Brandon Hackett
Darnell Lewis
Derek Livermont
Lindsey Yin

# CSD 310 Group 5

# Bacchus Case Study

Stan and Davis Bacchus, brothers and co-owners of Bacchus Winery, inherited the family winery from their father, George. Eager to modernize the business and enhance product quality and customer service, they retained the existing personnel, including Janet Collins (Finance and Payroll), Roz Murphy (Marketing), Bob Ulrich (Marketing Assistant), Henry Doyle (Production Manager), and Maria Costanza (Distribution).

The winery produces Merlot, Cabernet, Chablis, and Chardonnay wines, cultivating their own grapes. Facing challenges in supply management, Stan and Davis seek efficient methods for tracking and ordering supplies, contemplating internet-based solutions. Maria, responsible for distribution, envisions online ordering and shipment tracking for distributors.

As the yearly business snapshot approaches, the owners require insights into supplier reliability, wine distribution performance, and employee working hours. Key questions include the timeliness of supplier deliveries, wine sales analysis, and employee work hours over the past four quarters.

The case study emphasizes the need for a comprehensive database system to streamline operations, monitor supply chains, enhance distributor interactions, and assess employee productivity. It sets the stage for implementing technology solutions to propel Bacchus Winery into a new era of efficiency and success.

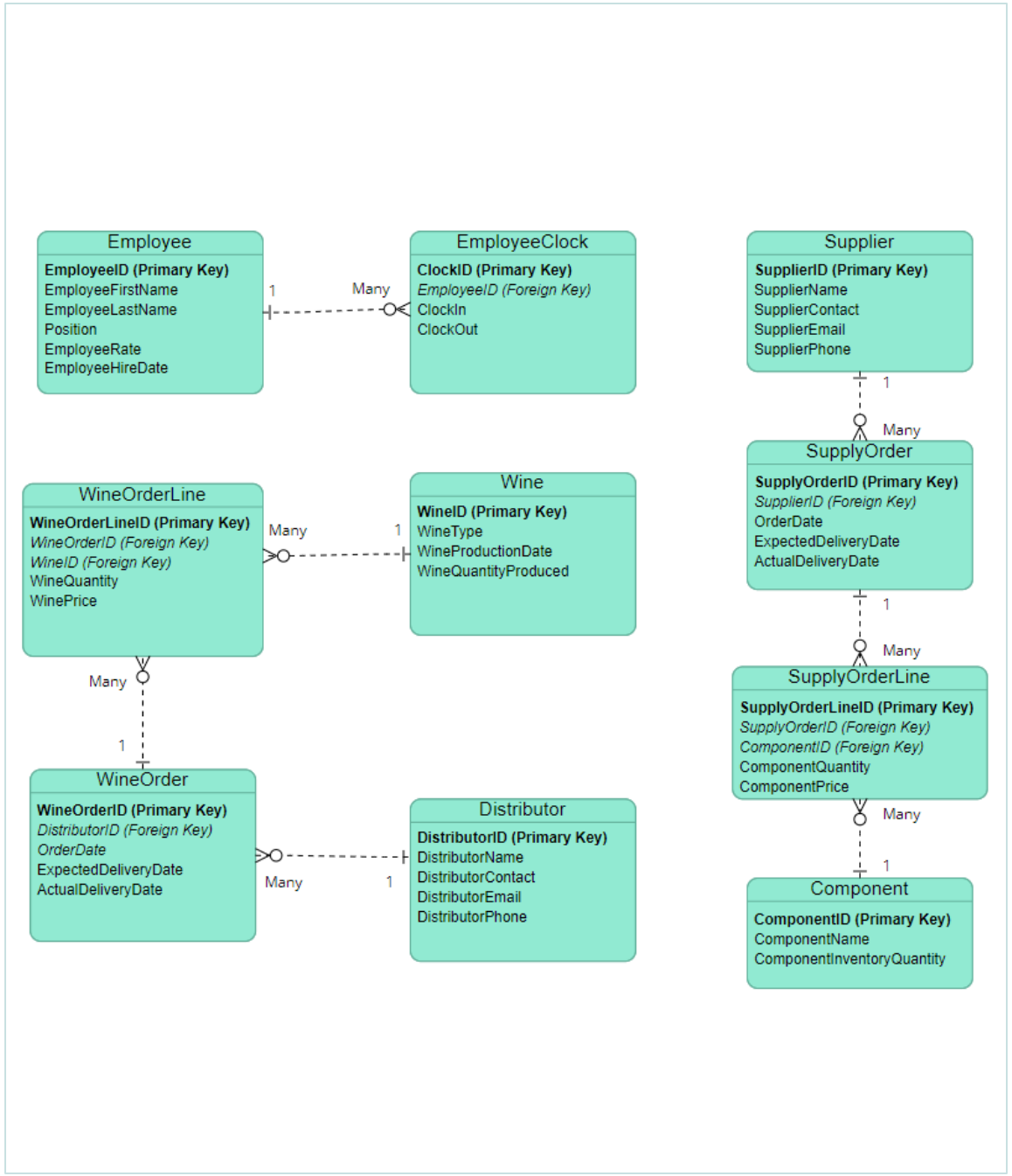
# Entity Relationship Diagram

Based on the needs stated by Stan and Davis, the database can be broken down into three main areas: employees, suppliers, and wines.

On the employee side, we maintain an employee and a clock-in table that can be queried to form time sheets and pull labor costs.

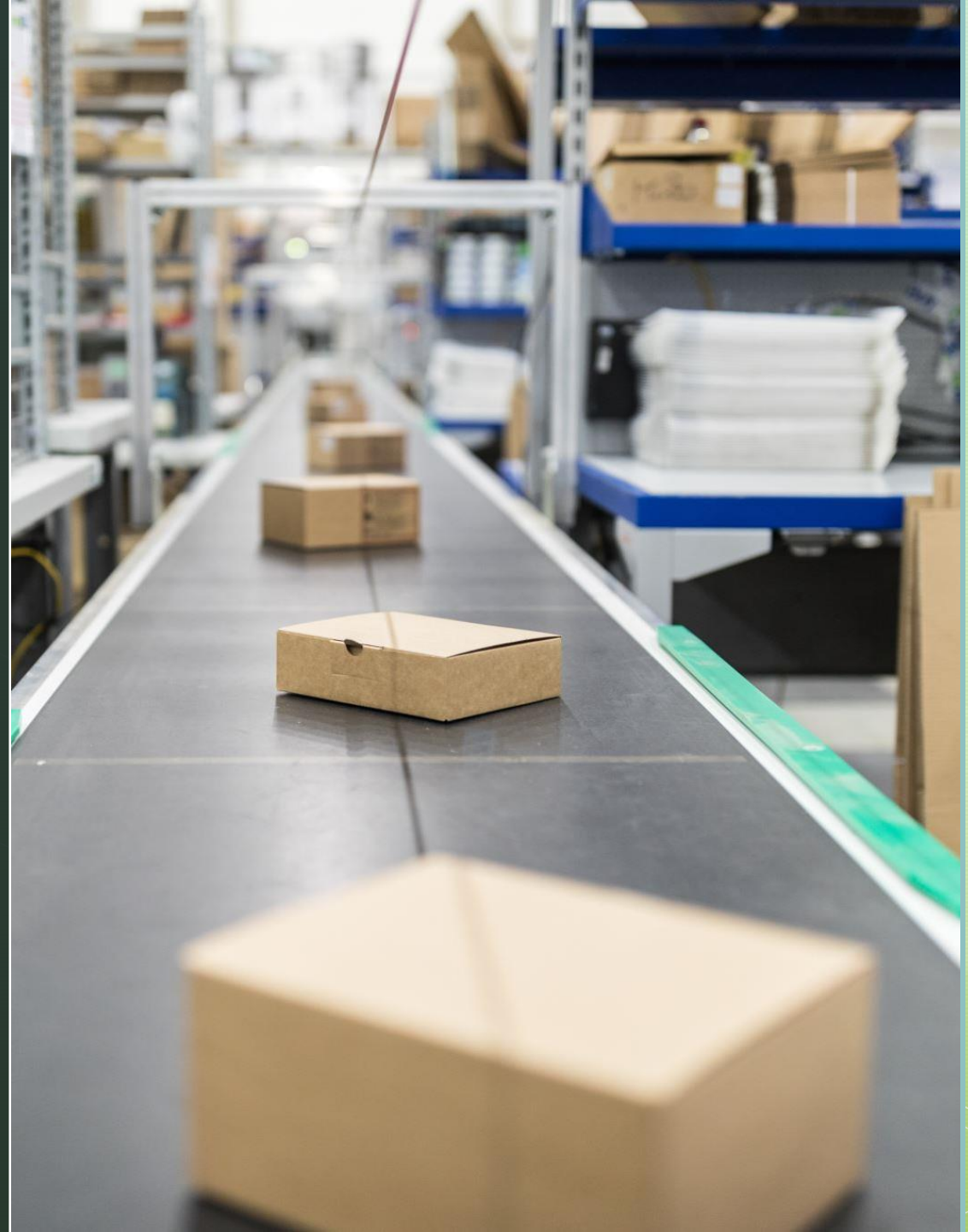
The supplier tables allow tracking of supply orders as well as components on-hand.

Finally, the wine tables track wines from production through the shipment process to our distributors.



# ERD Continued

- Using this database design, we intended to address the employee's time or yearly snapshot, wine deliveries, and supply deliveries.
- We purposely made our ERD more granular to track line items in wine orders and supplies. This also follows good database design.



# Assumptions

We structured our ERD in a way that line items being cancelled wouldn't impact the database.

An expected delivery schedule is communicated to vendors and buyers to establish an on-time delivery threshold.

An online system will streamline the tracking and ordering process, providing real-time updates on inventory levels and facilitating efficient ordering.

Implementing an online ordering system for distributors will enhance communication and streamline the distribution process.

Accurate timekeeping systems are in place, and employee work hours directly contribute to payroll, production, and overall business operations.

# Table Creation

```sql
-- create the employee table
CREATE TABLE employee (
    emp_id      INT             NOT NULL        AUTO_INCREMENT,
    emp_first    VARCHAR(50)      NOT NULL,
    emp_last    VARCHAR(50)      NOT NULL,
    emp_position    VARCHAR(50)      NOT NULL,
    emp_rate DECIMAL(6,2) NOT NULL,
    emp_hire_date    DATE           NOT NULL,

    PRIMARY KEY(emp_id)
);

-- create the employee_clock table
CREATE TABLE employee_clock (
    clock_id     INT             NOT NULL        AUTO_INCREMENT,
    emp_id    INT      NOT NULL,
    clock_in  DATETIME    NOT NULL,
    clock_out DATETIME,
    PRIMARY KEY(clock_id),
    CONSTRAINT fk_employee
    FOREIGN KEY(emp_id)
        REFERENCES employee(emp_id)
);
-- create the supplier table
CREATE TABLE supplier (
    supplier_id    INT             NOT NULL        AUTO_INCREMENT,
    supplier_name  VARCHAR(75)      NOT NULL,
    supplier_contact    VARCHAR(75),
    supplier_email VARCHAR(75),
    supplier_phone VARCHAR(15) NOT NULL,
    PRIMARY KEY(supplier_id)
);
-- create component table
CREATE TABLE component (
    component_id    INT             NOT NULL        AUTO_INCREMENT,
    component_name  VARCHAR(75)      NOT NULL,
    component_inv_qty INT NOT NULL,
    PRIMARY KEY(component_id)
);
```

```sql
-- create supply_order table
CREATE TABLE supply_order (
    supply_order_id    INT             NOT NULL        AUTO_INCREMENT,
    supplier_id  INT      NOT NULL,
    order_date DATE NOT NULL,
    expected_delivery DATE,
    actual_delivery DATE,
    PRIMARY KEY (supply_order_id),
    CONSTRAINT fk_supplier
    FOREIGN KEY(supplier_id)
        REFERENCES supplier(supplier_id)
);

CREATE TABLE supply_order_line (
    supply_order_line_id    INT             NOT NULL        AUTO_INCREMENT
    supply_order_id  INT      NOT NULL,
    component_id INT NOT NULL,
    component_quantity INT NOT NULL,
    component_price DECIMAL(8,2) NOT NULL,
    PRIMARY KEY (supply_order_line_id),
    CONSTRAINT fk_order_id
    FOREIGN KEY(supply_order_id)
        REFERENCES supply_order(supply_order_id),
    CONSTRAINT fk_component_id
    FOREIGN KEY(component_id)
        REFERENCES component(component_id)
);

CREATE TABLE distributor (
    distributor_id    INT             NOT NULL        AUTO_INCREMENT,
    distributor_name  VARCHAR(75)      NOT NULL,
    distributor_contact    VARCHAR(75),
    distributor_email VARCHAR(75),
    distributor_phone VARCHAR(15) NOT NULL,
    PRIMARY KEY(distributor_id)
);
-- create wine table
CREATE TABLE wine (
    wine_id    INT             NOT NULL        AUTO_INCREMENT,
    wine_type  VARCHAR(75)      NOT NULL,
    wine_production_date DATE NOT NULL,
    wine_qty_produced INT NOT NULL,
    PRIMARY KEY(wine_id)
);
```

```sql
-- create wine_order table
CREATE TABLE wine_order (
    wine_order_id    INT    NOT NULL        AUTO_INCREMENT,
    distributor_id  INT     NOT NULL,
    order_date DATE NOT NULL,
    expected_delivery DATE,
    actual_delivery DATE,
    PRIMARY KEY (wine_order_id),
    CONSTRAINT fk_distributor
    FOREIGN KEY(distributor_id)
        REFERENCES distributor(distributor_id)
);

CREATE TABLE wine_order_line (
    wine_order_line_id    INT             NOT NULL        AUTO_INCREMENT,
    wine_order_id  INT      NOT NULL,
    wine_id INT NOT NULL,
    wine_quantity INT NOT NULL,
    wine_price DECIMAL(8,2) NOT NULL,
    PRIMARY KEY (wine_order_line_id),
    CONSTRAINT fk_wine_order_id
    FOREIGN KEY(wine_order_id)
        REFERENCES wine_order(wine_order_id),
    CONSTRAINT fk_wine_id
    FOREIGN KEY(wine_id)
        REFERENCES wine(wine_id)
);
```

# Sample Data

```sql
INSERT INTO employee
    VALUES
    (NULL, "Debrah", "Messing","Production", 18.00,'2020-03-20'),
    (NULL, "Brad", "Pitt","Shift Lead", 20.00,'2018-08-21'),
    (NULL, "Tina", "Fey","Custodian", 17.00,'2019-04-07'),
    (NULL, "Debbie", "Reynolds","Production", 18.00,'2021-03-02'),
    (NULL, "Alec", "Baldwin","Production", 18.00,'2022-05-28'),
    (NULL, "Amy", "Poehler","Quality Assurance", 23.00,'2022-05-28');

INSERT INTO employee_clock
    VALUES
    (NULL, 1,'2023-11-27 08:00','2023-11-27 16:00'),
    (NULL, 2,'2023-11-27 07:30','2023-11-27 15:30'),
    (NULL, 3,'2023-11-27 12:00','2023-11-27 20:00'),
    (NULL, 4,'2023-11-27 08:00','2023-11-27 16:00'),
    (NULL, 5,'2023-11-27 08:00','2023-11-27 16:00'),
    (NULL, 6,'2023-11-27 09:00','2023-11-27 17:00');

INSERT INTO supplier
    VALUES
    (NULL, "Wine Time","Joe Smith", "joesmith@winetime.com", "8003224567"),
    (NULL, "Boxes and Things","Mary Johnson", "mjohnson@boxesnthings.com", "8004567892"),
    (NULL, "Brewing Buds","Jeff Field", "jeff.field@brewingbuds.com", "6053725555");

INSERT INTO supply_order
    VALUES
    (NULL, 1, '2023-11-01','2023-11-08','2023-11-09'),
    (NULL, 1, '2023-11-05','2023-11-12','2023-11-13'),
    (NULL, 2, '2023-11-02','2023-11-06','2023-11-06'),
    (NULL, 2, '2023-11-10','2023-11-16','2023-11-15'),
    (NULL, 3, '2023-11-03','2023-11-06','2023-11-07'),
    (NULL, 3, '2023-11-28','2023-12-02', NULL);

INSERT INTO component
    VALUES
    (NULL, "750ml Glass Bottle", 200),
    (NULL, "Standard Cork", 500),
    (NULL, "Labels", 1000),
    (NULL, "12-bottle Cardboard Box", 100),
    (NULL, "225 Liter Vat ", 10),
    (NULL, "Vinyl Tubing Ft.", 300);
```

```sql
INSERT INTO supply_order_line
    VALUES
    (NULL, 1, 1, 300, 0.98),
    (NULL, 2, 2, 200, 0.38),
    (NULL, 3, 3, 500, 0.22),
    (NULL, 4, 4, 100, 1.25),
    (NULL, 5, 5, 20, 638),
    (NULL, 6, 6, 200, 1.20);

INSERT INTO wine
    VALUES
    (NULL, "Chardonnay", '2023-09-15', 300),
    (NULL, "Chablis", '2023-08-15', 300),
    (NULL, "Merlot", '2023-08-30', 300),
    (NULL, "Cabernet", '2023-09-05', 300),
    (NULL, "Red Blend", '2023-08-20', 300),
    (NULL, "White Blend", '2023-10-01', 300);

INSERT INTO distributor
    VALUES
    (NULL, "Ben's Distribution Co", "Jeff Stevens", "jstevens@bensdistco.com", "6884598764"),
    (NULL, "Larry's Liquor Store", "Larry Jones", NULL, "6235761234"),
    (NULL, "Betty's Wine and Spirits Wholesale", "Jessica Lee", "jlee@bettyswinenspirits.com","6665551234"),
    (NULL, "Cask and Cork Wine Distribution", "Sam Rogers", "srogers@caskncork.com","3654511572"),
    (NULL, "Wine World", "Jeff Goldblum", "jeff.goldblum@wineworld.com","8887776543"),
    (NULL, "In Vino Veritas", "Vicki Vivacious", "v.vivacious@invinoveritas.com","6055558888");

INSERT INTO wine_order
    VALUES
    (NULL, 1, '2023-11-02','2023-11-09','2023-11-10'),
    (NULL, 2, '2023-11-04','2023-11-11','2023-11-12'),
    (NULL, 3, '2023-11-04','2023-11-08','2023-11-08'),
    (NULL, 4, '2023-11-11','2023-11-17','2023-11-17'),
    (NULL, 5, '2023-11-02','2023-11-07','2023-11-07'),
    (NULL, 6, '2023-11-30','2023-12-03', NULL);

INSERT INTO wine_order_line
    VALUES
    (NULL, 1, 1, 120, 24.50),
    (NULL, 2, 2, 24, 24.50),
    (NULL, 3, 3, 60, 24.50),
    (NULL, 4, 4, 100, 24.50),
    (NULL, 5, 5, 72, 24.50),
    (NULL, 6, 6, 96, 24.50);
```

# Business Reports

Based on the case study, we put together reports that would be meaningful to Bacchus Winery. These include:

- Total Sales By Wines
- Late Orders
- Wine Type by Distributor
- Employee Time Report

# Total Sales by Wine Report

```
-- Total Orders and Sales by Wine Type --
Wine: Chablis
Quantity: 24
Total Sales: 588.00

Wine: Merlot
Quantity: 60
Total Sales: 1470.00

Wine: Red Blend
Quantity: 72
Total Sales: 1764.00

Wine: White Blend
Quantity: 96
Total Sales: 2352.00

Wine: Cabernet
Quantity: 100
Total Sales: 2450.00

Wine: Chardonnay
Quantity: 120
Total Sales: 2940.00

Press Enter to Exit...
```

```python
from mysql.connector import errorcode

config = {
    "user":"bacchus_user",
    "password":"ILoveWine!",
    "host":"127.0.0.1",
    "database":"bacchus",
    "raise_on_warnings": True
}

try:
    db = mysql.connector.connect(**config)
    print("\n Database user {} connected to MySQL on host {} with database {}".format(config["user"], config["host"], config["database"]))

except mysql.connector.Error as err:
    if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("  The supplied username or password are invalid")

    elif err.errno == errorcode.ER_BAD_DB_ERROR:
        print("  The specified database does not exist")

    else:
        print(err)

cursor = db.cursor()


print("-- Total Orders and Sales by Wine Type --")

# Define the SQL query to group by one column and calculate the sum of another column
sql_query = "SELECT wine_order_line.wine_id, wine.wine_type, SUM(wine_order_line.wine_quantity), SUM(wine_order_line.wine_quantity * wine_order_line.wine_price) \
             FROM wine_order_line \
             JOIN wine ON wine.wine_id = wine_order_line.wine_id \
             GROUP BY wine_order_line.wine_id \
             ORDER BY SUM(wine_order_line.wine_quantity * wine_order_line.wine_price);"

# Execute the query
cursor.execute(sql_query)

# Fetch all the rows
result = cursor.fetchall()

# Display the results
for row in result:
    print("Wine: {}\nQuantity: {}\nTotal Sales: {}\n".format(row[1], row[2], row[3]))
```

This Python script generates a report that provides information on total wine sales by type. It achieves this by retrieving data from the `wine_order_line` and `wine` tables through SQL joins. The report displays the Each type of wine, quantity sold and the calculated sales for that wine. The report is sorted by lowest sales to highest. This report can be used by the owners to understand which wines are preforming the lowest and which wines are selling well.

# Late Orders Report

```
cursor = db.cursor()

print("-- LATE WINE ORDERS --")
cursor.execute("SELECT distributor_name, order_date, DATEDIFF(actual_delivery,expected_delivery) "
               "FROM wine_order INNER JOIN distributor ON wine_order.distributor_id=distributor.distributor_id;")
orders = cursor.fetchall()
for order in orders:
    try:
        if int(order[2]) > 0:
            print("Distributor: {}\nOrder Date: {}\nExpected vs. Actual Delivery: {} day(s) late\n".format(order[0],order[1],order[2]))
    except:
        continue

print("-- LATE SUPPLY ORDERS --")
cursor.execute("SELECT supplier_name, order_date, DATEDIFF(actual_delivery,expected_delivery) "
               "FROM supply_order INNER JOIN supplier ON supply_order.supplier_id=supplier.supplier_id;")
orders = cursor.fetchall()
for order in orders:
    try:
        if int(order[2]) > 0:
            print("Supplier: {}\nOrder Date: {}\nExpected vs. Actual Delivery: {} day(s) late\n".format(order[0],order[1],order[2]))
    except:
        continue

input("Press Enter to Exit...")
db.close()
```

```
-- LATE WINE ORDERS --
Distributor: Ben's Distribution Co
Order Date: 2023-11-02
Expected vs. Actual Delivery: 1 day(s) late

Distributor: Larry's Liquor Store
Order Date: 2023-11-04
Expected vs. Actual Delivery: 1 day(s) late

-- LATE SUPPLY ORDERS --
Supplier: Wine Time
Order Date: 2023-11-01
Expected vs. Actual Delivery: 1 day(s) late

Supplier: Wine Time
Order Date: 2023-11-05
Expected vs. Actual Delivery: 1 day(s) late

Supplier: Brewing Buds
Order Date: 2023-11-03
Expected vs. Actual Delivery: 1 day(s) late
```

This report pulls up every delivery where the actual delivery happened after the expected delivery. This pulls up every delivery within the sample data but could be easily adjusted into a front end where managers can enter specific periods that would be fed into a WHERE statement to narrow down results.

# Wine Types by Distributor Report

```
-- DISTRIBUTOR AND WINE INFORMATION --
Distributor: Ben's Distribution Co
Wine Type: Chardonnay

Distributor: Larry's Liquor Store
Wine Type: Chablis

Distributor: Betty's Wine and Spirits Wholesale
Wine Type: Merlot

Distributor: Cask and Cork Wine Distribution
Wine Type: Cabernet

Distributor: Wine World
Wine Type: Red Blend

Distributor: In Vino Veritas
Wine Type: White Blend
```

```python
103
104  # Execute SQL query to retrieve information on which distributor carries which wine
105  cursor.execute("SELECT distributor_name, wine_type FROM wine_order_line "
106                 "INNER JOIN wine_order ON wine_order_line.wine_order_id = wine_order.wine_order_id "
107                 "INNER JOIN distributor ON wine_order.distributor_id = distributor.distributor_id "
108                 "INNER JOIN wine ON wine_order_line.wine_id = wine.wine_id")
109
110  wine_distributor_info = cursor.fetchall()
111
112  # Display the report
113  print("-- DISTRIBUTOR AND WINE INFORMATION --")
114  for info in wine_distributor_info:
115      print("Distributor: {}\nWine Type: {}\n".format(info[0], info[1]))
116
117
118  input("Press Enter to Exit...")
119  db.close()
```

This Python script generates a report that provides information on which distributor carries which wine. It achieves this by retrieving data from the `wine_order_line`, `wine_order`, `distributor`, and `wine` tables through SQL joins. The report displays the distributor name along with the corresponding wine types they carry. This information is crucial for business decisions, as it allows the winery owners to understand the distribution network and popularity of different wines among distributors.

# Employee Time Report

```python
import mysql.connector
from mysql.connector import errorcode

config = {
    "user":"bacchus_user",
    "password":"ILoveWine!",
    "host":"127.0.0.1",
    "database":"bacchus",
    "raise_on_warnings": True
}

try:
    db = mysql.connector.connect(**config)
    print("\n Database user {} connected to MySQL on host {} with database {}".format(config["user"], config["host"], config["database"]))

except mysql.connector.Error as err:
    if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("  The supplied username or password are invalid")

    elif err.errno == errorcode.ER_BAD_DB_ERROR:
        print("  The specified database does not exist")

    else:
        print(err)

cursor = db.cursor()

cursor.execute("SELECT emp_first,emp_last,emp_rate, \
               DATE_FORMAT(clock_in,'%a %b %c'),TIME_TO_SEC(clock_out - clock_in) / 3600 \
               FROM employee INNER JOIN employee_clock ON employee.emp_id=employee_clock.emp_id \
               WHERE clock_in BETWEEN '2023-11-27 00:00:00' AND '2023-11-27 23:59:59'")
shifts = cursor.fetchall()

print("--DISPLAYING Shift RECORDS --")
laborcost = 0
for shift in shifts:
    print("Employee Name: {} {}\nPay Rate: {}\nShift Day: {}\nShift Length: {:.2f} hours\n".format(shift[0],shift[1],shift[2],shift[3],shift[4]))
    laborcost += shift[2] * shift[4]
print("Total labor cost: ${:.2f}".format(laborcost))

input("Press Enter to Exit...")
db.close()
```

```
--DISPLAYING Shift RECORDS --
Employee Name: Debrah Messing
Pay Rate: 18.00
Shift Day: Mon Nov 11
Shift Length: 8.00 hours

Employee Name: Brad Pitt
Pay Rate: 20.00
Shift Day: Mon Nov 11
Shift Length: 8.00 hours

Employee Name: Tina Fey
Pay Rate: 17.00
Shift Day: Mon Nov 11
Shift Length: 8.00 hours

Employee Name: Debbie Reynolds
Pay Rate: 18.00
Shift Day: Mon Nov 11
Shift Length: 8.00 hours

Employee Name: Alec Baldwin
Pay Rate: 18.00
Shift Day: Mon Nov 11
Shift Length: 8.00 hours

Employee Name: Amy Poehler
Pay Rate: 23.00
Shift Day: Mon Nov 11
Shift Length: 8.00 hours

Total labor cost: $912.00
Press Enter to Exit...
```

This report will draw up all shift records for a range of time periods. Although this is just one iteration of the script, it shows the flexibility of our database design. For example, as it's written, this script pulls up all the shift records for the one day of sample data we have. However, the BETWEEN statement could be attached to a front-end where a custom time-period could be pulled up. Additionally, a HAVING statement could be added to isolate a single employee for a payroll report.

# Conclusions

The Bacchus Winery was a great example of the challenges facing a growing production company. Reputation and on-time deliveries are balanced with costs of materials and labor. All thrse factors must be tracked and examined as time goes on.

Our design incorporates these challenges. By breaking down these flows to their most basic elements, we've built a database that can grow with Bacchus. The work we've done in these short two weeks lays the groundwork for an employee clock system, an order tracking system, and an inventory management that compares wine sold vs wine produced. With this information, Bacchus can continue doing what they love:  making great wine!