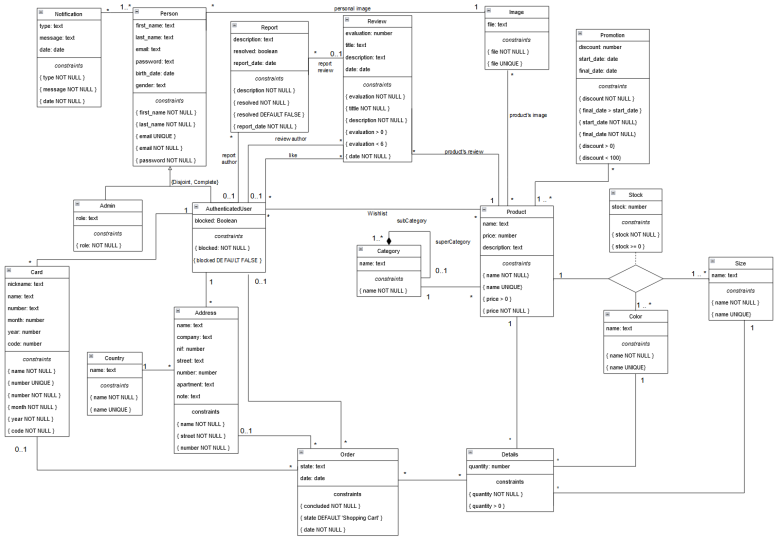
EBD: Database Specification Component

Oferecer aos adultos, uma alternativa mais prática e inovadora de adquirir artigos de moda de qualidade.

A4: Conceptual Data Model

O modelo de dados conceptual (utilizamos um diagrama UML para a sua representação) identifica e descreve todas as identidades que vão ser essenciais no sistema. Também serão representadas todas as relações entre estas identidades, os atributos de cada uma e as restrições aplicadas a alguns destes. O que não é possível representar no diagrama, é apresentado como uma *business rule*.

1. Class diagram



2. Additional Business Rules

Identificador	Descrição
BR01	O utilizador apenas pode fazer uma avaliação de um produto, caso este esteja presente numa das suas encomendas.
BR02	Quando um utilizador elimina a sua conta, as *reviews* efetuadas por ele passam a ter um autor anónimo.

Identificador	Descrição
BR03	A tabela Order é também usada para guardar o estado do carrinho de compras. Por este motivo há possibilidade de uma Order não ter nenhuma tabela Details (guarda o produto escolhido, o tamanho, a cor e a quantidade) associada, porque o carrinho está, inicialmente ou a uma dada altura, vazio. No entanto, se uma Order possuir o atributo state com outro valor, exceto 'Shopping Cart', necessita de ter pelo menos uma tabela Details associada, um Address e um Cart.
BR04	Um utilizador apenas pode dar *like* em *reviews* dadas por outros utilizadores.
BR05	Um utilizador não pode reportar a sua *review*

A5: Relational Schema, validation and schema refinement

O objetivo do modelo relacional é mapear todas as identidades e relações presentes no modelo conceptual. Neste modelo conseguimos ter uma forma mais clara de apresentar as identidades (como uma relação), os seus atributos, *primary keys*, *foreign keys* e todas as restrições impostas, que podem ser: NOT NULL, UNIQUE, CHECK e DEFAULT.

1. Relational Schema

Relation reference	Relation Compact Notation
R01	image(**id**, file **UK NN**)
R02	authenticated_user(**id**, first_name **NN**, last_name **NN**, email **UK NN**, password **NN**, birth_date, gender, blocked **NN DF** FALSE, id_image → image)

Relation reference	Relation Compact Notation
R03	admin(**id**, first_name **NN**, last_name **NN**, email **UK NN**, password **NN**, birth_date, gender, id_image → image, role **NN CK** role **IN** Admin Role Types)
R04	notification(**id**, type **NN CK** type **IN** Notification Types, message **NN**, date **NN**)
R05	admin_notification(**id_admin NN** → admin, **id_notification NN** → notification)
R06	user_notification(**id_user NN**→ authenticated_user, **id_notification NN** → notification)
R07	card(**id**, nickname , name **NN**, number **UK NN**, month **NN**, year **NN**, code **NN**, id_user → authenticated_user)
R08	country(**id**, name **UK NN**)
R09	address(**id**, name **NN**, company, nif, street **NN**, number **NN**, apartment, note, id_country **NN** → country, id_user **NN**→ authenticated_user)
R10	category(**id**, name **NN**, id_super_category → category)
R11	product(**id**, name **UK NN**, description, price **NN CK** price > 0, id_category **NN** → category)
R12	product_image(**id_product NN** → product, **id_image NN** → image)
R13	wishlist(**id_user NN** → authenticated_user, **id_product NN** → product)
R14	review(**id**, evaluation **NN CK** evaluation > 0 AND evaluation <= 5, title **NN**, description **NN**, date **NN** id_user → authenticated_user, id_product **NN** → product)
R15	like(**id_user NN** → authenticated_user, **id_review NN** → review)

Relation reference	Relation Compact Notation
R16	report(**id**, description **NN**, resolved **NN DF** FALSE, report_date **NN**, id_user → authenticated_user, id_review → review)
R17	promotion(**id**, discount **NN CK** discount > 0 AND discount < 100, start_date **NN**, final_date **NN CK** final_date > start_date)
R18	promotion_product(**id_promo NN** → promotion, **id_product NN** → product)
R19	size(**id**, name **UK NN**)
R20	color(**id**, name **UK NN**)
R21	stock(stock **NN CK** stock >= 0, **id_product NN** → product, **id_size NN** → size, **id_color NN** → color)
R22	details(quantity **NN CK** quantity > 0, **id_product NN** → product, **id_size NN** → size, **id_color NN** → color)
R23	order(**id**, state **NN DF** 'Shopping Cart' **CK** state **IN** Order State Types, date **NN**, id_user → authenticated_user, id_adress → adress, id_card → card)
R24	order_details(**id_order NN** → order, **id_details NN** → details)

Legenda: 1) NN = NOT NULL 2) UK = UNIQUE 3) CK = CHECK 4) DF = DEFAULT

Justificação da generalização Person para Admin ou AuthenticatedUser Optamos pela estratégia *object-oriented* porque a generalização é *disjoint*, ou seja, cada objeto apenas pertence a uma sub-árvore e terão associações diferentes. Assim conseguimos eliminar a super classe e associar as tabelas de uma forma mais eficaz.

2. Domains

Domain Name	Domain Specification
Admin Role Types	ENUM('Collaborator', 'Technician')

Domain Name	Domain Specification
Order State Types	ENUM('Shopping Cart', 'Pending', 'In Progress', 'Completed', 'Cancelled')
Notification Types	ENUM('New Promotion', 'New Collection', 'Recommended Product', 'Change in Order State', 'Payment accept', 'Product in Wishlist Available', 'Price Change of Item in Shopping Cart', 'Order', 'Report', 'Other')

3. Schema validation

TABLE R01	image
Keys	{id}, {file}
Functional Dependencies:	
FD0101	id → {file}
FD0102	file → {id}
NORMAL FORM	BCNF

TABLE R02	authenticated_user
Keys	{id}, {email}
Functional Dependencies:	
FD0201	id → {first_name, last_name, email, password, birth_date, gender, blocked, id_image}
FD0202	email → {first_name, last_name, id, password, birth_date, gender, blocked, id_image}
NORMAL FORM	BCNF

TABLE R03	admin
Keys	{id}, {email}
Functional Dependencies:	
FD0301	id → {first_name, last_name, email, password, birth_date, gender, id_image, role}

TABLE R03	admin
FD0302	email → {id, first_name, last_name, password, birth_date, gender, id_image, role}
NORMAL FORM	BCNF

TABLE R04	notification
Keys	{id}
Functional Dependencies:	
FD0401	id → {type, message, date}
NORMAL FORM	BCNF

TABLE R05	admin_notification
Keys	{id_admin, id_notification}
Functional Dependencies:	none
NORMAL FORM	BCNF

TABLE R06	user_notification
Keys	{id_user, id_notification}
Functional Dependencies:	none
NORMAL FORM	BCNF

TABLE R07	card
Keys	{id}, {number}, {code}
Functional Dependencies:	
FD0701	id → {nickname, name, number, month, year, code, id_user}
FD0702	number → {nickname, name, id, month, year, code, id_user}
FD0703	code → {nickname, name, number, month, year, id, id_user}
NORMAL FORM	BCNF

TABLE R08	country
Keys	{id}, {name}
Functional Dependencies:	
FD0801	id → {name}
FD0802	name → {id}

TABLE R08	country
NORMAL FORM	BCNF

TABLE R09	address
Keys	{id}
Functional Dependencies:	
FD0901	id → {name, company, nif, street, number, apartment, note, id_country, id_user}
NORMAL FORM	BCNF

TABLE R10	category
Keys	{id}
Functional Dependencies:	
FD1001	id → {name, id_super_category}
NORMAL FORM	BCNF

TABLE R11	product
Keys	{id}, {name}
Functional Dependencies:	
FD1101	id → {name, description, price, id_category}
FD1102	name → {id, description, price, id_category}
NORMAL FORM	BCNF

TABLE R12	product_image
Keys	{id_product, id_image}
Functional Dependencies:	none
NORMAL FORM	BCNF

TABLE R13	wishlist
Keys	{id_user, id_product}
Functional Dependencies:	none
NORMAL FORM	BCNF

TABLE R14	review
Keys	{id}
Functional Dependencies:	
FD1401	id → {evaluation, title, description, date, id_user, id_product}
NORMAL FORM	BCNF

TABLE R15	like
Keys	{id_user, id_review}
Functional Dependencies:	none
NORMAL FORM	BCNF

TABLE R16	report
Keys	{id}
Functional Dependencies:	
FD1601	id → {description, resolved, report_date, id_user, id_review}
NORMAL FORM	BCNF

TABLE R17	promotion
Keys	{id}
Functional Dependencies:	
FD1701	id → {discount, start_date, final_date}
NORMAL FORM	BCNF

TABLE R18	promotion_product
Keys	{id_promo, id_product}
Functional Dependencies:	none
NORMAL FORM	BCNF

TABLE R19	size
Keys	{id}, {name}
Functional Dependencies:	
FD1901	id → {name}
FD1902	name → {id}
NORMAL FORM	BCNF

TABLE R20	color
Keys	{id}, {name}
Functional Dependencies:	
FD2001	id → {name}
FD2002	name → {id}
NORMAL FORM	BCNF

TABLE R21	stock
Keys	{id_product, id_size, id_color}
Functional Dependencies:	
FD2101	id_product, id_size, id_color → {stock}
NORMAL FORM	BCNF

TABLE R22	details
Keys	{id_product, id_size, id_color}
Functional Dependencies:	
FD2201	id_product, id_size, id_color → {quantity}
NORMAL FORM	BCNF

TABLE R23	order
Keys	{id}
Functional Dependencies:	
FD2301	id → {state, date, id_user, id_adress, id_card}
NORMAL FORM	BCNF

TABLE R24	order_details
Keys	{id_order, id_details}
Functional Dependencies:	none
NORMAL FORM	BCNF

Justificação: Como todas as relações estão na forma normal de Boyce Codd, quer dizer que o modelo relacional cumpre também com a BCNF. Por esta razão, não há necessidade de normalizar o esquema relacional.

A6: Indexes, triggers, transactions and database population

Esta secção serve para demonstrar tudo o que é relacionado com a base de dados em si. É referido no ínicio a quantidade esperada de dados de cada tipo e o crescimento de cada um. Depois abordamos os índices que implementamos de forma a organizar alguns dos dados da nossa base de dados, para assim conseguirmos aceder a conteúdos importantes de forma rápida e mais eficaz. Apresentamos *triggers* e *user defined functions* que definimos para garantir o bom funcionamento, manutenção e cumprimento das nossas *business rules*, acima referidas. Por final, são descritas as *transactions* essenciais para o sistema da About Fashion.

1. Database Workload

Relation reference	Relation Name	Order of magnitude	Estimated growth
R01	image	10k (*tens of thousands*)	10 (tens)/day
R02	authenticated_user	10k (tens of thousands)	10/day
R03	admin	100	no growth
R04	notification	10k	100 (hundreds)/day
R05	admin_notification	1k (thousands)	100/day
R06	user_notification	10k	100/day
R07	card	10k	10/day
R08	country	100	no growth
R09	address	10k	10/day
R10	category	10	no growth
R11	product	1k	1/day
R12	product_image	1k	1/day
R13	wishlist	1k	10/day
R14	review	1k	10/day
R15	like	1k	10/day
R16	report	100 (hundreds)	1(units)/day
R17	promotion	10	1/month
R18	promotion_product	100	10/month
R19	size	1	no growth
R20	color	1	no growth
R21	stock	10k	1/day
R22	details	1k	10/day
R23	order	1k	10/day
R24	order_details	1k	10/day

2. Proposed Indexes

2.1. Performance Indexes

Index	IDX01
Relation	order
Attribute	id_user
Type	B-tree
Cardinality	Média
Clustering	Sim
Justification	Ao implementar um índice do tipo *B-tree* na tabela order com o atributo id_user, conseguimos ordenar as encomendas por utilizador, ou seja, conseguimos aceder a todas as encomendas realizadas por um certo utilizador de uma forma mais rápida e eficiente.

SQL code:

```
CREATE INDEX user_order_idx ON user_order USING btree (id_user);
CLUSTER user_order USING user_order_idx;
```

Index	IDX02
Relation	stock
Attribute	id_product
Type	Hash
Cardinality	Alta
Clustering	Não
Justification	A tabela 'stock' é frequentemente utilizada para obter a disponibilidade de produtos específicos. A filtragem é feita através da igualdade entre os respetivos id's, sendo assim, o tipo hash é o que se enquadra melhor neste caso.

SQL code:

```
CREATE INDEX product_stock_idx ON stock USING hash (id_product);
```

Index	IDX03
Relation	promotion
Attribute	final_date
Type	B-tree
Cardinality	Média
Clustering	Não
Justification	A tabela 'promotion' é frequentemente acedida para alocar/desalocar certas promoções a alguns artigos. Como uma das informações principais de uma promoção é a sua data de fim, para além da percentagem de desconto, utilizamos o tipo de índice *B-tree*, porque nos permite consultar os intervalos de datas de forma mais rápida.

SQL code:

```
CREATE INDEX final_date_promo_idx ON promotion USING btree (final_date);
```

Index	IDX04
Relation	authenticated_user
Attribute	first_name
Type	B-tree
Cardinality	Alta
Clustering	Não
Justification	Um índice do tipo *B-tree* implementado na relação authenticated_user utilizando o atributo first_name, é ótimo para organizar os utilizadores pelo primeiro nome. Assim já possuimos os dados significativamente organizados e torna a pesquisa pelos utilizadores mais rápida.

SQL code:

```
CREATE INDEX user_first_name_idx ON authenticated_user USING btree (first_name);
```

2.2. Full-text Search Indexes

Index	IDX04
Relation	product
Attribute	name, description
Type	GIN
Clustering	Não
Justification	A implementação deste índice tem como objetivo assegurar recursos de pesquisa de texto completo, a serem utilizados em produtos com determinados nomes e/ou descrições. No índice, foi dada maior importância ao campo do nome do produto e o tipo escolhido para a sua criação foi GIN, uma vez que os campos indexados não serão modificados frequentemente.

SQL code:

```
-- Função para utilização do trigger
CREATE FUNCTION product_search()
RETURNS TRIGGER AS
$$ BEGIN
    IF TG_OP = 'INSERT'
    THEN
        NEW.tsvectors = (
         setweight(to_tsvector('english', NEW.name), 'A') ||
         setweight(to_tsvector('english', NEW.description), 'B')
        );
    END IF;
    IF TG_OP = 'UPDATE' THEN
        IF (NEW.name <> OLD.name OR NEW.description <> OLD.description)
        THEN
           NEW.tsvectors = (
             setweight(to_tsvector('english', NEW.name), 'A') ||
             setweight(to_tsvector('english', NEW.description), 'B')
           );
        END IF;
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

-- Trigger para suportar o índice
CREATE TRIGGER product_search_update
```

```
BEFORE INSERT OR UPDATE ON product
FOR EACH ROW
EXECUTE PROCEDURE product_search();

-- Índice
CREATE INDEX search_idx ON product USING GIN (tsvectors);
```

3. Triggers and User Defined Functions

Trigger	TRIGGER01
Description	Trigger para dar suporte à implementação do INDEX04 (que é um Full-Text Search Index)

SQL code:

```
CREATE FUNCTION product_search()
RETURNS TRIGGER AS
$$ BEGIN
    IF TG_OP = 'INSERT'
    THEN
        NEW.tsvectors = (
         setweight(to_tsvector('english', NEW.name), 'A') ||
         setweight(to_tsvector('english', NEW.description), 'B')
        );
    END IF;
    IF TG_OP = 'UPDATE' THEN
        IF (NEW.name <> OLD.name OR NEW.description <> OLD.description)
        THEN
           NEW.tsvectors = (
             setweight(to_tsvector('english', NEW.name), 'A') ||
             setweight(to_tsvector('english', NEW.description), 'B')
           );
        END IF;
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER product_search_update
BEFORE INSERT OR UPDATE ON product
FOR EACH ROW
EXECUTE PROCEDURE product_search();
```

Trigger	TRIGGER02
Description	Verificar se um utilizador já comprou o produto antes de fazer uma review
Justification	Um utilizador apenas tem a possibilidade de fazer uma *review* a um produto que tenha anteriormente adquirido, ou seja, a um produto que esteja presente numa das suas encomendas. Este trigger garante que a *business rule* 1 é cumprida.

SQL code:

```
CREATE FUNCTION check_review_privileges()
RETURNS TRIGGER AS
$$ BEGIN
    IF NOT EXISTS (SELECT *
            FROM (SELECT DISTINCT id_user, id_product, id_size, id_color
            FROM user_order, order_details, details
            WHERE user_order.id = order_details.id_order AND
            order_details.id_details = details.id
            ORDER BY id_user, id_product, id_size, id_color) AS user_purchases
            WHERE NEW.id_user = user_purchases.id_user
            AND NEW.id_product = user_purchases.id_product)
    THEN
        RAISE EXCEPTION 'An item can only be reviewed if it has been purchased';
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER before_review_insert
BEFORE INSERT ON review
FOR EACH ROW
EXECUTE PROCEDURE check_review_privileges();
```

Trigger	TRIGGER03
Description	Ao apagar a conta de um utilizador, toda a informação partilhada (encomendas, gostos, reviews) deve ser mantida no sistema

Trigger	TRIGGER03
Justification	Precisamos de um trigger para garantir estes acontecimentos, porque não podemos perder informações como encomendas, *likes* e *reviews*. Estas terão que ter um autor *null* de forma a conseguirmos manter os dados que precisamos, sem continuar a armazenar a informação das pessoas que decidiram eliminar a conta. Este trigger garante que a *business rule* 2 é cumprida.

SQL code:

```
CREATE FUNCTION delete_user_information()
RETURNS TRIGGER AS
$$ BEGIN
    UPDATE report SET id_user = NULL WHERE id_user = OLD.id_user;
    UPDATE review SET id_user = NULL WHERE id_user = OLD.id_user;
    UPDATE user_like SET id_user = NULL WHERE id_user = OLD.id_user;
    DELETE FROM wishlist WHERE id_user = OLD.id_user;
    UPDATE user_order SET id_user = NULL,
                          id_address = NULL,
                          id_card = NULL
                          WHERE id_user = OLD.id_user;
    RETURN OLD;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER delete_user_account
AFTER DELETE ON authenticated_user
FOR EACH ROW
EXECUTE PROCEDURE delete_user_information();
```

Trigger	TRIGGER04
Description	Verificar se uma order com estado diferente de “Shopping Cart” tem todos os parâmetros preenchidos

Trigger	TRIGGER04
Justification	Como a tabela Order também é utilizada para representar o carrinho de compras quando possui o estado 'Shopping Cart', então temos que garantir que todos os parâmetros necessários para os restantes estados não são nulos quando esta tem um estado diferente. Este trigger garante que a *business rule* 3 é cumprida.

SQL code:

```
CREATE FUNCTION order_parameters()
RETURNS TRIGGER AS
$$ BEGIN
    IF (OLD.state = 'Shopping Cart' AND NEW.state <> 'Shopping Cart')
    THEN
        IF NEW.id_user IS NULL OR NEW.id_address IS NULL OR NEW.id_card IS NULL
        THEN
            RAISE EXCEPTION 'Order must have an user, an address and a card';
        END IF;
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER check_order_parameters
BEFORE UPDATE ON user_order
FOR EACH ROW
EXECUTE PROCEDURE order_parameters();
```

Trigger	TRIGGER05
Description	O utilizador não pode meter um *like* na própia review
Justification	Um utilizador não tem permissões para gostar da própria *review*, se não estaria a sobrevalorizar a sua opinião/avaliação. Este trigger garante que a *business rule* 4 é cumprida.

SQL code:

```
CREATE FUNCTION check_like_privileges()
RETURNS TRIGGER AS
$$ BEGIN
    IF EXISTS (SELECT id_user
               FROM review
               WHERE id_review = NEW.id_review AND id_user = NEW.id_user)
    THEN
        RAISE EXCEPTION 'A user cannot like his own review';
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER before_like_insert
BEFORE INSERT ON user_like
FOR EACH ROW
EXECUTE PROCEDURE check_like_privileges();
```

Trigger	TRIGGER06
Description	Um utilizador não pode reportar a sua review
Justification	Um utilizador não tem forma de reportar a sua *review*, porque em vez de o fazer pode editá-la para ir de encontro ao seu pensamento atual. Este trigger garante que a *business rule* 5 é cumprida.

SQL code:

```
CREATE FUNCTION check_report_privileges()
RETURNS TRIGGER AS
$$ BEGIN
    IF EXISTS (SELECT id_user
               FROM review
               WHERE id_review = NEW.id_review AND id_user = NEW.id_user)
    THEN
        RAISE EXCEPTION 'A user cannot report his own review';
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER before_report_insert
BEFORE INSERT ON report
```

```
FOR EACH ROW
EXECUTE PROCEDURE check_report_privileges();
```

User Defined Function	UDF01
Description	Verificar o stock dos produtos no momento da adição ao carrinho
Justification	Esta função permite verificar se existe *stock* do produto pretendido, para assim poder ser adicionado ao carrinho sem conflitos.

SQL code:

```
CREATE FUNCTION check_stock(Product details)
RETURNS INTEGER AS
$$ BEGIN
    IF Product.quantity = 0 THEN
        RAISE EXCEPTION 'Product out of stock';
    END IF;
    RETURN 1;
END; $$
LANGUAGE plpgsql;
```

User Defined Function	UDF02
Description	Adicionar um artigo ao carrinho
Justification	Esta função permite adicionar um produto ao carrinho de compras, quando existe *stock* do mesmo.

SQL code:

```
CREATE FUNCTION add_product_to_cart(Cart user_order, Product details)
RETURNS user_order AS
$$ BEGIN
  IF (Cart.state = 'Shopping Cart') THEN
    IF check_stock(Product) = 1 THEN
        INSERT INTO order_details VALUES (Cart.id, Product.id);
    ELSE
        RAISE EXCEPTION 'Error adding product to cart';
    END IF;
  ELSE
      RAISE EXCEPTION 'Error adding product to cart';
  END IF;
  RETURN Cart;
```

```
END; $$
LANGUAGE plpgsql;
```

User Defined Function	UDF03
Description	Remover um artigo ao carrinho
Justification	Esta função permite ao utilizador remover do carrinho de compras um produto que já não deseja.

SQL code:

```
CREATE FUNCTION remove_product_from_cart(Cart user_order, Product details)
RETURNS user_order AS
$$ BEGIN
    IF Cart.state = 'Shopping Cart' THEN
        DELETE FROM order_details WHERE id_order = Cart.id AND id_details = Product.id;
        RETURN Cart;
    ELSE
        RAISE EXCEPTION 'Error removing product from cart';
    END IF;
END; $$
LANGUAGE plpgsql;
```

User Defined Function	UDF04
Description	Aceder ao preço de um produto com a promoção aplicada
Justification	Com esta função é possível ver o preço final do produto com o desconto da respetiva promoção aplicado.

SQL code:

```
CREATE FUNCTION product_price_with_promotion(Product product, Promotion promotion)
RETURNS NUMERIC AS
$$ BEGIN
    RETURN Product.price * (1 - Promotion.discount);
END; $$
LANGUAGE plpgsql;
```

4. Transactions

Transaction	TRAN01
Description	Checkout do carrinho
Justification	De forma a manter a consistência da base de dados durante o checkout do carrinho, será preciso efeturar uma transaction de modo a atualizar o stock do produto depois de este ser associado a um cart. O nível de isolamento é REPEATABLE READ, pois queremos aceder à base de dados num estado anterior ao inicio da transaction.
Isolation level	REPEATABLE READ

SQL code:

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

INSERT INTO details(id, quantity, id_product, id_size, id_color)
VALUES ($id_details, $quantity, $id_product, $id_size, $id_color);

SELECT add_product_to_cart($id_order, $details);

-- Remove products from the stock
FROM (
    SELECT id_order, id_product, quantity, color, size
    FROM details INNER JOIN order_details
    ON details.id = order_details.id_details
    WHERE id_order = $id_order
) AS order_products
WHERE stock.id_product = order_products.id_product AND
      stock.size = order_products.size AND
      stock.color = order_products.color AND
      stock.stock >= order_products.quantity;
SET stock = stock - quantity;

END TRANSACTION;
```

Transaction	TRAN02
Description	Cancelar uma encomenda

Transaction	TRAN02
Justification	Para cancelar uma encomenda é preciso uma *transaction* para cobrir as alterações que devem ser feitas no *stock* e nas encomendas do utilizador. Para isso tem que se garantir que estas operações são realizadas no mesmo estado da base de dados. Assim o nível de isolamento prentendido será REPEATABLE READ.
Isolation level	REPEATABLE READ

SQL code:

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Delete order row
UPDATE user_order
SET status="Cancelled"
WHERE id=$id_order;

-- Restore the products from the cancelled order
UPDATE stock
SET stock = stock + quantity
FROM (
    SELECT id_order, id_product, quantity, color, size
    FROM details INNER JOIN order_details
    ON details.id = order_details.id_details
    WHERE id_order = $id_order
) AS order_products(id_order, id_product, quantity, color, size)
WHERE stock.id_product = order_products.id_product AND
      stock.size = order_products.size AND
      stock.color = order_products.color;

SELECT remove_product_from_cart($id_order, $details);

END TRANSACTION;
```

Transaction	TRAN03
Description	Adicionar um artigo

Transaction	TRAN03
Justification	Para manter a integridade e consistência da base de dados é necessária uma *transaction* para adicionar um novo artigo e garantir que todo o código executa sem erros. O nível de isolamento é REPEATABLE READ, porque ao inserir um novo produto, apenas se deve ter em conta o estado da base de dados antes da *transaction* começar.
Isolation level	REPEATABLE READ

SQL code:

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Insert product
INSERT INTO product (id, name, description, price, id_category)
VALUES ($id_product, $name, $description, $price, $id_category);

-- Insert image
INSERT INTO image (id, file)
VALUES ($id_image, $file);

-- Insert the product image
INSERT INTO product_image(id_product, id_image)
VALUES ($id_product, $id_image);

-- Insert product color if not exists
IF NOT EXISTS (SELECT * FROM color WHERE name=$name_color)
BEGIN
    INSERT INTO color (id, name)
    VALUES ($id, $name);
END

-- Insert product size if not exists
IF NOT EXISTS (SELECT * FROM size WHERE name=$name_size)
BEGIN
    INSERT INTO size (id, name)
    VALUES ($id, $name_size);
END
```

```
-- Insert the new product in stock
INSERT INTO stock (stock, id_product, id_size, id_color)
VALUES (1, $id_product, $id_size, $id_color)
ON DUPLICATE KEY UPDATE
  stock = stock + 1;

END TRANSACTION;
```

Transaction	TRAN04
Description	Remover um artigo
Justification	Ao remover um artigo temos de o apagar de várias tabelas, como o stock, product e também a sua imagem. Desta forma, é preciso uma transaction para remover um artigo, e para que transactions concurrentes não interferiram nesta operação será utilizado o nível de isolamento REPEATABLE READ.
Isolation level	REPEATABLE READ

SQL code:

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Delete product images
DELETE FROM product_image
WHERE id_product = $id_product;

DELETE FROM image
WHERE id IN (SELECT id FROM image WHERE id=$id_image);

-- Delete the product from stock
DELETE FROM stock
WHERE id_product = $id_product

-- Delete product
DELETE FROM product
WHERE id=$id_product

END TRANSACTION;
```

Annex A. SQL Code

A.1. Database schema

```
--DROP's
DROP TYPE IF EXISTS admin_type CASCADE;
DROP TYPE IF EXISTS order_state_type CASCADE;
DROP TYPE IF EXISTS notification_type CASCADE;
DROP TYPE IF EXISTS report_type CASCADE;
DROP TABLE IF EXISTS image CASCADE;
DROP TABLE IF EXISTS authenticated_user CASCADE;
DROP TABLE IF EXISTS admin CASCADE;
DROP TABLE IF EXISTS notification CASCADE;
DROP TABLE IF EXISTS admin_notification CASCADE;
DROP TABLE IF EXISTS user_notification CASCADE;
DROP TABLE IF EXISTS card CASCADE;
DROP TABLE IF EXISTS country CASCADE;
DROP TABLE IF EXISTS address CASCADE;
DROP TABLE IF EXISTS category CASCADE;
DROP TABLE IF EXISTS product CASCADE;
DROP TABLE IF EXISTS product_image CASCADE;
DROP TABLE IF EXISTS wishlist CASCADE;
DROP TABLE IF EXISTS review CASCADE;
DROP TABLE IF EXISTS report CASCADE;
DROP TABLE IF EXISTS promotion CASCADE;
DROP TABLE IF EXISTS promotion_product CASCADE;
DROP TABLE IF EXISTS size CASCADE;
DROP TABLE IF EXISTS color CASCADE;
DROP TABLE IF EXISTS stock CASCADE;
DROP TABLE IF EXISTS details CASCADE;
DROP TABLE IF EXISTS user_order CASCADE;
DROP TABLE IF EXISTS order_details CASCADE;
DROP TABLE IF EXISTS user_like CASCADE;
DROP FUNCTION IF EXISTS check_stock CASCADE;
DROP FUNCTION IF EXISTS add_product_to_cart CASCADE;
DROP FUNCTION IF EXISTS remove_product_from_cart CASCADE;
DROP FUNCTION IF EXISTS product_price_with_promotion CASCADE;
DROP FUNCTION IF EXISTS delete_user_information CASCADE;
DROP FUNCTION IF EXISTS product_search_update CASCADE;
DROP TRIGGER IF EXISTS delete_user_account on authenticated_user CASCADE;
DROP FUNCTION IF EXISTS check_review_privileges CASCADE;
DROP TRIGGER IF EXISTS before_review_insert on review CASCADE;
DROP FUNCTION IF EXISTS check_like_privileges CASCADE;
DROP TRIGGER IF EXISTS before_like_insert on user_like CASCADE;
DROP FUNCTION IF EXISTS check_report_privileges CASCADE;
DROP TRIGGER IF EXISTS before_report_insert on report CASCADE;
DROP FUNCTION IF EXISTS order_parameters CASCADE;
DROP TRIGGER IF EXISTS check_order_parameters on user_order CASCADE;
DROP FUNCTION IF EXISTS product_search CASCADE;
DROP TRIGGER IF EXISTS product_search_update on product CASCADE;
DROP INDEX IF EXISTS user_order_idx  CASCADE;
DROP INDEX IF EXISTS product_stock_idx CASCADE;
DROP INDEX IF EXISTS final_date_promo_idx  CASCADE;
DROP INDEX IF EXISTS user_first_name_idx  CASCADE; user_first_name_idx
DROP INDEX IF EXISTS search_idx  CASCADE;
```

```sql
--TYPE's
CREATE TYPE admin_type AS ENUM ('Collaborator', 'Technician');
CREATE TYPE order_state_type AS ENUM (
    'Shopping Cart',
    'Pending',
    'In Progress',
    'Completed',
    'Cancelled'
);
CREATE TYPE notification_type AS ENUM (
    'New Promotion',
    'New Collection',
    'Recommended Product',
    'Change in Order State',
    'Payment accept',
    'Product in Wishlist Available',
    'Price Change of Item in Shopping Cart',
    'Order',
    'Report',
    'Other'
);
```

```sql
CREATE TABLE image (
    id SERIAL PRIMARY KEY,
    file TEXT NOT NULL CONSTRAINT image_unique UNIQUE
);
CREATE TABLE authenticated_user (
    id SERIAL PRIMARY KEY,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    birth_date DATE,
    gender TEXT,
    blocked BOOLEAN NOT NULL DEFAULT FALSE,
    id_image INTEGER REFERENCES image(id) ON UPDATE CASCADE --VER TRIGGER
);
CREATE TABLE admin(
    id SERIAL PRIMARY KEY,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    birth_date DATE,
    gender TEXT,
    id_image INTEGER REFERENCES image(id) ON UPDATE CASCADE,
    --VER TRIGGER
    role admin_type NOT NULL
);
CREATE TABLE notification(
    id SERIAL PRIMARY KEY,
    type_notification notification_type NOT NULL,
    message TEXT NOT NULL,
    date TIMESTAMP NOT NULL
);
CREATE TABLE admin_notification(
    id_admin INTEGER NOT NULL REFERENCES admin(id) ON UPDATE CASCADE,
    id_notification INTEGER NOT NULL REFERENCES notification(id) ON UPDATE CASCADE,
    PRIMARY KEY (id_admin, id_notification)
);
CREATE TABLE user_notification(
    id_user INTEGER NOT NULL REFERENCES authenticated_user(id) ON UPDATE CASCADE,
    id_notification INTEGER NOT NULL REFERENCES notification(id) ON UPDATE CASCADE,
    PRIMARY KEY (id_user, id_notification)
);
```

```
CREATE TABLE card(
    id SERIAL PRIMARY KEY,
    nickname TEXT,
    name TEXT NOT NULL,
    number TEXT NOT NULL CONSTRAINT cart_unique UNIQUE,
    month SMALLINT NOT NULL,
    year SMALLINT NOT NULL,
    code SMALLINT NOT NULL,
    id_user INTEGER NOT NULL REFERENCES authenticated_user(id) ON UPDATE CASCADE
);
CREATE TABLE country(
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL CONSTRAINT country_unique UNIQUE
);
CREATE TABLE address(
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    company TEXT,
    nif TEXT,
    street TEXT NOT NULL,
    number SMALLINT NOT NULL,
    apartment TEXT,
    note TEXT,
    id_country INTEGER NOT NULL REFERENCES country(id) ON UPDATE CASCADE,
    id_user INTEGER NOT NULL REFERENCES authenticated_user(id) ON UPDATE CASCADE
);
CREATE TABLE category(
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    id_super_category INTEGER REFERENCES category(id) ON UPDATE CASCADE
);
CREATE TABLE product(
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL CONSTRAINT product_unique UNIQUE,
    description TEXT NOT NULL,
    price NUMERIC NOT NULL,
    id_category INTEGER NOT NULL REFERENCES category(id) ON UPDATE CASCADE
);
CREATE TABLE product_image(
    id_product INTEGER NOT NULL REFERENCES product(id) ON UPDATE CASCADE,
    id_image INTEGER NOT NULL REFERENCES image(id) ON UPDATE CASCADE,
    PRIMARY KEY (id_product, id_image)
);
CREATE TABLE wishlist(
    id_user INTEGER NOT NULL REFERENCES authenticated_user(id) ON UPDATE CASCADE
    id_product INTEGER NOT NULL REFERENCES product(id) ON UPDATE CASCADE
);
```

```sql
CREATE TABLE review(
    id SERIAL PRIMARY KEY,
    evaluation SMALLINT NOT NULL CHECK (
        evaluation > 0
        AND evaluation <= 5
    ),
    title TEXT NOT NULL,
    description TEXT NOT NULL,
    date TIMESTAMP NOT NULL,
    id_user INTEGER REFERENCES authenticated_user(id) ON UPDATE CASCADE,
    id_product INTEGER NOT NULL REFERENCES product(id) ON UPDATE CASCADE
);
CREATE TABLE user_like(
    id_user INTEGER REFERENCES authenticated_user(id) ON UPDATE CASCADE,
    id_review INTEGER NOT NULL REFERENCES review(id) ON UPDATE CASCADE,
    PRIMARY KEY (id_user, id_review)
);
CREATE TABLE report(
    id SERIAL PRIMARY KEY,
    description TEXT NOT NULL,
    resolved boolean NOT NULL DEFAULT FALSE,
    report_date TIMESTAMP NOT NULL,
    id_review INTEGER REFERENCES review(id) ON UPDATE CASCADE,
    id_user INTEGER REFERENCES authenticated_user(id) ON UPDATE CASCADE
);
CREATE TABLE promotion(
    id SERIAL PRIMARY KEY,
    discount NUMERIC NOT NULL CHECK (
        discount > 0
        AND discount < 100
    ),
    start_date  TIMESTAMP NOT NULL,
    final_date TIMESTAMP NOT NULL CHECK (final_date > start_date)
);
CREATE TABLE promotion_product(
    id_promotion INTEGER NOT NULL REFERENCES promotion(id) ON UPDATE CASCADE,
    id_product INTEGER NOT NULL REFERENCES product(id) ON UPDATE CASCADE,
    PRIMARY KEY (id_promotion, id_product)
);
CREATE TABLE size(
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL CONSTRAINT size_unique UNIQUE
);
CREATE TABLE color(
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL CONSTRAINT color_unique UNIQUE
);
```

```sql
CREATE TABLE stock(
    stock SMALLINT NOT NULL CHECK (stock >= 0),
    id_product INTEGER NOT NULL REFERENCES product(id) ON UPDATE CASCADE,
    id_size INTEGER NOT NULL REFERENCES size(id) ON UPDATE CASCADE,
    id_color INTEGER NOT NULL REFERENCES color(id) ON UPDATE CASCADE
);
CREATE TABLE details(
    id SERIAL PRIMARY KEY,
    quantity SMALLINT NOT NULL CHECK (quantity > 0),
    id_product INTEGER NOT NULL REFERENCES product(id) ON UPDATE CASCADE,
    id_size INTEGER NOT NULL REFERENCES size(id) ON UPDATE CASCADE,
    id_color INTEGER NOT NULL REFERENCES color(id) ON UPDATE CASCADE
);
CREATE TABLE user_order(
    id SERIAL PRIMARY KEY,
    status order_state_type NOT NULL DEFAULT 'Shopping Cart',
    date TIMESTAMP NOT NULL,
    id_user INTEGER REFERENCES authenticated_user(id) ON UPDATE CASCADE,
    id_address INTEGER REFERENCES address(id) ON UPDATE CASCADE,
    id_card INTEGER REFERENCES card(id) ON UPDATE CASCADE
);
CREATE TABLE order_details(
    id_order INTEGER NOT NULL REFERENCES user_order(id) ON UPDATE CASCADE,
    id_details INTEGER NOT NULL REFERENCES details(id) ON UPDATE CASCADE,
    PRIMARY KEY (id_order, id_details)
);
```

```sql
--INDICES

-- Index na tabela user_order no atributo id_user

CREATE INDEX user_order_idx ON user_order USING btree (id_user);
CLUSTER user_order USING user_order_idx;

-- Index na tabela stock no atributo id_product

CREATE INDEX product_stock_idx ON stock USING hash (id_product);

-- Index na tabela promotion no atributo final_date

CREATE INDEX final_date_promo_idx ON promotion USING btree (final_date);

-- Index na tabela authenticated_user no atributo first_name

CREATE INDEX user_first_name_idx ON authenticated_user USING btree (first_name);
```

```sql
--FULL-TEXT SEARCH INDICES

-- Full-text search indice (e trigger para suporte) na tabela product
-- nos atributos name e description

ALTER TABLE product
ADD COLUMN tsvectors TSVECTOR;

CREATE FUNCTION product_search()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT'
    THEN
        NEW.tsvectors = (
            setweight(to_tsvector('english', NEW.name), 'A') ||
            setweight(to_tsvector('english', NEW.description), 'B')
        );
    END IF;
    IF TG_OP = 'UPDATE'
    THEN
        IF (NEW.name <> OLD.name OR NEW.description <> OLD.description)
        THEN
            NEW.tsvectors = (
                setweight(to_tsvector('english', NEW.name), 'A') ||
                setweight(to_tsvector('english', NEW.description), 'B')
            );
        END IF;
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER product_search_update
BEFORE INSERT OR UPDATE ON product
FOR EACH ROW
EXECUTE PROCEDURE product_search();

CREATE INDEX search_idx ON product USING GIN (tsvectors);
```

```
--TRIGGERS and User Defined Functions

-- Verificar o stock dos produtos no momento da adição ao carrinho

CREATE FUNCTION check_stock(Product details)
RETURNS INTEGER AS
$$ BEGIN
    IF Product.quantity = 0 THEN
        RAISE EXCEPTION 'Product out of stock';
    END IF;
    RETURN 1;
END; $$
LANGUAGE plpgsql;

-- Adicionar um artigo ao carrinho

CREATE FUNCTION add_product_to_cart(Cart user_order, Product details)
RETURNS user_order AS
$$ BEGIN
  IF (Cart.state = 'Shopping Cart') THEN
      IF check_stock(Product) = 1 THEN
          INSERT INTO order_details VALUES (Cart.id, Product.id);
      ELSE
          RAISE EXCEPTION 'Error adding product to cart';
      END IF;
  ELSE
      RAISE EXCEPTION 'Error adding product to cart';
  END IF;
  RETURN Cart;
END; $$
LANGUAGE plpgsql;

-- Remover um artigo ao carrinho

CREATE FUNCTION remove_product_from_cart(Cart user_order, Product details)
RETURNS user_order AS
$$ BEGIN
    IF Cart.state = 'Shopping Cart' THEN
        DELETE FROM order_details WHERE id_order = Cart.id AND id_details = Product.id;
        RETURN Cart;
    ELSE
        RAISE EXCEPTION 'Error removing product from cart';
    END IF;
END; $$
LANGUAGE plpgsql;
```

```sql
-- Aceder ao preço de um produto com a promoção aplicada

CREATE FUNCTION product_price_with_promotion(Product product, Promotion promotion)
RETURNS NUMERIC AS
$$ BEGIN
    RETURN Product.price * (1 - Promotion.discount);
END; $$
LANGUAGE plpgsql;

-- Ao apagar a conta de um utilizador toda a informação partilhada (encomendas, gos

CREATE FUNCTION delete_user_information()
RETURNS TRIGGER AS
$$ BEGIN
    UPDATE report SET id_user = NULL WHERE id_user = OLD.id_user;
    UPDATE review SET id_user = NULL WHERE id_user = OLD.id_user;
    UPDATE user_like SET id_user = NULL WHERE id_user = OLD.id_user;
    DELETE FROM wishlist WHERE id_user = OLD.id_user;
    UPDATE user_order SET id_user = NULL,
                          id_address = NULL,
                          id_card = NULL
                          WHERE id_user = OLD.id_user;
    RETURN OLD;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER delete_user_account
AFTER DELETE ON authenticated_user
FOR EACH ROW
EXECUTE PROCEDURE delete_user_information();
```

```sql
-- Verificar se um utilizador já comprou o produto antes de fazer uma review

CREATE FUNCTION check_review_privileges()
RETURNS TRIGGER AS
$$ BEGIN
    IF NOT EXISTS (SELECT *
                   FROM (SELECT DISTINCT id_user, id_product, id_size, id_color
                        FROM user_order, order_details, details
                        WHERE user_order.id = order_details.id_order AND order_details.id_details = details.id
                        ORDER BY id_user, id_product, id_size, id_color) AS user_purchases
                   WHERE NEW.id_user = user_purchases.id_user AND NEW.id_product = user_purchases.id_product)
                   -- se quisermos especificar o tamanho e cor podemos adicionar aqui
    THEN
        RAISE EXCEPTION 'An item can only be reviewed if it has been purchased';
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER before_review_insert
BEFORE INSERT ON review
FOR EACH ROW
EXECUTE PROCEDURE check_review_privileges();

-- O utilizador não pode meter um like na própia review

CREATE FUNCTION check_like_privileges()
RETURNS TRIGGER AS
$$ BEGIN
    IF EXISTS (SELECT id_user
               FROM review
               WHERE id_review = NEW.id_review AND id_user = NEW.id_user)
    THEN
        RAISE EXCEPTION 'A user cannot like his own review';
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER before_like_insert
BEFORE INSERT ON user_like
FOR EACH ROW
EXECUTE PROCEDURE check_like_privileges();
```

```sql
-- Um utilizador não pode reportar a sua review

CREATE FUNCTION check_report_privileges()
RETURNS TRIGGER AS
$$ BEGIN
    IF EXISTS (SELECT id_user
               FROM review
               WHERE id_review = NEW.id_review AND id_user = NEW.id_user)
    THEN
        RAISE EXCEPTION 'A user cannot report his own review';
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER before_report_insert
BEFORE INSERT ON report
FOR EACH ROW
EXECUTE PROCEDURE check_report_privileges();

-- verificar se uma order com estado diferente de "Shopping Cart" tem todos os parametros preenchidos

CREATE FUNCTION order_parameters()
RETURNS TRIGGER AS
$$ BEGIN
    IF (OLD.state = 'Shopping Cart' AND NEW.state <> 'Shopping Cart')
    THEN
        IF NEW.id_user IS NULL OR NEW.id_address IS NULL OR NEW.id_card IS NULL
        THEN
            RAISE EXCEPTION 'Order must have an user, an address and a card';
        END IF;
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER check_order_parameters
BEFORE UPDATE ON user_order
FOR EACH ROW
EXECUTE PROCEDURE order_parameters();
```

A.2. Database population

```
--COLOR
insert into color (id, name) values (1, 'Fuscia');
insert into color (id, name) values (2, 'Crimson');
insert into color (id, name) values (3, 'Turquoise');
insert into color (id, name) values (4, 'Mauv');
insert into color (id, name) values (5, 'Maroon');
insert into color (id, name) values (6, 'Blue');
insert into color (id, name) values (7, 'Pink');
insert into color (id, name) values (8, 'Teal');
insert into color (id, name) values (9, 'Orange');
insert into color (id, name) values (10, 'Red');
```

```
--COUNTRY

insert into country (id, name) values (1, 'Japan');
insert into country (id, name) values (2, 'Panama');
insert into country (id, name) values (3, 'Poland');
insert into country (id, name) values (4, 'China');
insert into country (id, name) values (5, 'Ukraine');
insert into country (id, name) values (6, 'Morocco');
insert into country (id, name) values (7, 'Spain');
insert into country (id, name) values (8, 'Sweden');
insert into country (id, name) values (9, 'Pakistan');
insert into country (id, name) values (10, 'Indonesia');
```

```
-- SIZE

insert into size (id, name) values (1, 'Extra Small');
insert into size (id, name) values (2, 'Small');
insert into size (id, name) values (3, 'Medium');
insert into size (id, name) values (4, 'Large');
insert into size (id, name) values (5, 'Extra Large');
```

```
-- IMAGE
insert into image values (1, 'https://robohash.org/quodsitfuga.png?size=500x500&
insert into image values (2, 'https://robohash.org/velitmolestiaequi.png?size=50
insert into image values (3, 'https://robohash.org/iureipsamvoluptatem.png?size=
insert into image values (4, 'https://robohash.org/assumendaaliquamet.png?size=5
insert into image values (5, 'https://robohash.org/natusquaerem.png?size=500x500
insert into image values (6, 'https://robohash.org/culpavoluptasipsam.png?size=5
insert into image values (7, 'https://robohash.org/quiearumnon.png?size=500x500&
insert into image values (8, 'https://robohash.org/abautmolestias.png?size=500x5
insert into image values (9, 'https://robohash.org/idnostrumeaque.png?size=500x5
insert into image values (10, 'https://robohash.org/cupiditatequimollitia.png?si
```

```
-- Authenticated User

insert into authenticated_user values (1, 'Daniele', 'Groomebridge', 'dgroomebri
insert into authenticated_user values (2, 'Shela', 'Cianni', 'scianni1@myspace.c
insert into authenticated_user values (3, 'Cecelia', 'McIlraith', 'cmcilraith2@l
insert into authenticated_user values (4, 'Ruby', 'Pick', 'rpick3@oaic.gov.au',
insert into authenticated_user values (5, 'Katrine', 'Stubbins', 'kstubbins4@unc
insert into authenticated_user values (6, 'Russell', 'Daville', 'rdaville5@brave
insert into authenticated_user values (7, 'Phil', 'de Zamora', 'pdezamora6@go.co
insert into authenticated_user values (8, 'Edmund', 'Marchello', 'emarchello7@wa
insert into authenticated_user values (9, 'Egan', 'Sidnell', 'esidnell8@seattlet
insert into authenticated_user values (10, 'Kyle', 'Espadero', 'kespadero9@globo
```

```
-- Admin

insert into admin values (1, 'Philippe', 'OConnolly', 'poconnolly0@shareasale.com
insert into admin values (2, 'Reamonn', 'Crinage', 'rcrinage1@histats.com', 'c797
insert into admin values (3, 'Alberta', 'Doge', 'adoge2@wufoo.com', '301bb1ef786c
insert into admin values (4, 'Iorgo', 'Pontefract', 'ipontefract3@guardian.co.uk'
insert into admin values (5, 'Guthry', 'Boddington', 'gboddington4@bigcartel.com'
insert into admin values (6, 'Geno', 'Axelbey', 'gaxelbey5@mediafire.com', '92d9b
insert into admin values (7, 'Helenka', 'Fairholm', 'hfairholm6@cpanel.net', '169
insert into admin values (8, 'Teodoro', 'Blennerhassett', 'tblennerhassett7@goo.g
insert into admin values (9, 'Flossie', 'Shout', 'fshout8@ehow.com', 'b8d43a1c354
insert into admin values (10, 'Beth', 'Downing', 'bdowning9@is.gd', '1eace07b16fc
```

```
-- Notification

insert into notification values (1, 'Payment accept', 'Maecenas ut massa quis aug
Morbi porttitor lorem id ligula. Suspendisse ornare consequat lectus.', '2021-09
insert into notification values (2, 'Price Change of Item in Shopping Cart', 'In
venenatis, turpis enim blandit mi, in porttitor pede justo eu massa. Donec dapibu
insert into notification values (3, 'Price Change of Item in Shopping Cart', 'Al
ultrices posuere cubilia Curae; Nulla dapibus dolor vel est. Donec odio justo, so
insert into notificationvalues (4, 'New Collection', 'Donec dapibus. Duis at velit
Nulla neque libero, convallis eget, eleifend luctus, ultricies eu, nibh. Quisque
insert into notification values (5, 'Payment accept', 'Vestibulum rutrum rutrum n
hendrerit at, vulputate vitae, nisl.', '2022-01-15 17:20:20');
insert into notification values (6, 'Change in Order State', 'Duis bibendum. Morb
insert into notification values (7, 'Payment accept', 'Nulla justo. Aliquam quis
cubilia Curae; Nulla dapibus dolor vel est. Donec odio justo, sollicitudin ut, su
insert into notification values (8, 'Change in Order State', 'Maecenas leo odio,
ipsum. Aliquam non mauris. Morbi non lectus. Aliquam sit amet diam in magna bibe
insert into notification values (9, 'New Promotion', 'Donec odio justo, sollicitu
Cras non velit nec nisi vulputate nonummy.', '2022-02-19 10:27:14');
insert into notification values (10, 'Product in Wishlist Available', 'Nulla ut e
ut, nulla. Sed accumsan felis.', '2021-06-25 11:08:40');
```

```sql
-- Admin Notification

insert into admin_notification (id_admin, id_notification) values (39, 128);
insert into admin_notification (id_admin, id_notification) values (4, 106);
insert into admin_notification (id_admin, id_notification) values (26, 133);
insert into admin_notification (id_admin, id_notification) values (49, 145);
insert into admin_notification (id_admin, id_notification) values (25, 123);
insert into admin_notification (id_admin, id_notification) values (4, 112);
insert into admin_notification (id_admin, id_notification) values (37, 102);
insert into admin_notification (id_admin, id_notification) values (15, 110);
insert into admin_notification (id_admin, id_notification) values (26, 115);
insert into admin_notification (id_admin, id_notification) values (26, 112);
```

```sql
-- User Notification

insert into user_notification (id_user, id_notification) values (89, 93);
insert into user_notification (id_user, id_notification) values (37, 49);
insert into user_notification (id_user, id_notification) values (24, 31);
insert into user_notification (id_user, id_notification) values (25, 46);
insert into user_notification (id_user, id_notification) values (23, 22);
insert into user_notification (id_user, id_notification) values (70, 144);
insert into user_notification (id_user, id_notification) values (51, 50);
insert into user_notification (id_user, id_notification) values (52, 10);
insert into user_notification (id_user, id_notification) values (56, 86);
insert into user_notification (id_user, id_notification) values (11, 128);
```

```sql
-- Card

insert into card values (1, null, 'Adorne Gorini', '5100175084456910', 11, 38, 87
insert into card values (2, null, 'Stirling McEntagart', '5100134149511680', 3, 4
insert into card values (3, 'jkytter2', 'Jdavie Kytter', '5100130893648801', 7, 4
insert into card values (4, null, 'Blondell Becker', '5100133054663767', 6, 44, 8
insert into card values (5, null, 'Linnea Dibbs', '5100144666005107', 7, 29, 475,
insert into card values (6, null, 'Ofelia Morpeth', '5100145790278841', 3, 24, 88
insert into card values (7, null, 'Gray Morais', '5100175147501942', 1, 50, 725,
insert into card values (8, 'ltranter7', 'Lorri Tranter', '5100178519588620', 3,
insert into card values (9, 'mpaulson8', 'Morissa Paulson', '5100139660931258', 2
insert into card values (10, 'pfrichley9', 'Phelia Frichley', '5100137547333946',
```

```sql
--Category
insert into category (id, name) values (1, 'Man');
insert into category (id, name) values (2, 'Woman');
insert into category values (3, 'Clothing', 1);
insert into category values (4, 'Footwear', 1);
insert into category values (5, 'Accessories', 1);
insert into category values (6, 'Clothing', 2);
insert into category values (7, 'Footwear', 2);
insert into category values (8, 'Accessories', 2);
insert into category values (9,'Coats & Jackets' ,3);
insert into category values (10,'Jumpers' ,3);
```

```sql
--Products

insert into product values (1, 'Emarcpo Cxpexi', 'ut rhoncus aliquet pulvinar se
insert into product values (2, 'Xtqyptd Jjioln', 'elementum ligula vehicula cons
insert into product values (3, 'Rbpyxvg Syntne', 'dapibus dolor vel est donec od
insert into product values (4, 'Tznlcrg Oalrhy', 'diam id ornare imperdiet sapie
insert into product values (5, 'Ksjghmc Dehprk', 'aenean auctor gravida sem prae
insert into product values (6, 'Rbgutwt Rnbajv', 'sagittis dui vel nisl duis ac
insert into product values (7, 'Aqlnbxc Ukibke', 'aenean auctor gravida sem prae
insert into product values (8, 'Rncpxbs Zgeeop', 'congue etiam justo etiam preti
insert into product values (9, 'Swbzkex Eehvmo', 'dui luctus rutrum nulla tellus
insert into product values (10, 'Hutyngw Anaieh', 'vulputate elementum nullam va
```

```sql
-- Wishlist

insert into wishlist (id_user, id_product) values (85, 192);
insert into wishlist (id_user, id_product) values (64, 183);
insert into wishlist (id_user, id_product) values (186, 51);
insert into wishlist (id_user, id_product) values (108, 130);
insert into wishlist (id_user, id_product) values (199, 24);
insert into wishlist (id_user, id_product) values (188, 117);
insert into wishlist (id_user, id_product) values (79, 207);
insert into wishlist (id_user, id_product) values (194, 14);
insert into wishlist (id_user, id_product) values (137, 222);
insert into wishlist (id_user, id_product) values (78, 218);
```

```sql
--Product Image

insert into product_image (id_product, id_image) values (1, 250);
insert into product_image (id_product, id_image) values (2, 251);
insert into product_image (id_product, id_image) values (3, 252);
insert into product_image (id_product, id_image) values (4, 253);
insert into product_image (id_product, id_image) values (5, 254);
insert into product_image (id_product, id_image) values (6, 255);
insert into product_image (id_product, id_image) values (7, 256);
insert into product_image (id_product, id_image) values (8, 257);
insert into product_image (id_product, id_image) values (9, 258);
insert into product_image (id_product, id_image) values (10, 259);
```

```sql
-- Promotion

insert into promotion values (1, 53, '2022-10-14 05:49:20', '2023-06-20 06:46:50
insert into promotion values (2, 54, '2022-08-16 22:44:29', '2022-12-03 13:10:53
insert into promotion values (3, 68, '2022-10-06 12:09:26', '2022-12-18 22:34:48
insert into promotion values (4, 69, '2022-09-10 04:43:46', '2023-04-14 07:40:38
insert into promotion values (5, 75, '2022-09-04 15:51:06', '2023-04-28 07:58:40
insert into promotion values (6, 34, '2022-09-16 03:30:39', '2023-05-17 22:18:19
insert into promotion values (7, 34, '2022-09-20 01:29:45', '2022-12-21 08:02:11
insert into promotion values (8, 64, '2022-09-07 12:15:34', '2022-12-03 03:36:22
insert into promotion values (9, 71, '2022-10-10 11:56:55', '2023-05-06 22:53:57
insert into promotion values (10, 56, '2022-09-10 13:01:05', '2022-11-07 22:43:1
```

```sql
--Promotion Product

insert into promotion_product (id_promotion, id_product) values (40, 245);
insert into promotion_product (id_promotion, id_product) values (8, 22);
insert into promotion_product (id_promotion, id_product) values (13, 142);
insert into promotion_product (id_promotion, id_product) values (15, 149);
insert into promotion_product (id_promotion, id_product) values (44, 133);
insert into promotion_product (id_promotion, id_product) values (30, 145);
insert into promotion_product (id_promotion, id_product) values (19, 55);
insert into promotion_product (id_promotion, id_product) values (11, 32);
insert into promotion_product (id_promotion, id_product) values (36, 76);
insert into promotion_product (id_promotion, id_product) values (5, 204);
```

```sql
--Stock

insert into stock (stock, id_product, id_size, id_color) values (182, 198, 5, 5)
insert into stock (stock, id_product, id_size, id_color) values (515, 72, 3, 14)
insert into stock (stock, id_product, id_size, id_color) values (408, 174, 4, 13
insert into stock (stock, id_product, id_size, id_color) values (642, 227, 2, 1)
insert into stock (stock, id_product, id_size, id_color) values (940, 57, 1, 6);
insert into stock (stock, id_product, id_size, id_color) values (376, 186, 3, 5)
insert into stock (stock, id_product, id_size, id_color) values (541, 217, 2, 5)
insert into stock (stock, id_product, id_size, id_color) values (282, 22, 2, 1);
insert into stock (stock, id_product, id_size, id_color) values (354, 46, 2, 5);
insert into stock (stock, id_product, id_size, id_color) values (227, 91, 1, 3);
```

```sql
-- Details

insert into details (id, quantity, id_product, id_size, id_color) values (1, 3,
insert into details (id, quantity, id_product, id_size, id_color) values (2, 1,
insert into details (id, quantity, id_product, id_size, id_color) values (3, 2,
insert into details (id, quantity, id_product, id_size, id_color) values (4, 2,
insert into details (id, quantity, id_product, id_size, id_color) values (5, 6,
insert into details (id, quantity, id_product, id_size, id_color) values (6, 7,
insert into details (id, quantity, id_product, id_size, id_color) values (7, 3,
insert into details (id, quantity, id_product, id_size, id_color) values (8, 4,
insert into details (id, quantity, id_product, id_size, id_color) values (9, 3,
insert into details (id, quantity, id_product, id_size, id_color) values (10, 7,
```

```sql
--ADDRESS

insert into address values (1, 'Eadith Caulket', null, '466296634', 'Crest Line'
insert into address values (2, 'Alessandra Gellert', null, '305969005', 'Ramsey'
insert into address values (3, 'Lorilyn Presswell', 'Pixoboo', '224798511', 'Loe
insert into address values (4, 'Hazel Jirik', null, '824290248', 'Memorial', 159
insert into address values (5, 'Othilie Woolnough', null, '320396561', 'Division
insert into address values (6, 'Jamal Treker', 'Dablist', '608206992', 'Farwell'
insert into address values (7, 'Isabella Britee', null, '011562730', 'Green Ridg
insert into address values (8, 'Cloe Boog', null, '618724413', 'Melrose', 97, nu
insert into address values (9, 'Delmore Robert', null, '876594609', 'Schurz', 20
insert into address values (10, 'Leone Tyndall', null, '605712949', 'Village Gre
```

```sql
--User Order

insert into user_order values (1, 'Cancelled', '2023-05-16 13:11:17', 1, 1, 1);
insert into user_order values (2, 'Pending', '2022-12-24 17:05:43', 2, 2, 2);
insert into user_order values (3, 'Shopping Cart', '2023-06-01 02:31:04', 3, 3,
insert into user_order values (4, 'Shopping Cart', '2023-06-14 09:11:23', 4, 4,
insert into user_order values (5, 'Shopping Cart', '2023-05-04 05:24:23', 5, 5,
insert into user_order values (6, 'Cancelled', '2023-08-18 14:36:00', 6, 6, 6);
insert into user_order values (7, 'Completed', '2023-07-01 04:38:08', 7, 7, 7);
insert into user_order values (8, 'Completed', '2022-11-18 13:12:42', 8, 8, 8);
insert into user_order values (9, 'Completed', '2023-09-17 04:00:43', 9, 9, 9);
insert into user_order values (10, 'Shopping Cart', '2023-04-11 20:20:25', 10, 1
```

```sql
--Order details

insert into order_details (id_order, id_details) values (1, 1);
insert into order_details (id_order, id_details) values (2, 2);
insert into order_details (id_order, id_details) values (3, 3);
insert into order_details (id_order, id_details) values (4, 4);
insert into order_details (id_order, id_details) values (5, 5);
insert into order_details (id_order, id_details) values (6, 6);
insert into order_details (id_order, id_details) values (7, 7);
insert into order_details (id_order, id_details) values (8, 8);
insert into order_details (id_order, id_details) values (9, 9);
insert into order_details (id_order, id_details) values (10, 10);
```

```sql
-- Review

insert into review values (1, 3, 'ornare consequat lectus in est risus auctor se
insert into review values (2, 3, 'odio justo sollicitudin ut suscipit a feugiat
insert into review values (3, 3, 'montes nascetur ridiculus mus vivamus', 'torto
insert into review values (4, 5, 'non velit donec diam neque', 'vulputate vitae
insert into review values (5, 1, 'sociis natoque penatibus et magnis dis parturi
insert into review values (6, 4, 'eu massa donec dapibus duis', 'eros suspendiss
insert into review values (7, 3, 'sodales sed tincidunt eu felis', 'rhoncus sed
insert into review values (8, 1, 'in hac habitasse platea dictumst etiam faucibu
insert into review values (9, 4, 'a pede posuere nonummy integer non velit donec
insert into review values (10, 1, 'justo sollicitudin ut suscipit a feugiat et',
```

```sql
--Like

insert into user_like (id_user, id_review) values (90, 56);
insert into user_like (id_user, id_review) values (138, 9);
insert into user_like (id_user, id_review) values (165, 48);
insert into user_like (id_user, id_review) values (60, 6);
insert into user_like (id_user, id_review) values (161, 40);
insert into user_like (id_user, id_review) values (158, 19);
insert into user_like (id_user, id_review) values (29, 11);
insert into user_like (id_user, id_review) values (131, 10);
insert into user_like (id_user, id_review) values (170, 58);
insert into user_like (id_user, id_review) values (140, 6);
```

```sql
-- Report

insert into report values (1, 'hac habitasse platea dictumst aliquam augue quam
insert into report values (2, 'volutpat eleifend donec ut dolor morbi vel lectus
insert into report values (3, 'justo pellentesque viverra pede ac diam cras pell
insert into report values (4, 'suscipit a feugiat et eros vestibulum ac est laci
insert into report values (5, 'justo lacinia eget tincidunt eget tempus vel pede
insert into report values (6, 'orci luctus et ultrices posuere cubilia curae mau
insert into report values (7, 'lacinia nisi venenatis tristique fusce congue dia
insert into report values (8, 'non velit donec diam neque vestibulum eget vulput
insert into report values (9, 'ante ipsum primis in faucibus orci luctus et ultr
insert into report values (10, 'nascetur ridiculus mus vivamus vestibulum sagitt
```

Revision history

(não aplicável de momento)

GROUP2251, 26/10/2022

- Membro 1 - Alexandre Correia, up202007042@edu.fe.up.pt
- Membro 2 - Ana Sofia Costa, up202007602@edu.fe.up.pt
- Membro 3 - Daniel Rodrigues, up202006562@edu.fe.up.pt (Editor)
- Membro 4 - Diogo Fonte, up202004175@edu.fc.up.pt