**COEN 380 Advanced Database Systems**

**Final Report Project - 2**

**Car Sales Database System**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Title** | **Pg. No.** |
| **1.** | **ABSTRACT** | **2** |
| **2.** | **DATABASE SCHEMA** | **3** |
| **3.** | **INDEXES AND TRIGGERS** | **4** |
| **4.** | **STORED PROCEDURES** | **5** |
| **5.** | **SELECT QUERIES** | **6** |
| **6.** | **IUDs** | **23** |
| **7.** | **VIEWS** | **24** |
| **8.** | **TRANSACTION** | **25** |

**ABSTRACT**

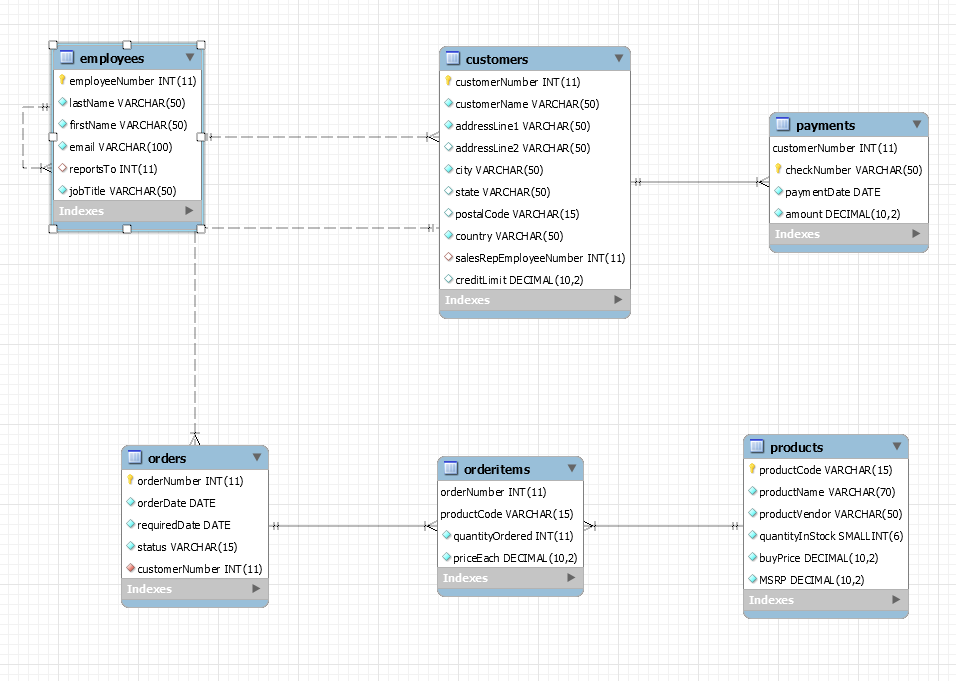
An organized and symmetric office solution is essential for any car sales business. There are departments of administration responsible for the maintenance of information of customers, their products inventory and their sales. Our project aims at building a database system on MySQL for data retrieval, data manipulation, and analysis of such data. This is based on a system wherein a customer buys antique cars from employee representatives.

**DATABASE SCHEMA**

The classicmodels database is a retailer of scale models of classic cars database. It contains typical business data such as customers, products, sales orders, sales order line items, etc.

Database Size : 352KB

Count of columns : 32



**Fig. Extended ER Diagram of our Database**

**TABLES-**

1. **Customers**: stores customer’s data.
2. **Products**: stores a list of scale model cars..
3. **Orders**: stores sales orders placed by customers.
4. **Orderitems**: stores sales order line items for each sales order.
5. **Payments**: stores payments made by customers based on their accounts.
6. **Employees**: stores all employee information as well as the organization structure such as who reports to whom.

**INDEXES AND TRIGGERS**

**(i) Indexes**

|  |  |  |
| --- | --- | --- |
| **No.** | **Table Name** | **Column Name** |
| 1 | customers | salesRepEmployeeNumber |
| 2 | customers | customerNumber |
| 3 | employees | reportsTo |
| 4 | employees | employeeNumber |
| 5 | orderitems | productCode |
| 6 | orderitems | orderNumber |
| 7 | orderitems | productCode |
| 8 | orders | customerNumber |
| 9 | orders | orderNumber |
| 10 | payments | customerNumber |
| 11 | payments | checkNumber |
| 12 | products | productCode |

**(ii) Trigger: check\_price**

DELIMITER $$

CREATE TRIGGER productCheckData Before insert on products

for each row

BEGIN

call check\_price(new.buyPrice, new.MSRP);

End$$

DELIMITER ;

**STORED PROCEDURE**

CREATE DEFINER=`root`@`localhost` PROCEDURE `check\_price`(IN buyPrice decimal(10, 2), MSRP decimal(10, 2))

BEGIN

if buyPrice < 0 then

signal sqlstate '45000'

set message\_text = 'check constraint on products.buyPrice failed';

end if;

if MSRP < 0 then

signal sqlstate '45001'

set message\_text = 'check constraint on products.MSRP failed';

end if;

if buyPrice < MSRP then

signal sqlstate '45002'

set message\_text = 'buyPrice value cannot be lower than MSRP';

end if;

END

**SELECT QUERIES**

1. **Find the Status of the orders where the number of orders per customer is more than 2.**

SELECT

c.customerNumber,

c.customerName,

COUNT(o.orderNumber) OrderCount,

o.status

FROM

customers c

LEFT JOIN

orders o ON c.customerNumber = o.customerNumber

GROUP BY c.customerName , o.status

HAVING OrderCount > 2;

**Results:**





**2. Find the total no of employees reporting to each Manager.**

SELECT

CONCAT(m.lastname, ', ', m.firstname) AS 'Manager',

COUNT(CONCAT(e.lastname, ', ', e.firstname)) AS 'Direct report'

FROM

employees e

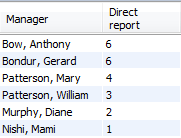
INNER JOIN

employees m ON m.employeeNumber = e.reportsto

GROUP BY CONCAT(m.lastname, ', ', m.firstname)

ORDER BY COUNT(CONCAT(e.lastname, ', ', e.firstname)) DESC;

**Result:**



3.  **Find the details of products bought more than 25 times.**

SELECT

p.productName, COUNT(p.productCode) Total\_Products\_Sold

FROM

orders o

JOIN

orderitems oi ON o.orderNumber = oi.orderNumber

JOIN

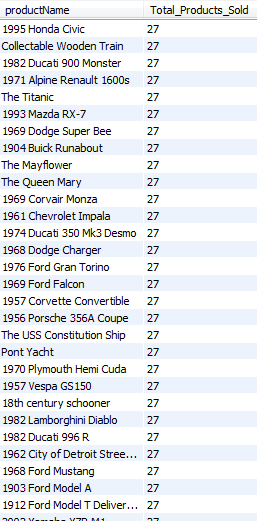
products p ON oi.productCode = p.productCode

GROUP BY oi.productCode

HAVING COUNT(p.productCode) > 25

ORDER BY Total\_Products\_Sold;

**Result:**









**View : ProductsPerOrder :**

CREATE VIEW ProductsPerOrder AS

SELECT

o.orderNumber,

p.productName,

o.customerNumber,

COUNT(p.productCode) No\_of\_products,

o.status,

oi.quantityOrdered,

p.productVendor,

p.MSRP

FROM

orders o

JOIN

orderitems oi ON o.orderNumber = oi.orderNumber

JOIN

products p ON oi.productCode = p.productCode

GROUP BY oi.productCode;

**4. Find the details of the employee who sold the maximum no. of Ford cars**

SELECT

e.employeeNumber,

CONCAT(e.firstName, ' ', e.lastName) AS 'Employee Name',

SUM(ppo.No\_of\_products) AS 'Total Products'

FROM

employees e

LEFT JOIN

customers c ON c.salesRepEmployeeNumber = e.employeeNumber

JOIN

productsperorder ppo ON c.customerNumber = ppo.customerNumber

WHERE

ppo.productName LIKE '%Ford%'

GROUP BY e.employeeNumber

ORDER BY SUM(ppo.No\_of\_products) DESC

LIMIT 1;

**Result:**



**5. Find the latest(most recent) order details of customers between 1st Jan 2003 to 1st Nov 2003**

SELECT

o.orderDate, o.orderNumber, c.customerName

FROM

orders o

JOIN

customers c ON c.customerNumber = o.customerNumber

WHERE

orderDate IN (SELECT

MAX(o.orderDate)

FROM

orders o

JOIN

customers c ON o.customerNumber = c.customerNumber

WHERE

o.orderDate BETWEEN CAST('2003-01-01' AS DATE) AND CAST('2003-11-01' AS DATE)

GROUP BY c.customerName

ORDER BY c.customerName)

ORDER BY c.customerName , o.orderNumber;

**Result:**







1. **Find the total no. of customers from each country who ordered products with buyPrice > 50**

SELECT

COUNT(c.customerNumber) No\_of\_Customers, c.country

FROM

customers c

LEFT JOIN

orders o ON o.customerNumber = c.customerNumber

JOIN

orderitems oi ON o.orderNumber = oi.orderNumber

JOIN

products p ON oi.productCode = p.productCode

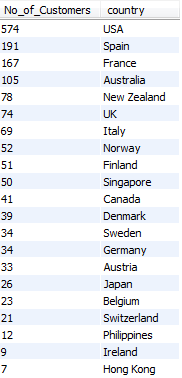
WHERE

p.buyPrice > 50

GROUP BY c.country

ORDER BY COUNT(c.customerNumber) DESC;

**Result:**

****

1. **Find the customer who has placed the maximum order by price.**

SELECT

o.customerNumber,

customerName,

SUM((priceEach \* quantityOrdered)) Total\_Price,

productName

FROM

orders o,

orderitems oi,

customers c,

products p

WHERE

c.customerNumber = o.customerNumber

AND o.orderNumber = oi.orderNumber

AND p.productCode = oi.productCode

GROUP BY c.customerNumber

ORDER BY Total\_Price DESC

LIMIT 1;

**Result:**



1. **Display all the customers who bought products from vendor Red Start Diecast and whose quantity in stock in less than 1000;**

SELECT

c.customerNumber, c.customerName

FROM

orders o,

orderitems oi,

customers c,

products p

WHERE

c.customerNumber = o.customerNumber

AND o.orderNumber = oi.orderNumber

AND p.productCode = oi.productCode

AND p.productVendor LIKE '%Red Start%'

AND p.quantityInStock < 1000;

**Result:**



1. **The vendor who sold the maximum number of products in the USA.**

SELECT

o.customerNumber,

customerName,

COUNT(productName) Product\_Count,

c.country,

p.productVendor

FROM

orders o,

orderitems oi,

customers c,

products p

WHERE

c.customerNumber = o.customerNumber

AND o.orderNumber = oi.orderNumber

AND p.productCode = oi.productCode

AND c.country LIKE '%USA%'

GROUP BY p.productVendor

ORDER BY Product\_Count DESC

LIMIT 1;

**Result:**



1. **Find the number of products sold by each employee.**

SELECT

e.firstName,

COUNT(p.productCode) AS 'Number of products sold'

FROM

employees e

LEFT JOIN

customers c ON e.employeeNumber = c.salesRepEmployeeNumber

LEFT JOIN

orders o ON c.customerNumber = o.customerNumber

LEFT JOIN

orderitems oi ON o.orderNumber = oi.orderNumber

LEFT JOIN

products p ON oi.productCode = p.productCode

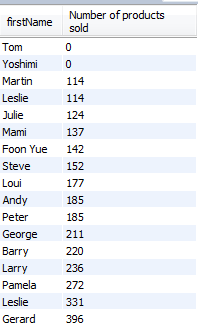
WHERE

e.jobTitle = 'Sales Rep'

GROUP BY e.employeeNumber

ORDER BY COUNT(p.productCode);

Result:



**IUDs**

**Insert Query:**

* **Insert New Entry in Customer Table**

insert into customers (customerNumber,customerName, addressLine1,addressLine2,city,state, postalCode,country,salesRepEmployeeNumber,creditLimit) values

(500,'John Sequal','500, Royale',NULL,'Santa Clara', 'CA','50000','USA',1370,'5000.00');

**Update Query:**

* **Update the ordered quantity from 30 to 35 for orderId = 10100 and productCode = S18\_1749**

update orderitems set quantityOrdered = 35

where orderNumber = 10100 and productCode = 'S18\_1749';

**Delete Query:**

* **Delete payment records made before the year 2004 -**

DELETE FROM payments where paymentDate <'20040101';

**VIEWS**

**VIEW: Customer\_View**

CREATE OR REPLACE FORCE VIEW "CUSTOMER\_VIEW" ("CUSTOMERNUMBER", "CUSTOMERNAME", "CREDITLIMIT") AS

select customerNumber, customerName, creditLimit

from customers;

**Update View:**

CREATE OR REPLACE TRIGGER "UPDATE\_ON\_CUSTOMER"

INSTEAD OF UPDATE ON Customer\_View **Event**

FOR EACH ROW

BEGIN

Update CUSTOMERS **Action**

set CUSTOMERNAME = :new.CUSTOMERNAME

where CUSTOMERNUMBER = :old.CUSTOMERNUMBER; **Condition**

DBMS\_OUTPUT.PUT\_LINE('updated');

END;

/

**EXAMPLE OF VIEW UPDATE:**

**Before Update:**



**Update Query on View:**

update Customer\_View

set customerName = 'supriya'

where CUSTOMERNUMBER = 103;

Select CUSTOMERNUMBER, CUSTOMERNAME, CREDITLIMIT from Customer\_View where CUSTOMERNUMBER = 103;

**Result**



Select \* from customers where customerNumber = 103



**TRANSACTION**

**Normal Transaction Execution**

**Transaction that either execute totally or roll backs complete transaction if error occurred anywhere in the transaction:**

**Positive case (Without Error):**

CREATE DEFINER=`root`@`localhost` PROCEDURE `transaction\_Employee\_1`()

BEGIN

DECLARE exit handler for sqlexception

BEGIN

ROLLBACK;

signal sqlstate '45000'

set message\_text = 'Error - Rollback';

END;

DECLARE exit handler for sqlwarning

BEGIN

ROLLBACK;

signal sqlstate '45001'

set message\_text = 'Warning rollback';

END;

START TRANSACTION;

delete from employees where employeeNumber = 1000;

insert into employees (employeeNumber, firstName, lastName, email, reportsTo, jobTitle) values (1000, 'Allen', 'Richey', 'allen@abc.com', 1002, 'VP Product');

insert into employees (employeeNumber, firstName, lastName, email, reportsTo, jobTitle) values (1001, 'Shravya', NULL, 'allen@abc.com', 1000, 'VP Product');

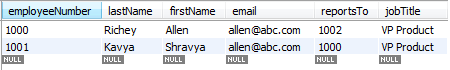
update employees set lastName = 'Richey' where employeeNumber = 1000;

COMMIT;

END

****

****

****

**Transaction Execution with Violating Constraint:**

**Negative Case (With Error):**

CREATE DEFINER=`root`@`localhost` PROCEDURE `transaction\_Employee\_1`()

BEGIN

DECLARE exit handler for sqlexception

BEGIN

ROLLBACK;

signal sqlstate '45000'

set message\_text = 'Error - Rollback';

END;

DECLARE exit handler for sqlwarning

BEGIN

ROLLBACK;

signal sqlstate '45001'

set message\_text = 'Warning rollback';

END;

START TRANSACTION;

insert into employees (employeeNumber, firstName, lastName, email, reportsTo, jobTitle)

values (1000, 'Allen', 'Richey', 'allen@abc.com', 1002, 'VP Product');

insert into employees (employeeNumber, firstName, lastName, email, reportsTo, jobTitle)

values (1001, 'Shravya', NULL, 'allen@abc.com', 1000, 'VP Product');

update employees set lastName = 'Richey' where employeeNumber = 1000;

COMMIT;

END

****

****