Projet Bases de Données

Bernigaud Noé, Fissore Davide et Antoine Venturelli

M1 Informatique, année 2021-2022

—

Objectif du projet : réaliser une base de données pour la gestion d'un magasin

Source image : https://www.lebigdata.fr/base-de-donnees

Panoramique

Langage utilisé SQL3 par le logiciel *sqldeveloppper* de *Oracle*.

Description du sujet

Le projet a pour but la réalisation d'une base de données qui vise à maîtriser :

- Les factures et les tickets
- Les fournisseurs et les clients
- Les stocks
- Les employés

d'un magasin. Avec des méthodes utilitaires, on peut récupérer d'autres informations sans lancer de complexes requêtes SQL comme par exemple lister toutes les factures non payées d'un client tout en utilisant les avantages de la programmation par référence proposée par la version 3 de SQL.

Structures de la base de données et dictionnaire de données MERISE

Article_t

Article_t est le type utilisé pour la représentation des articles du magasin.

Caractérisé par :

Nom Attribut	Type	Description	Contraintes
codebarre	VARCHAR(13)	la clé primaire	not null
quantité	NUMBER	la quantité présente dans le stock, elle est mise automatiquement à jour en cas de vente ou achat grâce à un trigger	not null >= 0
nom	VARCHAR(50)	La description du produit	not null
prix_achat	NUMBER	Le prix d'achat de l'article	not null
prix_vente	NUMBER	Le prix de vente de l'article	not null >= prix_achat

ligne_ticket_avec_this	listrefligneticket_t	La liste des références des lignes des tickets qui référencent le ticket courant	not null

Les méthodes de ce type :

- **MAP MEMBER FUNCTION COMPARTICLE RETURNS VARCHAR2**
  Elle permet de trier les articles par la concaténation du nom, du prix de vente et du code barre.
- **MEMBER FUNCTION GET_QUANTITE_VENDUE RETURNS NUMBER**
  Elle renvoie le nombre d'articles vendus à partir de l'instance d'un article : c'est la somme des quantités des "ligneticket" qui sont référencés par un ticket de vente.
- **MEMBER FUNCTION GET_QUANTITE_ACHETEE RETURNS NUMBER**
  C'est la méthode spéculaire à la précédente, elle renvoie la quantité de cet article achetée.
- **Les trois méthodes de mise à jour des liens de l'attribut ligne_ticket_avec_this**
  (add_ligne_ticket, delete_ligne_ticket, update_ligne_ticket)

LigneTicket_t

Ce type représente une ligne d'un ticket, il crée un pont entre le type *article_t* et *ticket_t* elle permet aussi de connaître la quantité de l'article vendu ou acheté.

Nom Attribut	Type	Description	Contraintes
numeroligne	NUMBER	la clé primaire	not null
quantité	NUMER	c'est la quantité qui de l'article A qui va être vendu ou acheté	not null > 0
article	REF article_t	la réf. vers l'article de la ligne courante	not null
parentticket	REF ticket_	la réf vers le ticket dans lequel se trouve la *ligneticket* courante	not null

Les méthodes de ce type :

- **MAP MEMBER FUNCTION COMPARLIGNETICKET RETURNS VARCHAR2**
  Méthode d'ordre sur numeroligne.
- **MEMBER FUNCTION GET_COUT RETURNS NUMBER**
  Renvoie le coût de la ligne courante, c'est le produit entre la quantité et le prix de vente si le *parentticket* est un ticket de vente ou le prix d'achat sinon.

- **MEMBER FUNCTION GET_TICKET RETURNS TICKET_T**
  Renvoie l'instance du ticket qui contient la ligne courante.

Ticket_t

Il représente les tickets émis par le magasin. Comme notre magasin peut avoir des intéractions avec des fournisseurs ou des clients qui veulent des factures et non des "simples tickets" alors on a créé aussi les types **factureEmise_t** et **factureRecue_t** qui représentent respectivement les factures de vente et les factures d'achat et qui étendent Ticket_t.

Ticket_t est caractérisé par :

Nom Attribut	Type	Description	Contraintes
id	NUMBER	la clé primaire	not null
estvente	NUMER	c'est un "boolean" qui indique si le ticket est un ticket de vente ou d'achat	not null in (0, 1)
ligneticket	listrefligneticket_t	la liste des réf. des *ligneticket* qui composent les tickets	not null
paiement	VARCHAR2(30)	le moyen de paiement	not null in ('espece', 'cb', 'cheque', 'autre')
employeemmetteur	REF empl_t	c'est la réf vers l'employé qui a émis le ticket en cas de vente, null sinon	
carte_reduction	REF carte_t	la référence vers une carte si elle est utilisée, null sinon	
dateemission	DATE	la date d'émission	not null

Les méthodes de la classe ticket_t :

- **MAP MEMBER FUNCTION COMPTICKET RETURNS VARCHAR2**
  Méthode d'ordre sur dateemission || id
- **MEMBER FUNCTION PRINT_TICKET RETURNS VARCHAR2**
  Renvoie une chaîne de caractères représentant le ticket
- **MEMBER FUNCTION GETTOTAL RETURNS NUMBER**
  Calcule le coût total du ticket courant (si ticket de vente on considère le prix_vente de l'article, sinon le prix_achat)

- **Les trois méthodes de mise à jour des liens de l'attribut ligneticket**

FactureRecue_t (resp. FactureEmise_t) ajoute au ticket les infos suivantes :

- Fournisseur de type REF FOURNISSEUR_T (resp. Client de type REF CLIENT _T)
- DateLimite qui représente la date limite du paiement de la facture reçue (resp. on a la date limite où le client doit payer la facture que nous avons émise)
- Payeounon de type NUMBER, si égal à 1 alors la facture a été payée

Empl_t

Le type empl_t représente les employés.

Il est caractérisé par :

Nom Attribut	Type	Description	Contraintes
numsecu	NUMBER	clé primaire	not null
nom	VARCHAR2(30)	le nom de l'employé	not null
prenom	VARCHAR2(30)	les prenom de l'emp	not null
job	VARCHAR2(30)	la mansion de l'emp	not null in ('Caissier', 'Polyvalent', 'Responsable', 'Directeur')
adresse	REF adresse_t	l'adresse de l'emp	not null
naissance	DATE	la date de naissance	not null
embauche	DATE	la date d'embauche	not null
salaire	NUMBER	le salaire perçu	not null BETWEEN(1500,15000)
cv	CLOB	le curriculum vitae	
ticket_emis	listrefticket_t	la liste des réf. des tickets émis par l'emp	not null

Les méthodes de ce type :

- **ORDER MEMBER FUNCTION COMPEMPLOYE RETURNS NUMBER**
  Méthode d'ordre sur les types de job d'un employé, suivant l'ordre (du plus important) : Directeur - Responsable - Caissier - Polyvalent

- **STATIC FUNCTION GET_EMPL_QUI_A_APPORTE_LES_PLUS_DARGENT RETURNS EMPL_T**
  C'est une méthode statique qui renvoie l'employé qui à fait gagner le plus à l'entreprise en regardant les factures émises
- **MEMBER FUNCTION GET_LA_PLUS_CHERE_FACTURE_EMISE RETURNS TICKET_T**
  Renvoie la facture la plus chère émise par l'employé courant
- **Les trois méthodes de mise à jour des liens de l'attribut ticket_emis**

Fournisseur_t

Il représente les fournisseurs de notre magasin.

Il est caractérisé par :

Nom Attribut	Type	Description	Contraintes
siret	NUMBER	la clé primaire (le siret est l'identificateur des entreprises en France)	not null
nom	VARCHAR2(30)	le nom du PDG de l'entreprise	
prenom	VARCHAR2(30)	le prénom du PDG de l'entreprise	
adresse	REF adresse_t	l'adresse du PDG	not null
naissance	DATE	la date de naissance du PDF	
facture_du_fourn	listrefticket_t	la liste des factures qu'on a reçu de ce fournisseur	not null

Les informations du PDG ne sont pas obligatoires, on se contente du numéro de SIRET.

Les méthode de ce type :

- **MAP MEMBER FUNCTION COMPFOURNISSEUR RETURNS VARCHAR2**
  La méthode d'ordre sur nom || prenom || siret;
- **MEMBER FUNCTION GET_FACTURES_A_PAYER RETURN LISTREFTICKET_T**
  Renvoie la liste des réf. vers les tickets qui sont encore à payer à l'instance du fournisseur courant
- **MEMBER FUNCTION GET_CATALOGUE RETURN LISTREFARTICLE_T**
  Renvoie la liste d'article que nous avons acheté du fournisseur courant
- **Les trois méthodes de mise à jour des liens de l'attribut facture_du_fourn**

Client_t

Il représente les clients de notre entreprise.

Il est caractérisé par :

Nom Attribut	Type	Description	Contraintes
id	NUMBER	la clé primaire	not null
nom	VARCHAR2(30)	le nom du client	not null
prenom	VARCHAR2(30)	le prénom	
adresse	REF adresse_t	l'adresse	not null
naissance	DATE	la date de naissance	
facture_du_client	listrefticket_t	la liste des réf. des ticket que nous avons émis vers le client courant	not null

Les méthodes de ces types :

- **MAP MEMBER FUNCTION COMPCLIENT RETURNS VARCHAR2**
  Méthode d'ordre sur nom || prenom || siret
- **MEMBER FUNCTION GET_ARGENT_APPORTE_EN_ENTREPRISE RETURNS NUMBER**
  C'est la somme d'argent que le client a apporté en entreprise
- **MEMBER FUNCTION GET_FACTURES_A_ENCAISSER RETURN LISTREFTICKET_T**
  C'est la liste des réf. des ticket que le client courant doit encore nous payer
- **Les trois méthodes de mise à jour des liens de l'attribut facture_du_client**

Carte_t

Il représente les cartes de réduction de notre magasin, la remise est appliquée sur les tickets.

Il est caractérisé par :

Nom Attribut	Type	Description	Contraintes
nom	VARCHAR2(30)	le nom de la carte	not null, in ('bronze', 'silver', 'gold', 'platinum', 'diamond', 'VIP', 'VIP+')
remise	NUMBER	le pourcentage de remise	not null, between (0.01 AND 0.95)

clients	listrefclients_t	la liste des clients qui possède cette carte	not null

Les méthodes de ce type :

- **MAP MEMBER FUNCTION COMPCARTE RETURN NUMBER**
  Méthode d'ordre en fonction de la remise de la carte
- **STATIC FUNCTION GET_MOST_USED_CARD RETURN LISTREFCARTE_T**
  Renvoie la liste des réf. des cartes les plus utilisées
- **STATIC FUNCTION GET_NB_OF_CL_FROM_NOM (nom1 VARCHAR2) RETURN NUMBER**
  Renvoie le nombre des cartes qui ont nom = nom1 (:= les cartes émises)

Adresse_t

Adresse_t est un type utilitaire qui a pour seul but de représenter les adresses des employés, clients et fournisseurs.

Caractérisé par :

Nom Attribut	Type	Description	Contraintes
pays	VARCHAR2(30)	Le pays de l'adresse	not null
ville	VARCHAR2(60),	La ville	not null
codepostal	VARCHAR2(5)	Le code postal	not null
rue	VARCHAR2(100)	La rue	not null
numero	NUMBER	Le numéro de la rue	not null

Comme cette information n'est pas trop utilisée dans notre base, on a préféré de ne pas créer un attribut exprès pour la clé primaire, alors on a décidé de mettre en clé primaire la concaténation de toutes les informations de l'adresse.

Ce type possède une seule méthode de type MAP : celle qui permet de trier les adresses.

Les index

Un index a été créé sur chaque attribut contenant une référence ainsi que sur les tables de références afin d'accéder plus rapidement aux données. Voici une liste des attributs sur lesquels un index a été mis en place:

- l'attribut *article* de la table *ligneticket_o*

- l'attribut *parentticket* de la table *ligneticket_o*
- la table de références *tablelistrefclients*
- l'attribut *adresse* de la table *empl_o*
- la table de références *tablelistrefticketarticles*
- l'attribut *employeemetteur* de la table *ticket_o*
- l'attribut *carte_reduction* de la table *ticket_o*
- l'attribut *adresse* de la table *client_o*
- l'attribut *carte* de la table *client_o*
- l'attribut *adresse* de la table *fournisseur_o*
- la table de références *tablelistrefticketemis*
- la table de référence *tablerefticket_du_client*
- la table de références *tablelistref_facture_du_fourn*
- ta table de références *listref_facture_avec_this*

Description textuelles des requêtes de mise à jour

Requêtes impliquant 1 table

Mettre à jour la ligneticket numéro 2 à 5 (le trigger mets automatiquement à jour le nombre d'article dans le stock)

- *UPDATE ligneticket_o SET quantite = 5 WHERE numeroligne = 2;*

Les employés qui travaillent depuis le 15-10-2000 reçoivent une augmentation de salaire de 1000 €.

- *UPDATE empl_o SET salaire = salaire + 1000 WHERE embauche > TO_DATE('15-10-2000', 'DD-MM-YYYY');*

Requêtes impliquant 2 tables

Les employés qui ne travaillent pas à Nice reçoivent 10 € de bonus pour les transports

- UPDATE empl_o SET salaire = salaire + 10 WHERE deref(adresse).ville != 'Nice';

Les employés qui ont émis des factures de plus de 500 € reçoivent un bonus de 50 €

- UPDATE empl_o SET salaire = salaire + 50 WHERE numsecu IN (
  SELECT deref(employeemmetteur).numsecu
  FROM ticket_o t WHERE t.gettotal() > 500
  );

Requêtes impliquant plus de 2 tables

Les articles que nous avons achetés du fournisseur 1234 ont une augmentation du 2 % du prix de vente

- UPDATE article_o art SET art.prix_vente = art.prix_vente * 1.02
  WHERE codebarre IN (
  SELECT lre.column_value.codebarre FROM TABLE (
  SELECT f.get_catalogue() FROM fournisseur_o f WHERE siret = 1234) lre);

Les articles présents dans la facture 1 ont une réduction de 5 % du prix de vente.

1. UPDATE article_o art SET art.prix_vente = art.prix_vente * 0.95 WHERE codebarre IN (
   SELECT lre.column_value.article.codebarre FROM TABLE (
   SELECT ligneticket FROM ticket_o WHERE id = 1) lre);

Description textuelles des requêtes de suppression

Requêtes impliquant 1 table

Suppression d'un client qui n'a ni carte ni facture.

- DELETE FROM client_o
  WHERE id = 7;

Suppression d'une carte n'étant affectée à aucun client.

- DELETE FROM carte_o
  WHERE remise > 0.35;

Requêtes impliquant 2 tables

Suppression d'une carte et mise à jour du pointeur des clients vers celle-ci.

- DELETE FROM carte_o
  WHERE nom = 'gold';
  UPDATE client_o clt
  SET clt.carte = NULL
  WHERE carte IS DANGLING;

Requêtes impliquant plus de 2 tables

On supprime un ticket qui n'est pas une facture et est vieux de plus de 10 ans, on supprime ses ligneticket, on met à jour ligne_ticket_avec_this dans les articles concernés par ce ticket, on met à jour ticket_emis dans l'employé concerné par ce ticket.

- ```
  DECLARE
    ref_ticket      REF ticket_t;
    article         article_t;
    ref_ligneticket REF ligneticket_t;
    ref_l_tick      setligneticket_t;
    employe         empl_t;
    ticket_id       NUMBER := 17;
  BEGIN
    SELECT deref(t.employeemmetteur), ref(t)
    INTO employe, ref_ticket
    FROM ticket_o t
    WHERE t.id = ticket_id;
    employe.delete_ticket_emis(ref_ticket);

    DELETE FROM ticket_o
        WHERE
        id = 17;
    SELECT CAST(COLLECT(value(l)) AS setligneticket_t)
    INTO ref_l_tick
    FROM ligneticket_o l
    WHERE parentticket IS DANGLING;
    FOR i IN ref_l_tick.first..ref_l_tick.last LOOP
      SELECT deref(l.article), ref(l)
      INTO article, ref_ligneticket
      FROM ligneticket_o l
      WHERE value(l) = ref_l_tick(i);
      article.delete_ligne_ticket(ref_ligneticket);
      DELETE FROM ligneticket_o l
      WHERE value(l) = ref_l_tick(i);
    END LOOP;
  END;
  ```

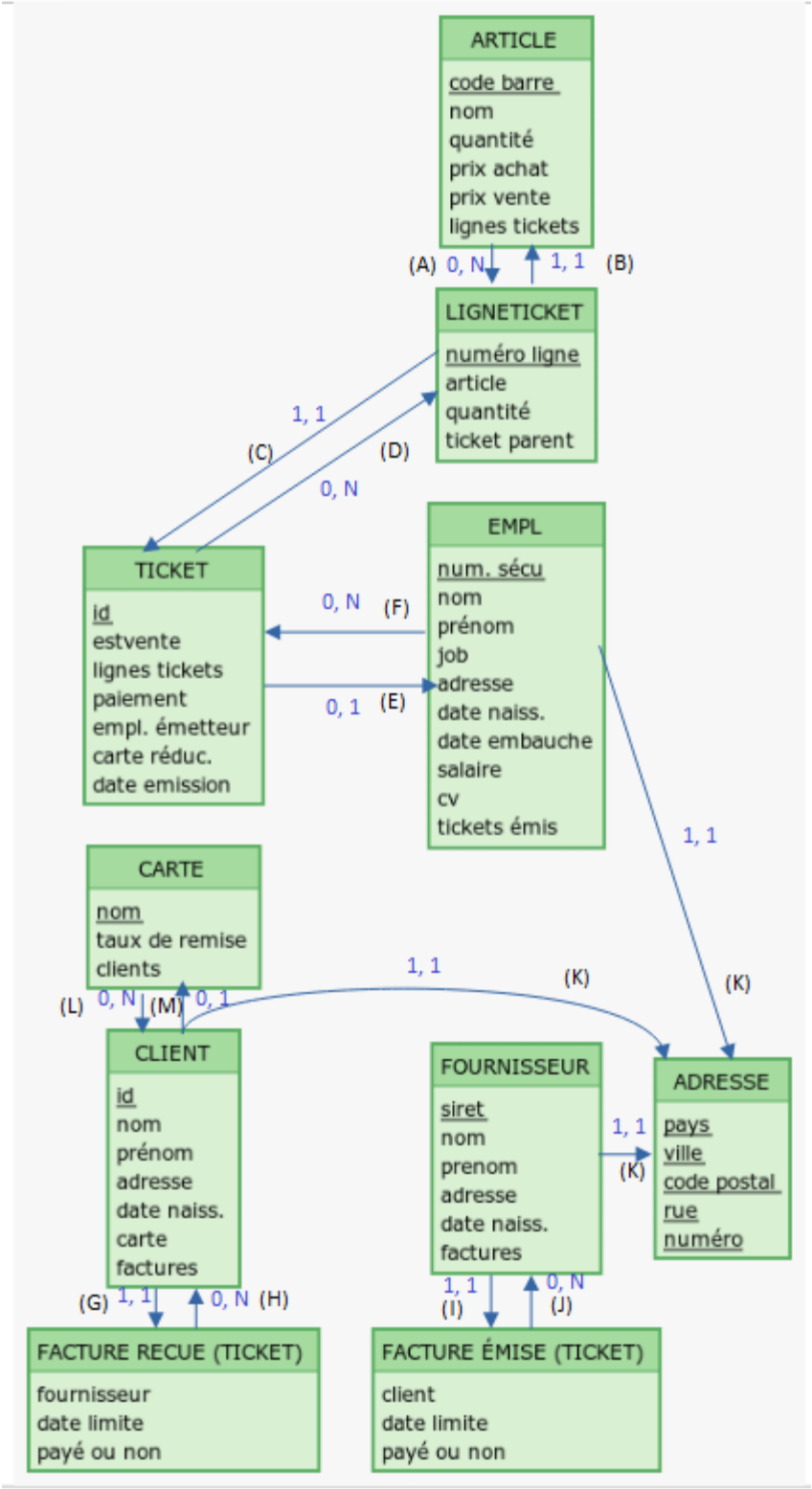
On supprime le client 1 qui a une carte et sur lequel on a emis une facture, on met à jour donc listrefclients_t dans la carte du client 1, et on supprime les factures emises sur ce
  ```

client, (Attention au trigger delete_facture_checker car on ne peut pas supprimer des factures de moins de 10 ans).

- DECLARE
  ref_client REF client_t;
  carte carte_t;
  ref_fact_e setfactureemise_t;
  client_id NUMBER := 1;
  BEGIN
  SELECT deref(v.carte), ref(v)
  INTO carte, ref_client
  FROM client_o v
  WHERE v.id = client_id;
  carte.deleteclient(ref_client);
  DELETE FROM client_o c
  WHERE id = 1;
  SELECT
  CAST(COLLECT(TREAT(value(t) AS factureemise_t)) AS setfactureemise_t)
  INTO ref_fact_e
  FROM ticket_o t
  WHERE TREAT(value(t) AS factureemise_t).client IS dangling;
  FOR i IN ref_fact_e.first..ref_fact_e.last LOOP
  DELETE FROM ticket_o t
  WHERE value(t) = ref_fact_e(i);
  END LOOP;
  end;

Description textuelles des requêtes de consultation

Requêtes impliquant 1 table

Les cartes dont la remise est >= à 0.2.

- SELECT oc.nom FROM carte_o oc WHERE oc.remise >= 0.2;

Les articles dont il en reste plus de 3 et qui coûtent 50 euros ou moins.

- SELECT nom FROM article_o WHERE quantite > 3 AND prix_vente <= 50;

Nom et prénom de chaque fournisseur.

- SELECT nom, prenom FROM fournisseur_o;

Requête avec regroupement sur les jobs en faisant la somme des salaires.

- SELECT job, SUM(salaire) FROM empl_o oe GROUP BY job;

Requête avec tri sur les adresses utilisant notre méthode map.

- SELECT oa.pays, oa.ville
  FROM adresse_o oa
  WHERE oa.numero = 17000
  OR oa.numero = 19000
  OR oa.numero = 12
  ORDER BY value(oa);

La date limite de toutes les factures reçues.

- SELECT TREAT(value(ot) AS factureрecue_t).datelimite AS datelimite
  FROM ticket_o ot
  WHERE value(ot) IS OF (factureрecue_t);

Requêtes impliquant 2 tables

Fournisseur dont les factures que nous avons reçues ont été toutes payées.

- SELECT o.siret AS siret
  FROM fournisseur_o o
  LEFT JOIN (
  SELECT deref(TREAT(deref(t.column_value) AS factureрecue_t).fournisseur).siret AS ss
  FROM TABLE (SELECT b.get_factures_a_payer()
  FROM fournisseur_o b
  WHERE b.siret = 1234) t)
  ON siret = ss
  WHERE ss IS NULL;

Client dont les factures que nous avons émises ont été toutes payées.

- SELECT o.id AS id
  FROM client_o o
  LEFT JOIN (
  SELECT deref(TREAT(deref(t.column_value) AS factureemise_t).client).id AS id2
  FROM TABLE (SELECT oc.get_factures_a_encaisser()
  FROM client_o oc
  WHERE oc.id = 2) t)
  ON id = id2
  WHERE id2 IS NULL;

Informations employé émetteur du ticket 11.

- SELECT ot.id, oe.numsecu, oe.nom, oe.embauche
  FROM ticket_o ot
  LEFT JOIN empl_o oe
  ON oe.numsecu = ot.employeemmetteur.numsecu
  WHERE ot.id = 11;

Les tickets émis (id et date d'emmission) par les employés triés par ordre ante-chronologique.

- SELECT oe.nom, oe.prenom, ot.id, ot.dateemission
  FROM empl_o oe
  LEFT JOIN ticket_o ot
  ON oe.numsecu = ot.employeemmetteur.numsecu
  WHERE ot.id IS NOT NULL
  ORDER BY ot.dateemission DESC;

Pour chaque employé, la quantité totale d'argent encaissé depuis son enregistrement (c-à-d, le total des totaux de chacun de ses tickets). On ne considère pas les employés n'ayant émis aucun ticket.

- SELECT oe.numsecu, oe.nom, SUM(ot2.total) AS total
  FROM empl_o oe
  INNER JOIN (SELECT o.employeemmetteur.numsecu n, o.gettotal() total
  FROM ticket_o o) ot2
  ON oe.numsecu = ot2.n
  GROUP BY oe.numsecu, oe.nom;

Requêtes impliquant plus de 2 tables

Le nom de l'article vendu avec l'id du ticket dans lequel il a été enregistré associé à l'employé ayant scanné l'article (l'employé qui a remis le ticket).

- SELECT oe.nom, oe.prenom, ticket_id, nom_article
  FROM empl_o oe
  INNER JOIN (SELECT ot.id AS ticket_id,
  ot.employeemmetteur.numsecu AS n,
  ol.article.nom AS nom_article
  FROM ticket_o ot
  INNER JOIN ligneticket_o ol
  ON ot.id = ol.parentticket.id)
  ON oe.numsecu = n;

Id des factures émises du client 1 sachant qu'il a la carte gold en utilisant 2 sous-jointures externes.

- ```
  SELECT tab1.id_client, tab2.nom AS nom_carte, tab1.id_facture
  FROM (SELECT id_facture, oc.id AS id_client
        FROM client_o oc
        LEFT JOIN (SELECT
        deref(TREAT(deref(tc.column_value) AS factureemise_t).client).idAS idss,
        TREAT(deref(tc.column_value) AS factureemise_t).id AS id_facture
            FROM TABLE ( SELECT c.get_factures_a_encaisser()
                FROM client_o c
                WHERE c.id = 1) tc)
        ON id = idss
        WHERE idss IS NOT NULL) tab1
  INNER JOIN ( SELECT id_client2, o.nom AS nom
            FROM carte_o o
            LEFT JOIN (SELECT
                TREAT(deref(t.column_value) AS client_t).carte.nom AS ssnom,
                TREAT(deref(t.column_value) AS client_t).id AS id_client2
                    FROM TABLE (SELECT car.clients
                                FROM carte_o car
                                WHERE car.nom = 'bronze') t)
            ON nom = ssnom
            WHERE ssnom IS NOT NULL) tab2
  ON tab1.id_client = tab2.id_client2;
  ```

Le nom de l'article associé au numéro de sécurité social de celui qui l'a vendu et ce trier par nom d'article.

- ```
  SELECT oe.numsecu, nom_article
  FROM empl_o oe
  INNER JOIN ( SELECT ot.employeemmetteur.numsecu AS n,
                ol.article.nom AS nom_article
            FROM ticket_o ot
            INNER JOIN ligneticket_o ol
            ON ot.id = ol.parentticket.id )
  ON oe.numsecu = n
  ORDER BY nom_article;
  ```

Pour chaque employé, on calcule la moyenne (en pourcentage) de la proportion du coût d'un article dans le total du ticket dans lequel l'article est enregistré (on ne considère pas les employés n'ayant émis aucun ticket)

- ```
  SELECT oe.numsecu, oe.nom, oe.prenom,
        AVG(round(prix / total * 100, 2)) AS mean_total_percent_proportion
  ```
  ```

```
FROM empl_o oe
INNER JOIN (  SELECT ot.employeemmetteur.numsecu   AS n,
                     ol.article.prix_vente         AS prix,
                     ot.gettotal()                 AS total,
                     ol.article.nom                AS nom_article
              FROM ticket_o ot
              INNER JOIN ligneticket_o ol
              ON ot.id = ol.parentticket.id)
ON oe.numsecu = n
GROUP BY oe.numsecu, oe.nom, oe.prenom
ORDER BY mean_total_percent_proportion DESC;
```

Pour chaque employé, on calcule le bénéfice brut total à partir des tickets. On utilise la fonction **nvl** qui substitue la valeur nulle par 0 (si les employés n'ont émis aucun ticket).

-
```
SELECT oe.numsecu, oe.nom, oe.prenom,
       nvl(SUM(prix - prix_fournisseur), 0) AS benefice_brut
FROM empl_o oe
LEFT JOIN (   SELECT ot.employeemmetteur.numsecu AS n,
                     ol.article.prix_vente     AS prix,
                     ol.article.prix_achat     AS prix_fournisseur,
                     ol.article.nom            AS nom_article
              FROM ticket_o ot
              INNER JOIN ligneticket_o ol
              ON ot.id = ol.parentticket.id)
ON oe.numsecu = n
GROUP BY oe.numsecu, oe.nom, oe.prenom
ORDER BY benefice_brut DESC;
```

La définition du Modèle Entité-Association MERISE



La description textuelles des associations

Ici on va expliquer les liens des associations existants entre les différents types :

- **FLECHE A :** 1 article peut apparaître dans 0..N ligneticket
- **FLECHE B :** 1 ligneticket contient 1 et 1 seul article
- **FLECHE C :** 1 ligneticket apparaît dans 1 et 1 seul ticket
- **FLECHE D :** 1 ticket contient 0..N ligneticket
- **FLECHE E :** 1 ticket est émis par 0 ou 1 employé (0 si ticket d'achat, 1 sinon)
- **FLECHE F :** 1 employé émet de 0 à N tickets
- **FLECHE G :** 1 facture émise contient 0 ou 1 client
- **FLECHE H :** 1 client apparaît dans 0..N factures émises
- **FLECHE I :** 1 facture reçues contient 0 ou 1 fournisseur
- **FLECHE J :** 1 fournisseur apparaît dans 0..N factures reçues
- **LES FLECHES K :** 1 client, fournisseur, client ont 1 et 1 seule adresse
- **FLECHE L :** 1 type de carte est possédée par 0..N client
- **FLECHE M :** 1 client possède 0 ou 1 carte de réduction

Quelque précision par rapport à nos choix de conception :

Les articles ne connaissent pas directement les tickets où ils sont cités, mais, on peut trouver ces tickets à travers les "lignes tickets", car elles connaissent les tickets auxquels elles appartiennent.

Un autre choix a été celui de ne pas créer des VARRAY, car on ne voulait pas limiter une table à un nombre fixé de ligne, c'est pour ça qu'on a opté pour des nested table à leur place.

Enfin, on n'a pas réussi à trouver des relations N-M cohérentes car les choix conceptuels qu'on a faits ne permettent pas de réaliser une telle liaison sans casser la logique de notre base de données.

Mapping Oracle JDBC / Java

Le dossier JavaJDBC contient un projet Java qui permet la lecture des types et donc des instances de notre base de données. Le main est situé dans la classe Main et, une fois lancé, il affiche sur la sortie standard tous les objets présents dans notre base en bouclant sur tous les types de existant.

Note : Pour que le main compile, il faut aussi inclure dans le jar "ojdbc11.jar" situé dans le répertoire JDBCjar