ceu-economics-and-business.github.io

Data warehouse architectures. – ECBS 5146 SQL and Different Shapes of Data

7–8 minutes

Overview

Teaching: 90 min

Questions

- What is a Data Warehouse?
- As analyst, how can you create a simple analytical platform using SQL DB?

Objectives

- Understanding data warehouse architectures
- Building a denormalized analytical data store
- Building an ETL pipeline using MySQL Triggers and Events
- Building data marts with MySQL View

Keywords

#DATA WAREHOUSE ARCHITECTURE

#VIEWS

#TRIGGERS

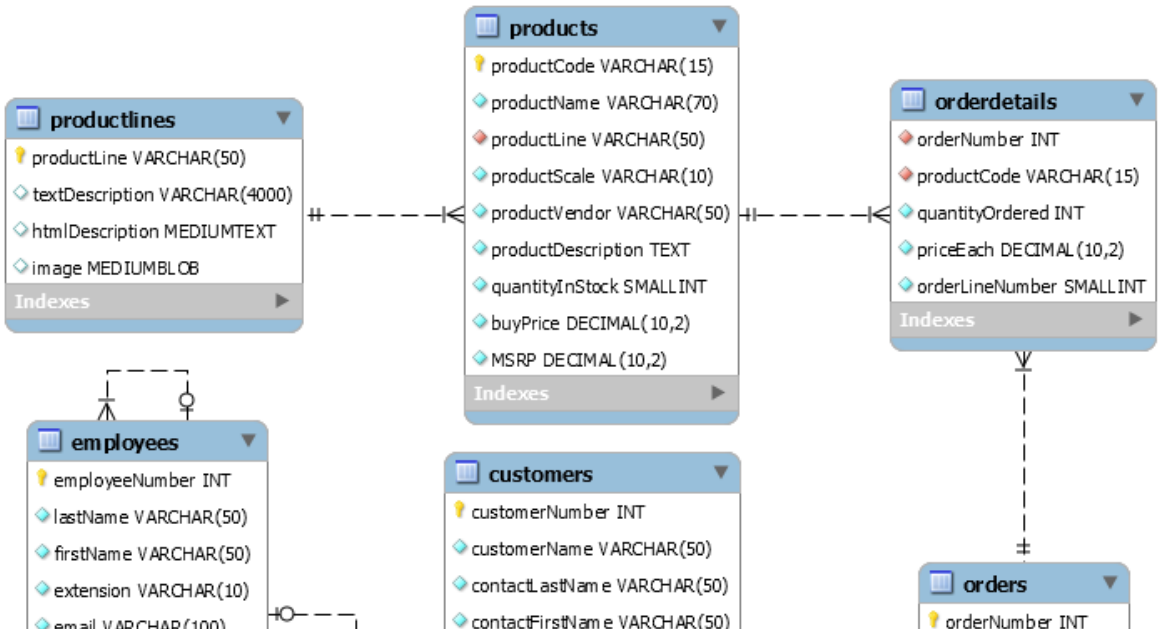
#ETL

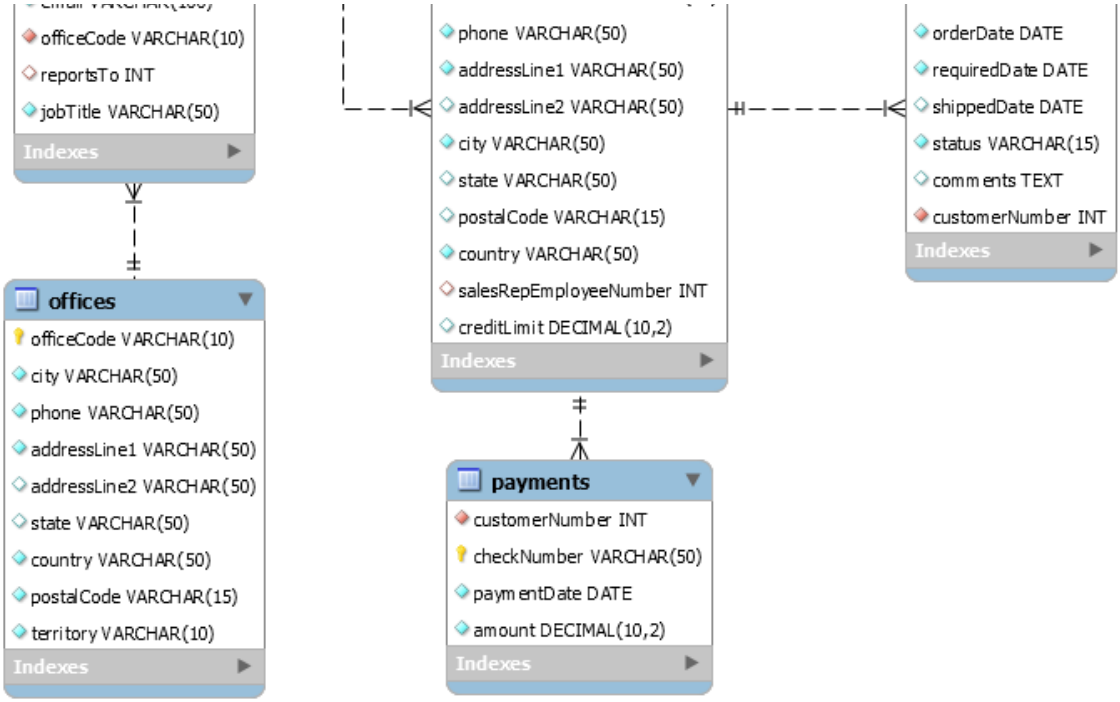
#DATA MARTS

Table of Content

Session setup

No need to load new data, in this chapter we will use the same sample db we used in the last chapter:





Creating the analytical data store

```
CREATE TABLE new_order LIKE orders;

DROP TABLE new_order;

CREATE TABLE new_order AS SELECT * FROM orders;
```

Exercise1

Create a physical copy of birdstrikes with records where state is Oklahoma

We will use a query created in Homework 3. This creates a denormalized snapshot of the operational tables for product_sales subject. We will embed the creation in a stored procedure.

```
DROP PROCEDURE IF EXISTS CreateProductSalesStore;

DELIMITER //
```

```
CREATE PROCEDURE CreateProductSalesStore()
BEGIN

        DROP TABLE IF EXISTS product_sales;

        CREATE TABLE product_sales AS
        SELECT
           orders.orderNumber AS SalesId,
           orderdetails.priceEach AS Price,
           orderdetails.quantityOrdered AS Unit,
           products.productName AS Product,
           products.productLine As Brand,
           customers.city As City,
           customers.country As Country,
           orders.orderDate AS Date,
           WEEK(orders.orderDate) as WeekOfYear
        FROM
                orders
        INNER JOIN
                orderdetails USING (orderNumber)
        INNER JOIN
                products USING (productCode)
        INNER JOIN
                customers USING (customerNumber)
        ORDER BY
                orderNumber,
                orderLineNumber;

END //
```

```
DELIMITER ;



CALL CreateProductSalesStore();
```

Events to schedule ETL jobs

Event engine runs scheduled jobs/tasks. We can us it for scheduling ETL processes.

Basics on how to check the state of the scheduler. Check if scheduler is running

```
SHOW VARIABLES LIKE "event_scheduler";
```

Turn it on if not

```
SET GLOBAL event_scheduler = ON;
```

This is how you turn it OFF

```
SET GLOBAL event_scheduler = OFF;
```

Format:

```
CREATE EVENT [IF NOT EXIST] event_name
ON SCHEDULE schedule
DO
event_body
```

Exercise3

create a scheduler which writes the current time in messages in every second

Event which is calling CreateProductSalesStore every 1 minute in the next 1 hour.

```
DELIMITER $$

CREATE EVENT CreateProductSalesStoreEvent
ON SCHEDULE EVERY 1 MINUTE
STARTS CURRENT_TIMESTAMP
ENDS CURRENT_TIMESTAMP + INTERVAL 1 HOUR
DO
        BEGIN
                INSERT INTO messages SELECT
CONCAT('event:',NOW());
                CALL CreateProductSalesStore();
        END$$
DELIMITER ;
```

Listing all events stored in the schema

Deleting an event

```
DROP EVENT IF EXISTS CreateProductSalesStoreEvent;
```

Trigger as ETL

Format:

```
DELIMITER $$

CREATE TRIGGER trigger_namex
    AFTER INSERT ON table_namex FOR EACH ROW
BEGIN
    -- statements
    -- NEW.orderNumber, NEW.productCode etc
END$$
```

```
DELIMITER ;
```

> ## Exercise3
>
> Copy the birdstrikes structure into a new table called birdstrikes2. Insert into birdstrikes2 the line where id is 10. Hints:
>
> - Use the samples from Chapter2 for copy
> - For insert use the format like: INSERT INTO bla SELECT blabla

Empty log table:

The trigger

Creating a trigger which is activated if an insert is executed into orderdetails table. Once triggered will insert a new line in our previously created data store.

```
DROP TRIGGER IF EXISTS after_order_insert;

DELIMITER $$

CREATE TRIGGER after_order_insert
AFTER INSERT
ON orderdetails FOR EACH ROW
BEGIN

        -- log the order number of the newley
inserted order
        INSERT INTO messages SELECT CONCAT('new
orderNumber: ', NEW.orderNumber);
```

```
        -- archive the order and assosiated table
entries to product_sales
        INSERT INTO product_sales
        SELECT
           orders.orderNumber AS SalesId,
           orderdetails.priceEach AS Price,
           orderdetails.quantityOrdered AS Unit,
           products.productName AS Product,
           products.productLine As Brand,
           customers.city As City,
           customers.country As Country,
           orders.orderDate AS Date,
           WEEK(orders.orderDate) as WeekOfYear
        FROM
                orders
        INNER JOIN
                orderdetails USING (orderNumber)
        INNER JOIN
                products USING (productCode)
        INNER JOIN
                customers USING (customerNumber)
        WHERE orderNumber = NEW.orderNumber
        ORDER BY
                orderNumber,
                orderLineNumber;

END $$

DELIMITER ;
```

`E` - Extract: Joining the tables for the operational layer is an extract operation

`T` - Transform: We don't have glamorous transformations here, only a WeekOfYear covering this part. Nevertheless, please note that you call a store procedure form trigger or even use procedural language to do transformation in the trigger itself.

`L` - Load: Inserting into product_sales represents the load part of the ETL

Activating the trigger

Listing the current state of the product_sales. Please note that, there is no orderNumber 16.

```
SELECT * FROM product_sales ORDER BY SalesId;
```

Now will activate the trigger by inserting into orderdetails:

```
INSERT INTO orders
VALUES(16,'2020-10-01','2020-10-01','2020-10-
01','Done','',131);
INSERT INTO orderdetails
VALUES(16,'S18_1749','1','10',1);
```

Check product_sales again, you should have orderNumber 16:

```
SELECT * FROM product_sales ORDER BY SalesId;
```

`Note` Triggers are not the only way to initiate an ETL process. In fact for performance reasons, it is advised to use the Event engine on large data sets. For more information check: https://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/

Data marts with Views

With views we can define sections of the datastore and prepare them for a BI operation such as reporting.

View of sales for a specific brand (Vintage_Cars)

```
DROP VIEW IF EXISTS Vintage_Cars;

CREATE VIEW `Vintage_Cars` AS
SELECT * FROM product_sales WHERE product_sales.Brand
= 'Vintage Cars';
```

View of sales in USA:

```
DROP VIEW IF EXISTS USA;

CREATE VIEW `USA` AS
SELECT * FROM product_sales WHERE country = 'USA';
```

Note the content of Views are generated on-the-fly. For performance reasons, in analytics, so called materialized views are preferred on large data set. This is not supported by MySQL, but there are several ways to implemented. Here is an example: https://fromdual.com/mysql-materialized-views

Exercise4

Create a view, which contains product_sales rows of 2003 and 2005.