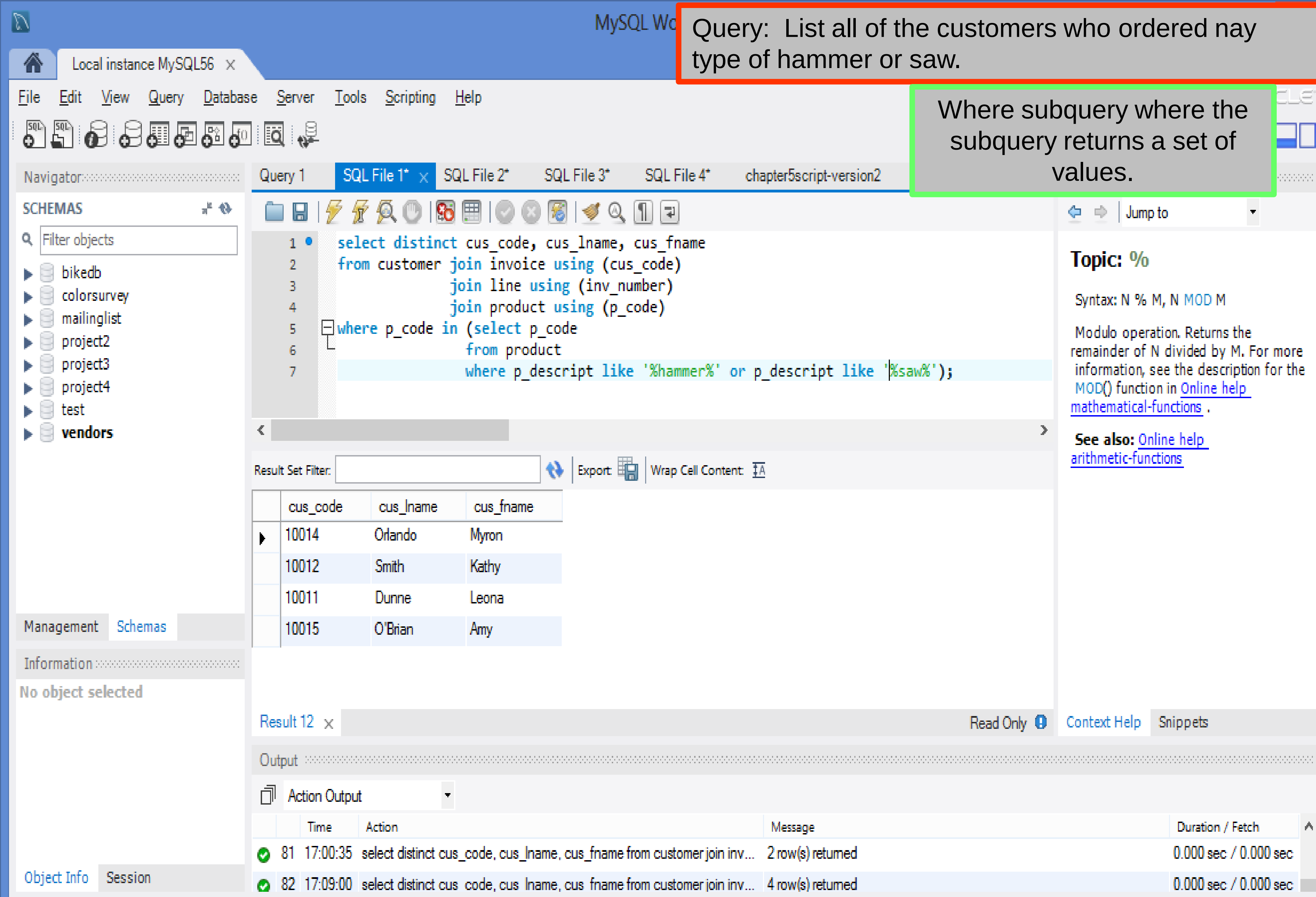
Note that if the subquery were to encounter two or more products with a description of claw hammer, an error would be returned.

WHERE Clause Subqueries

- In the previous example, if the subquery had found more than one p_code corresponding to a claw hammer, the DBMS would have generated an error due to the = condition on p_code.
- If the inner query might generate more than one value the IN operator must be used. In this fashion the subquery is assumed to generate a set of values and the comparison operator needs only to check for set membership.
- The next slide illustrates this type of subquery.

Query: List all of the customers who ordered nay type of hammer or saw.

Where subquery where the subquery returns a set of values.



HAVING Clause Subqueries

- Subqueries can also be included inside the HAVING clause of a GROUP BY clause. Recall that the HAVING clause cannot stand alone and must appear only in the presence of a GROUP B clause.
- Recall that the HAVING clause is used to restrict the output of a GROUP BY clause by applying conditional criteria to the grouped rows.
- The query on the following page lists all of the products with a total quantity sold greater than the average quantity sold.

Query: List all of the products with a total quantity sold greater than the average quantity sold.

HAVING clause subquery

The average quantity sold in this database is currently 2.55

```
select p_code, sum(line_units)
from line
group by p_code
having sum(line_units) > (select avg(line_units)
                          from line);
```

p_code	sum(line_units)
13-Q2/P2	8.00
23109-HB	5.00
54778-2T	6.00
PVC23DRT	17.00
SM-18277	3.00
WR3/TT3	3.00

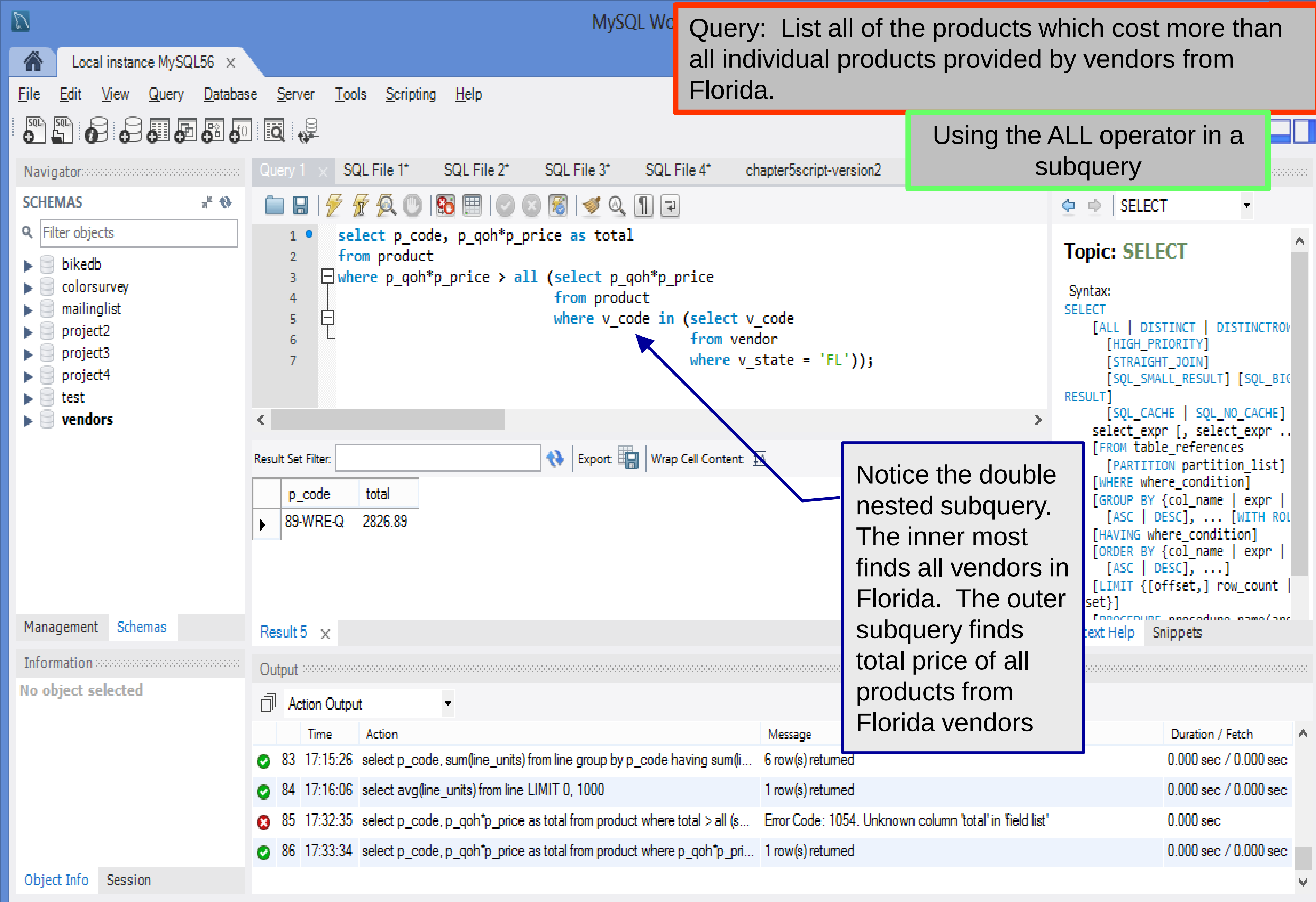
	Time	Action	Message	Duration / Fetch
83	17:15:26	select p_code, sum(line_units) from line group by p_code having sum(lin...	6 row(s) returned	0.000 sec / 0.000 sec
84	17:16:06	select avg(line_units) from line LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Multirow Subquery Operators

- The IN subquery operator allows you to check for set inclusion, but it uses an equality operation; that is, it selects only those rows that are equal to at least one of the values in the set.
- What happens if you need to make an inequality comparison (< or >) of one value to a list of values?
- SQL provides two operators for these cases: ANY and ALL.
- The use of the ALL operator allows you to compare a single value with a list of values returned by the inner query using a comparison operator other than equals.
- The ANY operator allows you to compare a single value to a list of values and select only the rows for which it is greater or less than any value in the list.
- The next couple of slides illustrate both operators.

Query: List all of the products which cost more than all individual products provided by vendors from Florida.

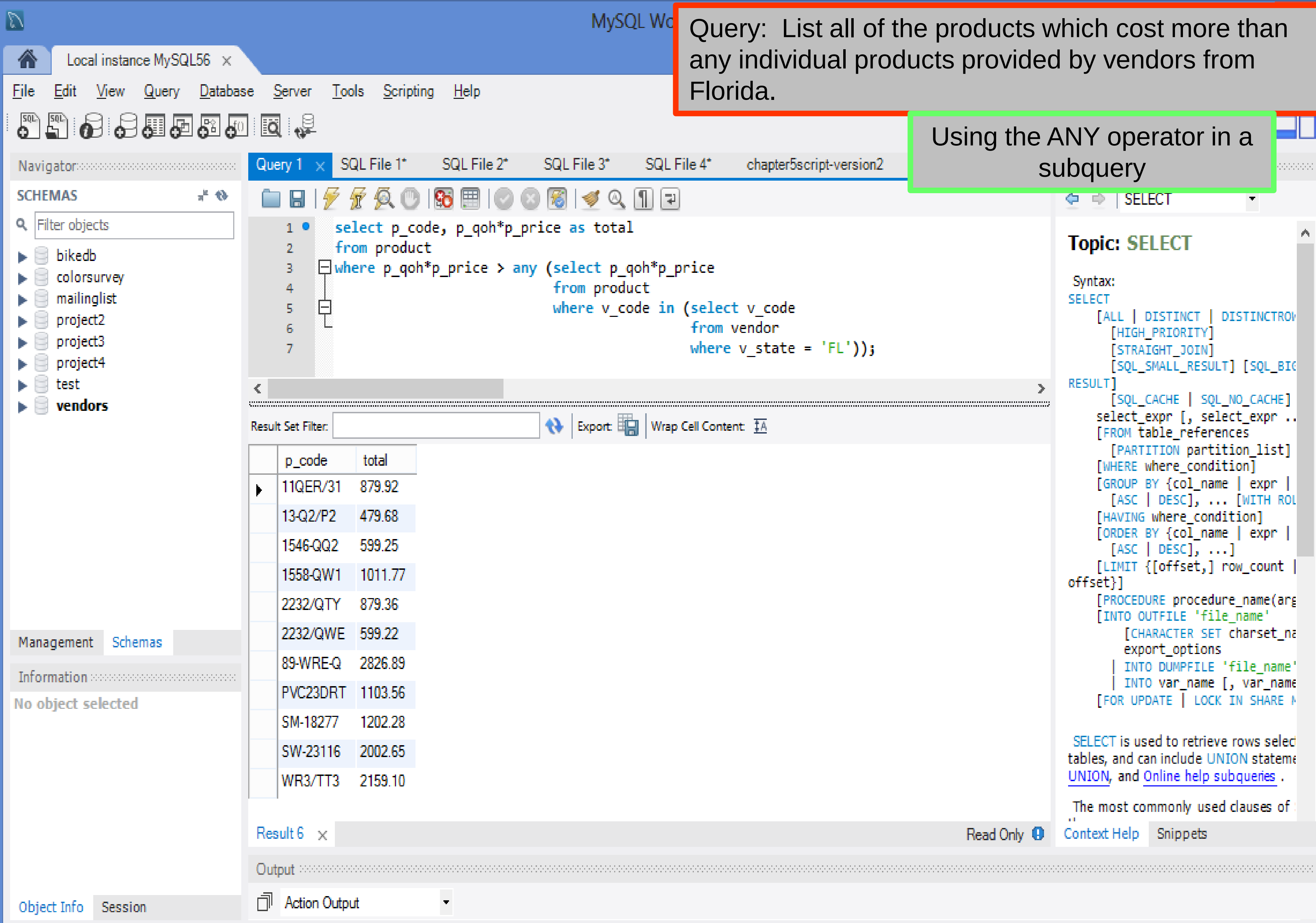
Using the ALL operator in a subquery



Notice the double nested subquery. The inner most finds all vendors in Florida. The outer subquery finds total price of all products from Florida vendors

Query: List all of the products which cost more than any individual products provided by vendors from Florida.

Using the ANY operator in a subquery



FROM Clause Subqueries

- So far we seen how the SELECT statement uses subqueries in the WHERE, HAVING, and IN statements along with the ANY and ALL operators for multirow subqueries. In all of those cases, the subquery was part of a conditional expression, and it always appeared on the right hand side of the expression.

- The FROM clause specifies the table(s) from which the data will be drawn in a SELECT statement. Because the output of a SELECT statement is another table (or more precisely, a "virtual" table), you can use a SELECT subquery in the FROM clause.

FROM Clause Subqueries

- Consider the following case:
  - You want to know all the customer who have purchased products 13-Q2/P2 and 23109-HB.
  - All product purchases are stored in the LINE table, so you can determine who purchased any product by searching the P_CODE attribute in the LINE table.
  - In this case, however, you want to know all customers who purchased both products, not just one.
  - The next page illustrates how this query can be answered using a subquery in the FROM clause.

Query: List customer details for customers who have purchased both product '13-Q2/P2' and '23109-HB'.

Using a FROM clause subquery

```sql
select distinct customer.cus_code, customer.cus_lname
from customer,
     (select invoice.cus_code
       from invoice natural join line
       where p_code = '13-Q2/P2') as cp1,
     (select invoice.cus_code
       from invoice natural join line
       where p_code = '23109-HB') as cp2
where customer.cus_code = cp1.cus_code
     and customer.cus_code = cp2.cus_code;
```

cus_code	cus_lname
10014	Orlando

Note that since the old style join was used, explicit join conditions are included in the WHERE clause.

Query: Same query as previous page.

Using a FROM clause subquery

```
select distinct customer.cus_code, customer.cus_lname
from customer natural join
    (select invoice.cus_code
      from invoice natural join line
      where p_code = '13-Q2/P2') as cp1  natural join
    (select invoice.cus_code
      from invoice natural join line
      where p_code = '23109-HB') as cp2;
```

cus_code	cus_lname
10014	Orlando

Note that new style natural join was used in this case eliminating the need for the WHERE clause on the outer selection. However, notice that the “virtual” tables are required to have aliases even though they are not explicitly referenced in this case.

Attribute List Subqueries

- The SELECT statement uses the attribute list to indicate what columns to project in the resulting set.
- The columns in the attribute list can be attributes of base tables, computed attributes, or the result of an aggregate function, as we've already seen.
- The attribute list can also include a subquery expression, which is also referred to as an inline query.
- An inline query must return one value; otherwise, an error is generated.
- An example of a inline query is shown on the next page.

Query: For each product show the difference in its price compared to the average price of all products.

An inline query
(A subquery in the attribute list)

```sql
select p_code, p_price,
       (select avg(p_price) from product) as average_price,
        p_price - (select avg(p_price) from product) as difference
from product;
```

This query contains two inline subqueries plus a computed value (difference).

p_code	p_price	average_price	difference
11QER/31	109.99	56.421250	53.568750
13-Q2/P2	14.99	56.421250	-41.431250
14-Q1/L3	17.49	56.421250	-38.931250
1546-QQ2	39.95	56.421250	-16.471250
1558-QW1	43.99	56.421250	-12.431250
2232/QTY	109.92	56.421250	53.498750
2232/QWE	99.87	56.421250	43.448750
2238/QPD	38.95	56.421250	-17.471250
23109-HB	9.95	56.421250	-46.471250
23114-AA	14.40	56.421250	-42.021250
54778-2T	4.99	56.421250	-51.431250
89-WRE-Q	256.99	56.421250	200.568750
PVC23DRT	5.87	56.421250	-50.551250
SM-18277	6.99	56.421250	-49.431250
SW-23116	8.45	56.421250	-47.971250
WR3/TT3	119.95	56.421250	63.528750

Query: Same query as previous page.

An inline query
(A subquery in the attribute list)

```sql
select p_code, p_price,
       (select avg(p_price) from product) as average_price,
        p_price - average_price as difference
from product;
```

Since the average price is an alias defined inside the same attribute list, it cannot be used in this expression. Notice how the complete expression was used in this position on the previous page.

	Time	Action	Message	Duration / Fetch
89	14:30:36	select e.emp_num, e.emp_lname, e.emp_mgr as manager_num, m.emp...	14 row(s) returned	0.000 sec / 0.000 sec
90	11:51:20	select distinct customer.cus_code, customer.l_name from customer, ...	Error Code: 1054. Unknown column 'customer.l_name' in 'field list'	0.015 sec
91	11:51:52	select distinct customer.cus_code, customer.cus_l_name from custom...	Error Code: 1054. Unknown column 'customer.cus_l_name' in 'field list'	0.000 sec
92	11:52:08	select distinct customer.cus_code, customer.cus_lname from customer...	1 row(s) returned	0.000 sec / 0.000 sec
93	11:55:16	select distinct customer.cus_code, customer.cus_lname from customer...	1 row(s) returned	0.000 sec / 0.000 sec
94	11:56:06	select distinct customer.cus_code, customer.cus_lname from customer...	Error Code: 1248. Every derived table must have its own alias	0.000 sec
95	11:56:49	select distinct customer.cus_code, customer.cus_lname from customer...	1 row(s) returned	0.000 sec / 0.000 sec
96	12:07:18	select p_code, p_price, (select avg(p_price) from product) as aver...	16 row(s) returned	0.000 sec / 0.000 sec
97	12:12:59	select p_code, p_price, (select avg(p_price) from product) as aver...	Error Code: 1054. Unknown column 'average_price' in 'field list'	0.000 sec

Attribute List Subqueries

- One more example illustrating the use of attribute subqueries and column aliases.
  - Suppose that you want to know the product code, the total sales by product, and contribution of each employee of each product’s sales.
  - To get the sales by product, you need to use only the LINE table.
  - To compute the contribution by each employee, you need to know the number of employees (from the EMPLOYEE table). If you look at the table schemas, you’ll notice that EMPLOYEE and LINE do not share a common attribute. In fact, you do not need a common attribute. You only need to know the total number of employees, no the total employees related to each product.
  - The answer to this query is shown on the next page.

Query: For each product, list the total sales and the contribution per employee to the total sales.

An inline query
(A subquery in the attribute list)

```
select p_code, sum(line_units * line_price) as total_sales,
       (select count(*) from employee) as emp_count,
       sum(line_units * line_price)/(select count(*) from employee) as contribution
from line
group by p_code;
```

p_code	total_sales	emp_count	contribution
13-Q2/P2	119.9200	17	7.05411765
1546-QQ2	39.9500	17	2.35000000
2232/QTY	109.9200	17	6.46588235
2238/QPD	38.9500	17	2.29117647
23109-HB	49.7500	17	2.92647059
54778-2T	29.9400	17	1.76117647
89-WRE-Q	256.9900	17	15.11705882
PVC23DRT	99.7900	17	5.87000000
SM-18277	20.9700	17	1.23352941
WR3/TT3	359.8500	17	21.16764706

As you can see, the number of employees remains the same for each row in the result set. The use of this type of subquery is limited to certain instanced when you need to include data from other tables that are not directly related to the main table or tables in the query. The value will remain the same for each row, like a constant in a programming language.
Note again, that you cannot use an alias in the attribute list to write the expression that computes the contribution per employee.

Query: For each product, list the total sales and the contribution per employee to the total sales.

An alternative way to do the same query using FROM clause subqueries.

```
select p_code, total_sales, emp_count, total_sales/emp_count as contribution
from (select p_code, sum(line_units * line_price) as total_sales,
            (select count(*) from employee) as emp_count
       from line
    group by p_code)as virtual_table;
```

Every “virtual” table requires an alias.

p_code	total_sales	emp_count	contribution
13-Q2/P2	119.9200	17	7.05411765
1546-QQ2	39.9500	17	2.35000000
2232/QTY	109.9200	17	6.46588235
2238/QPD	38.9500	17	2.29117647
23109-HB	49.7500	17	2.92647059
54778-2T	29.9400	17	1.76117647
89-WRE-Q	256.9900	17	15.11705882
PVC23DRT	99.7900	17	5.87000000
SM-18277	20.9700	17	1.23352941
WR3/TT3	359.8500	17	21.16764706

Correlated Subqueries

- Up to this point, every subquery that we've seen executed independently. That is, each subquery in a command sequence executed in serial fashion, one after another. Nested queries of this type are referred to as a non-correlated query.
  - The inner subquery executed first; its output was used by the outer query, when then executes until the last outer query finishes (the first SQL statement in the code).
- In contrast, a correlated query is a subquery that executes once for each row in the outer query. The process is similar to a nested loop in a programming language (see below).

```
for x = 1 to 2
   for y = 1 to 3
      print "X = ", X, " Y = ", Y
   end
end
```

A nested loop

produces

```
X = 1   Y = 1
X = 1   Y = 2
X = 1   Y = 3
X = 2   Y = 1
X = 2   Y = 2
X = 2   Y = 3
```

The output

Correlated Subqueries

- A correlated subquery is processed in the following fashion:
  - The outer query is initiated.
  - For each row of the outer query result set, the inner query is executed by passing the outer row to the inner query.
- The process is exactly the opposite of a non-correlated query in which the inner query is completely processed before any rows of the outer query result set are generated.
- A correlated query is called such, because the inner query is related to the outer query; the inner query references a column of the outer subquery.
- The following page illustrates how a correlated query is processed.

Correlated Subqueries

- Suppose that you want to know all product sales in which the units sold value is greater than the average units sold value for that product (as opposed to the average for all products). In other words, for every sale of a product, you only want to list those products where a specific sale is for more units of that product than the average number of units sold in all sales for that product.

- To process this query you need to do the following:

  - Compute the average units sold for a given product.

  - Compare the average computed in step 1 to the units sold in each sale row, and then select only those rows in which the number of units sold is greater than that average.

- The following page illustrates the SQL correlated query expression that correctly answers this query.

Query: For each product list the details where that product is sold in a quantity greater than the average number of units sold for that product.

A correlated subquery

```
select inv_number, p_code, line_units
from line as line1
where line1.line_units > (select avg(line_units)
                          from line as line2
                          where line2.p_code = line1.p_code);
```

inv_number	p_code	line_units
1003	13-Q2/P2	5.00
1004	54778-2T	3.00
1004	23109-HB	2.00
1005	PVC23DRT	12.00

The p_code from the outer query is used to select the product currently being examined. Since line1 refers to the table in the outer query, this is a correlated query.

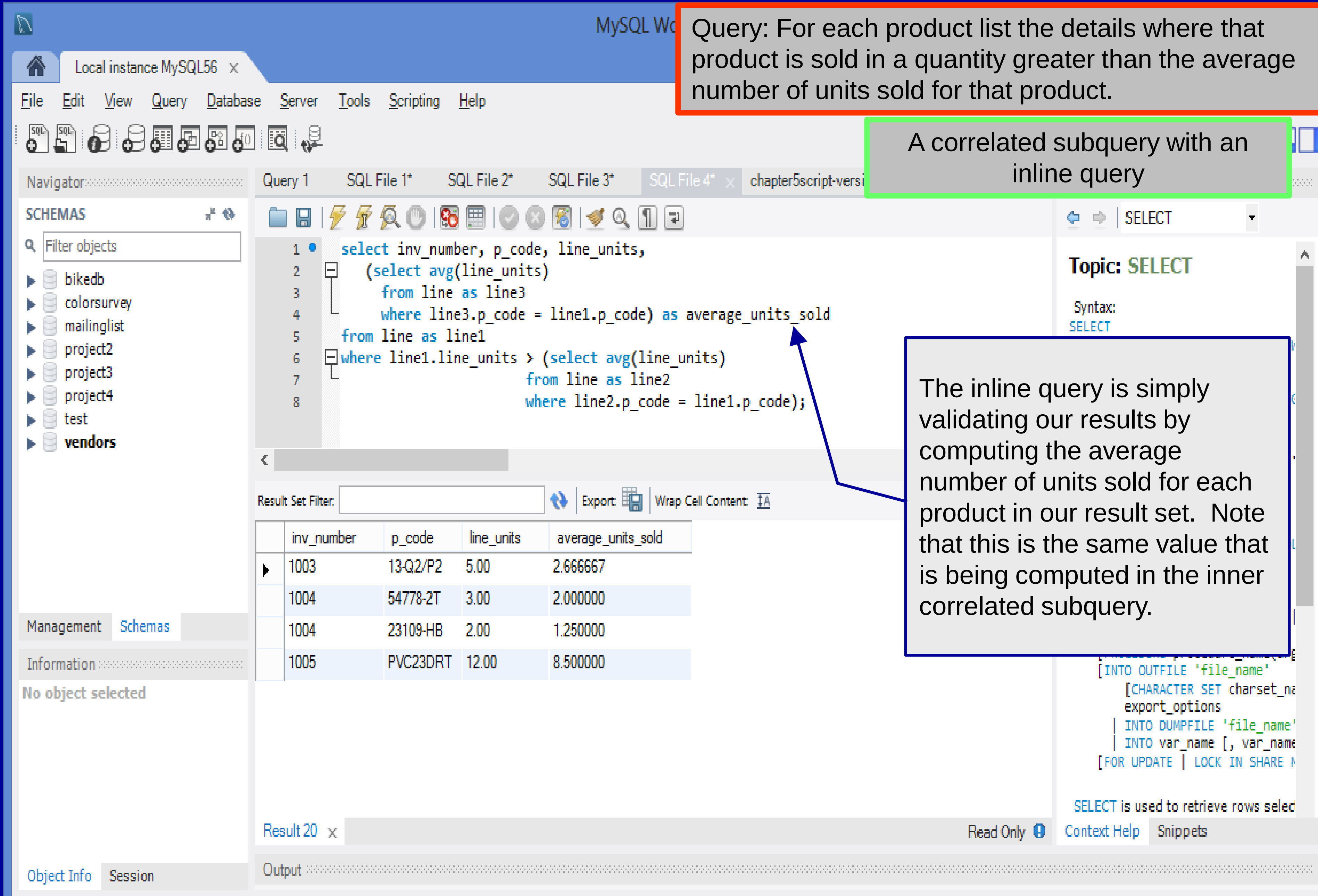
The inner query computes the average units sold of the product that matches the p_code of the outer query p_code. Thus, the inner query executes once, using the first product code found in the outer LINE table, and returns the average number of units sold for that product. When the number of units sold in the outer LINE row is greater than the average computed, the row is added to the output.

Correlated Subqueries

- To further illustrate the use of subqueries, the next example shows how subquery results can be combined.
- How do you know that the result set in the previous query is correct? In other words, how do you know that those products that were returned along with the units sold were for sales that were greater than the average sold for that produt?
- One way would be to run another query that computed the average number of units sold for each product and then compare that result to those products in the result set of the previous query.
- Instead of this approach, let's take the approach of writing a single query that produces both results for us. We'll do this by combining an inline query that produces the average number of units sold for the product in addition to the original query, so we get not only our answer, but also verification.
- This is shown on the next slide.

Query: For each product list the details where that product is sold in a quantity greater than the average number of units sold for that product.

A correlated subquery with an inline query



The inline query is simply validating our results by computing the average number of units sold for each product in our result set. Note that this is the same value that is being computed in the inner correlated subquery.

Correlated Subqueries

- Correlated queries can also be used with the EXISTS operator.
- For example, suppose that you want to know the names of all customers who have placed an order after January 1, 2012.
- In this case, a correlated query works quite nicely, as shown on the next page.

Query: List the customer details for customers who have placed an order after January 1, 2012.

A correlated subquery using the EXISTS operator.

```sql
select cus_code, cus_lname, cus_fname
from customer
where exists (select cus_code
              from invoice
              where invoice.cus_code = customer.cus_code and
                    invoice.inv_date > '2012-01-01');
```

cus_code	cus_lname	cus_fname
10011	Dunne	Leona
10018	Farriss	Anne
10015	O'Brian	Amy
10014	Orlando	Myron
10012	Smith	Kathy

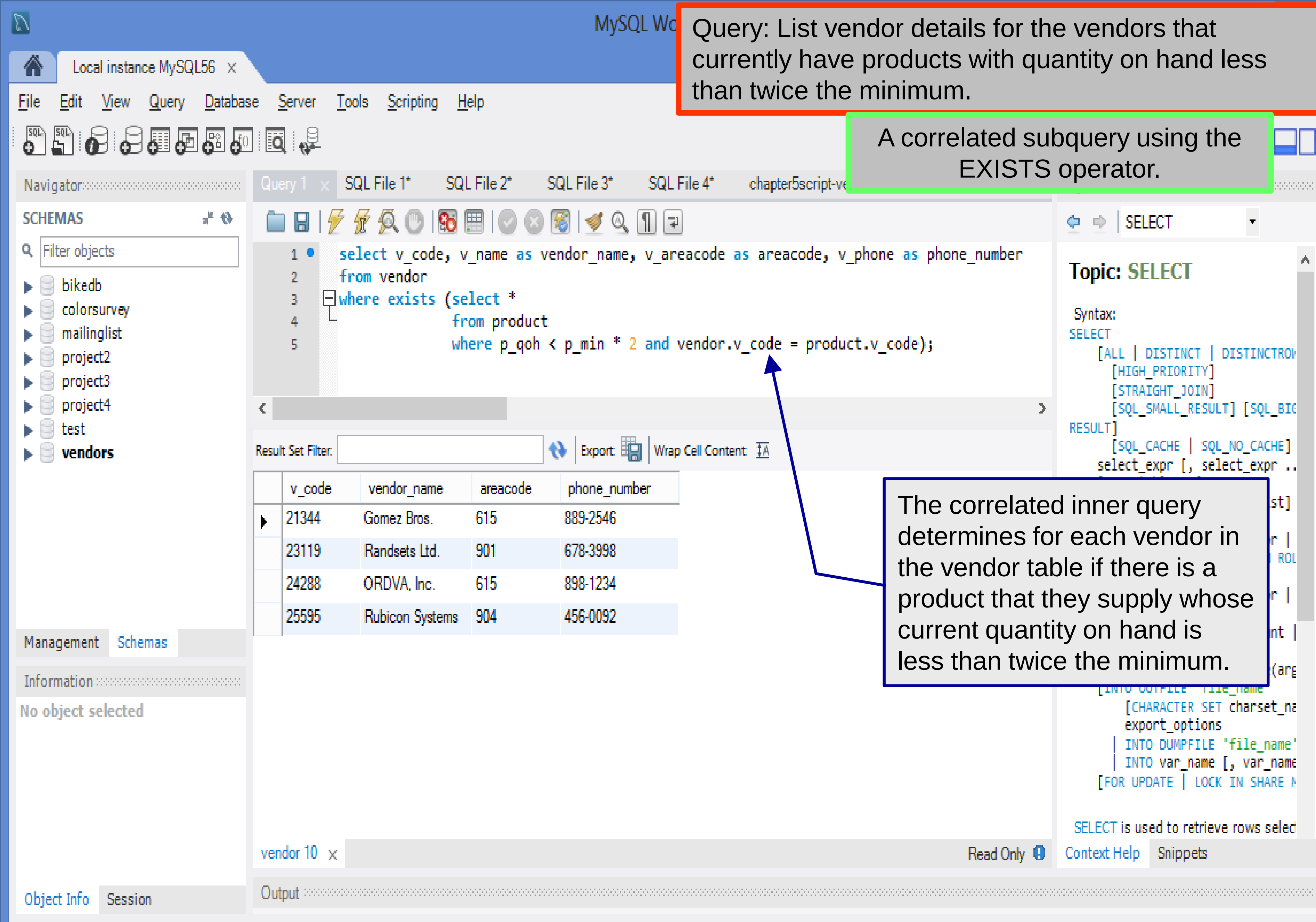
The correlated inner query determines for each customer in the customer table if there is an invoice belonging to that customer with an invoice date after January 1, 2012.

Correlated Subqueries

- Another correlated query example.
- Suppose that you want to know what vendors you need to contact to order products that are approaching the minimum quantity on hand value. In particular, you want to know the vendor code, vendor name, and vendor telephone number for products with a quantity on hand that is less than double the minimum quantity.
- The solution is shown on the next page. Note how the inner correlated subquery runs using the first vendor. If any products match the condition (quantity on hand is less than double the minimum quantity), the vendor information is listed in the output. The correlated subquery then runs the next vendor and the process repeats until all vendors have been examined.

Query: List vendor details for the vendors that currently have products with quantity on hand less than twice the minimum.

A correlated subquery using the EXISTS operator.



The correlated inner query determines for each vendor in the vendor table if there is a product that they supply whose current quantity on hand is less than twice the minimum.

Relational Set Operations In SQL

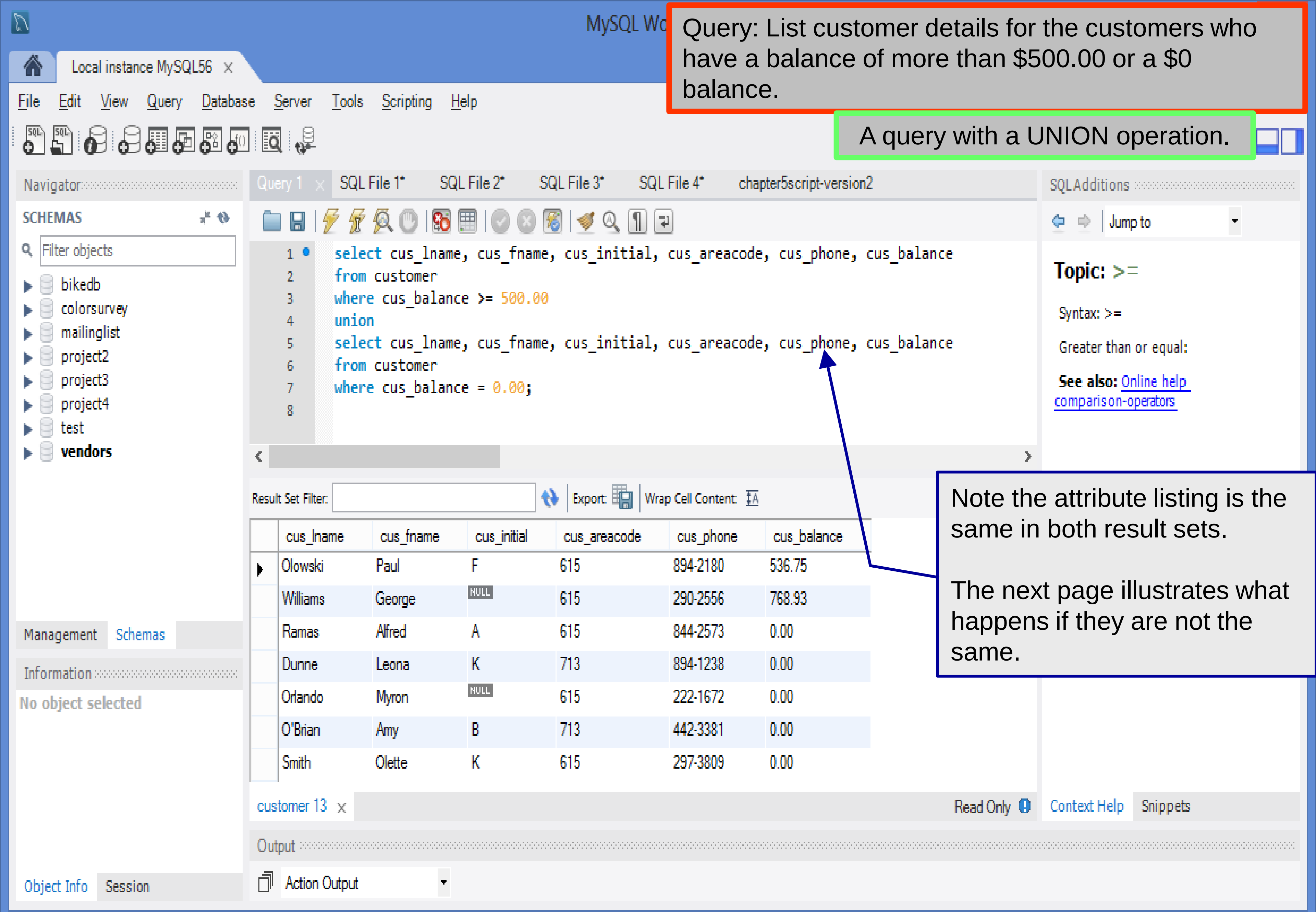
- SQL data manipulation commands are set-oriented; they operate over entire sets of rows and columns (tables) at once.
- ANSI standard SQL supports the UNION, INTERSECT, and MINUS operations, which operate exactly as their relational algebra counterparts.
- Recall that these operators require union compatible sets in order for the operation to be defined. Some DBMSs will require identical data types in a one to one correspondence of attributes, while other DBMSs will simply require compatible data types in a one to one correspondence to ensure union compatibility.
- The following pages illustrate how MySQL implements these operations.

Relational Set Operations In SQL

- SQL data manipulation commands are set-oriented; they operate over entire sets of rows and columns (tables) at once.
- ANSI standard SQL supports the UNION, INTERSECT, and MINUS operations, which operate exactly as their relational algebra counterparts.
- Recall that these operators require union compatible sets in order for the operation to be defined. Some DBMSs will require identical data types in a one to one correspondence of attributes, while other DBMSs will simply require compatible data types in a one to one correspondence to ensure union compatibility.
- The following pages illustrate how MySQL implements these operations.

Query: List customer details for the customers who have a balance of more than $500.00 or a $0 balance.

A query with a UNION operation.

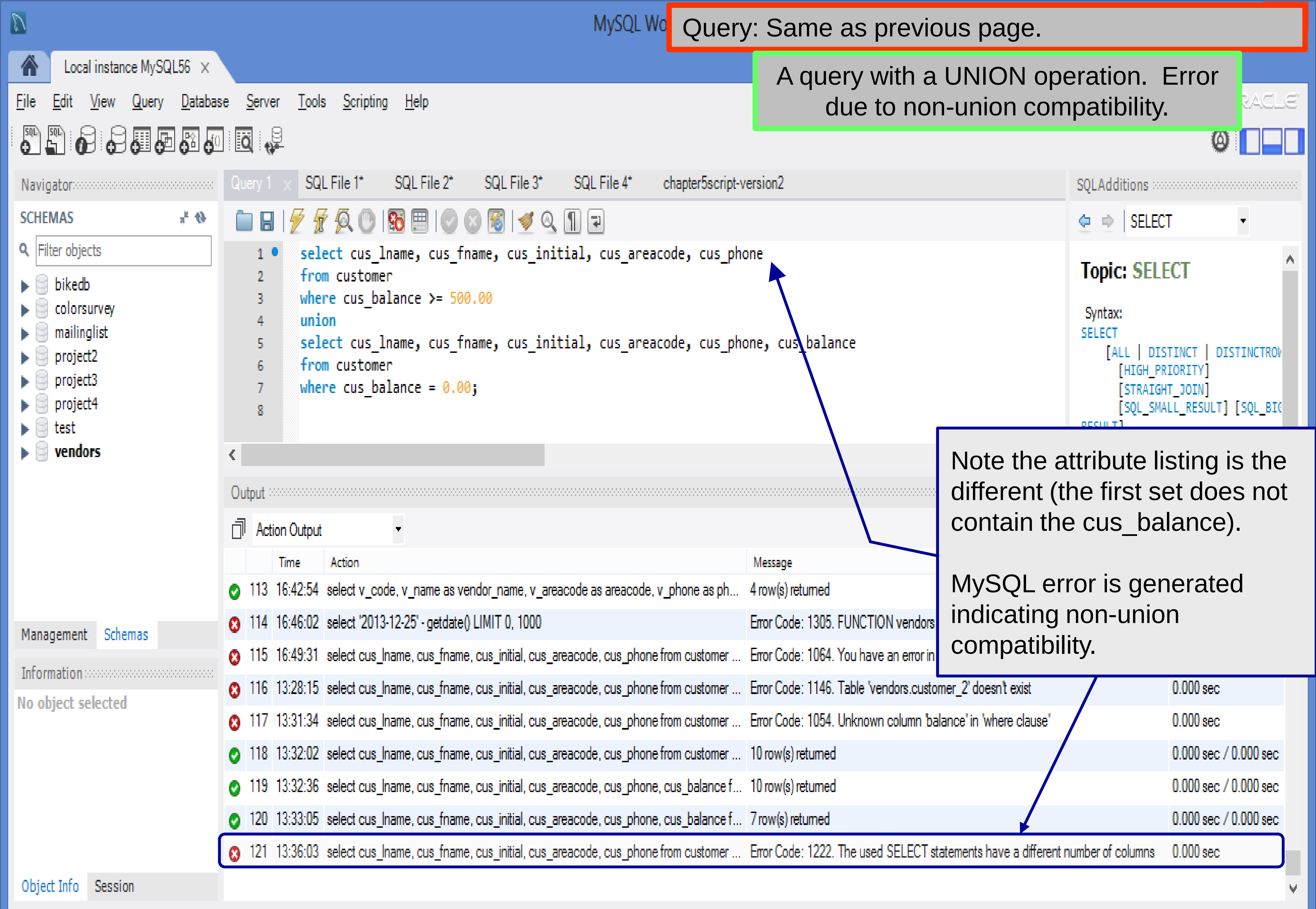


Note the attribute listing is the same in both result sets.

The next page illustrates what happens if they are not the same.

Query: Same as previous page.

A query with a UNION operation. Error due to non-union compatibility.



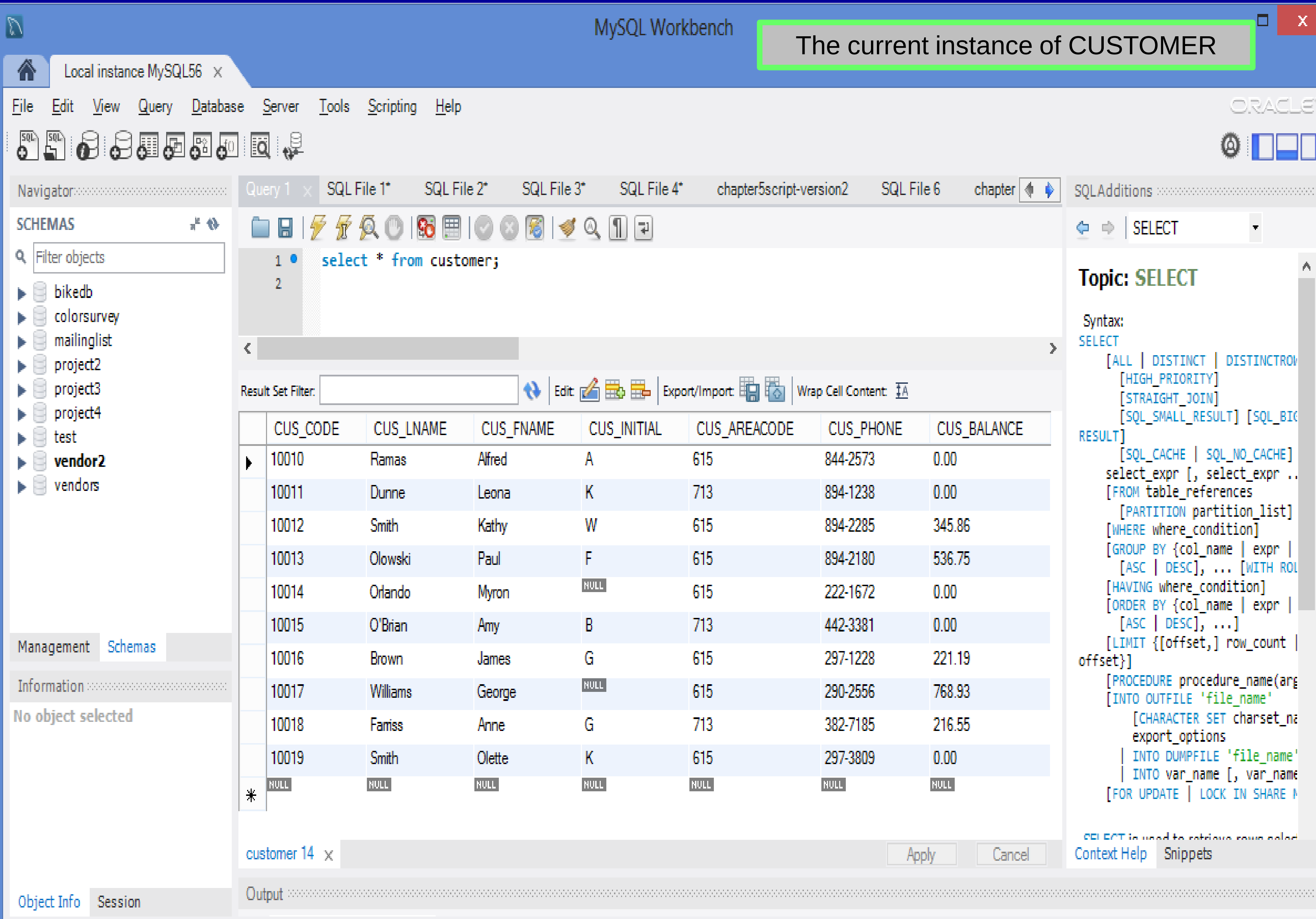
Note the attribute listing is the different (the first set does not contain the cus_balance).

MySQL error is generated indicating non-union compatibility.

Relational Set Operations In SQL

- For the next couple of examples, I modified the database we've been using so that the queries would make more sense.
- I created a second customer table with a schema identical to that of the customer table but missing the customer balance.
- The next two slides illustrate the current instances of the CUSTOMER and CUSTOMER2 tables. Note that two customers, Olowski and Dunne appear in both tables.

The current instance of CUSTOMER



CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
10010	Ramas	Alfred	A	615	844-2573	0.00
10011	Dunne	Leona	K	713	894-1238	0.00
10012	Smith	Kathy	W	615	894-2285	345.86
10013	Olowski	Paul	F	615	894-2180	536.75
10014	Orlando	Myron	NULL	615	222-1672	0.00
10015	O'Brian	Amy	B	713	442-3381	0.00
10016	Brown	James	G	615	297-1228	221.19
10017	Williams	George	NULL	615	290-2556	768.93
10018	Farriss	Anne	G	713	382-7185	216.55
10019	Smith	Olette	K	615	297-3809	0.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL

The current instance of CUSTOMER2

```sql
select * from customer2;
```

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
345	Terrell	Justine	H	615	322-9870
347	Olowski	Paul	F	615	894-2180
351	Hernandez	Carlos	J	723	123-7654
352	McDowell	George	NULL	723	123-7768
365	Tirpin	Amy	G	723	123-9876
368	Lewis	Sherrie	J	734	322-2357
369	Dunne	Leona	K	713	894-1238
NULL	NULL	NULL	NULL	NULL	NULL

Query: List customer details for the customers who appear in either the CUSTOMER or the CUSTOMER2 table.

A query with a UNION operation.

```
select cus_code, cus_lname, cus_fname, cus_initial, cus_areacode, cus_phone
from customer
union
select cus_code, cus_lname, cus_fname, cus_initial, cus_areacode, cus_phone
from customer2;
```

cus_code	cus_lname	cus_fname	cus_initial	cus_areacode	cus_phone
10010	Ramas	Alfred	A	615	844-2573
10011	Dunne	Leona	K	713	894-1238
10012	Smith	Kathy	W	615	894-2285
10013	Olowski	Paul	F	615	894-2180
10014	Orlando	Myron	NULL	615	222-1672
10015	O'Brian	Amy	B	713	442-3381
10016	Brown	James	G	615	297-1228
10017	Williams	George	NULL	615	290-2556
10018	Farriss	Anne	G	713	382-7185
10019	Smith	Olette	K	615	297-3809
345	Terrell	Justine	H	615	322-9870
347	Olowski	Paul	F	615	894-2180
351	Hernandez	Carlos	J	723	123-7654
352	McDowell	George	NULL	723	123-7768
365	Tirpin	Amy	G	723	123-9876
368	Lewis	Sherrie	J	734	322-2357
369	Dunne	Leona	K	713	894-1238

The SQL standard states that duplicates are to be removed from the result of a UNION operation. MySQL does not adopt that standard and you will need to explicitly remove duplicates from a UNION.

The ANSI standard provides a UNION ALL operation that includes duplicate values in a UNION operation.

Relational Set Operations In SQL

- ANSI-standard SQL provides an INTERSECT operation for which the syntax is:

```
query1 INTERSECT query2;
```

- The result set contains the rows that appear in the result of both `query1` and `query2`.
- MySQL does not support the INTERSECT operator. In MySQL the INTERSECT operation is simulated with an INNER JOIN.
- Suppose you want to see the customers who appear in both the CUSTOMER and CUSTOMER2 tables. The query expressions to answer this query are shown on the next page.

Relational Set Operations In SQL

- ANSI-standard SQL:

```
select cus_lname, cus_fname
from customer
INTERSECT
select cus_lname, cus_fname
from customer2;
```

- In MySQL this query would be expressed as:

```
select cus_lname, cus_fname
from customer INNER JOIN customer2
     using (cus_lname, cus_fname);
```

- See next slide.

Query: List customer last name and first name for customers who appear in both the CUSTOMER or the CUSTOMER2 table.

Simulating an INTERSECT operation in MySQL with an INNER JOIN and USING clause

```sql
select customer.cus_lname, customer.cus_fname
from customer inner join customer2
    using (cus_lname, cus_fname);
```

cus_lname	cus_fname
Dunne	Leona
Olowski	Paul

Query: List customer codes for customers in the 615 area code who have made a purchase.

Simulating an INTERSECT operation in MySQL with an INNER JOIN and USING clause

```sql
select distinct customer.cus_code, customer.cus_lname, customer.cus_fname
from customer inner join invoice
    using (cus_code)
where cus_areacode = '615'
```

cus_code	cus_lname	cus_fname
10014	Orlando	Myron
10012	Smith	Kathy

Relational Set Operations In SQL

- ANSI-standard SQL provides a MINUS operation for which the syntax is:

```
query1 MINUS query2;
```

- The result set contains the rows that appear only in the result of `query1`.
- MySQL does not support the MINUS operator. In MySQL the MINUS operation can be simulated with two different scenarios. These two scenarios are illustrated on the next page.

Relational Set Operations In SQL

- ANSI-standard SQL:

```
select cus_lname, cus_fname
from customer
MINUS
select cus_lname, cus_fname
from customer2;
```

- In MySQL this query could be expressed as:

```
select distinct cus_lname, cus_fname
from customer
where (cus_lname, cus_fname) not in
        (select cus_lname, cus_fname
         from customer2);
```

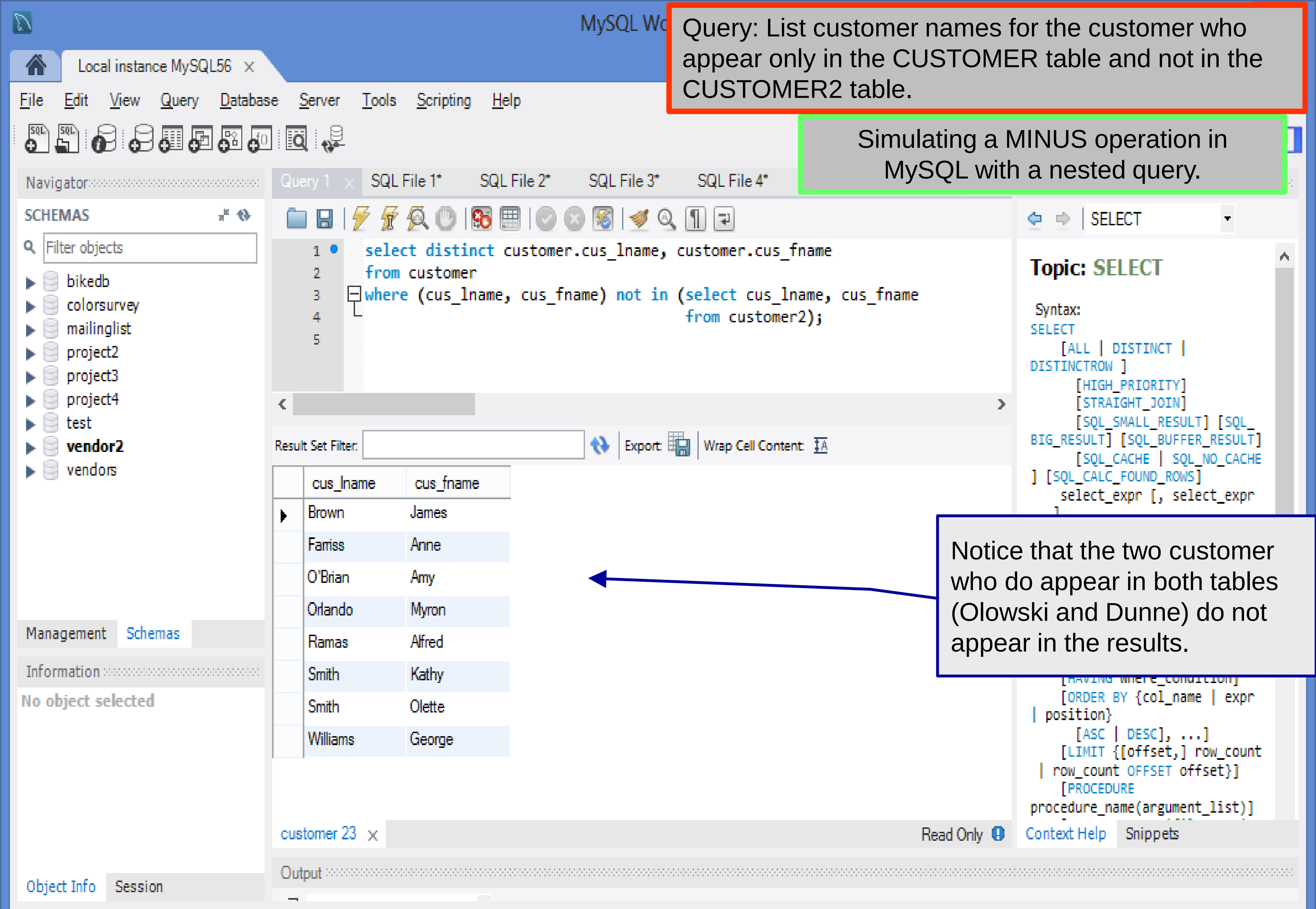
or as:

```
select distinct cus_lname, cus_fname
from customer left outer join customer2
     using (cus_lname, cus_fname)
where customer2.lname is null;
```

- See next slide.

Query: List customer names for the customer who appear only in the CUSTOMER table and not in the CUSTOMER2 table.

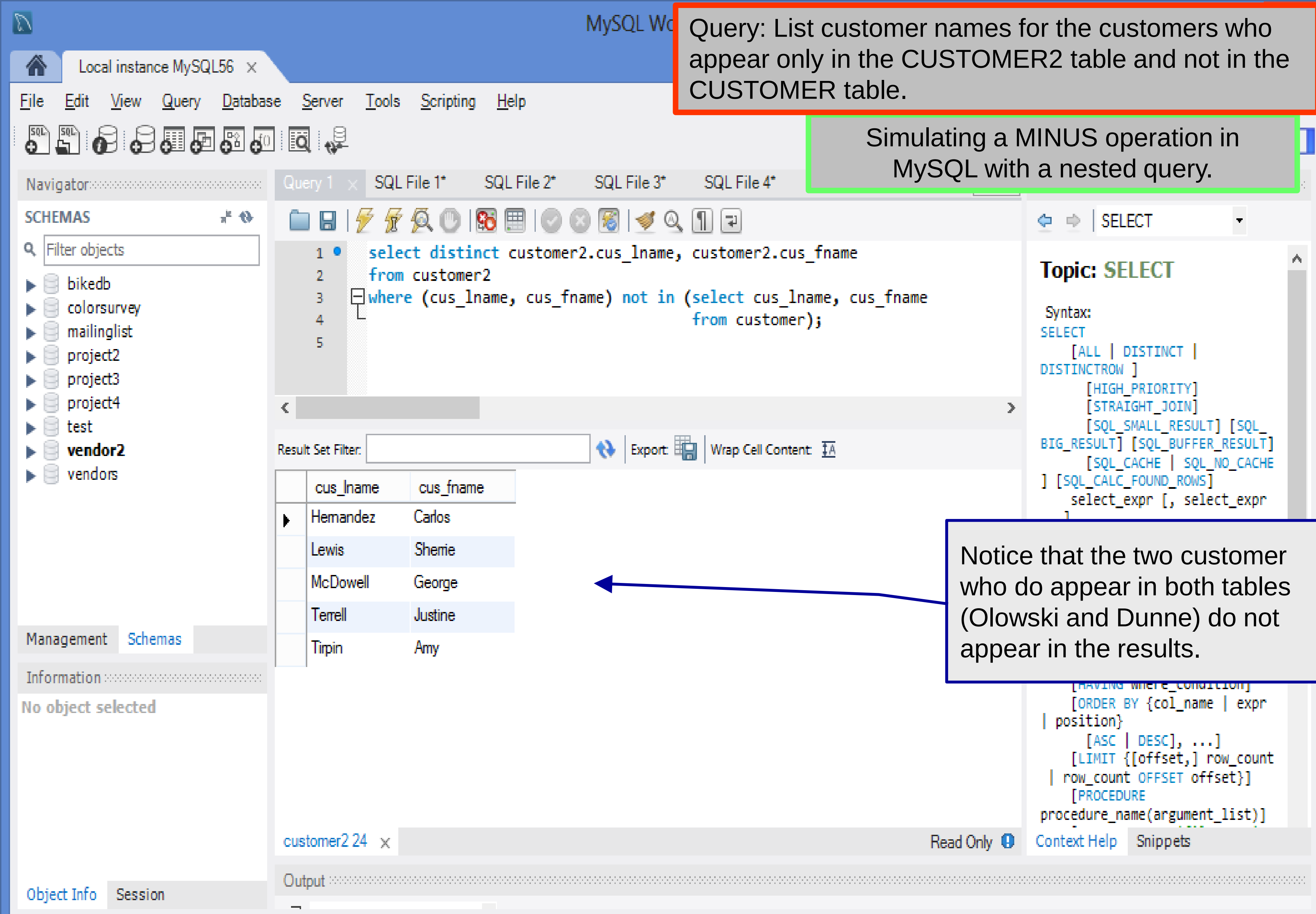
Simulating a MINUS operation in MySQL with a nested query.



Notice that the two customer who do appear in both tables (Olowski and Dunne) do not appear in the results.

Query: List customer names for the customers who appear only in the CUSTOMER2 table and not in the CUSTOMER table.

Simulating a MINUS operation in MySQL with a nested query.



Notice that the two customer who do appear in both tables (Olowski and Dunne) do not appear in the results.

Query: List customer names for the customers who appear only in the CUSTOMER table and not in the CUSTOMER2 table.

Simulating a MINUS operation in MySQL using a LEFT OUTER JOIN

```sql
select distinct customer.cus_lname, customer.cus_fname
from customer left outer join customer2
   using(cus_lname, cus_fname)
where customer2.cus_lname is null;
```

cus_lname	cus_fname
Brown	James
Farriss	Anne
O'Brian	Amy
Orlando	Myron
Ramas	Alfred
Smith	Kathy
Smith	Olette
Williams	George

Notice that the two customer who do appear in both tables (Olowski and Dunne) do not appear in the results.