LAB05: View and Subquery (V2)

Submission:

- Submit a lab file named "int205_lab05_xxxxxxxxxxx.docx/.pdf" into the LEB2 system. xxxxxxxxxxx = your student id

Due Date & Time:

- Lecturer will inform the LAB05 due date and time in lab class.

The View WITH CHECK OPTION Clause

The WITH CHECK OPTION is an integrity constraint on **an updatable view** to prevent inserts to rows for which the WHERE clause in the select_statement is not true. WITH CHECK OPTION testing is standard-compliant.

Syntax for creating a view

```
CREATE[OR REPLACE]
    VIEW view_name [(column_list)]
    AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION];
```

The Syntax of CREATE VIEW statement:
Documentation: https://dev.mysql.com/doc/refman/8.0/en/create-view.html

The WITH CHECK OPTION clause can be given for an updatable view to prevent inserts or updates to rows except those for which the WHERE clause in the select_statement is true.

In a WITH CHECK OPTION clause for an updatable view, the LOCAL and CASCADED keywords determine the scope of check testing when the view is defined in terms of another view. The LOCAL keyword restricts the CHECK OPTION only to the view being defined. CASCADED causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is CASCADED.

Resource: https://www.mysqltutorial.org/mysql-view-local-cascaded-in-with-check-option

Example:

```
CREATE TABLE t1 (a INT);

CREATE OR REPLACE VIEW v1
AS SELECT *
    FROM t1
    WHERE a < 2;
```

```
CREATE OR REPLACE VIEW v2
AS SELECT *
    FROM v1 WHERE a > 1
    WITH LOCAL CHECK OPTION;

CREATE OR REPLACE VIEW v3
AS SELECT *
    FROM v1
    WHERE a > 0
    WITH CASCADED CHECK OPTION;
```

Evaluate the following INSERT statements:

```
-- 1. What is the result?
INSERT INTO v2 VALUES (1);
```

-- The “CHECK OPTION failed” error is returned because the ”a > 1” WHERE condition of V2 is False.

```
-- 2. What is the result?
INSERT INTO v2 VALUES (3);
```

-- The INSERT statement is executed successfully because the ”a > 1” WHERE condition of V2 is True.

```
-- 3. What is the result?
INSERT INTO v3 VALUES (1);
```

-- The INSERT statement is executed successfully because both the ”a > 0” WHERE condition of V3 and the “a < 2” WHERE condition of V1 are True.

```
-- 4. What is the result?
INSERT INTO v3 VALUES (3);
```

-- The “CHECK OPTION failed” error is returned because only the ”a > 0” WHERE condition of V3 is True while the “a < 2” WHERE condition of V1" is False.

Subquery Review:

A subquery is a SELECT statement within another statement.

- Type 1 – **Nested Subquery**: Database evaluates the whole query in two steps:
  - First, execute the subquery (inner query).
  - Second, use the result of the subquery in the parent statement (outer query).
- Type 2 - **Correlated Subquery**: Database evaluated once for each row processed by the parent statement.
  - This operation is used when a subquery refers to a column from a table in an outer query.
  - The unqualified columns in the subquery are resolved by looking in the tables named in the inner query and then in the tables named in the outer query.

Subquery Documentation:https://dev.mysql.com/doc/refman/8.0/en/subqueries.html

Subquery in DML statements

- INSERT statement – adds new rows of data to a table
- UPDATE statement – modifies existing data in a table
- DELETE statement – removes rows of data from a table

-- Syntax --

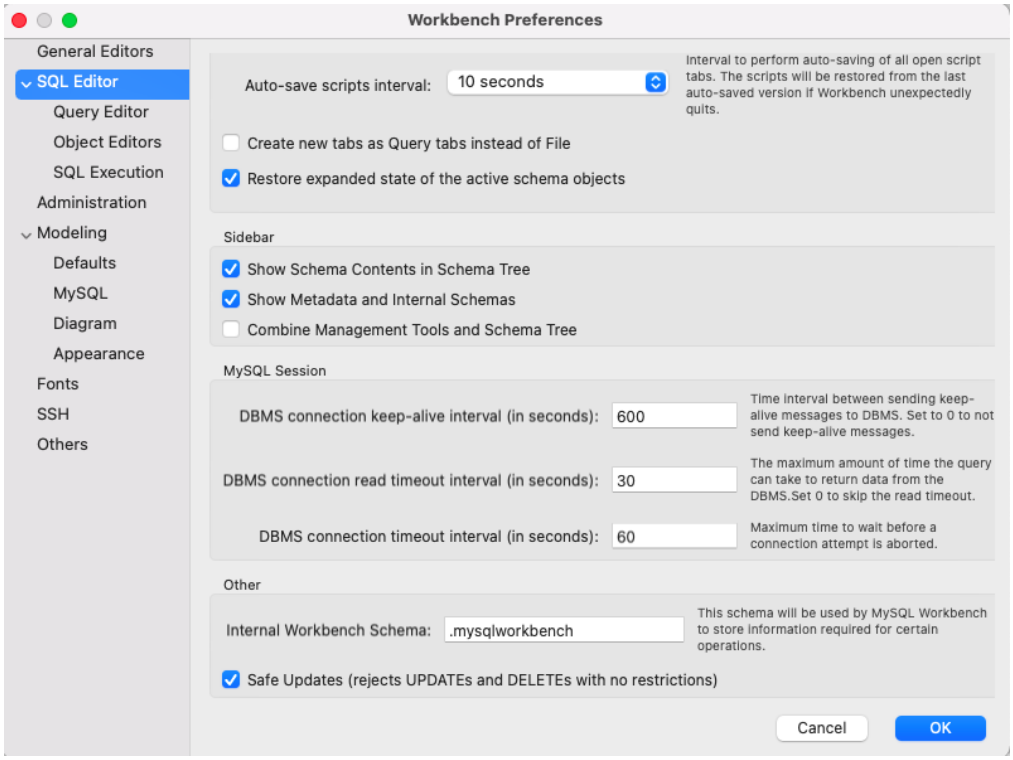
```
INSERT INTO table_name|view_name [(column_list)]
   SELECT column(s)
   FROM table_name| view_name
   [WHERE condition(s)];

UPDATE table_name|view_name
SET  column = value [,column2 = value2,...]
[WHERE condition(s)];

DELETE table_name|view_name
[WHERE condition(s)];
```

SAFE-UPDATES option

MySQL session has the safe-updates option set (SET SQL_SAFE_UPDATES = 1). This means that you can't update or delete records without specifying a key (ex. primary key) in the WHERE clause. If you want to disable the safe-updates option, you can set SET SQL_SAFE_UPDATES = 0.

MySQL Workbench: Checking the safe-updates option

Menu => Tools/MySQLWorkbench => Preferences => SQL Editor => Safe-updates

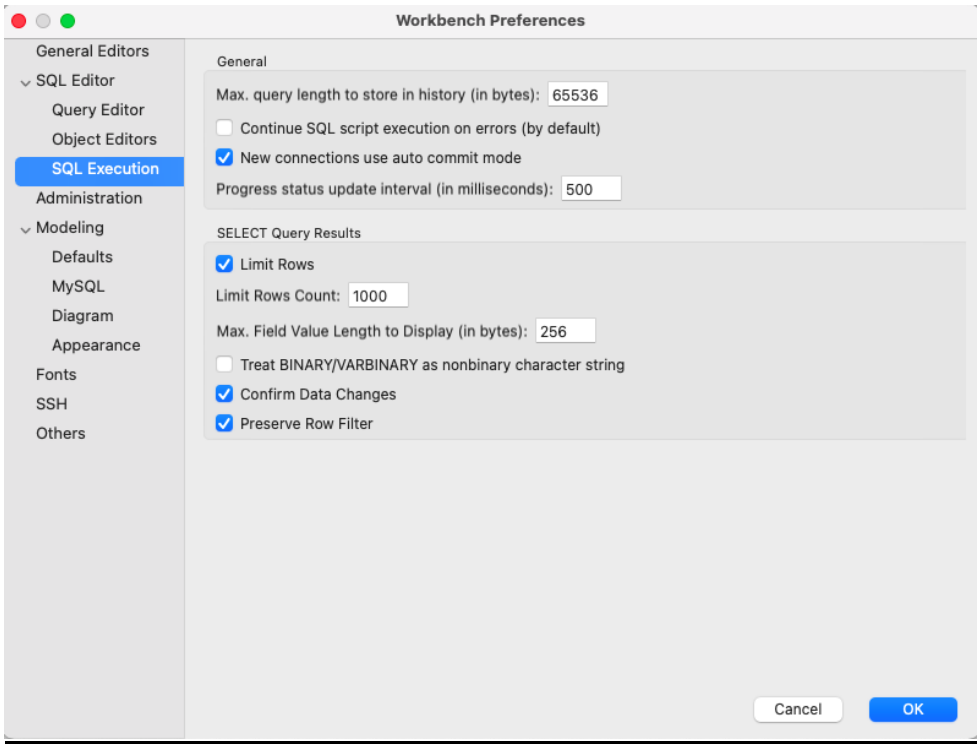


AUTOCOMMIT Mode

By default, MySQL starts the session for each new connection with autocommit enabled, so MySQL does a commit after each SQL statement if that statement did not return an error.

If **autocommit** mode is disabled within a session with SET autocommit = 0, the session always has a transaction open. A COMMIT or ROLLBACK statement ends the current transaction and a new one starts. If a session that has autocommit disabled ends without explicitly committing the final transaction, MySQL rolls back that transaction.

MySQL Workbench: Checking the autocommit mode
Menu => Tools/MySQLWorkbench => Preferences => SQL Execution



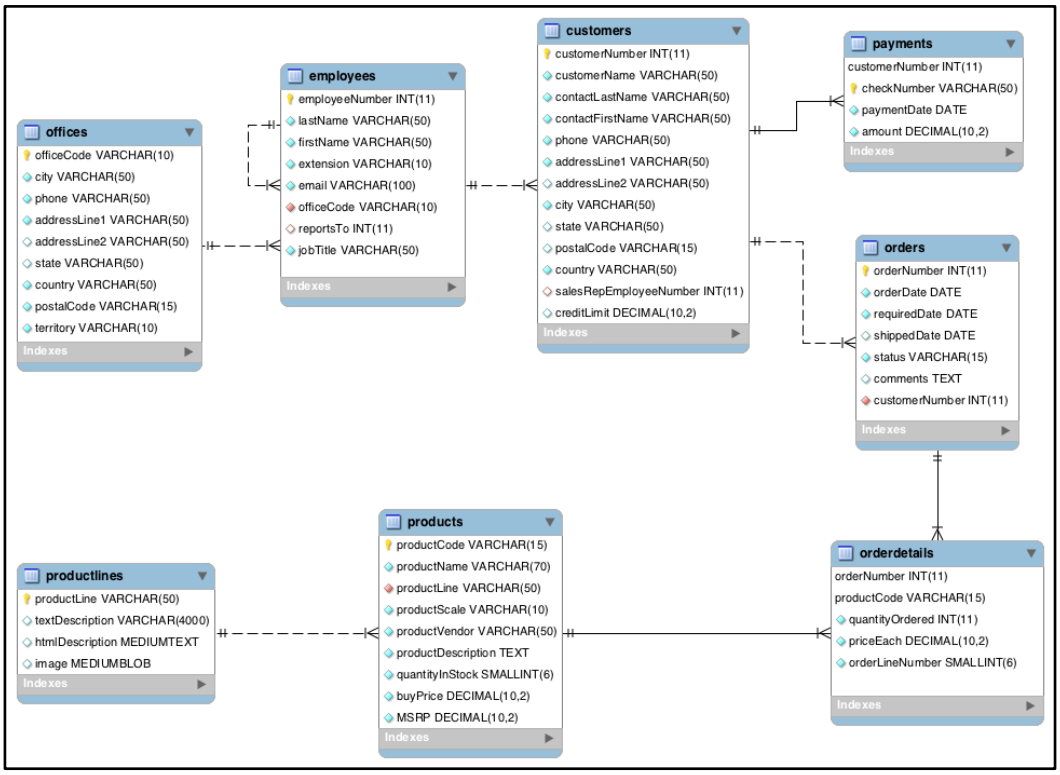
Switch to SQL Editor

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

  USE db_name;

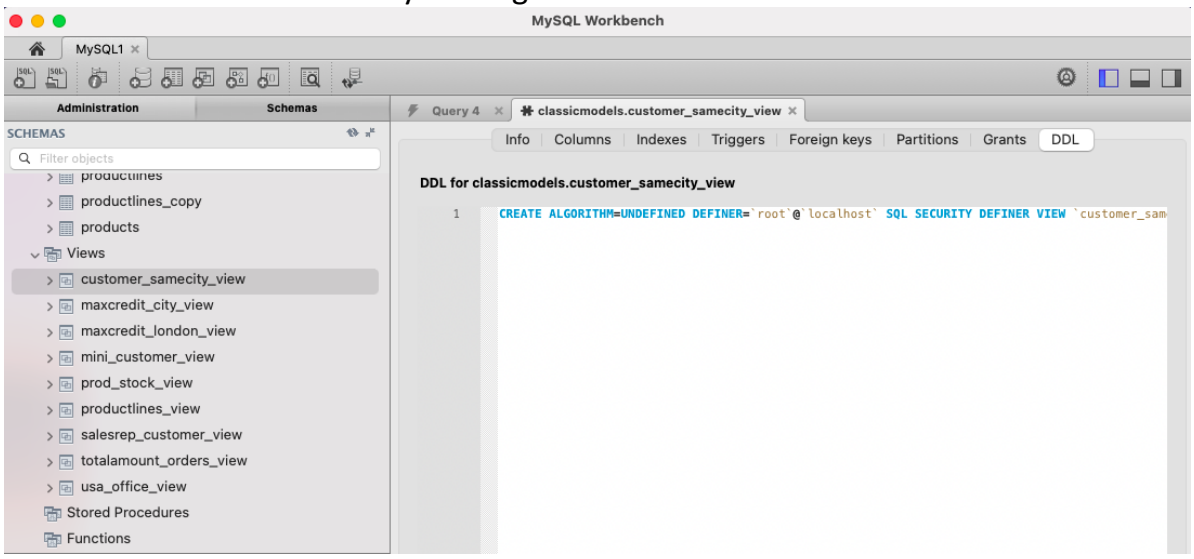
The USE statement tells MySQL to use the named database as the default (current) database for subsequent statements. This statement requires some privilege for the database or some object within it.

The ER diagram for the classicmodels.
Note: The MSRP is “Manufacturer's suggested retail price” (ราคาขายปลีกแนะนำของผู้ผลิต).



MySQL Workbench:

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



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

use classicmodels;

1. Create a new table named ''usa_customers'' with copying only the structure of four columns: customernumber, customername, city, country of the ''customers'' table. Do not copy any data from the ''customers'' table. Please verify by querying data from the table.
-- Write a statement(s) here

2. Insert data by copying the existing data of all customers who live in the USA from the ''customers'' table into the “usa_customers” table. Please verify by querying data from the table. How many rows are inserted into the "usa_customers" table.
-- Write a statement(s) here and also capture the screen of querying data from the table.

3. Based on the ''usa_customers'' table, modify the city of the customername "Mini Wheels Co." to the same city of the customer number 344 of the "customers" table. Please verify your data modification.
-- Write a statement(s) here and also capture the screen of querying data from the table.

4. Based on the ''usa_customers'' table, modify the city of all customers who have a sales representative (employee) last named "Patterson" to "Bangmod". Please verify your data modification.
Hint: you may use the customers and employees tables to find out “who have a sales representative (employee) last named "Patterson".
-- Write a statement(s) here and also capture the screen of querying data from the table.

5. Modify an existing view named "mini_customer_view" to display the customer number, customer name, city and country of all customers whose names start with the word “Mini” from the ''usa_customers'' table. Name four columns of this view to "cno", ''cname'', "city" and ''country'', respectively. Please verify by querying data from this view.
-- Write a statement here(s) and also capture the screen of querying data from the table/view.

6. Create a view named "miniltd_customer_view" to display the customer number, customer name, city and country of all customers whose names end with the word “Ltd.” from the “mini_customer_view” view. Please ensure that the rows that are being changed through this view are conformable to the definition of the "miniltd_customer_view" view. Name four columns of this view to “custno", "custname”, “custcity” and "custcountry", respectively. Please verify by querying data from this view.
-- Write a statement(s) here and also capture the screen of querying data from the table/view.

7. Insert new data {customer number "9000", customer name ''SUNISA Ltd.'', city ''Texas'' and country ''USA''} through the “miniltd_customer_view” view. Please verify by querying data from both this view and the base table. Can the data be inserted through this view? If not, please explain.
-- Write a statement(s) here

8. Insert new data {customer number "9001", customer name ''Mini SUNISA'', city = ''Texas'' and country ''USA''} through the “miniltd_customer_view” view. Please verify by querying data from both this view and the base table. Can the data be inserted through this view? If not, please explain.
-- Write a statement(s) here

9. Modify an existing view named the "miniltd_customer_view" created in Question 6 to ensure that the rows that are being changed through this view are conformable to the definition of the "miniltd_customer_view" view and also the definition of the underlying views recursively.
-- Write a statement(s) here and also capture the screen of querying data from the table/view.

10. Try to insert the same data of Question 7-8 again.

What happened to the row of the customer name ''SUNISA Ltd.''? Please verify by querying data from both this view and the base table. Can the data be inserted through this view? If not, please explain.
-- Write a statement(s) here

What happened to the row of the customer name ''Mini SUNISA'' ? Please verify by querying data from both this view and the base table. Can the data be inserted through this view? If not, please explain.
-- Write a statement(s) here

11. Please insert one row through the “miniltd_customer_view” view. You should create the data by yourself that can be inserted through this view. Please verify by querying data from both the views and the base table.
-- Write a statement(s) here and also capture the screen of querying data from the table/view.

12. Remove two existing views that were created in Lab04. You can select two views by yourself.
-- Write a statement(s) here