Tecnológico de Monterrey

Homework 1.1: Liverpool Data model

Students:

Salvador Orozco Villalever - A07104218

Aranzza Abascal Fararoni - A01329203

Professor:

Dr. Daniel Pérez Rojas

Subject:

Advanced Database

Due date:

January 26th, 2018

ITESM Campus Puebla

1. Requirements description:

The following system is to be used at Liverpool and its branches. The idea is to model all of the company's products (by category, department and branch), employees (by type, branch, salary) and sales (by product list, customer, total amount and date). The following sentences cover all the requirements:

1. The system must be able to manage products in the database.
2. The system must be able to manage employees in the database.
3. The system must be able to manage the employees' dependents.
4. The system must be able to manage the employees' assistance.
5. The system must be able to manage sales including modes of payment, discounts and special offers.
6. The system must be able to manage suppliers, purchase orders, product deliveries, invoices and expenditures.
7. The system must be able to manage all these operations in every branch.
8. The system must be able to manage shopping carts and gift registries.
9. The system must allow the user to make queries on the data to retrieve useful information according to the relationship between the entities.
10. The system must be robust enough to handle operations like sales and product returns without corrupting data.
11. The system must be able to perform automatic operations in response to certain events.

2. ER Diagram:

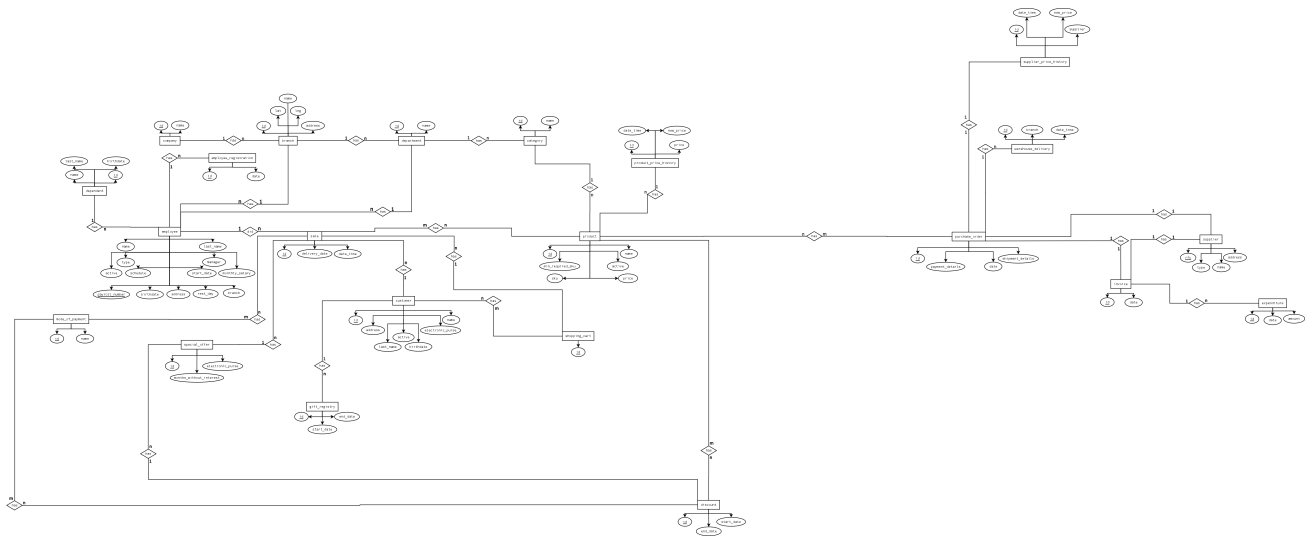

Figura 1: *Entity-Relationship Diagram*

3. Relational Diagram:

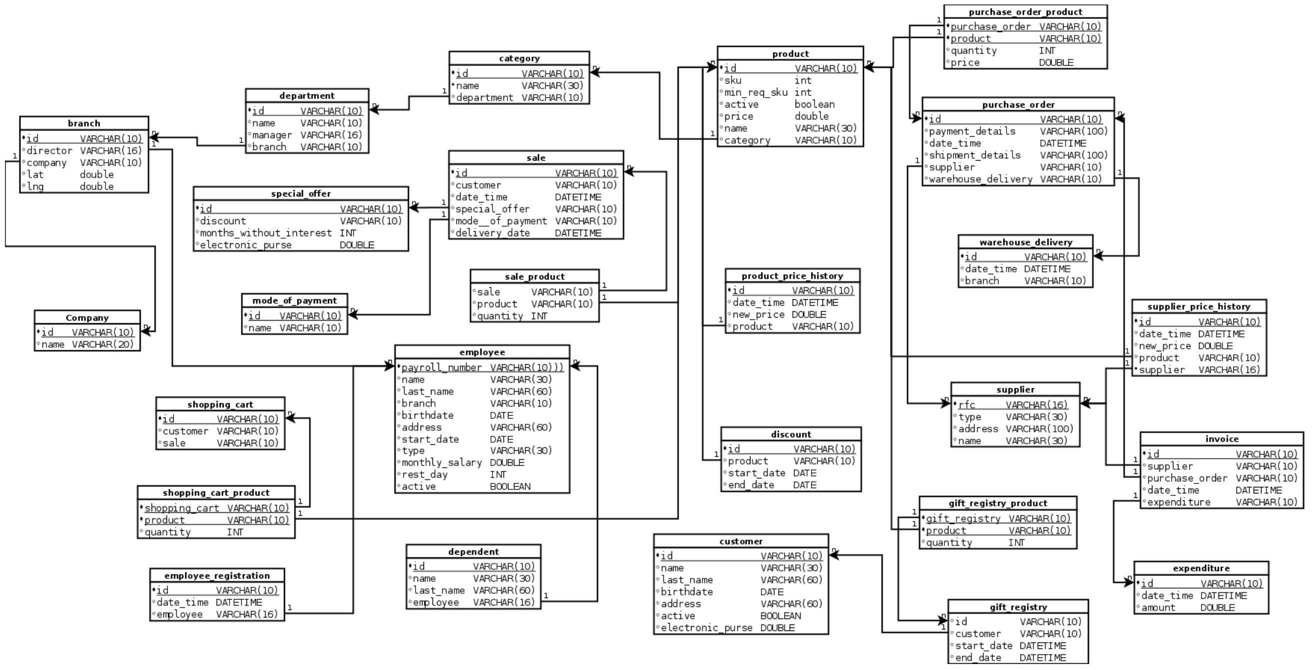

Figura 2: *Relational Diagram*

4. Special operations:

4.1. Transactions:

1. Sales operations must be implemented as transactions to avoid products being taken off from the stock without confirming the payment, or in case the system crashes, the Internet connection is lost, etc..

2. Product returns must be implemented as transactions to avoid products being returned to the stock in case the system crashes, the Internet connection is lost, the money can't be returned, etc..

4.2. Stored procedures:

1. A stored procedure must be implemented to apply discounts to certain products according to their category, department, price, etc..

2. A stored procedure must be implemented to deactivate products according to category, department, supplier, etc..

4.3. Triggers:

1. A trigger must be implemented to avoid the stock of a product reaching zero. To solve this, each product has a minimum required stock and in case its actual stock reaches this value, a purchase order is created.

2. A trigger must be implemented to check whether a person is buying more than 1000 MXN in products; in such a case, the customer receives 15 % of the purchase in an electronic purse.

4.4. Functions:

1. A function must be implemented to calculate the age of a given employee.

2. A function must be implemented to calculate the years of service of a given employee.

3. A stored procedure must be implemented to get a table of the employees that currently are inside the building of a given store.

4.5. Programmed tasks:

1. A programmed task must be implemented to compute the payroll on the days 15th and 30th of each month.

2. A programmed task must be implemented to get a summary of the previous day's sales. This would be done in the early morning where no sales are being done in a given branch.

5. Useful queries:

```
SELECT category.id, category.name, department.name
FROM category
INNER JOIN department ON category.department = department.id
WHERE department.branch = "B0710";
```

Query 1. Query to get the categories of a given branch

```
SELECT e.payroll_number, e.name, e.last_name, e.birthdate
FROM employee e
WHERE e.branch = "B1210";
```

Query 2. Query to get the employees of a given branch

```
SELECT p.id, p.name, p.sku, d.name,
FROM product p
JOIN category c
ON p.category = c.id
JOIN department d
ON c.department = d.id
JOIN branch b
ON d.branch = "B0710"
GROUP BY p.id
ORDER BY d.name;
```

Query 3. Query to show products in a branch

```
SELECT e.payroll_number, e.name, e.last_name, e.birthdate
FROM employee e
WHERE e.branch = "B0710" AND e.birthdate = (SELECT MIN(e1.birthdate)
                                            FROM employee e1
                                            WHERE e1.branch = "B0710")
                                            GROUP BY e.payroll_number;
```

Query 4. Query to get the oldest employee

```
SELECT e.payroll_number, e.name, e.last_name, e.birthdate
FROM employee e
WHERE e.branch = "B0710" AND e.birthdate = (SELECT MAX(e1.birthdate)
                                            FROM employee e1
                                            WHERE e1.branch = "B0710")
                                            GROUP BY e.payroll_number;
```

Query 5. Query to get the youngest employee

```
SELECT e.payroll_number, e.name, e.last_name, e.birthdate, e.monthly_salary
FROM employee e
WHERE e.branch = "B0710" AND e.monthly_salary =
(SELECT MAX(e1.monthly_salary)
FROM employee e1
WHERE e1.branch = "B0710")
GROUP BY e.payroll_number;
```

Query 6. Query to get the employee with the highest salary in a branch

```
SELECT e.payroll_number, e.name, e.last_name, e.birthdate, e.monthly_salary
FROM employee e
WHERE e.branch = "B0710" AND e.monthly_salary =
(SELECT MIN(e1.monthly_salary)
FROM employee e1
WHERE e1.branch = "B0710")
GROUP BY e.payroll_number;
```

Query 7. Query to get the employee with the lowest salary in a branch

```
SELECT p.id, p.name, p.sku, p.category, p.price
FROM product p
WHERE p.price = (SELECT MAX(p1.price)
                FROM product p1 JOIN category c
                ON p1.category = c.id
                JOIN department d
                ON c.department = d.id
                JOIN branch b
                ON d.branch = "B0710");
```

Query 8. Query to get the most expensive product in a branch

```
SELECT p.id, p.name, p.sku, p.category, p.price
FROM product p
WHERE p.price = (SELECT MIN(p1.price)
                FROM product p1
                JOIN category c
                ON p1.category = c.id
                JOIN department d
                ON c.department = d.id
                JOIN branch b
                ON d.branch = "B0710");
```

Query 9. Query to get the cheapest product in a branch

```
SELECT p.id, p.name, p.sku, p.category, p.price
FROM product p
JOIN category c
ON p.category = c.id
JOIN department d
ON c.department = d.id
JOIN branch b
ON d.branch = "B0710"
GROUP BY p.id
WHERE p.price < 200.0;
```

Query 10. Query to get the products below a given price p in a given branch b

```
SELECT p.id, p.name, p.sku, p.price, d.name, b.name
FROM product p
JOIN category c
ON p.category = c.id
JOIN department d
ON c.department = d.id
JOIN branch b
ON d.branch = b.id
WHERE p.name = "Body";
```

Query 11. Query to get the status of one specific product.

```
SELECT sale.customer, SUM(sale_product.quantity) as 'products'
FROM sale, sale_product
WHERE sale_product.sale = sale.id
GROUP BY sale.customer
ORDER BY products DESC LIMIT 1;
```

Query 12. Query to get the customer who has purchased more products.

```
SELECT product.id, product.name, product.price, sale_product.quantity,
branch.name
FROM sale_product, product
JOIN category
ON product.category = category.id
JOIN department
ON category.department = department.id
JOIN branch
ON department.branch = branch.id
WHERE sale_product.product = product.id
AND sale_product.quantity = (SELECT MAX(s.quantity)
                             FROM sale_product s);
```

Query 13. Query to get the most purchased product.