Information Systems assignment 1

Koen Bolhuis (s3167895)
Germán Calcedo (s5381541)

November 29, 2022

1 Task 1

We use <u>underline</u> to indicate primary keys, and **bold** to indicate foreign keys. We assume product codes are only unique for a given supplier, ie. multiple suppliers can have unrelated products with the same product code.

1.1 Unnormalized Schema

customer
[<u>custno</u>, cust_name, cust_addr, cust_phone, (supplier_id, supplier_name, product_code, product_title, purchase_date, sales_price)]

1.2 First Normal Form

For the First Normal Form, we ensure that each row and column in the relation only contains atomic values. Effectively, this means that groups of (supplier_id, supplier_name, product_code, product_title, purchase_date, sales_price) are split up, with (cust_no, cust_name, cust_addr, cust_phone) repeated for each. In order to still uniquely identify each row, we add a sale_id column. sale_id is unique for each customer (ie. sale_id can overlap between customers), so the combination of (sale_id, custno) is a candidate key.

sales
[<u>custno</u>, cust_name, cust_addr, cust_phone, <u>sale_id</u>, supplier_id, supplier_name, product_code, product_title, purchase_date, sales_price]

1.3 Second Normal Form

For the Second Normal Form, we make sure that every non-key attribute is fully functionally dependent on the primary key, eg. needs the primary key in order to be uniquely identifiable. This means we split off the customer information into its own relation, where every attribute depends on the custno.

customer
[<u>custno</u>, cust_name, cust_addr, cust_phone]

sales
[**<u>custno</u>**, <u>sale_id</u>, supplier_id, supplier_name, product_code, product_title, purchase_date, sales_price]

1.4 Third Normal Form

Achieving the third normal form is relatively straight forward. Both supplier_name and product_title can be inferred without the need of the primary key. That is, these columns can be determined with one or more non-key attributes. To solve this, we simply extract suppliers and products into separate tables.
We do this based on the assumption that product_code is not unique, multiple suppliers can use the same code internally. We therefore need both supplier_id and product_code to form the primary key of the product table.

customer
[custno, cust_name, cust_addr, cust_phone]

sales
[custno, sale_id, supplier_id, product_code, purchase_date, sales_price]

supplier
[supplier_id, supplier_name]

product
[supplier_id, product_code, product_title]

1.5 Boyce-Codd Normal Form

Finally, for BCNF, as product_code is not unique, we modify the product table so that it no longer is part of the primary key. Instead, we create a unique product_id, which determines both the product_title and product_code unequivocally. In order to not lose the relation between suppliers and products, we add a table that connects both of them.

customer
[custno, cust_name, cust_addr, cust_phone]

sales
[**custno**, sale_id, **supplier_id**, **product_id**, purchase_date, sales_price]

supplier
[supplier_id, supplier_name]

product
[product_id, product_title, product_code]

supplier_product
[**supplier_id**, **product_id**]

1.6 Creation in PostgreSQL

```
CREATE TABLE customer (
    custno SERIAL PRIMARY KEY,
    cust_name VARCHAR(50),
    cust_addr VARCHAR(100),
    cust_phone VARCHAR(20)
);

CREATE TABLE supplier (
    supplier_id SERIAL PRIMARY KEY,
    supplier_name VARCHAR(50)
);

CREATE TABLE product (
    product_id SERIAL PRIMARY KEY,
    product_title VARCHAR(50),
    product_code VARCHAR(50)
);

CREATE TABLE supplier_product (
    supplier_id INT NOT NULL,
    product_id INT NOT NULL,
   PRIMARY KEY (supplier_id, product_id),
   FOREIGN KEY (supplier_id) REFERENCES supplier (supplier_id),
   FOREIGN KEY (product_id) REFERENCES product (product_id)
);

CREATE TABLE sales (
    custno INT NOT NULL,
    sale_id INT NOT NULL,
    supplier_id INT NOT NULL,
    product_id INT NOT NULL,
    purchase_date DATE,
    sales_price INT,
   PRIMARY KEY (custno, sale_id),
   FOREIGN KEY (custno) REFERENCES customer (custno),
   FOREIGN KEY (supplier_id) REFERENCES supplier (supplier_id),
   FOREIGN KEY (product_id) REFERENCES product (product_id)
);

-- Sale ID counter
CREATE OR REPLACE FUNCTION set_sale_id_counter() RETURNS TRIGGER
LANGUAGE plpgsql AS
$$
DECLARE sale_count INT;
BEGIN
   SELECT COUNT(*) + 1
   INTO sale_count
```

```
    FROM sales
    WHERE sales.custno = NEW.custno;

    NEW.sale_id := sale_count;

    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER sale_id_counter
BEFORE INSERT ON sales
FOR EACH ROW
EXECUTE FUNCTION set_sale_id_counter();
```

2 Task 2

```
-- Customer
CREATE OR REPLACE FUNCTION set_cust_name_uppercase() RETURNS TRIGGER
LANGUAGE plpgsql AS
$$
BEGIN
    NEW.cust_name := UPPER(NEW.cust_name);
    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER cust_name_uppercase
BEFORE INSERT OR UPDATE ON customer
FOR EACH ROW
EXECUTE FUNCTION set_cust_name_uppercase();

-- Supplier
CREATE OR REPLACE FUNCTION set_supplier_name_uppercase() RETURNS TRIGGER
LANGUAGE plpgsql AS
$$
BEGIN
    NEW.supplier_name := UPPER(NEW.supplier_name);
    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER supplier_name_uppercase
BEFORE INSERT OR UPDATE ON supplier
FOR EACH ROW
EXECUTE FUNCTION set_supplier_name_uppercase();
```

3 Task 3

```
CREATE OR REPLACE FUNCTION check_sales_price_not_negative() RETURNS TRIGGER
LANGUAGE plpgsql AS
$$
BEGIN
     IF (NEW.sales_price < 0) THEN
          RAISE EXCEPTION 'Sales price cannot be negative';
     END IF;
     RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER sales_price_not_negative
BEFORE INSERT OR UPDATE ON sales
FOR EACH ROW
EXECUTE FUNCTION check_sales_price_not_negative();
```

4 Task 4

To make sure every fifth sale receives a discount, we can check that the number of sales is divisible by 5. If this is the case, the 10% discount is applied.

```
CREATE OR REPLACE FUNCTION set_discount_fifth_sale() RETURNS TRIGGER
LANGUAGE plpgsql AS
$$
DECLARE sale_count INT;
BEGIN
     SELECT COUNT(*) + 1
     INTO sale_count
     FROM sales
     WHERE sales.custno = NEW.custno;

     NEW.sale_id := sale_count;

     IF (MOD(NEW.sale_id, 5) = 0) THEN
          NEW.sales_price := NEW.sales_price * 0.9;
     END IF;
     RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER discount_fifth_sale
BEFORE INSERT ON sales
FOR EACH ROW
EXECUTE FUNCTION set_discount_fifth_sale();
```