# COMP 2350

Lab Report:

Advanced SELECTs, DML and Web Security

By Yeseol (Sol) Kim

Submitted to: Patrick Guichon

Introduction:

The goal of today's lab session, I tried updating, deleting, and reassembling contents within a database. In a SELECT statement, you can use the keywords in the following order: SELECT, FROM, JOIN, WHERE, GROUP BY, HAVING, ORDER BY. The INSERT statement allows you to add new content. DELETE enables the removal of content, with the option to specify values using the WHERE clause.

This part was little bit tricky to try the first time, Left Join retrieves all the records and the data from the left table and all matching records from the right table. Right Join retrieves all the records and the data from the right table and all matching records from the left table.

The GROUP BY statement is used to group rows with identical values into summary rows. This statement is often combined with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set based on one or more columns.

Finally, I established a connection between the render server and MySQL, enabling the addition, editing, and deletion of data from both the server and SQL. To facilitate error identification in case of connection issues, I inputted distinct error codes, making it more convenient for troubleshooting.

Screenshot #1:

This is how to insert contents into table. The **INSERT** command can add new individual rows to single table one at a time (using multiple INSERTs) or multiple rows at a time (using a single INSERT).

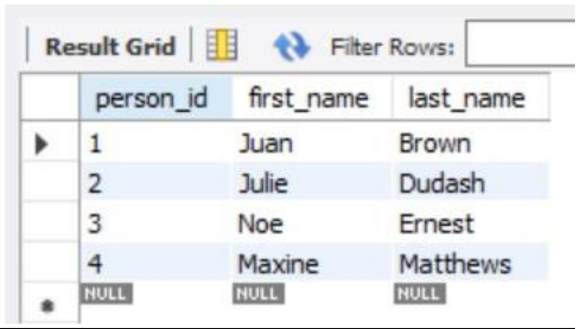
**Create table**

```
CREATE TABLE person (
person_id int auto_increment NOT NULL,
first_name varchar(50) NOT NULL,
last_name varchar(50) NOT NULL,
 CONSTRAINT PK_Person PRIMARY KEY (person_id)
);
```

**INSERT command**

```
INSERT INTO Person
(first_name, last_name)
VALUES
('Juan', 'Brown'),
('Julie', 'Dudash'),
('Noe','Ernest'),
('Maxine','Matthews');
```

Filename: 01_New_People.jpg

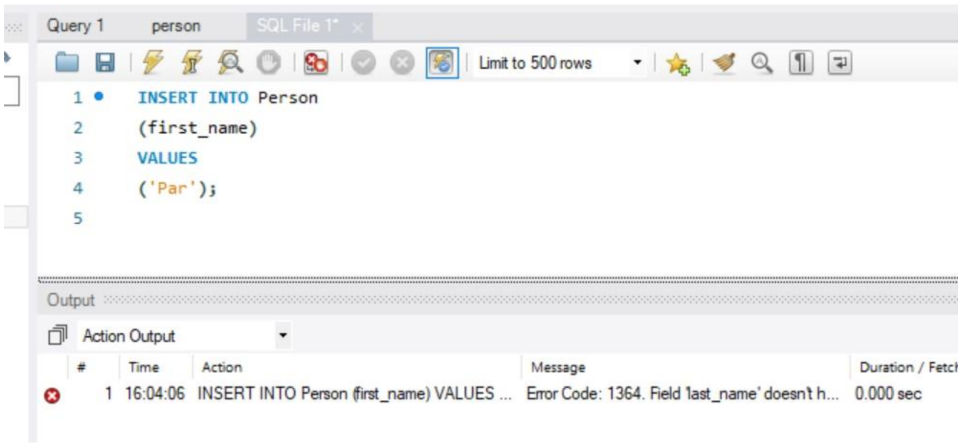


| person_id | first_name | last_name |
|---|---|---|
| 1 | Juan | Brown |
| 2 | Julie | Dudash |
| 3 | Noe | Ernest |
| 4 | Maxine | Matthews |
| NULL | NULL | NULL |

Screenshot #2:

The error arises when the last_name value is absent or null. If the data type for last_name is non-nullable, it is imperative to include this information in the contents; otherwise, it cannot be added to the table.

Filename: 02_Null_Last_Name.jpg



```
INSERT INTO Person
(first_name)
VALUES
('Par');
```

```
#  Time      Action                                        Message                                            Duration / Fetch
1  16:04:06  INSERT INTO Person (first_name) VALUES ...    Error Code: 1364. Field 'last_name' doesn't h...   0.000 sec
```

Screenshot #3:

Alter the table by adding a new column using the following command.

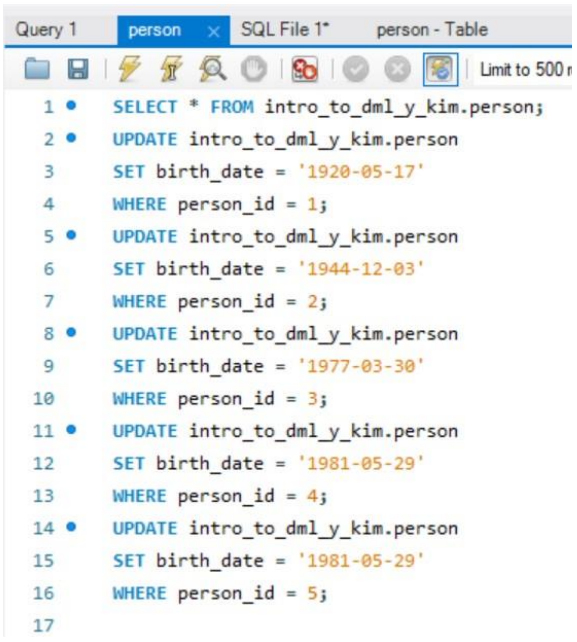
Add a new column to table, also add type and execute.

*ALTER TABLE Person*

*ADD birth_date datetime NULL;*

The **UPDATE** command is employed to alter values within existing rows. It allows for the modification of a single value in a specific column or multiple columns simultaneously. This can be applied to a single row or multiple rows. To selectively apply the UPDATE command to specific rows, a WHERE clause is utilized, similar to what is used in the SELECT statement. The WHERE clause serves to filter and determine which rows will be affected by the UPDATE operation. For singular row modifications, it is advisable to ensure that the WHERE clause is tailored to a single Primary Key or Unique Column value. While the WHERE clause is technically optional, its inclusion is strongly recommended to avoid unexpected outcomes. This is the example of how to update to birth_date column.

Filename: 03_Non_Null_Birth_Dates.jpg



```
SELECT * FROM intro_to_dml_y_kim.person;
UPDATE intro_to_dml_y_kim.person
SET birth_date = '1920-05-17'
WHERE person_id = 1;
UPDATE intro_to_dml_y_kim.person
SET birth_date = '1944-12-03'
WHERE person_id = 2;
UPDATE intro_to_dml_y_kim.person
SET birth_date = '1977-03-30'
WHERE person_id = 3;
UPDATE intro_to_dml_y_kim.person
SET birth_date = '1981-05-29'
WHERE person_id = 4;
UPDATE intro_to_dml_y_kim.person
SET birth_date = '1981-05-29'
WHERE person_id = 5;
```

Screenshot #4:

Set a default value for the birth_date column to the current timestamp using the following command:

*ALTER TABLE person*

*MODIFY COLUMN birth_date datetime NOT NULL DEFAULT CURRENT_TIMESTAMP;*

Then, attempt to insert a person and observe that they are assigned today's date as their birth date with the following command:

Type and execute:

*INSERT INTO Person*

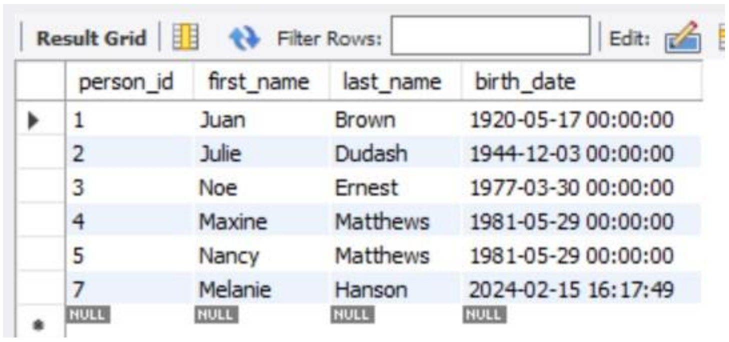
*(first_name, last_name)*

*VALUES*

*('Melanie', 'Hanson');*

Melanie Hanson will be added with today's date as the default value for the birth_date column.

Filename: 04_Default_Birth_Date.jpg



| person_id | first_name | last_name | birth_date |
|---|---|---|---|
| 1 | Juan | Brown | 1920-05-17 00:00:00 |
| 2 | Julie | Dudash | 1944-12-03 00:00:00 |
| 3 | Noe | Ernest | 1977-03-30 00:00:00 |
| 4 | Maxine | Matthews | 1981-05-29 00:00:00 |
| 5 | Nancy | Matthews | 1981-05-29 00:00:00 |
| 7 | Melanie | Hanson | 2024-02-15 16:17:49 |
| NULL | NULL | NULL | NULL |

Screenshot #5:

The **DELETE** command is utilized to eliminate complete rows from a table. Similar to the UPDATE command, the DELETE command incorporates a WHERE clause to specify the rows to be affected. Deleting a singular row requires a WHERE clause narrowed down to a single Primary Key or Unique Column value. While technically optional, it is highly advisable to include the WHERE clause to prevent unexpected outcomes.

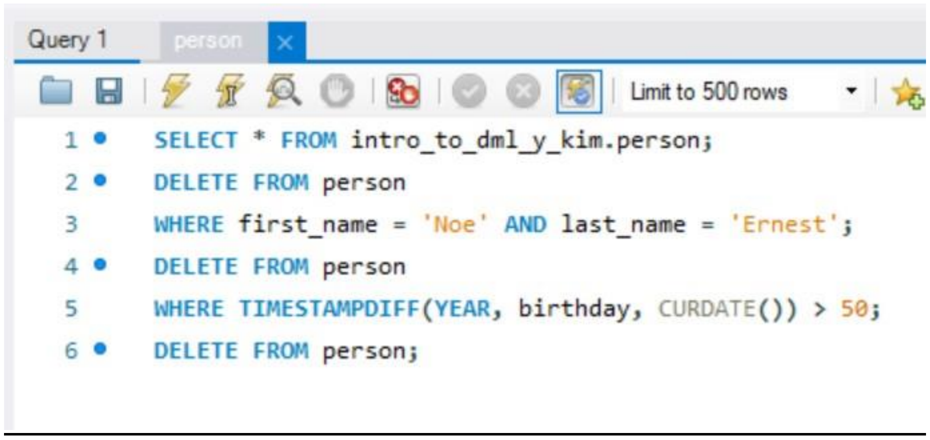
These commands are derived from the provided screenshots.

Write a DELETE command to delete just Noe Ernest.

Write a DELETE command to delete all people older than 50.

Write a DELETE command to delete everyone else (no WHERE clause).

Filename: 05_Delete_the_People.jpg



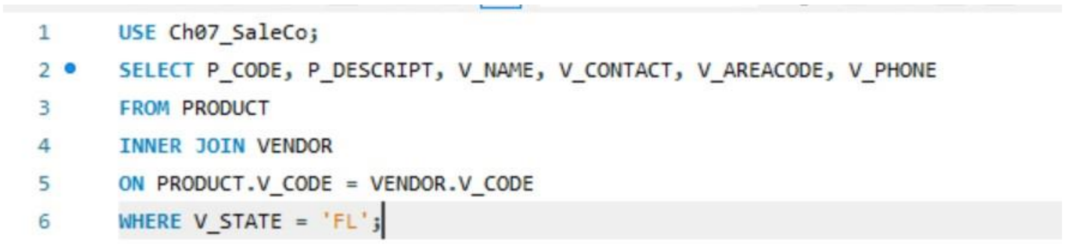
```
SELECT * FROM intro_to_dml_y_kim.person;
DELETE FROM person
WHERE first_name = 'Noe' AND last_name = 'Ernest';
DELETE FROM person
WHERE TIMESTAMPDIFF(YEAR, birthday, CURDATE()) > 50;
DELETE FROM person;
```

Screenshot #6:

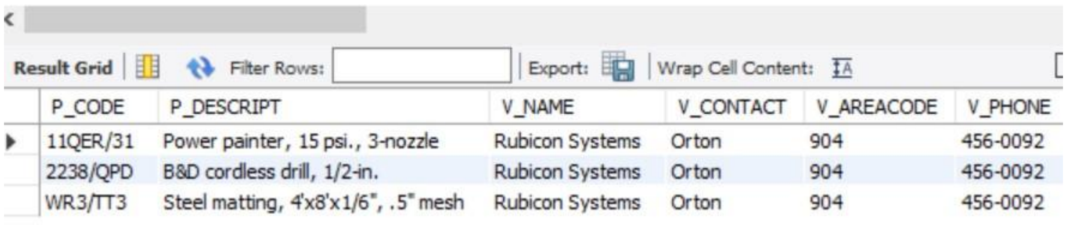
The **INNER JOIN** combines each row from the first table with corresponding rows from the second table, linking them based on the specified column values in the ON section.

List all products with their product code and description along with vendor details, including Vendor Company Name, Contact Person, area code, and phone number. Modify the query to include only products sold by vendors in Florida. (where V_STATE = 'FL')

Filename: 06_Products_from_Florida.jpg



```
USE Ch07_SaleCo;
SELECT P_CODE, P_DESCRIPT, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM PRODUCT
INNER JOIN VENDOR
ON PRODUCT.V_CODE = VENDOR.V_CODE
WHERE V_STATE = 'FL';
```



| P_CODE | P_DESCRIPT | V_NAME | V_CONTACT | V_AREACODE | V_PHONE |
|---|---|---|---|---|---|
| 11QER/31 | Power painter, 15 psi., 3-nozzle | Rubicon Systems | Orton | 904 | 456-0092 |
| 2238/QPD | B&D cordless drill, 1/2-in. | Rubicon Systems | Orton | 904 | 456-0092 |
| WR3/TT3 | Steel matting, 4'x8'x1/6", .5" mesh | Rubicon Systems | Orton | 904 | 456-0092 |

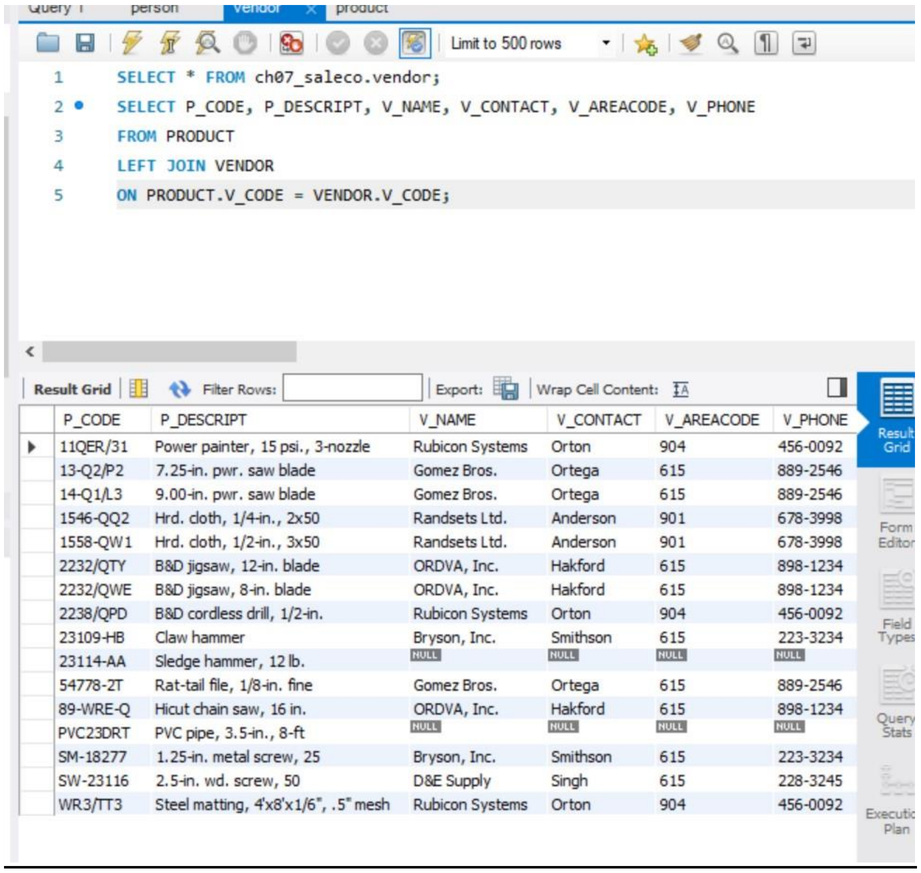
Screenshot #7:

**LEFT** and **RIGHT JOINs**, often termed **OUTER JOINs**, handle optional relationships where foreign keys are permitted to be NULL. In an INNER JOIN, rows in tables are matched based on values specified in the ON section. However, when foreign keys are optional and can be NULL, there might be no matches for these values in the other table. It's essential to note that NULL, by definition, lacks a value and cannot be matched with any other value.

**LEFT JOINs** is type of join in SQL that returns all records from the left table.

The query includes all records from the PRODUCT table. For each record, it tries to find a matching record in the VENDOR table based on the V_CODE. If a match is found, columns from the VENDOR table are included in the result; otherwise, the VENDOR columns will have NULL values.

Filename: 07_All_Products_and_Vendors_Optional.jpg



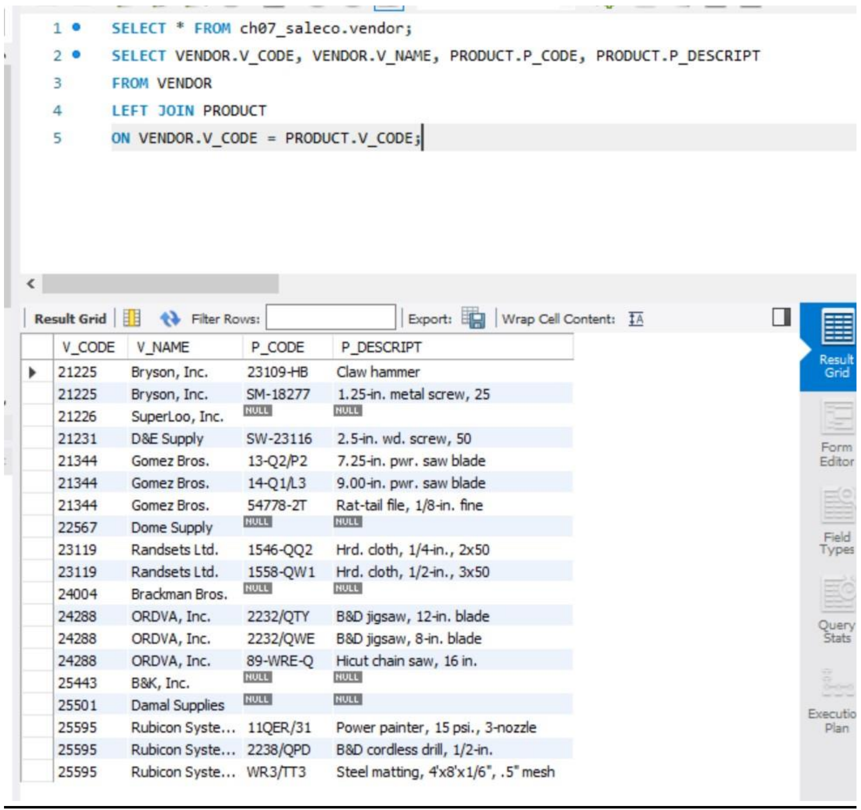
```
SELECT * FROM ch07_saleco.vendor;
SELECT P_CODE, P_DESCRIPT, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM PRODUCT
LEFT JOIN VENDOR
ON PRODUCT.V_CODE = VENDOR.V_CODE;
```

| P_CODE | P_DESCRIPT | V_NAME | V_CONTACT | V_AREACODE | V_PHONE |
|---|---|---|---|---|---|
| 11QER/31 | Power painter, 15 psi., 3-nozzle | Rubicon Systems | Orton | 904 | 456-0092 |
| 13-Q2/P2 | 7.25-in. pwr. saw blade | Gomez Bros. | Ortega | 615 | 889-2546 |
| 14-Q1/L3 | 9.00-in. pwr. saw blade | Gomez Bros. | Ortega | 615 | 889-2546 |
| 1546-QQ2 | Hrd. cloth, 1/4-in., 2x50 | Randsets Ltd. | Anderson | 901 | 678-3998 |
| 1558-QW1 | Hrd. cloth, 1/2-in., 3x50 | Randsets Ltd. | Anderson | 901 | 678-3998 |
| 2232/QTY | B&D jigsaw, 12-in. blade | ORDVA, Inc. | Hakford | 615 | 898-1234 |
| 2232/QWE | B&D jigsaw, 8-in. blade | ORDVA, Inc. | Hakford | 615 | 898-1234 |
| 2238/QPD | B&D cordless drill, 1/2-in. | Rubicon Systems | Orton | 904 | 456-0092 |
| 23109-HB | Claw hammer | Bryson, Inc. | Smithson | 615 | 223-3234 |
| 23114-AA | Sledge hammer, 12 lb. | NULL | NULL | NULL | NULL |
| 54778-2T | Rat-tail file, 1/8-in. fine | Gomez Bros. | Ortega | 615 | 889-2546 |
| 89-WRE-Q | Hicut chain saw, 16 in. | ORDVA, Inc. | Hakford | 615 | 898-1234 |
| PVC23DRT | PVC pipe, 3.5-in., 8-ft | NULL | NULL | NULL | NULL |
| SM-18277 | 1.25-in. metal screw, 25 | Bryson, Inc. | Smithson | 615 | 223-3234 |
| SW-23116 | 2.5-in. wd. screw, 50 | D&E Supply | Singh | 615 | 228-3245 |
| WR3/TT3 | Steel matting, 4'x8'x1/6", .5" mesh | Rubicon Systems | Orton | 904 | 456-0092 |

## Screenshot #8:

Get a list of vendors showing their Vendor Code and Company Name along with the products they sell (if any). If a vendor doesn't have any products, the product details will appear as NULL. The LEFT JOIN ensures all vendors are included, and any corresponding products are added to the list. If a vendor has no associated products, the PRODUCT columns will be NULL.

Filename: 08_All_Vendors_and_Products_Optional.jpg



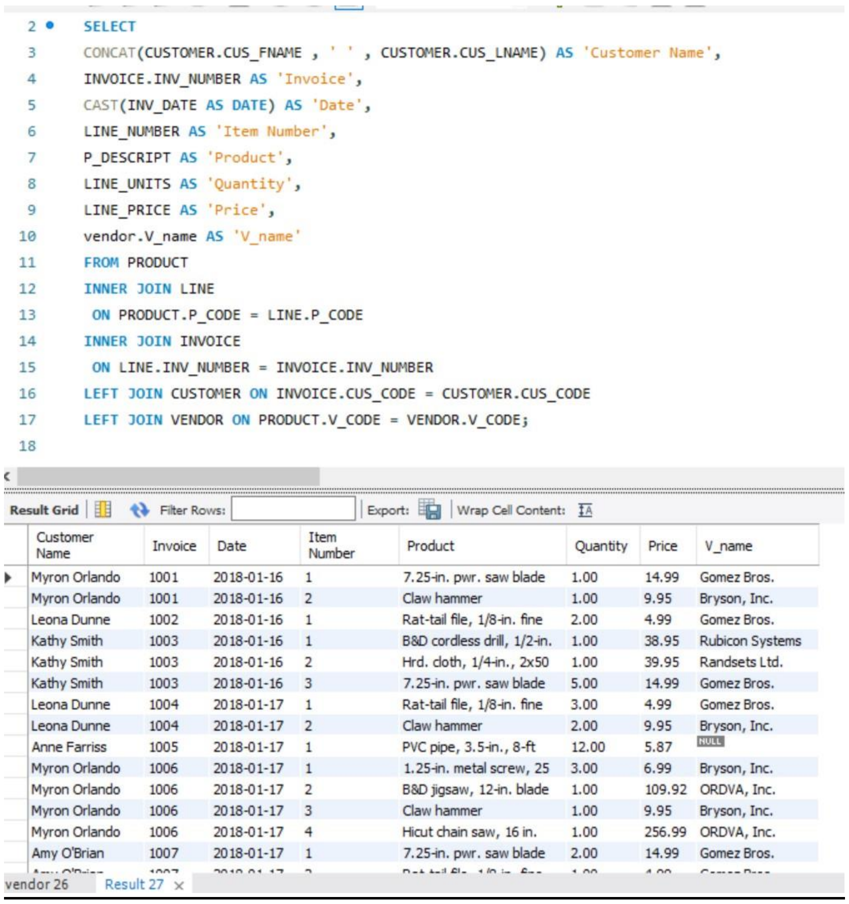
```sql
SELECT * FROM ch07_saleco.vendor;
SELECT VENDOR.V_CODE, VENDOR.V_NAME, PRODUCT.P_CODE, PRODUCT.P_DESCRIPT
FROM VENDOR
LEFT JOIN PRODUCT
ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

| V_CODE | V_NAME | P_CODE | P_DESCRIPT |
|---|---|---|---|
| 21225 | Bryson, Inc. | 23109-HB | Claw hammer |
| 21225 | Bryson, Inc. | SM-18277 | 1.25-in. metal screw, 25 |
| 21226 | SuperLoo, Inc. | NULL | NULL |
| 21231 | D&E Supply | SW-23116 | 2.5-in. wd. screw, 50 |
| 21344 | Gomez Bros. | 13-Q2/P2 | 7.25-in. pwr. saw blade |
| 21344 | Gomez Bros. | 14-Q1/L3 | 9.00-in. pwr. saw blade |
| 21344 | Gomez Bros. | 54778-2T | Rat-tail file, 1/8-in. fine |
| 22567 | Dome Supply | NULL | NULL |
| 23119 | Randsets Ltd. | 1546-QQ2 | Hrd. cloth, 1/4-in., 2x50 |
| 23119 | Randsets Ltd. | 1558-QW1 | Hrd. cloth, 1/2-in., 3x50 |
| 24004 | Brackman Bros. | NULL | NULL |
| 24288 | ORDVA, Inc. | 2232/QTY | B&D jigsaw, 12-in. blade |
| 24288 | ORDVA, Inc. | 2232/QWE | B&D jigsaw, 8-in. blade |
| 24288 | ORDVA, Inc. | 89-WRE-Q | Hicut chain saw, 16 in. |
| 25443 | B&K, Inc. | NULL | NULL |
| 25501 | Damal Supplies | NULL | NULL |
| 25595 | Rubicon Syste... | 11QER/31 | Power painter, 15 psi., 3-nozzle |
| 25595 | Rubicon Syste... | 2238/QPD | B&D cordless drill, 1/2-in. |
| 25595 | Rubicon Syste... | WR3/TT3 | Steel matting, 4'x8'x1/6", .5" mesh |

Screenshot #9:

Select the necessary values and use INNER JOIN to combine columns from both the LINE and INVOICE tables. Present the customer's full name in a single column and include the vendor's name. If there is no current vendor for an invoice, leave the Vendor Name as NULL.

Filename: 09_All_Invoices_with_Vendors_and_Customers.jpg



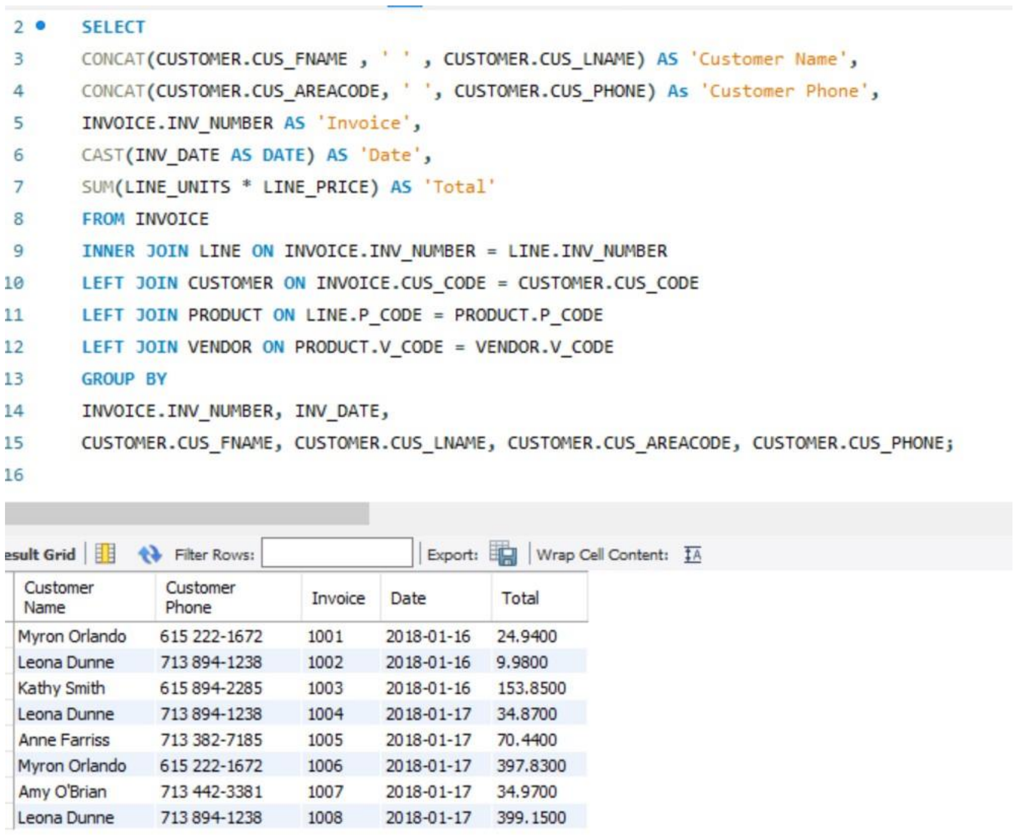
```sql
SELECT
CONCAT(CUSTOMER.CUS_FNAME , ' ' , CUSTOMER.CUS_LNAME) AS 'Customer Name',
INVOICE.INV_NUMBER AS 'Invoice',
CAST(INV_DATE AS DATE) AS 'Date',
LINE_NUMBER AS 'Item Number',
P_DESCRIPT AS 'Product',
LINE_UNITS AS 'Quantity',
LINE_PRICE AS 'Price',
vendor.V_name AS 'V_name'
FROM PRODUCT
INNER JOIN LINE
 ON PRODUCT.P_CODE = LINE.P_CODE
INNER JOIN INVOICE
 ON LINE.INV_NUMBER = INVOICE.INV_NUMBER
LEFT JOIN CUSTOMER ON INVOICE.CUS_CODE = CUSTOMER.CUS_CODE
LEFT JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE;
```

| Customer Name | Invoice | Date | Item Number | Product | Quantity | Price | V_name |
|---|---|---|---|---|---|---|---|
| Myron Orlando | 1001 | 2018-01-16 | 1 | 7.25-in. pwr. saw blade | 1.00 | 14.99 | Gomez Bros. |
| Myron Orlando | 1001 | 2018-01-16 | 2 | Claw hammer | 1.00 | 9.95 | Bryson, Inc. |
| Leona Dunne | 1002 | 2018-01-16 | 1 | Rat-tail file, 1/8-in. fine | 2.00 | 4.99 | Gomez Bros. |
| Kathy Smith | 1003 | 2018-01-16 | 1 | B&D cordless drill, 1/2-in. | 1.00 | 38.95 | Rubicon Systems |
| Kathy Smith | 1003 | 2018-01-16 | 2 | Hrd. cloth, 1/4-in., 2x50 | 1.00 | 39.95 | Randsets Ltd. |
| Kathy Smith | 1003 | 2018-01-16 | 3 | 7.25-in. pwr. saw blade | 5.00 | 14.99 | Gomez Bros. |
| Leona Dunne | 1004 | 2018-01-17 | 1 | Rat-tail file, 1/8-in. fine | 3.00 | 4.99 | Gomez Bros. |
| Leona Dunne | 1004 | 2018-01-17 | 2 | Claw hammer | 2.00 | 9.95 | Bryson, Inc. |
| Anne Farriss | 1005 | 2018-01-17 | 1 | PVC pipe, 3.5-in., 8-ft | 12.00 | 5.87 | NULL |
| Myron Orlando | 1006 | 2018-01-17 | 1 | 1.25-in. metal screw, 25 | 3.00 | 6.99 | Bryson, Inc. |
| Myron Orlando | 1006 | 2018-01-17 | 2 | B&D jigsaw, 12-in. blade | 1.00 | 109.92 | ORDVA, Inc. |
| Myron Orlando | 1006 | 2018-01-17 | 3 | Claw hammer | 1.00 | 9.95 | Bryson, Inc. |
| Myron Orlando | 1006 | 2018-01-17 | 4 | Hicut chain saw, 16 in. | 1.00 | 256.99 | ORDVA, Inc. |
| Amy O'Brian | 1007 | 2018-01-17 | 1 | 7.25-in. pwr. saw blade | 2.00 | 14.99 | Gomez Bros. |

Screenshot #10:

Get the customer's full name using concat, their complete phone number, invoice number, invoice date, and the total invoice amount, calculated as the sum of each product's price multiplied by its quantity. You need to use GROUP BY on the invoice to avoid getting a separate row for each item in the invoice.

Filename: 10_All_Customer_Invoice_Totals.jpg



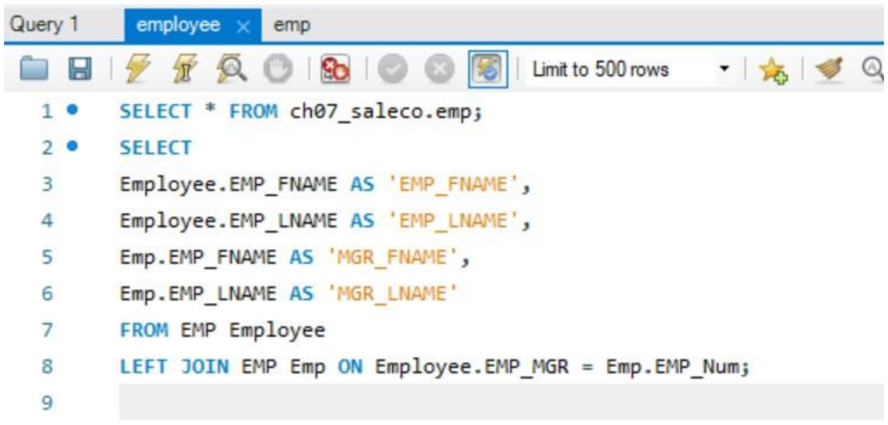
```
SELECT
CONCAT(CUSTOMER.CUS_FNAME , ' ' , CUSTOMER.CUS_LNAME) AS 'Customer Name',
CONCAT(CUSTOMER.CUS_AREACODE, ' ', CUSTOMER.CUS_PHONE) As 'Customer Phone',
INVOICE.INV_NUMBER AS 'Invoice',
CAST(INV_DATE AS DATE) AS 'Date',
SUM(LINE_UNITS * LINE_PRICE) AS 'Total'
FROM INVOICE
INNER JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
LEFT JOIN CUSTOMER ON INVOICE.CUS_CODE = CUSTOMER.CUS_CODE
LEFT JOIN PRODUCT ON LINE.P_CODE = PRODUCT.P_CODE
LEFT JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE
GROUP BY
INVOICE.INV_NUMBER, INV_DATE,
CUSTOMER.CUS_FNAME, CUSTOMER.CUS_LNAME, CUSTOMER.CUS_AREACODE, CUSTOMER.CUS_PHONE;
```

| Customer Name | Customer Phone | Invoice | Date | Total |
|---|---|---|---|---|
| Myron Orlando | 615 222-1672 | 1001 | 2018-01-16 | 24.9400 |
| Leona Dunne | 713 894-1238 | 1002 | 2018-01-16 | 9.9800 |
| Kathy Smith | 615 894-2285 | 1003 | 2018-01-16 | 153.8500 |
| Leona Dunne | 713 894-1238 | 1004 | 2018-01-17 | 34.8700 |
| Anne Farriss | 713 382-7185 | 1005 | 2018-01-17 | 70.4400 |
| Myron Orlando | 615 222-1672 | 1006 | 2018-01-17 | 397.8300 |
| Amy O'Brian | 713 442-3381 | 1007 | 2018-01-17 | 34.9700 |
| Leona Dunne | 713 894-1238 | 1008 | 2018-01-17 | 399.1500 |

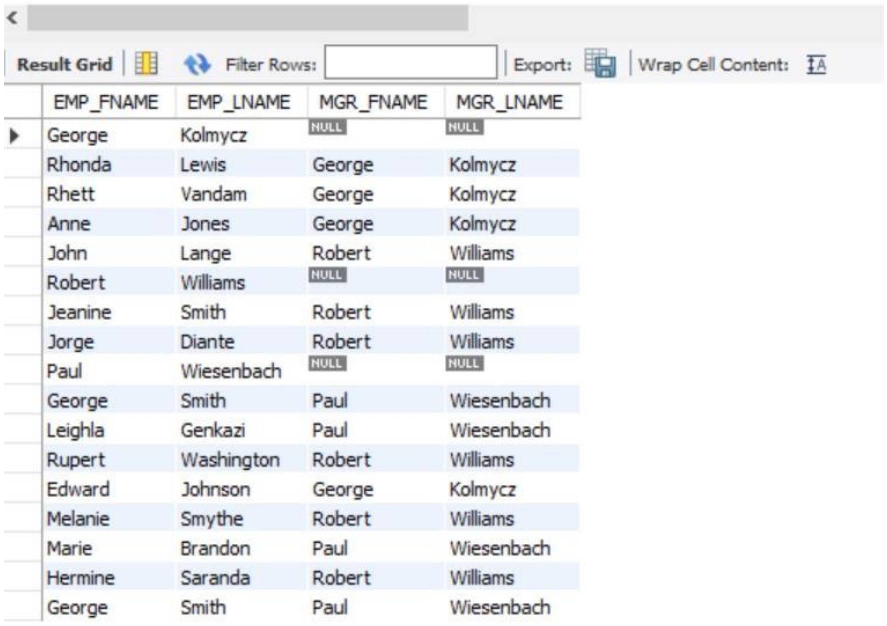
Screenshot #11:

Generate a query to showcase all employees from the EMP table along with their respective managers. Display the full name of employees and managers. If employees do not have managers, their manager's name will be presented as NULL.

Filename: 11_All_Employees_and_their_Managers.jpg



```
SELECT * FROM ch07_saleco.emp;
SELECT
Employee.EMP_FNAME AS 'EMP_FNAME',
Employee.EMP_LNAME AS 'EMP_LNAME',
Emp.EMP_FNAME AS 'MGR_FNAME',
Emp.EMP_LNAME AS 'MGR_LNAME'
FROM EMP Employee
LEFT JOIN EMP Emp ON Employee.EMP_MGR = Emp.EMP_Num;
```

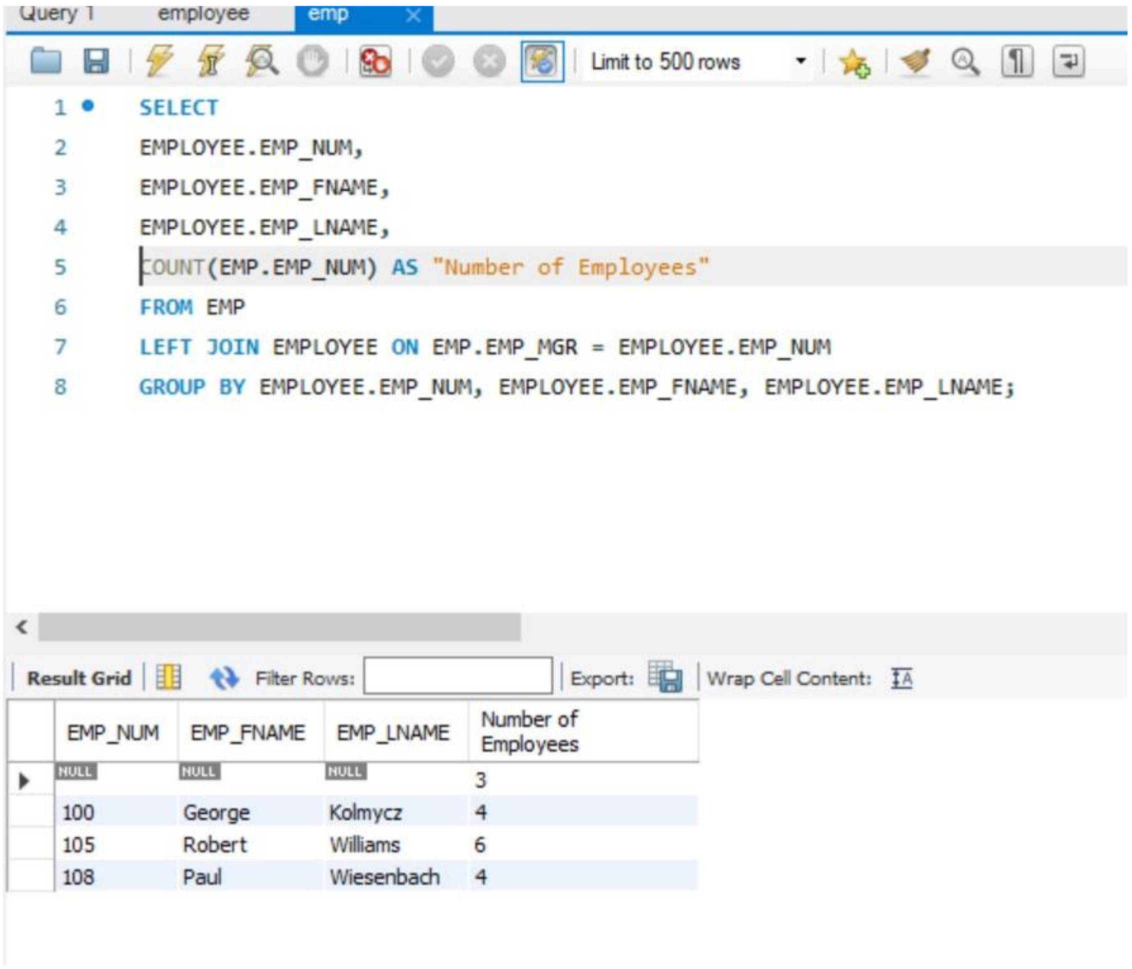


| EMP_FNAME | EMP_LNAME | MGR_FNAME | MGR_LNAME |
|---|---|---|---|
| George | Kolmycz | NULL | NULL |
| Rhonda | Lewis | George | Kolmycz |
| Rhett | Vandam | George | Kolmycz |
| Anne | Jones | George | Kolmycz |
| John | Lange | Robert | Williams |
| Robert | Williams | NULL | NULL |
| Jeanine | Smith | Robert | Williams |
| Jorge | Diante | Robert | Williams |
| Paul | Wiesenbach | NULL | NULL |
| George | Smith | Paul | Wiesenbach |
| Leighla | Genkazi | Paul | Wiesenbach |
| Rupert | Washington | Robert | Williams |
| Edward | Johnson | George | Kolmycz |
| Melanie | Smythe | Robert | Williams |
| Marie | Brandon | Paul | Wiesenbach |
| Hermine | Saranda | Robert | Williams |
| George | Smith | Paul | Wiesenbach |

Screenshot #12:

Adjust the query to include a count of employees for each manager. Additionally, make sure to include employees without managers in the list. Display the full names of employees and their managers, showing NULL for those without managers.

Filename: 12_How_Many_Employees_Do_You_Manage.jpg



```
SELECT
EMPLOYEE.EMP_NUM,
EMPLOYEE.EMP_FNAME,
EMPLOYEE.EMP_LNAME,
COUNT(EMP.EMP_NUM) AS "Number of Employees"
FROM EMP
LEFT JOIN EMPLOYEE ON EMP.EMP_MGR = EMPLOYEE.EMP_NUM
GROUP BY EMPLOYEE.EMP_NUM, EMPLOYEE.EMP_FNAME, EMPLOYEE.EMP_LNAME;
```

| EMP_NUM | EMP_FNAME | EMP_LNAME | Number of Employees |
|---|---|---|---|
| NULL | NULL | NULL | 3 |
| 100 | George | Kolmycz | 4 |
| 105 | Robert | Williams | 6 |
| 108 | Paul | Wiesenbach | 4 |

<u>Screenshot #13:</u>

The text file contains a Render link connecting SQL with freeDB. This link facilitates data interaction between the two databases.

Filename: 13_Render_Web_Users_Site.txt

13_Render_Web_
Users_Site.txt

<u>Screenshot #14:</u>

The code zipfile includes a collection of code that connects the web user database to the rendering server. This code allows for the addition and deletion of new user values on the server. The information entered on the server is automatically updated in SQL tables..

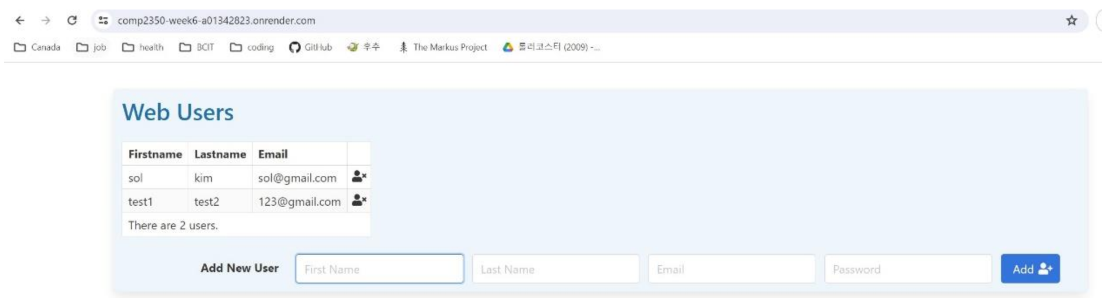
Filename: 14_Render_Web_Users_Code.zip.zip

14_Render_Web_Users_Code.zip.zip

<u>Screenshot #15:</u>

This is an example for the 14th screenshot description. I added a new user function, enabling data input directly from the browser to the database. This practice helped me understand how data is transferred between the server and the database.

Filename: 15_Reder_Web_Users_Screenshot.jpg



Web Users

| Firstname | Lastname | Email | |
|---|---|---|---|
| sol | kim | sol@gmail.com | |
| test1 | test2 | 123@gmail.com | |
| There are 2 users. | | | |