RAMADAN OMAR

Proiect BD - ASE

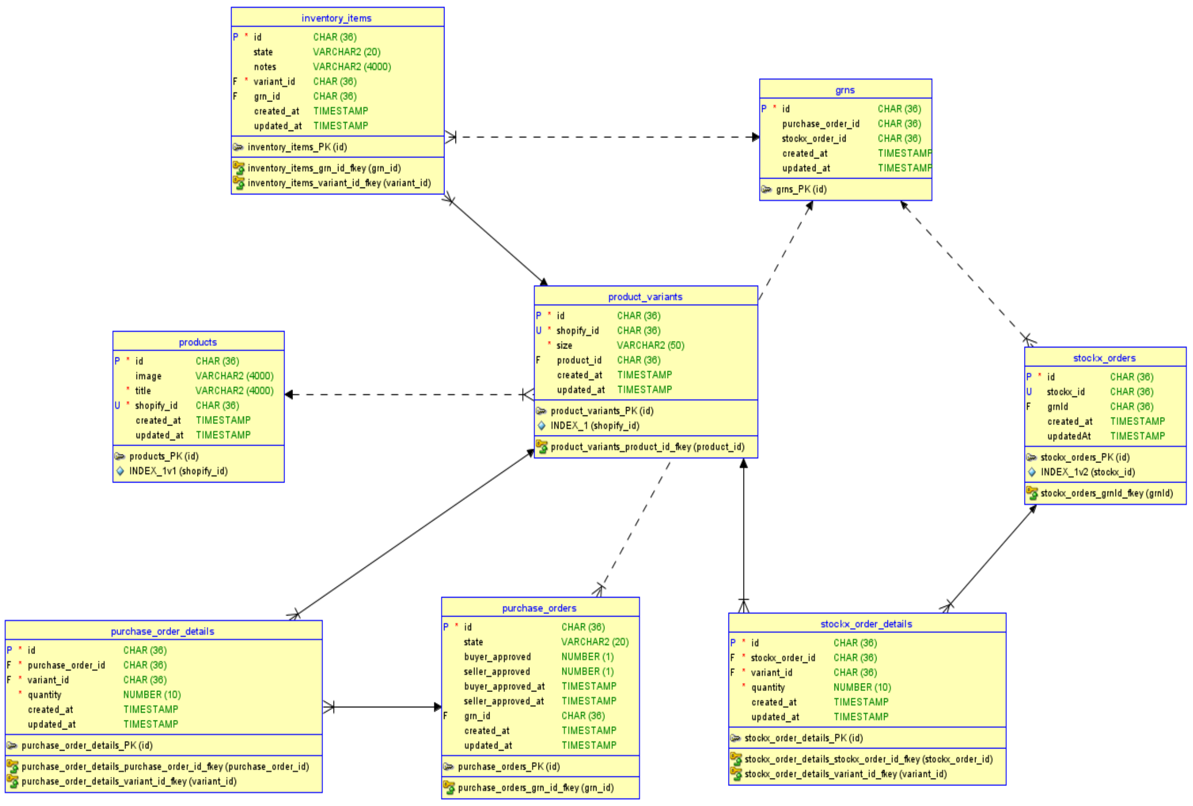
Prezentare Problema Economica:

O problema pe care o intampina comerciantii cu magazine pe Shopify (cea mai populara platforma de e-commerce din lume) este pastrarea gestiunii stocului pentru magazinele care practica “Consigmentul” si/sau ofera posibilitatea de Preorders.

Pentru a adresa aceasta problema, vom crea un inventory management system care va genera NIR-uri pentru toate comenzile primite de la furnziori. Sistemul permite generarea NIR-urilor prin mai multe metode, acesta avand mai mult rolul de “decorator”, encapsuland toate metodele pe care le doreste magazinul pentru primirea inventarului.

In acest proiect vom prezenta doar fluxul de intrare / primire a inventarului. Vom ignora partile de autentificare, detalii comenzi, loguri pentru audit si analtica.

Schema Bazei de Date: (Poza este atasata si in repo ca sa fie mai clara)



DDL:

```sql
-- Create Tables
CREATE TABLE inventory_items (
    id CHAR(36) PRIMARY KEY,
    state VARCHAR2(20) CHECK (
        state IN (
            'AVAILABLE',
            'COMMITTED',
            'UNAVAILABLE',
            'INCOMING',
            'NEEDED'
        )
    ),
    notes VARCHAR2(4000),
    variant_id CHAR(36) NOT NULL,
    grn_id CHAR(36),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP
);


-- NIR uri in engleza
CREATE TABLE grns (
    id CHAR(36) PRIMARY KEY,
    purchase_order_id CHAR(36),
    stockx_order_id CHAR(36),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP
);

CREATE TABLE stockx_orders (
    id CHAR(36) PRIMARY KEY,
    stockx_id CHAR(36) UNIQUE,
    grnId CHAR(36),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updatedAt TIMESTAMP
);

CREATE TABLE stockx_order_details (
    id CHAR(36) PRIMARY KEY,
    stockx_order_id CHAR(36) NOT NULL,
    variant_id CHAR(36) NOT NULL,
    quantity NUMBER(10) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP
);

CREATE TABLE purchase_orders (
    id CHAR(36) PRIMARY KEY,
    state VARCHAR2(20) CHECK (
        state IN (
            'AWAITING_APPROVAL',
            'APPROVED',
            'SHIPPED',
            'AWAITING_PAYMENT',
            'FINISHED',
            'CANCELLED'
        )
    ),
    buyer_approved NUMBER(1) DEFAULT 0,
    seller_approved NUMBER(1) DEFAULT 0,
    buyer_approved_at TIMESTAMP,
    seller_approved_at TIMESTAMP,
    grn_id CHAR(36),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP
);

CREATE TABLE purchase_order_details (
    id CHAR(36) PRIMARY KEY,
    purchase_order_id CHAR(36) NOT NULL,
    variant_id CHAR(36) NOT NULL,
    quantity NUMBER(10) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP
);

CREATE TABLE products (
    id CHAR(36) PRIMARY KEY,
    image VARCHAR2(4000),
    title VARCHAR2(4000) NOT NULL,
    shopify_id CHAR(36) NOT NULL UNIQUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP
);

CREATE TABLE product_variants (
    id CHAR(36) PRIMARY KEY,
    shopify_id CHAR(36) NOT NULL UNIQUE,
    "size" VARCHAR2(50) NOT NULL,
    product_id CHAR(36),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP
);

-- Foreign Key Constraints
ALTER TABLE inventory_items
ADD CONSTRAINT inventory_items_variant_id_fkey FOREIGN KEY (variant_id)
REFERENCES product_variants(id);
ALTER TABLE inventory_items
ADD CONSTRAINT inventory_items_grn_id_fkey FOREIGN KEY (grn_id) REFERENCES
grns(id);

ALTER TABLE stockx_orders
ADD CONSTRAINT stockx_orders_grnId_fkey FOREIGN KEY (grnId) REFERENCES
grns(id);

ALTER TABLE stockx_order_details
ADD CONSTRAINT stockx_order_details_stockx_order_id_fkey FOREIGN KEY
(stockx_order_id) REFERENCES stockx_orders(id);
ALTER TABLE stockx_order_details
ADD CONSTRAINT stockx_order_details_variant_id_fkey FOREIGN KEY
(variant_id) REFERENCES product_variants(id);

ALTER TABLE purchase_orders
ADD CONSTRAINT purchase_orders_grn_id_fkey FOREIGN KEY (grn_id) REFERENCES
grns(id);

ALTER TABLE purchase_order_details
ADD CONSTRAINT purchase_order_details_purchase_order_id_fkey FOREIGN KEY
(purchase_order_id) REFERENCES purchase_orders(id);
```

```sql
ALTER TABLE purchase_order_details
ADD CONSTRAINT purchase_order_details_variant_id_fkey FOREIGN KEY
(variant_id) REFERENCES product_variants(id);

ALTER TABLE product_variants
ADD CONSTRAINT product_variants_product_id_fkey FOREIGN KEY (product_id)
REFERENCES products(id);
```

Tabele:

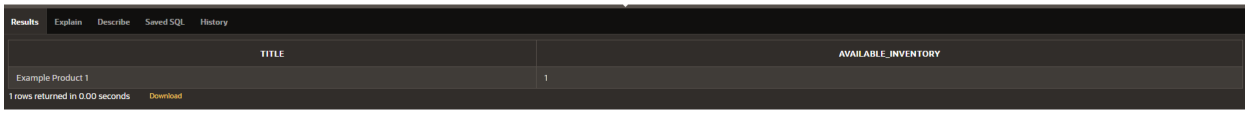
Pentru toate obiectele, vom folosi un uuid v4. Din pacate, Oracle SQL nu are un UUID type asa ca vom pasa generarea in logica API a aplicatiei. Folosim CHAR(36) ca uuid in practica este doar un sir de 32 de caractere hexadecimale separate cu - in formatul: 8-4-4-4-12

Exemple de interogari:

Listam toate produsele cu stocul lor disponibil:

```sql
SELECT p.title,
    COUNT(ii.id) AS available_inventory
FROM products p
    JOIN product_variants pv ON p.id = pv.product_id
    JOIN inventory_items ii ON pv.id = ii.variant_id
WHERE ii.state = 'AVAILABLE'
GROUP BY p.title;
```

Rezultat:



TITLE	AVAILABLE_INVENTORY
Example Product 1	1

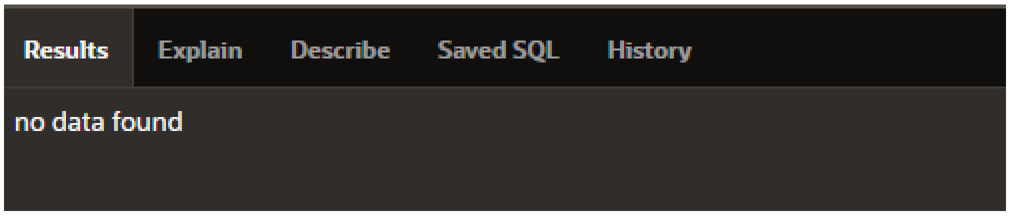
1 rows returned in 0.00 seconds

Toate Ordinele de cumparare care asteapta approval cu cantitatea lor totala:

```sql
SELECT po.id,
    (
        SELECT SUM(quantity)
        FROM purchase_order_details
        WHERE purchase_order_id = po.id
    ) AS total_quantity
```

```
FROM purchase_orders po
WHERE po.state = 'AWAITING_APPROVAL';
```

Rezultat:



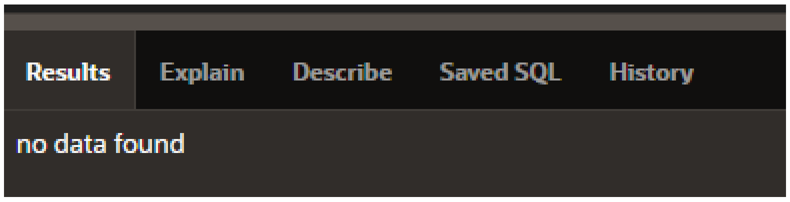
no data found

Nu avem astfel de intrare. Sintaxa este corecta deci va merge in productie 😸

Purchase Orders care au fost trimise de furnizori dar nu au ajuns inca la noi in ultima saptamana:

```
SELECT id,
    updated_at
FROM purchase_orders
WHERE state = 'SHIPPED'
    AND updated_at >= TRUNC(SYSDATE, 'IW');
```

Rezultat:



no data found

Nu avem astfel de intrare. Sintaxa este corecta deci va merge in productie 😸

Niste Update Statements care ar fi utile in frontend:

Aprobare Ordin de Cumparare (Purchase Order)

```
UPDATE purchase_orders
SET buyer_approved = 1,
    buyer_approved_at = CURRENT_TIMESTAMP
WHERE id = 'specific_purchase_order_id'
    AND state = 'AWAITING_APPROVAL';
```

Daca nu ar fi trebuit sa ne gandim la preorders, am fi putut face un update de gen sa updatam toate produsele cand s-a schimbat un purchase order in Shipped pentru iteme. Tinand cont ca nu vrem sa generam inventory items pentru purchase orders care nu au fost confirmate, vom pastra logica de generare a purchase order urilor la nivelul API ului. Am fi putut crea logica direct in baza de date daca aveam cum sa generam uuid uri unice.

Ar fi aratat ceva de gen (folosind un subquery sa luam toate produsele din purchase order)

```sql
UPDATE inventory_items
SET state = 'INCOMING'
WHERE variant_id IN (
        SELECT pod.variant_id
        FROM purchase_order_details pod
            JOIN purchase_orders po ON pod.purchase_order_id = po.id
        WHERE po.state = 'SHIPPED'
    )
    AND state NOT IN ('COMMITTED', 'UNAVAILABLE');
```

Tot codul este disponibil pe github:

https://github.com/ramadanomar/seminar-db