

LAB03: View (Solution)**Submission:**

- Submit a lab file named “int205_lab03_XXXXXXXXXX.docx/.pdf” into the LEB2 system. XXXXXXXXXXXX = your student id

Due Date & Time:

- Lecturer will inform the LAB03 due date and time in lab class.

What is a View?

A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called **base tables**. The view is stored as a SELECT statement in the data dictionary.

EMPLOYEES table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_FRES	2400
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	1700
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	1700
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	900
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	600
107	Diana	Lorentz	DLORENTZ	590.423.4567	07-FEB-99	IT_PROG	420
124	Kevin	Mourgos	KMOURGOS	650.126.5234	16-NOV-99	ST_MAN	580
141	Trenna	Rais	TRAIRS	650.121.3009	17-OCT-95	ST_CLERK	350
143	Curtis	Davies	CDAVIES	650.121.2994	05-JAN-97	ST_CLERK	310
143	Randall	Mates	RMATES	650.121.2974	15-MAR-98	ST_CLERK	290
149		Zlotkey			JUL-95	ST_CLERK	250
174		Abel		10500	JAN-00	SA_MAN	10500
176		Taylor		11000	MAY-96	SA_REP	11000
176				0600	MAR-98	SA_REP	860
176	Kimberly	Grant	KGRANT	611.44.1044,429203	24-MAY-99	SA_REP	700
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	440
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	1300
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	600
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	1200
206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	830

20 rows selected.

Syntax for creating a view

```
CREATE[OR REPLACE]
  VIEW view_name [(column_list)]
  AS select_statement ;
```

Updatable Views:

- A simple view is one that:
 - Derives data from only one table
 - Contains no functions or groups of data
 - Can perform DML operations through the view

Non-updatable Views:

- A complex view is one that:
 - Derives data from many tables
 - Contains functions or groups of data
 - Does not always allow DML operations through the view

The Syntax of CREATE VIEW statement:

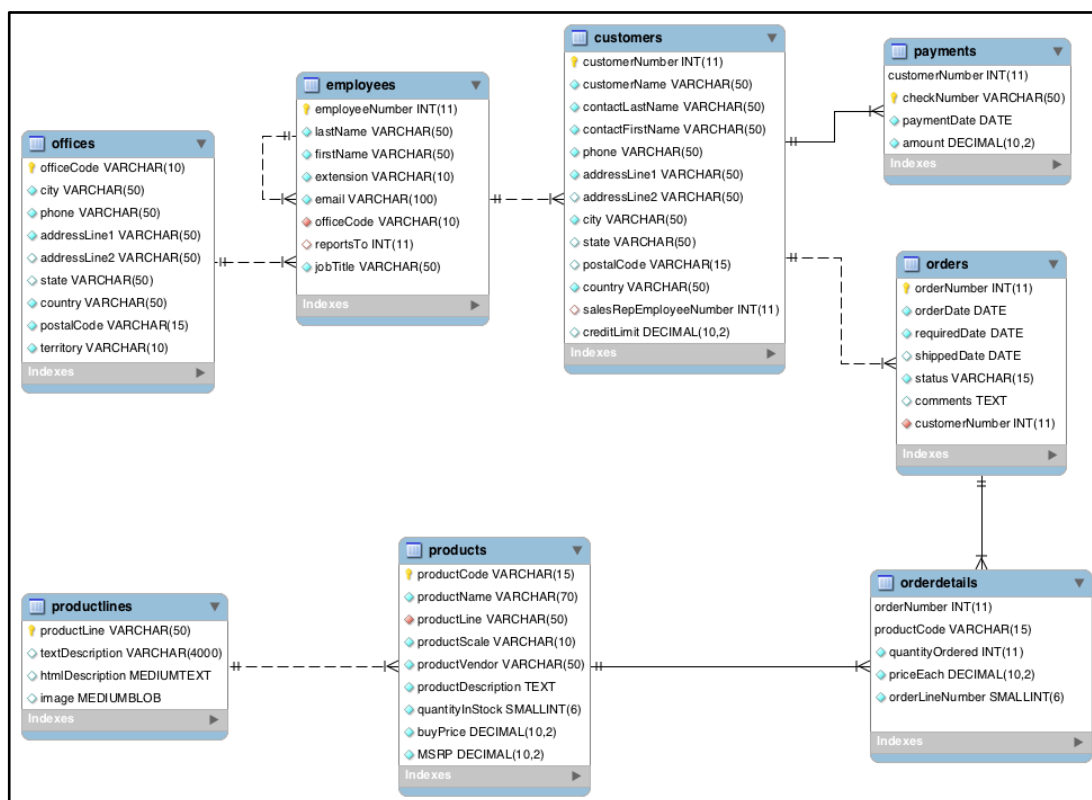
Documentation: <https://dev.mysql.com/doc/refman/8.0/en/create-view.html>

Note: The MySQL error code 1064 is a syntax error. This means the reason there's a problem is because MySQL doesn't understand what you're asking it to do.

Switch to SQL Editor

- You should specify the classicmodels database before writing SQL statements using the following command:
USE db_name;

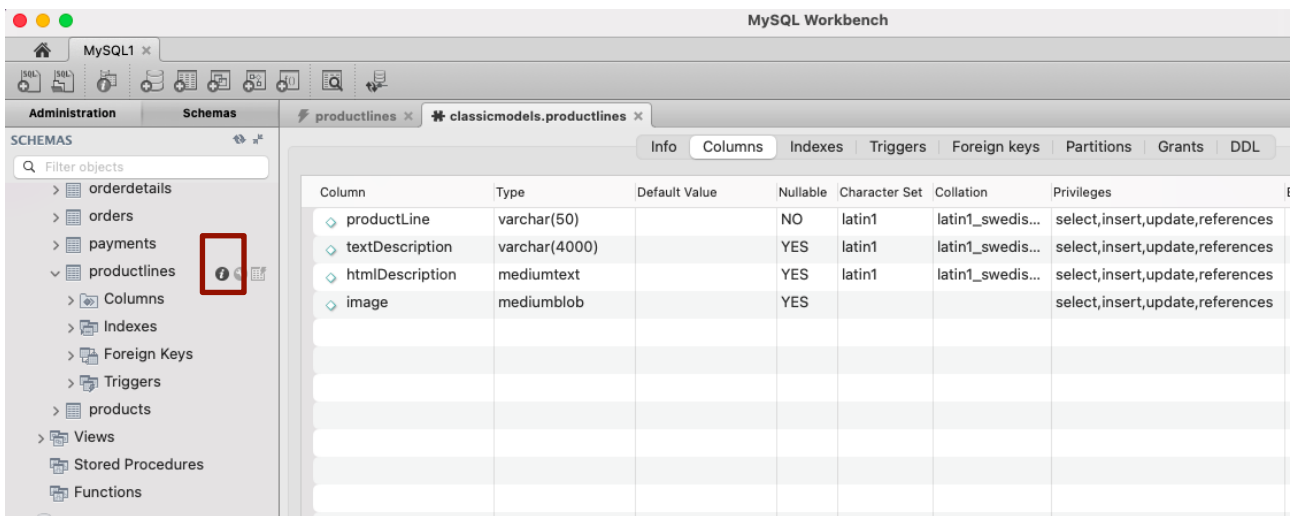
The USE statement tells MySQL to use the named database as the default (current) database for subsequent statements. This statement requires some privilege for the database or some object within it.

The ER diagram for the classicmodels.

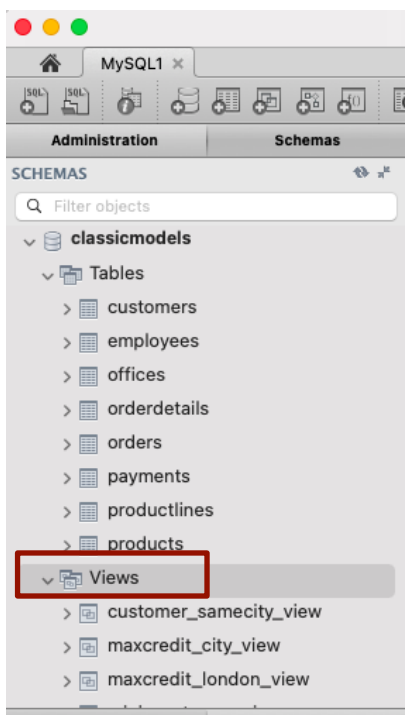
Note: The MSRP is “Manufacturer's suggested retail price” (ราคาขายปลีกแนะนำของผู้ผลิต).

MySQL Workbench:

- You can see details of a table by clicking i button below:



- You can see the existing view by clicking “Views” menu below:



Task 1: Using the “classicmodels” database and write SQL statements to answer the following questions.

use classicmodels;

1. Create a view named "mini_customer_view" to display the customer name of all customers whose names start with the word “Mini”. Please verify by querying data from this view.

-- Write a statement here, the screen of querying data from this view is optional

```
create view mini_customer_view
as
select customername
from customers
where customername like 'Mini%';
```

```
select * from mini_customer_view;
```

The screenshot shows a database management interface with the following components:

- SCHEMAS Panel:** A tree view on the left showing the database structure. The 'classicmodels' database is expanded, showing 'Tables' and 'Views'. The 'mini_customer_view' view is selected.
- SQL Editor:** The main area contains the following SQL statements:


```
1 • use classicmodels;
2   -- 63199999999
3 • create view mini_customer_view
4   as
5   select customername
6   from customers
7   where customername like 'Mini%';
8
9 • select * from mini_customer_view;
```
- Result Grid:** Below the SQL editor, a table displays the results of the final query. The table has one column, 'customername', and six rows of data.

customername
Mini Gifts Distributors Ltd.
Mini Wheels Co.
Mini Caravy
Mini Classics
Mini Creations Ltd.
Mini Auto Werke
- Action Output:** At the bottom, a log shows the execution of the SQL statements:

	Time	Action	Response
1	18:03:34	use classicmodels	0 row(s) affected
2	18:03:51	create view mini_customer_view as select customername from customers where customername like 'Mini%';	0 row(s) affected
3	18:04:30	select * from mini_customer_view LIMIT 0, 1000	6 row(s) returned

2. Create a view named "prod_minstock_view " to display the product name and quantity in stock of the product that has the minimum quantities in stock. Please verify by querying data from this view.

```
create view prod_minstock_view
as
select productname, quantityInStock
from products
where quantityinstock = (select min(quantityinstock) from products);
```

```
select * from prod_minstock_view;
```

The screenshot shows a database management interface with the following components:

- Top Bar:** Displays the user ID '63199999999' and various tool icons.
- Left Panel (SCHEMAS):** A tree view showing the database structure. Under 'classicmodels', 'Views' are expanded, showing 'mini_customer_view' and 'prod_minstock_view' (selected).
- SQL Editor:** Contains the following SQL script:


```

10 • select * from mini_customer_view;
11
12 -- 63199999999
13 -- 2
14 • create view prod_minstock_view
15 as
16 select productname, quantityInStock
17 from products
18 where quantityInStock = (select min(quantityInStock) from products);
19
20 • select * from prod_minstock_view;
21

```
- Result Grid:** Displays the results of the selected query. It shows two columns: 'productname' and 'quantityInStock'. One row is visible: '1960 BSA Gold Star DBD34' with a value of '15'.

productname	quantityInStock
1960 BSA Gold Star DBD34	15
- Object Info:** Shows details for the selected view 'prod_minstock_view':
 - Columns: productname (varchar(70)), quantityInStock (smallint)
- Action Output:** A log showing the execution of the query: 'select * from prod_minstock_view LIMIT 0, 1000' at 18:07:14, resulting in '1 row(s) returned'.

3. Create a view named "totalamount_orders_view" to display the order number, order date and the total amount of sales of all orders and sort the results in descending order by the total amount of sales. Name three columns of the view to orderno, orderdate and total_amount, respectively. Please verify by querying data from this view.

-- A1

```

create view totalamount_orders_view (orderno,orderdate,total_amount)
as
select o.ordernumber, orderdate, sum(quantityordered*priceeach)
from orderdetails d join orders o
where d.ordernumber = o.ordernumber
group by o.ordernumber, orderdate
order by 3 desc ;

```

-- A2

```

create or replace view totalamount_orders_view
as
select o.ordernumber as orderno, orderdate, sum(quantityordered*priceeach) as total_amount
from orderdetails d join orders o
where d.ordernumber = o.ordernumber
group by o.ordernumber, orderdate
order by total_amount desc ;

```

```

select * from totalamount_orders_view;

```

The screenshot shows the SQL Enterprise Manager interface. On the left, the 'SCHEMAS' tree is expanded to show 'classicmodels' > 'Views' > 'totalamount_orders_view'. The main pane displays the SQL script for creating and querying the view. The 'Result Grid' shows the output of the query, displaying columns: orderno, orderdate, and total_amount. The 'Action Output' pane shows the execution log with three steps: selecting from prod_minstock_view, creating the view, and selecting from the view.

```

22 -- 6319999999
23 -- 3
24 • create view totalamount_orders_view (orderno,orderdate,total_amount)
25 as
26 select o.ordernumber, orderdate, sum(quantityordered*priceeach)
27 from orderdetails d join orders o
28 where d.ordernumber = o.ordernumber
29 group by o.ordernumber, orderdate
30 order by 3 desc ;
31
32 • select * from totalamount_orders_view;
33

```

orderno	orderdate	total_amount
10165	2003-10-22	67392.85
10287	2004-08-30	61402.00
10310	2004-10-16	61234.67
10212	2004-01-16	59830.55
10207	2003-12-09	59265.14
10127	2003-06-03	58841.35

	Time	Action	Response
1	18:07:14	select * from prod_minstock_view LIMIT 0, 1000	1 row(s) returned
2	18:08:49	create view totalamount_orders_view (orderno,orderdate,total_amount) as select o.ordernumb...	0 row(s) affected
3	18:09:04	select * from totalamount_orders_view LIMIT 0, 1000	326 row(s) returned

4. Create a view named "customer_samecity_view" to display the customer name and city of all customers who live in the same city of their sales rep employee's office city. Name two view columns to cust_name and cust_city, respectively. Please verify by querying data from this view.

-- A1

```

create view customer_samecity_view (cust_name, cust_city)
as
select customername, city
from customers c join employees e
on c.salesRepEmployeeNumber = e.employeeNumber
where exists (select city
              from offices
              where officeCode = e.officeCode
              and city = c.city);

```

-- A2

```

create or replace view customer_samecity_view
as
select customername as cust_name, c.city as cust_city
from customers c join employees e
on c.salesRepEmployeeNumber = e.employeeNumber
join offices o
on o.officeCode = e.officeCode
where o.city = c.city;

select * from customer_samecity_view;

```

The screenshot shows a database management interface with a left sidebar containing a tree view of schemas. The 'Schemas' section is expanded, showing 'classicmodels' and its sub-items: 'Tables', 'Views', 'Stored Procedures', and 'Functions'. The 'Views' folder is selected, and 'customer_samecity_view' is highlighted. The main pane displays the SQL code for creating and querying the view. The 'Result Grid' shows the output of the query, displaying columns 'cust_name' and 'cust_city' with data for Mini Wheels Co., Corporate Gift Ideas Co., Diecast Collectables, and Gifts4AllAges.com. The 'Action Output' pane at the bottom shows a log of database actions, including the creation of the view and the execution of the query, with the response indicating that 14 rows were returned.

```

34 -- 63199999999
35 -- 4
36 • create view customer_samecity_view (cust_name, cust_city)
37 as
38 select customername, city
39 from customers c join employees e
40 on c.salesRepEmployeeNumber = e.employeeNumber
41 where exists (select city
42               from offices
43               where officeCode = e.officeCode
44                 and city = c.city);
45 • select * from customer_samecity_view;
46

```

cust_name	cust_city
Mini Wheels Co.	San Francisco
Corporate Gift Ideas Co.	San Francisco
Diecast Collectables	Boston
Gifts4AllAges.com	Boston

Time	Action	Response
18:08:49	create view totalamount_orders_view (orderno,orderdate,total_amount) as select o.ordernumb...	0 row(s) affected
18:09:04	select * from totalamount_orders_view LIMIT 0, 1000	326 row(s) returned
18:10:53	create view customer_samecity_view (cust_name, cust_city) as select customername, city fro...	0 row(s) affected
18:11:14	select * from customer_samecity_view LIMIT 0, 1000	14 row(s) returned

5. Create a view named "maxcredit_city_view" to display the city and the maximum credit limit of all customers in each city. Please verify by querying data from this view.

```

create view maxcredit_city_view
as
select city, max(creditlimit) as max_credit
from customers
group by city;

select * from maxcredit_city_view;

```

The screenshot shows the SQL Enterprise Manager interface. The left pane displays the 'SCHEMAS' tree with 'classicmodels' expanded, showing 'Views' and 'maxcredit_city_view' selected. The right pane shows the SQL script for creating and querying the view.

```

46
47
48 -- 63199999999
49 -- 5
50 • create view maxcredit_city_view
51 as
52 select city, max(creditlimit) as max_credit
53 from customers
54 group by city;
55
56 • select * from maxcredit_city_view;
57

```

The 'Result Grid' shows the output of the query, displaying columns 'city' and 'max_credit'.

city	max_credit
Nantes	118200.00
Las Vegas	71800.00
Melbourne	117300.00
Stavern	81700.00
San Rafael	210500.00

The 'Action Output' pane shows the execution history:

	Time	Action	Response
4	18:10:53	create view customer_samecity_view (cust_name, cust_city) as select customername, city fro...	0 row(s) affected
5	18:11:14	select * from customer_samecity_view LIMIT 0, 1000	14 row(s) returned
6	18:13:40	create view maxcredit_city_view as select city, max(creditlimit) as max_credit from customers g...	0 row(s) affected
7	18:13:52	select * from maxcredit_city_view LIMIT 0, 1000	95 row(s) returned

6. Create a view named "maxcredit_london_view" to display the city and the maximum credit limit of all customers who live in London city. You should create this view from the "maxcredit_city_view" view in [Question 5](#). Please verify by querying data from this view.

-- A1

```
create view maxcredit_london_view
as
```

```
select city, max_credit
from maxcredit_city_view
where city = 'London';
```

-- A2

```
create or replace view maxcredit_london_view
as
```

```
select *
from maxcredit_city_view
where city = 'London';
```

```
select * from maxcredit_london_view;
```


The screenshot shows the SQL Enterprise Manager interface. On the left, the 'SCHEMAS' tree is expanded to 'Views', and 'maxcredit_london_view' is selected. The main pane displays the SQL script for creating and querying the view:

```

59
60 -- 63199999999
61 -- 6
62 • create view maxcredit_london_view
63 as
64 select city, max_credit
65 from maxcredit_city_view
66 where city = 'London';
67
68 • select * from maxcredit_london_view;
69
70

```

Below the script, the 'Result Grid' shows the output of the query:

city	max_credit
London	77000.00

The 'Action Output' pane at the bottom shows the execution log:

	Time	Action	Response
6	18:13:40	create view maxcredit_city_view as select city, max(creditlimit) as max_credit from customers g...	0 row(s) affected
7	18:13:52	select * from maxcredit_city_view LIMIT 0, 1000	95 row(s) returned
8	18:15:24	create view maxcredit_london_view as select city, max_credit from maxcredit_city_view where c...	0 row(s) affected
9	18:15:38	select * from maxcredit_london_view LIMIT 0, 1000	1 row(s) returned

7. Create a table named "offices_copy" with copying the structure and data from the "offices" table using the following commands:

```

create table offices_copy
as select * from offices;

```

Create a view named "usa_office_view" to display office code, city and state of the country "USA" from the "offices_copy" table. Please verify by querying data from this view.

```

create table offices_copy
as select * from offices;

```

```

create view usa_office_view
as
select officecode, city, state
from offices_copy
where country = 'USA';

```

```

select * from usa_office_view;

```

The screenshot shows the SQL Enterprise Manager interface. The left pane displays the 'SCHEMAS' tree with 'usa_office_view' selected. The middle pane shows the SQL script for creating and querying the view. The right pane shows the 'Result Grid' with three rows of data from the view. The bottom pane shows the 'Action Output' log with three entries.

SQL Script:

```

74 -- 63199999999
75 -- 7
76 • create view usa_office_view
77 as
78 select officecode, city, state
79 from offices_copy
80 where country = 'USA';
81
82 • select * from usa_office_view;
83
84

```

Result Grid:

	officecode	city	state
1		San Francisco	CA
2		Boston	MA
3		NYC	NY

Action Output:

	Time	Action	Response
9	18:15:38	select * from maxcredit_london_view LIMIT 0, 1000	1 row(s) returned
10	18:16:52	create table offices_copy as select * from offices	7 row(s) affected Records :
11	18:17:00	create view usa_office_view as select officecode, city, state from offices_copy where country =...	0 row(s) affected
12	18:17:02	select * from usa_office_view LIMIT 0, 1000	3 row(s) returned

8. Try to insert a new row through the "usa_office_view" view created in [Question 7](#). What happens about the data insertion into the "offices_copy" table? Please explain.
Hint: You can create data for a new row by yourself.

insert into usa_office_view
values (8, 'NYC', 'NY');

-- Error Code: 1423. Field of view 'classicmodels.usa_office_view' underlying table doesn't have a default value
-- The NOT NULL columns of the "offices_copy" table were not in the view definition. Therefore, you cannot insert new rows through this view.

select * from usa_office_view;
select * from offices_copy;

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane shows the 'usa_office_view' view under the 'classicmodels' database. The 'Columns' pane shows the view has columns: officecode (varchar(10)), city (varchar(50)), and state (varchar(50)). The main pane shows the SQL script:

```

75 -- 7
76 • create view usa_office_view
77 as
78 select officecode, city, state
79 from offices_copy
80 where country = 'USA';
81
82 • select * from usa_office_view;
83
84
85 -- 63199999999
86 -- 8
87 • insert into usa_office_view
88 values (8, 'NYC', 'NY');
89
90
91
92

```

The 'Action Output' pane at the bottom shows the execution results:

	Time	Action	Response
7	18:13:52	select * from maxcredit_city_view LIMIT 0, 1000	95 row(s) returned
8	18:15:24	create view maxcredit_london_view as select city, ma...	0 row(s) affected
9	18:15:38	select * from maxcredit_london_view LIMIT 0, 1000	1 row(s) returned
10	18:16:52	create table offices_copy as select * from offices	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0
11	18:17:00	create view usa_office_view as select officecode, cit...	0 row(s) affected
12	18:17:02	select * from usa_office_view LIMIT 0, 1000	3 row(s) returned
13	18:20:04	insert into usa_office_view values (8, 'NYC', 'NY')	Error Code: 1423. Field of view 'classicmodels.usa_office_view' underlying table doesn't have a default value

9. To resolve the problem found in [Question 8](#), Please modify the "usa_office_view" view to ensure that you can insert a new row through this view (an updatable view). Please show the data insertion of the "offices_copy" table.

Hint: You can create data for a new row by yourself.

```

create or replace view usa_office_view
as
select officecode, city, phone, addressline1, country, postalcode, territory
from offices_copy
where country = 'USA';

```

```

insert into usa_office_view
values (8, 'NYC', '9999', '126 KMUTT', 'USA', 10022, 'NA');
commit;

```

```

select * from usa_office_view;
select * from offices_copy;

```

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane shows the 'usa_office_view' under the 'classicmodels' database. The 'Object Info' pane shows the columns: officecode, city, phone, addressline1, addressline2, state, country, postalcode, and territory. The main pane shows the SQL script for creating the view and inserting data. The 'Result Grid' shows the data inserted into the view.

```

-- 6319999999
-- 9
91 * create or replace view usa_office_view
92 as select officecode, city, phone,addressline1,country,postalcode,territory
93     from offices_copy
94     where country = 'USA';
95 * insert into usa_office_view
96 values (8, 'NYC', '9999','126 KMUTT', 'USA', 10022,'NA');
97 * select * from usa_office_view;
98 * select * from offices_copy;

```

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA
4	Paris	+33 14 723 4404	43 Rue Jouffroy D'abbans			France	75017	EMEA
5	Tokyo	+81 33 224 5000	4-1 Kioicho			Chiyoda-Ku	102-8578	Japan
6	Sydney	+61 2 9264 2451	5-11 Wentworth Avenue	Floor #2		Australia	NSW 2010	APAC
7	London	+44 20 7877 2041	25 Old Broad Street	Level 7		UK	EC2N 1HN	EMEA
8	NYC	9999	126 KMUTT			USA	10022	NA

The 'Action Output' pane shows the following actions:

Time	Action	Response
14 18:21:44	create or replace view usa_office_view as select offic...	0 row(s) affected
15 18:21:47	insert into usa_office_view values (8, 'NYC', '9999','1...	1 row(s) affected
16 18:21:53	select * from offices_copy LIMIT 0, 1000	8 row(s) returned

10 Please delete both the structure and data of the "offices_copy" table. What happens to an existing view that references the "offices_copy" table? Please explain.

drop table offices_copy;

select * from usa_office_view;

-- Error Code: 1356. View 'classicmodels.usa_office_view' references invalid table(s) or column(s) or function(s) or definer/invoke of view lack rights to use them

-- The "usa_office_view" still exists but it does not work because the base table does not exist.

The screenshot shows the SQL Server Enterprise Manager interface after the deletion of the 'offices_copy' table. The 'SCHEMAS' pane shows the 'usa_office_view' still present. The main pane shows the SQL script for creating the view, inserting data, and dropping the table. The 'Result Grid' is empty. The 'Action Output' pane shows the following actions:

Time	Action	Response
14 18:21:44	create or replace view usa_office_view as select offic...	0 row(s) affected
15 18:21:47	insert into usa_office_view values (8, 'NYC', '9999','12...	1 row(s) affected
16 18:21:53	select * from offices_copy LIMIT 0, 1000	8 row(s) returned
17 18:24:16	drop table offices_copy	0 row(s) affected
18 18:24:20	select * from usa_office_view LIMIT 0, 1000	Error Code: 1356. View 'classicmodels.usa_office_view' references invalid table(s) or column(s) o...