

ENSIMAG 2ème année

Projet de bases de données

**Application de gestion de tirages
de photos numériques**

**SIBILLEAU Michaël
SPORTOUCH David**

Sommaire

Application de gestion de tirages de photos numériques

I. Modélisation du problème

11. Identification des propriétés.....	2
12. Définition des entités	2
13. Dépendances fonctionnelles	3
14. Schéma entité association et CIF.....	3
141. Entités et associations	3
142. Choix des CIF	5

II. Passage au schéma relationnel et implantation.

21. Algorithme de passage	6
22. Schéma relationnel	9
221. Résultat de l'algorithme	9
222. Normalisation	10
223. Contraintes d'intégrité référentielles	10
23. Traduction des structures en SQL	10
24. Implantation en Java	11

III. Bilan

31. Bilan d'activité	11
32. Enseignements tirés	13

IV. Annexes

41. Schéma entités – associations	
42. Implantation SQL	
43. Implantation Java	

Note liminaire : ce document traite de l'analyse, la conception et la validation du projet de gestion de base de données. Il présente en détail les différents travaux réalisés en respectant un ordre chronologique depuis la modélisation du problème jusqu'à l'implantation en SQL et Java.

I. Modélisation du problème.

11. Identification des propriétés.

Dans cette étape, nous contentons d'établir un listing consistant à fournir un identificateur à chaque propriété :

- un client est identifié par son adresse internet (**mail**), son nom (**nom**), son prénom (**prenom**), son adresse postale (**adresse**), un identifiant (**login**) et un mot de passe (**passwd**)
- un fichier image se trouve dans le répertoire **chemin**, et comporte un certain nombre d'informations sur la prise de vue : l'appareil photo (**appareil**), l'objectif utilisé (**objectif**), la distance focale (**distancefoc**), la sensibilité ISO (**sensibilite**), l'ouverture (**ouverture**), la vitesse d'obturation (**vitessobtu**) et la résolution de l'image (**resolution**).
- un album est une collection de photos identifié de manière unique par un code (**codalbum**). Il comporte un titre (**titralbum**) et éventuellement un sous-titre (**stitalbum**). Un cas particulier d'album est un livre dans lequel le client peut ajouter une préface (**preface**) ou une postface (**postface**).
- les photos d'un album doivent être identifiées par un numéro d'ordre unique (**numordre**) et sont associées à un titre (**titrephoto**) et un commentaire (**commentaire**).
- une commande établie par un client à une certaine **date** doit être identifiée de manière unique par un numéro (**numcommande**). Cette commande est établie pour une quantité (**quantite**) dont le coût global s'élève à **prixtotal**. Préalablement, le client doit choisir le format de développement identifié par un code (**codeformat**) dont le coût unitaire est **prix**.

12. Définition des entités.

Les entités suivantes sont immédiates :

- **client** : entité identifiée par mail et un mot de passe (passwd)
- **filepicture** : fichier image identifié par un chemin.
- **commande** : entité identifiée par numcommande.
- **album** : entité caractérisée par un numéro codalbum.
- **livre** : sous-type de l'entité album.
- **photos** : entité caractérisée par un numéro numordre.

Par ailleurs, il est nécessaire de créer deux entités supplémentaires :

- comme un album contient un titre et, de façon facultative un sous titre, nous introduisons une entité **sous_titre** ayant stitalbum comme propriété.
- Dans un même ordre d'idée, et de façon à assurer un prix unitaire à chaque format, nous créons l'entité **format** ayant codeformat (pour l'unicité) et prix comme propriétés.

N.B. : dans toute la suite, nous appellerons « propriétaire », le client qui dépose ses fichiers photos.

13. Dépendances fonctionnelles.

Notations : les clés sont soulignées et les dépendances sont matérialisées par une flèche. Ainsi, l'écriture : A \longrightarrow B signifie « A détermine B ».

Certaines dépendances fonctionnelles sont immédiates :

client : mail \longrightarrow nom, prenom, adresse, login, passwd

filepicture : chemin \longrightarrow appareil, objectif, distancefoc, sensibilite, ouverture, vitesseobtu, resolution

commande : numcommande \longrightarrow date, prixtotal

format : codeformat \longrightarrow prix

album : codalbum \longrightarrow titralbum, preface, postface

photos : numordre \longrightarrow titrephoto, commentaire

On peut également identifier des dépendances fonctionnelles inter-entités que nous les matérialiserons (pour la plupart) par des cardinalités (1,1) lors de la construction du schéma entités-associations.

numcommande \longrightarrow mail

chemin \longrightarrow mail

codalbum \longrightarrow chemin, mail

14. Schéma entités-associations et contraintes d'intégrité fonctionnelle.

Le schéma entité-association est présenté en annexe. Nous y avons indiqué les cardinalités et les contraintes d'intégrité fonctionnelle (CIF). L'objet de ce paragraphe est de justifier les choix qui ont été retenus pour l'élaboration de ce schéma.

14.1. Entités et associations.

Il est nécessaire d'établir un premier lien entre client et filepicture au travers d'une association que nous baptisons « dépose » (**fig.1**) Il s'agit ici du téléchargement dont il est question dans l'énoncé.

La sémantique du verbe télécharger étant trop vague, nous retenons verbe déposer qui, en français, est plus explicite.

On note que les éléments apportés en fin d'énoncé visant à faciliter travail de la tireuse numérique (objectif, distance focale, etc.) sont introduits comme propriétés de filepicture, ce qui est logique : filepicture est l'emplacement « physique » de l'image.

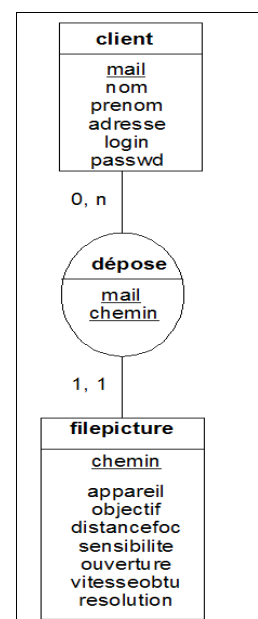


Fig. 1

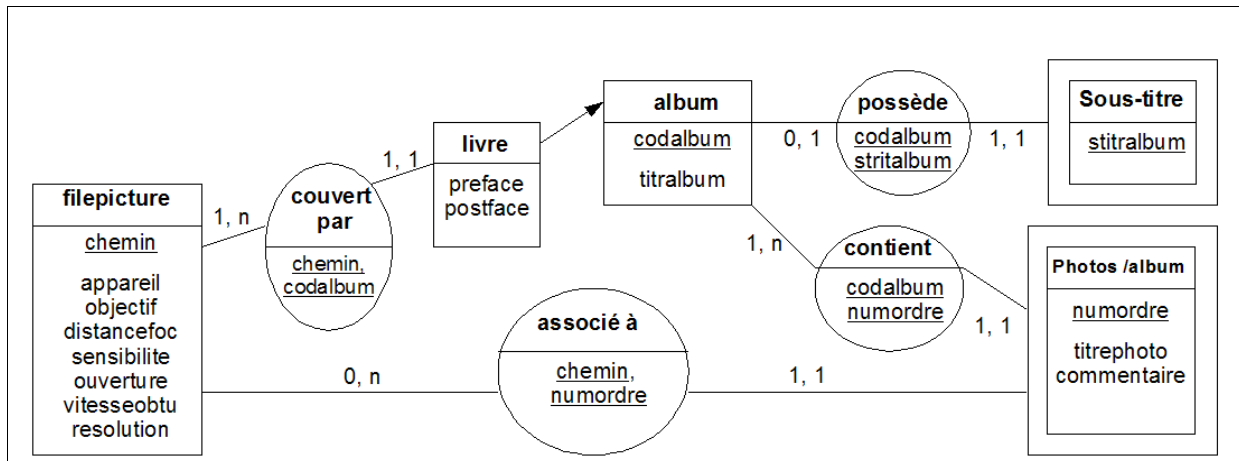


Fig. 2

La **figure 2** appelle plusieurs remarques :

➤ S'il est possible de représenter photo au moyen d'une association, nous avons préféré créer une entité photos avec un lien injectif (cardinalité 1,1 de droite) sur filepicture. Ce choix nous a paru plus simple puisqu'il permet de décliner des propriétés essentielles : numordre qui définit l'unicité de la photo dans un album donné, titrephoto et commentaire.

La cardinalité 0,n de gauche entre filepicture et photos permet qu'une même photo puisse se retrouver dans plusieurs albums différents, sans préjudice d'unicité au sein d'un même album.

➤ Un livre est un cas particulier d'album, il s'agit donc clairement d'un sous type d'entité. Sur ce livre, le client doit pouvoir ajouter une photo en couverture, d'où l'association « couvert par ». La cardinalité 1,n de gauche permet qu'une même photo couvre plusieurs livres distincts. Quant à celle de droite, elle assure une seule et unique couverture par livre (la fin de phrase « doit figurer sur la couverture » lève toute ambiguïté).

➤ Tel que nous avons conçu le schéma, il est nécessaire de rendre unique chaque photo au sein d'un même album (resp. livre), ce qui peut être fait au travers d'une association d'appartenance (« contient »). Photos devient alors une entité faible par rapport à album.

➤ Un album contient éventuellement un sous titre. Pour satisfaire cette éventualité, nous avons créé l'entité « sous-titre » également faible par rapport à « album ». Comme il s'agit d'une possibilité et non d'un impératif, il suffit de définir une cardinalité 0,1 à gauche de l'association « possède ». La cardinalité assure l'unicité d'un sous-titre au sein d'un même album.

Figure 3 : l'association entre client et commande se décline naturellement. Il est cependant nécessaire d'ajouter une association directe entre client et album : un client peut décider que ses fichiers soient visibles par d'autres clients.

La cardinalité 0,n de gauche traduit bien qu'un client puisse ne pas avoir d'album. Le cas échéant, il peut accepter de les rendre visibles par d'autres. Quant à la cardinalité 1,n de droite : un album est visible par 1 (le propriétaire) ou plusieurs clients selon les choix du propriétaire.

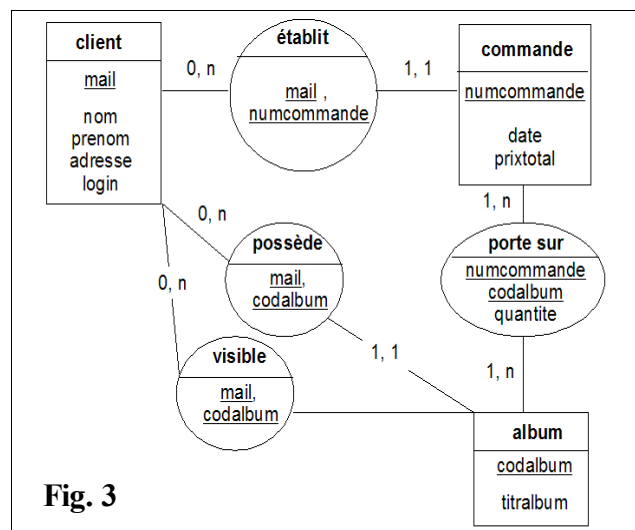


Fig. 3

L'association « possède » intègre la dépendance fonctionnelle « codalbum détermine mail » au travers de la cardinalité (1,1) de droite. Inversement, un client possède 0 ou plusieurs albums. Il reste à résoudre un problème : la nécessité, pour commander, d'avoir visibilité sur les albums (resp. livres). Comme nous ne pouvons pas apporter de solution par cardinalité avec le schéma retenu, nous devons introduire une CIF (voir § 142).

Figure 4 :Plusieurs versions de schémas entités-associations propres à cette figure ont été étudiées afin d'établir des liens entre les entités « commande », « format » et « album ». Après avoir conçu une association à trois branches, nous avons finalement opté pour le schéma ci-dessus en raison de sa plus grande clarté.

La création d'une entité « format » était indispensable pour permettre à la fois l'unicité de prix et la possibilité que ces prix unitaires varient sans influencer les commandes effectuées. Il est facile de vérifier que chacune des cardinalité doit être 1,n.

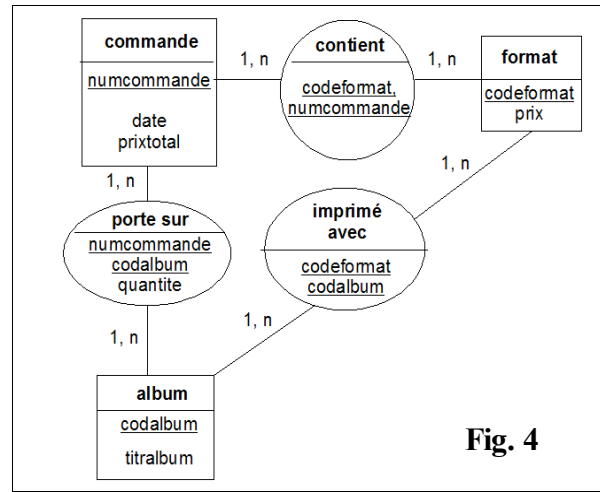


Fig. 4

Avec un tel schéma, on retrouve le dispositif habituel de commande sur internet formé d'un panier dans lequel le client dépose le choix de ses articles et de leur quantité, chacun des articles valant un prix unitaire. Puis, lorsque le client s'apprête à valider sa commande, il décaisser par CB le prix total du panier.

142. Choix des CIF.

La CIF indiquée sur la **figure 5** est justifiée par le dernier paragraphe de l'énoncé.

Il est indiqué qu'un client peut décider de partager tout ou partie de ses albums (resp. livres) impliquant une forme de visibilité (traduite par une association « visible » et des cardinalités adaptées⁽¹⁾) et que dans ce cas, il est possible à la personne ayant visibilité d'effectuer une commande.

La commande doit donc être subordonnée à la visibilité de l'album (resp. du livre). Naturellement, le client qui dépose ses fichiers a visibilité dessus. Il est par ailleurs le seul à attribuer les visibilités aux autres clients.

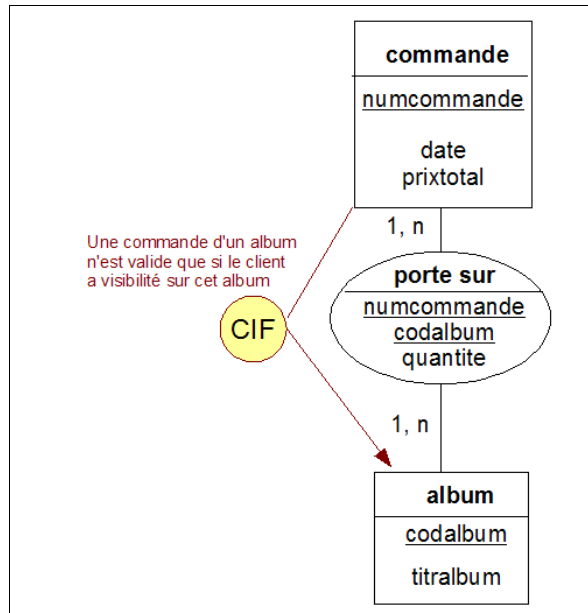
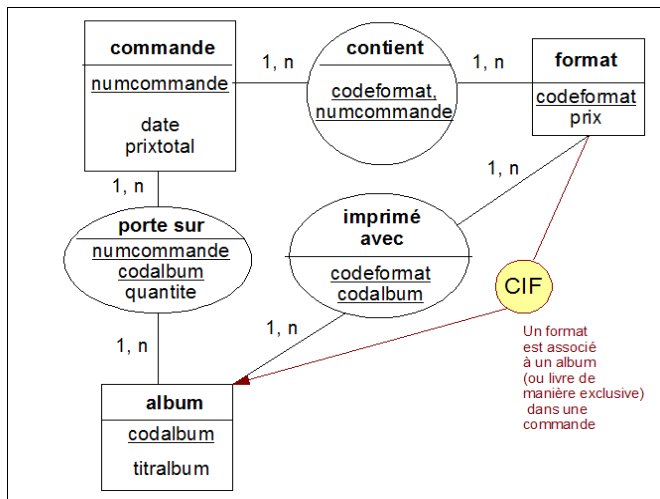


Fig. 5

(1) la cardinalité 1,n située à droite de l'association « visible » indique bien le fait qu'un album est visible par au moins un client (le propriétaire) et éventuellement par d'autres clients sur décision du propriétaire.



La CIF de la *figure 6* traduit le fait qu'un format est associé à un album (resp. livre) d'une commande donnée.

Elle complète convenablement la partie du schéma présenté dans la figure 4.

On notera que le codeformat s'applique à un livre ou à un album de manière exclusive et qu'il y a différents formats pour les albums et pour les livres.

Fig. 6

II. Passage au schéma relationnel et SQL.

21. Algorithme de passage.

Dans cette étape, on applique l'algorithme de passage du modèle entités-associations vers le schéma relationnel sans se préoccuper de la normalisation.

211. Première étape de l'algorithme.

On crée une relation par entité en lui donnant pour attributs les propriétés et pour clé l'identifiant. Nous obtenons facilement :

client (mail, nom, prenom, adresse, login, passwd)

filepicture (chemin, appareil, objectif, distancefoc, sensibilite, ouverture, vitesseobtu, resolution)

commande (numcommande, date, prixtotal)

format (codeformat, prix)

album (codalbum, titralbum)

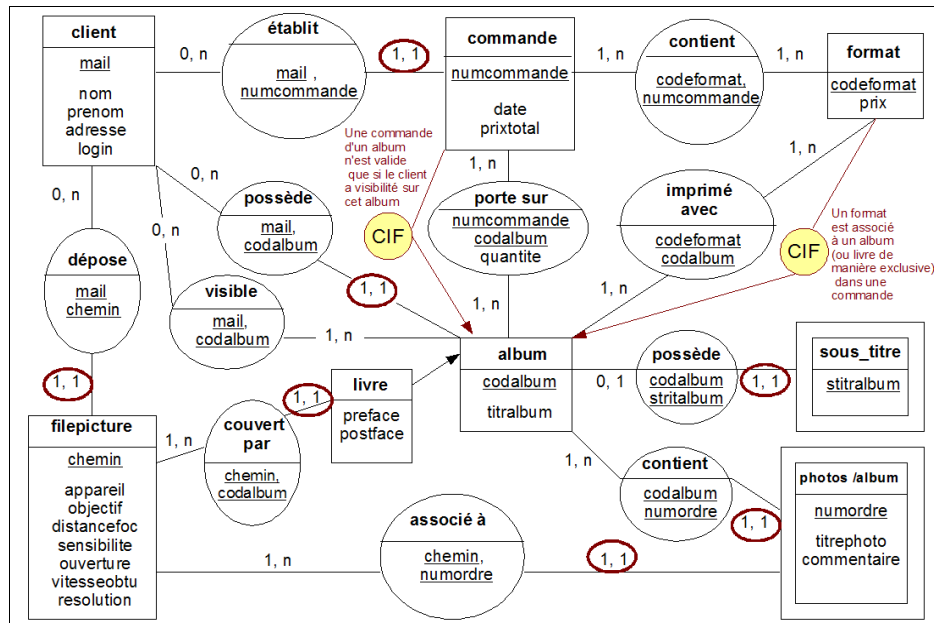
livre (codalbum, titralbum, preface, postface)

sous_titre (codalbum, stitalbum)

photos (codalbum, numordre, titrephoto, commentaire)

212. 2ème étape : Examen des associations de cardinalité (1,1).

Il s'agit d'appliquer la partie 2 de l'algorithme vu en cours.



On intègre dans « commande » la clé de client et les propriétés de « établi ». Cet union se restreint à la propriété « mail ». On obtient donc :

commande (numcommande, mail, date, prixtotal)

Il ne faut pas perdre de vu que mail est une clé étrangère. On a donc une contrainte d'intégrité relationnelle que nous notons :

$$\Pi_{mail}(commande) \subseteq \Pi_{mail}(client)$$

Un raisonnement semblable conduit aux résultats suivants :

album (codalbum, mail, chemin, titralbum)
 livre (codalbum, mail, chemin, titralbum, preface, postface)

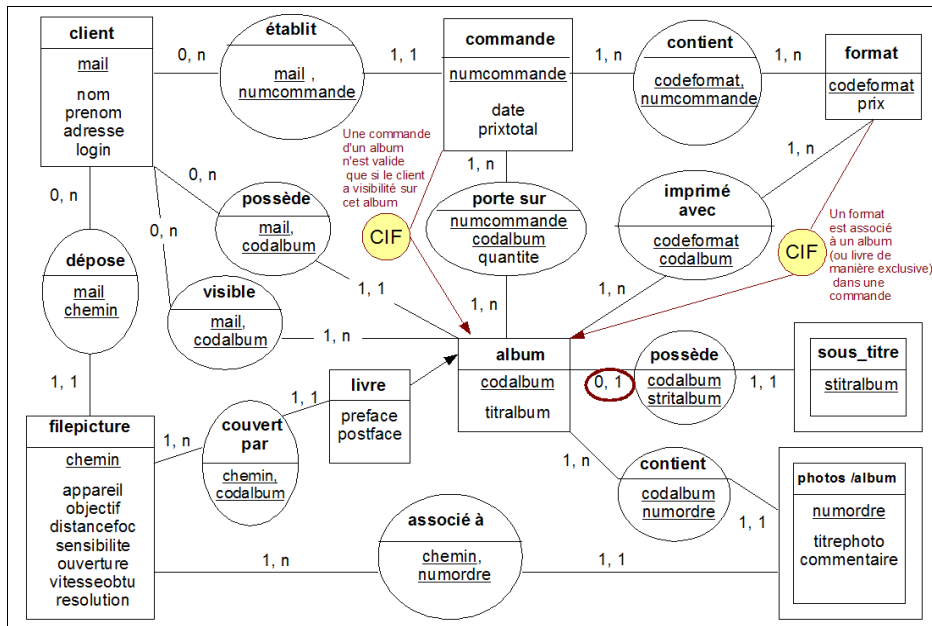
avec : $\Pi_{mail}(album) \subseteq \Pi_{mail}(client)$ et $\Pi_{chemin}(album) \subseteq \Pi_{chemin}(filepicture)$
 $\Pi_{mail}(livre) \subseteq \Pi_{mail}(client)$ et $\Pi_{chemin}(livre) \subseteq \Pi_{chemin}(filepicture)$

filepicture (chemin, mail, appareil, objectif, distancefoc, sensibilitè, ouverture, vitesseobtu, resolution) avec $\Pi_{mail}(filepicture) \subseteq \Pi_{mail}(client)$

photos (numordre, codalbum, chemin, titrephoto, commentaire)
 avec : $\Pi_{chemin}(photos) \subseteq \Pi_{chemin}(filepicture)$

sous_titre (codalbum, stitalbum)

212. Examen des associations de cardinalité (0,1)

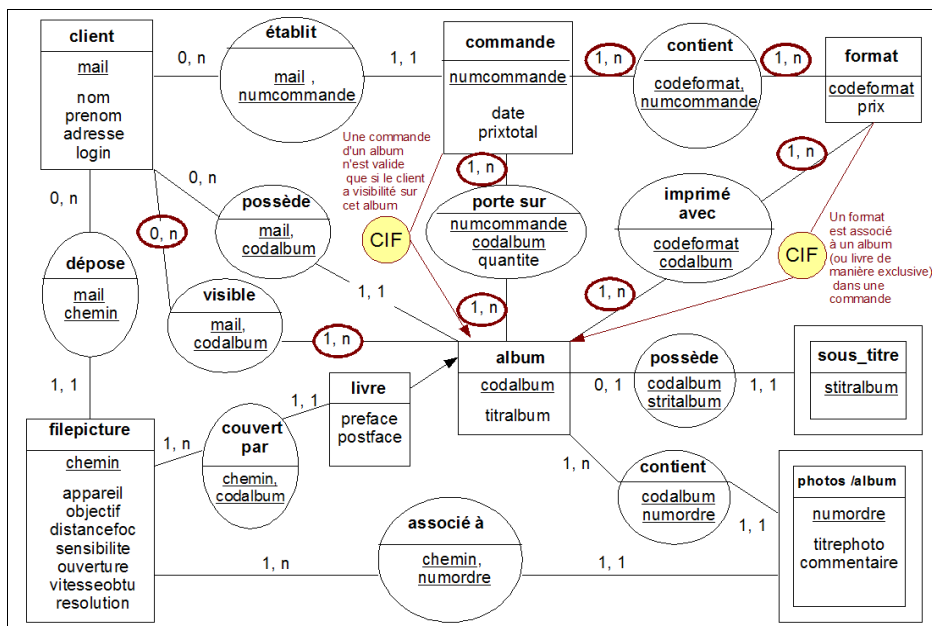


Ce traitement n'est pas nécessaire dans la mesure où la cardinalité (1,1) située à droite de l'association « possède » a déjà été traitée au § 212. On peut s'en apercevoir en appliquant l'algorithme du cours : on crée une nouvelle relation « possède » ayant comme attributs les propriétés de l'association possède et les clés d'album et de sous_titre. La clé de cette nouvelle relation est la clé d'album :

possède (codalbum, stitralbum)

Or cette relation contient les mêmes informations que sous_titre, ce qui termine la démonstration.

213. Examen des associations de cardinalité (x,n).



Les cardinalités (x,n) correspondant aux associations non traitées sont répertoriées dans le graphe ci-dessus.

Pour chacune des associations concernées, on crée une nouvelle relation ayant pour attributs les propriétés de cette association et les clés des entités situées de part et d'autre de l'association.

On obtient les relations suivantes :

visible (mail, codalbum)

avec : $\Pi_{mail}(visible) \subseteq \Pi_{mail}(client)$, $\Pi_{codalbum}(visible) \subseteq \Pi_{codalbum}(album)$

porteSur (numcommande, codalbum, quantité)

$\Pi_{numcommande}(porteSur) \subseteq \Pi_{numcommande}(commande)$ $\Pi_{codalbum}(porteSur) \subseteq \Pi_{codalbum}(album)$

contient (codeformat, numcommande)

$\Pi_{codeformat}(contient) \subseteq \Pi_{codeformat}(format)$ $\Pi_{numcommande}(contient) \subseteq \Pi_{numcommande}(commande)$

imprimeAvec (codeformat, codalbum, numcommande)

$\Pi_{codeformat}(imprimeAvec) \subseteq \Pi_{codeformat}(format)$ $\Pi_{codalbum}(imprimeAvec) \subseteq \Pi_{codalbum}(album)$
 $\Pi_{numcommande}(imprimeAvec) \subseteq \Pi_{numcommande}(commande)$

22. Schémas relationnel.

221. Résultat de l'algorithme.

L'algorithme que nous avons appliqué dans les paragraphes ci-dessus nous permet de lister les relations suivantes :

client (mail, nom, prenom, adresse, login, passwd)

filepicture (chemin, mail, appareil, objectif, distancefoc, sensibilite, ouverture, vitesseobt, resolution) où mail est une clé étrangère.

commande (numcommande, mail, date, prixtotal) où mail est une clé étrangère

format (codeformat, prix)

album (codalbum, chemin, mail, titralbum) où chemin et mail sont des clés étrangères

livre (codalbum, chemin, mail, titralbum, preface, postface) où chemin et mail sont des clés étrangères

photos (codalbum, numordre, chemin, titrephoto, commentaire) où chemin est une clé étrangère

visible (mail, codalbum)

porteSur (numcommande, codalbum, quantité)

contient (codeformat, numcommande)

imprimeAvec (codeformat, codalbum, numcommande)

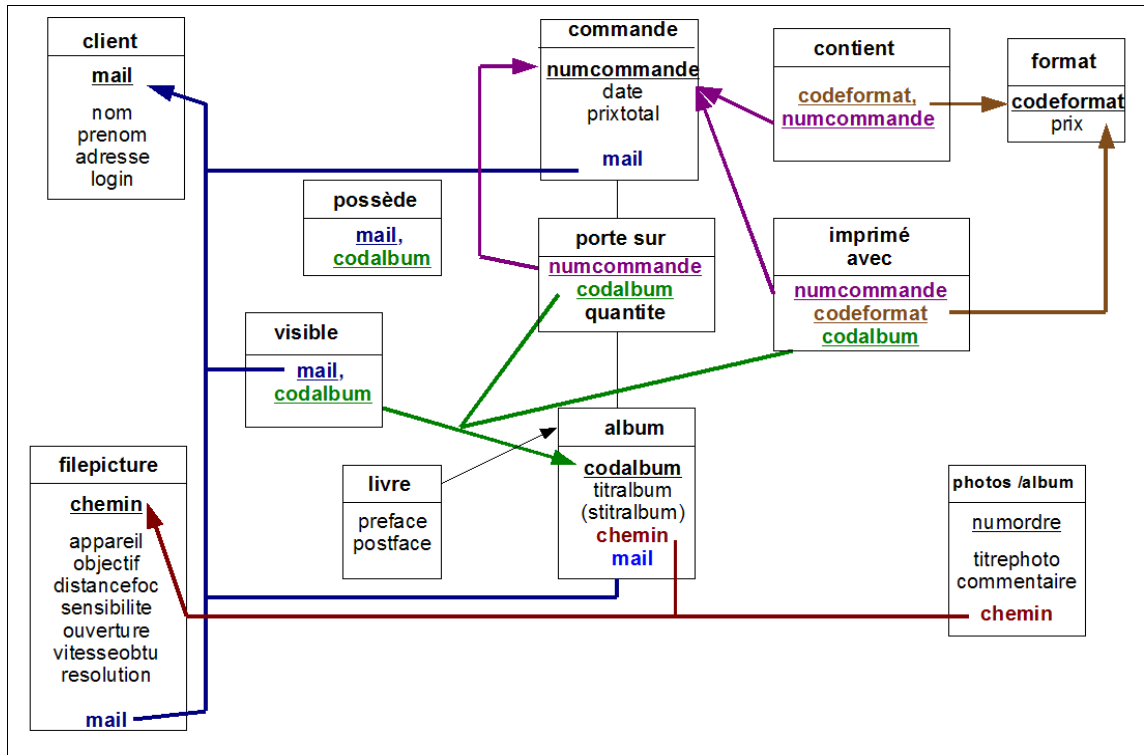
N.B. : L'absence d'une relation sous_titre sera expliquée au § 231.

222. Normalisation.

Il est facile de vérifier que les dépendances fonctionnelles associées aux relations listées en 221 sont en troisième forme normale. Nous pouvons même affirmer qu'elles sont FNBC (Boyce Codd).

223. Contraintes d'intégrité référentielles

Nous schématisons ci-dessous et de façon non formelle les contraintes d'intégrité référentielles :



23. Traduction des structures en SQL.

Le script SQL est présenté en annexe. L'implantation appelle quelques remarques relatives à certaines tables :

- table **CLIENT** : l'implantation de cette table ne pose pas de problème particulier. Il faut cependant remarquer que nous avons introduit une contrainte statique sous forme de prédicat « check » imposant que les adresses e-mail comportent bien le caractère @. L'interface devra cependant rappeler à l'utilisateur la syntaxe à respecter.

Dernière remarque : tous les attributs dispose d'une contrainte not null indispensable.

- Table **COMMANDE** : cette table comporte une information portant sur une date. Nous l'avons donc simplement baptisée par ce terme pour éviter toute ambiguïté. Cependant, nous nous sommes aperçu que ce mot est un mot clé du langage SQL et qu'il correspond exactement au type que nous voulons créer. Nous avons donc été contraint, pour l'implémentation, de rebaptiser l'attribut clé en cle_sql.

- tables **ALBUM** : nous avons remarqué que LIVRE était un sous type d'entité d'ALBUM. Il y a donc plusieurs façon de le traduire en SQL. La solution que nous avons retenu consiste à créer une seule table ALBUM ayant deux attributs caractéristiques de LIVRE. Une contrainte statique check permet d'effectuer une distinction entre LIVRE et ALBUM.

De la même façon, le traitement de l'entité sous_type, faible par rapport à album, est inclus dans l'implantation de la table ALBUM (présence de stitralbum sans contrainte not null).

– table **PHOTOS** : l'entité photos est faible par rapport à album. La solution que nous utilisons ici est de créer une table spécifique incluant la clé d'album (ce qui nous paraît être la solution la plus simple).

24. Implantation Java.

Dans cette phase de conception, beaucoup d'importance a été accordée à la programmation défensive et à l'aspect pratique d'utilisation de l'interface.

En effet, lors de l'implémentation de cette interface nous avons mis l'accent sur une gestion optimale des erreurs pouvant être commises par l'utilisateur, notamment lors des requêtes. Pour ce faire, nous avons utilisé un système d'étiquettes et de boucle donnant l'opportunité à l'utilisateur de revenir en arrière (en cas d'erreur) sans avoir à relancer le programme concerné depuis le début.

Ce système de boucles et d'étiquettes est accompagné de nombreux tests effectués sur les tables. Ainsi l'utilisateur ne doit pas être confronté à la levée d'une exception SQL susceptible de le faire sortir du gestionnaire à la moindre petite erreur.

Les codes source des fonctionnalités suivantes sont présentés en annexe :

La première fonctionnalité implantée permet la création d'un nouveau client (Creation_Client.java) en entrant ses coordonnées. Chaque client est authentifié au début du programme par son login et son mot de passe. Cette fonctionnalité remplit la table CLIENT.

Pour la seconde (téléchargement de photos et création de livres), nous avons volontairement scindé la fonctionnalité en deux programmes : un pour le téléchargement et un pour la création de livre. Ces deux programmes remplissent les tables FILEPICTURE , PHOTOS ET ALBUM.

La troisième fonctionnalité affiche d'abord la liste des albums consultables par l'utilisateur et permet à ce dernier de consulter un album en affichant la liste des photos qui y sont incluses.

La dernière fonctionnalité permet la saisie d'une commande accompagné de la facturation et assure le remplissage des tables PORTE_SUR, IMPRIME_AVEC,CONTIENT et COMMANDE.

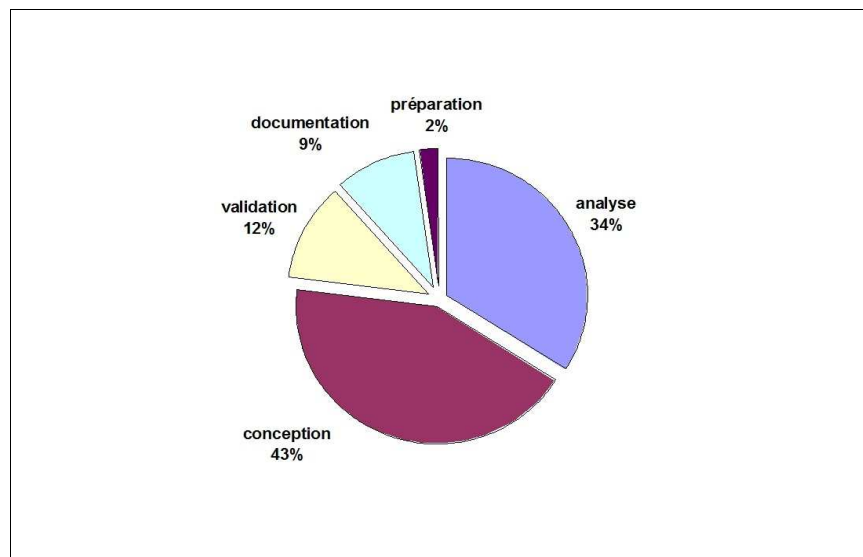
Ce type de programmation avec de nombreux tests sur les paramètres en entrée nous a permis de construire un interface facile d'utilisation pour l'utilisateur.

III. Bilan.

31. Bilan d'activité.

<i>Date</i>	<i>Activités</i>	<i>Volume horaire (en heure étudiant)</i>
30/03	Préparation et tests.	1h30
03/04	Analyse : identification des propriétés, définition des entités et des dépendances fonctionnelles.	3h00
	Documentation : début de rédaction de la partie I du rapport.	1h00
05/04	Analyse : Première ébauche du schéma entité-association.	3h00
06/04	Analyse : schéma entité-association.	6h00
	Documentation : réalisation du schéma. Fin de rédaction de la partie I du rapport.	0h30

<i>Date</i>	<i>Activités</i>	<i>Volume horaire (en heure étudiant)</i>
	Validation : schéma entité-association.	0h30
10/04	Analyse : passage au relationnel. Documentation : passage au relationnel	2h00 0h30
11/04	Analyse : passage au relationnel et normalisation Documentation : paragraphe 21 Conception : début du script	4h00 1h00 1h00
13/04	Documentation : corrections et rédaction des paragraphes 22 et 23 Conception : étude de l'interface et fin du script Validation : corrections (introduction d'une association « possède » entre client et album)	1h00 5h00 1h00
18/04	Conception : implémentation de l'interface Java	8h00
19/04 et 20/04	Conception : implémentation Java Validation : corrections apportées au code sql (changement de stratégie sur la table album).	3h00 3h00
25/04	Conception : implémentation Java Validation : débogage Java et tests	2h00 2h00
26/04	Documentation : préparation de la soutenance Conception : Java Validation : Tests	2h00 9h00 3h00
TOTAL		63h00



32. Enseignements tirés.

321. Analyse.

Les dépendances fonctionnelles obtenues en 221 ont été directement établies en troisième forme normale sans qu'il soit nécessaire d'opérer une décomposition. Ce résultat repose sur le travail d'analyse lors de la création du schéma entités-associations : une décomposition initiale avait été opérée avec minutie et toute redondance avait été exclue.

322. Développement-conception.

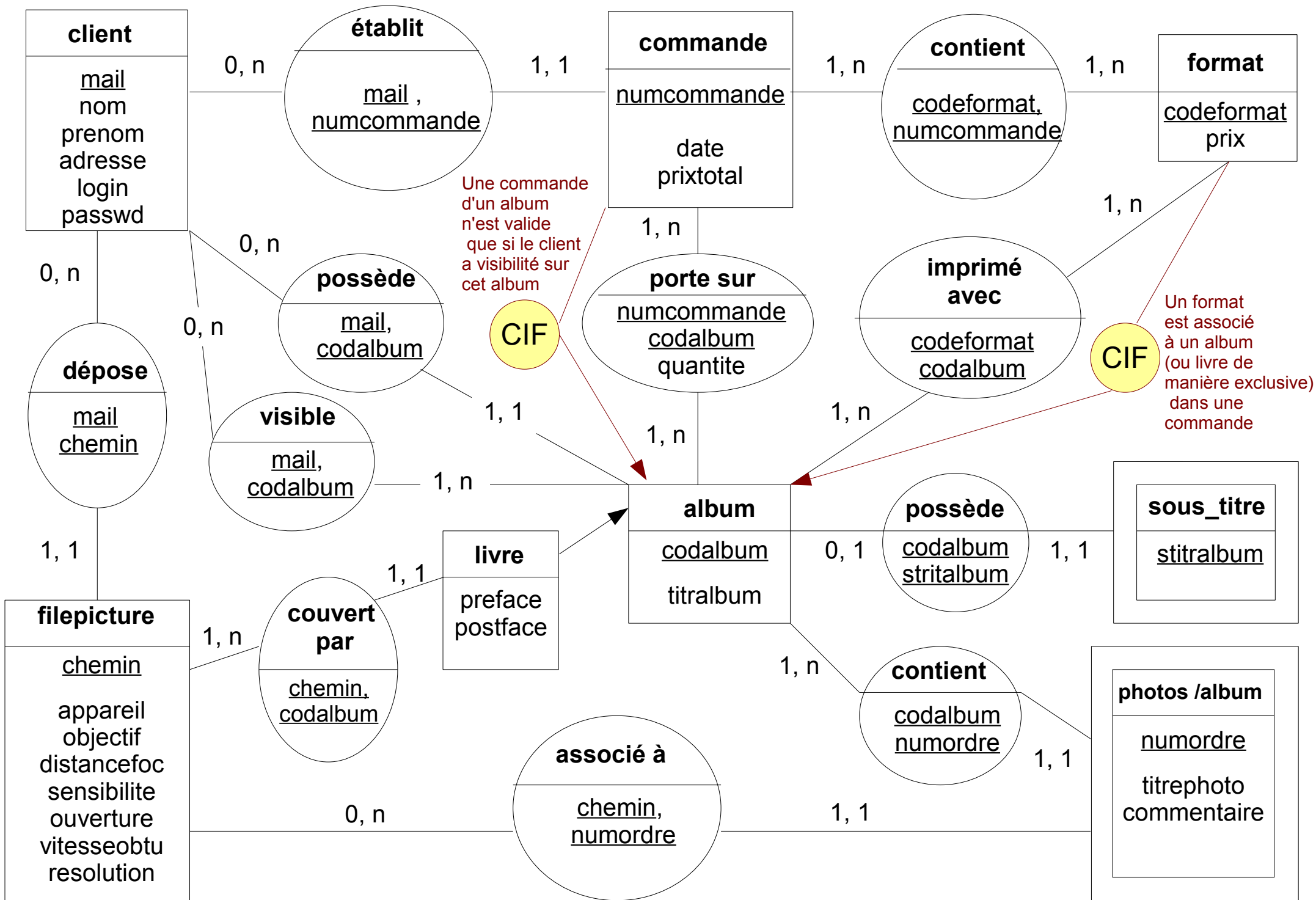
L'implantation de la table ALBUM a fait l'objet d'un changement de stratégie. Dans un premier temps, nous avons créé deux tables distinctes ayant des clés (primaires et référentielles) identiques et des attributs optionnels pour la préface et la postface (par défaut).

Mais cette façon de faire s'est rapidement avérée lourde de conséquences sur les autres tables en raison de nombreux effets de bord. Pour simplifier le programme et éviter toute erreur à l'exécution, nous avons modifié notre implantation en incluant la notion de LIVRE dans ALBUM lui-même. Cette nouvelle stratégie devait alors s'accompagner d'une contrainte statique comme expliquée *supra*.

323. validation.

La dernière phase de réalisation (codage Java), outre son objectif de création d'une interface, a permis d'apporter quelques corrections essentielles sur des contraintes non identifiées auparavant : en particulier l'ajout d'une clé "numcommande" dans la table IMPRIME_AVEC et d'un champ passwd dans la table CLIENT. Dans le premier cas, il s'agissait d'éviter qu'un objet de la table IMPRIME_AVEC ne soit pas parfaitement déterminé (un format est forcément relatif à une commande lorsqu'il est en rapport avec un album).

IV. Annexes.



42. Implantation SQL (script).

```
/*
*****
*/
implantation des structures en SQL
/*
*/
Michael Sibilleau
David Sportouch
*****

// creation de la table CLIENT
// -----

create table CLIENT (
    mail varchar(40) not null,
    nom varchar(40) not null,
    prenom varchar(40) not null,
    adresse varchar(100) not null,
    login varchar(20) not null,
    passwd varchar(20) not null,
        primary key (mail),
        check (mail like '%@%'));

// creation de la table filepicture
// -----

create table FILEPICTURE (
    chemin varchar(60) not null,
    mail varchar(40) not null,
    appareil varchar(40),
    objectif varchar(40),
    distancefoc real,
    sensibilite real,
    ouverture real,
    vitessobtu real,
    resolution real,
        primary key (chemin),
        foreign key (mail) references CLIENT);

// creation de la table commande
// -----

create table COMMANDE (
    numcommande integer not null,
    mail varchar(40) not null,
    date_sql date,
    prixtotal real not null,
        primary key (numcommande),
        foreign key (mail) references CLIENT);

// creation de la table format
// -----

create table FORMAT (
    codeformat integer not null,
    prix real not null,
        primary key (codeformat));

// creation de la table album
// -----

create table ALBUM (
    codalbum integer not null,
    chemin varchar(60) not null,
    mail varchar(40) not null,
    titralbum varchar(60) not null,
    stitralbum varchar(60),
    preface varchar (250),
    postface varchar (250),
        primary key (codalbum),
        foreign key (chemin) references FILEPICTURE,
        foreign key (mail) references CLIENT,
        check ((preface is null and postface is null) or
(preface is not null and postface is not null));
```

```

// creation de la table photos
// -----

create table PHOTOS (
    codalbum integer not null,
    numordre integer not null,
    chemin varchar(60) not null,
    titrephoto varchar(60) not null,
    commentaire varchar(250),
        primary key (codalbum,numordre),
        foreign key (chemin) references FILEPICTURE);

// creation de la table visible
// -----

create table VISIBLE (
    mail varchar(40) not null,
    codalbum integer not null,
        primary key (mail,codalbum),
        foreign key (codalbum) references ALBUM,
        foreign key (mail) references CLIENT);

// creation de la table porte_sur
// -----

create table PORTE_SUR (
    numcommande integer not null,
    codalbum integer not null,
    quantite integer not null,
        primary key (numcommande,codalbum),
        foreign key (codalbum) references ALBUM,
        foreign key (numcommande) references COMMANDE);

// creation de la table contient
// -----

create table CONTIENT (
    codeformat integer not null,
    numcommande integer not null,
        primary key (codeformat,numcommande),
        foreign key (numcommande) references COMMANDE,
        foreign key (codeformat) references FORMAT);

// creation de la table imprime_avec
// -----

create table IMPRIME_AVEC (
    codeformat integer not null,
    numcommande integer not null,
    codalbum integer not null,
        primary key (codeformat,codalbum),
        foreign key (numcommande) references COMMANDE,
        foreign key (codeformat) references FORMAT,
        foreign key (codalbum) references ALBUM);

```

43. Implantation Java.

Creation_Client.java

```
import java.sql.*;
import java.io.*;

public class Creation_Client{

    static final String CONN_URL = "jdbc:oracle:thin:@ensibm.imag.fr:1521:ensi2";
    static final String USER = "sportoud"; // A remplacer pour votre compte, sinon
    genere une exception
    static final String PASSWD = "toto";

    public Creation_Client(){
        int a=0;

        try {
            // Enregistrement du driver Oracle
            System.out.println("Loading Oracle thin driver...");
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            System.out.println("loaded.");

            // Etablissement de la connexion
            System.out.println("Connecting to the database...");
            Connection conn = DriverManager.getConnection(CONN_URL, USER, PASSWD);
            System.out.println("connected.");

            // Demarrage de la transaction (implicite dans notre cas)
            conn.setAutoCommit(false);
            conn.setTransactionIsolation(conn.TRANSACTION_SERIALIZABLE);

            // appel à l'utilisateur pour rentrer les données concernant le client
            EasyIn easy = new EasyIn();

            System.out.println("***login du client***");
            String login = easy.readString();

            // on vérifie que ce login la n'existe pas déjà
            PreparedStatement pstmt0 = conn.prepareStatement("select count(*) from CLIENT
where login='"+login+"' ");
            ResultSet rset0 = pstmt0.executeQuery();
            while(rset0.next()){
                a=rset0.getInt(1);
            }
            conn.commit();
            pstmt0.close();
            if (a==1){
                System.out.println(">>Ce login existe déjà");
                return ;
            }

            rset0.close();

            System.out.println("***password du client***");
            String passwd = easy.readString();
            System.out.println("***adresse mail du client***");
            String mail = easy.readString();
            System.out.println("***nom du client***");
            String nom = easy.readString();
            System.out.println("***prenom du client***");
            String prenom = easy.readString();
            System.out.println("***adresse du client***");
            String adresse = easy.readString();

            // modification de la base de donnée : rajout

            Statement stmt=conn.createStatement();
            int rest = stmt.executeUpdate("insert into
CLIENT(mail,nom,prenom,adresse,login,passwd)
values ('"+mail+"', '"+nom+"', '"+prenom+"', '"+adresse+"', '"+login+"', '"+passwd+"')");

            // terminaison de la modification
            conn.commit();
```

```

        //fermeture
        stmt.close();
        conn.close();
    }
    catch (SQLException e) {
        System.err.println("ERREUR A LA CREATION DU CLIENT: vérifiez les données entrées");
        e.printStackTrace();
    }
}

public static void main(String args[]) {
    new Creation_Client();
}
}

```

Creation_livre.java

```

import java.sql.*;
import java.io.*;

public class Creation_Livres {

    static final String CONN_URL = "jdbc:oracle:thin:@ensibm.imag.fr:1521:ensi2";
    static final String USER = "sportoud"; // A remplacer pour votre compte, sinon
    genere une exception
    static final String PASSWD = "toto";

    public Creation_Livres() {
        int num_photo=1;
        int test=0;
        int cover=0;
        int codalbum_photo=0;
        int a2=0;
        int a3=0;
        int a4=0;
        int a5=0;
        int a7=0;
        int a9=0;
        int all=0;
        int codalbum_livre=0;
        String chemin_livre="init";
        String mail_test="init";
        String mail_livre="init";
        String chemin_file="init";

        try{
            // Enregistrement du driver Oracle
            System.out.println("Loading Oracle thin driver...");
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            System.out.println("loaded.");

            // Etablissement de la connexion
            System.out.println("Connecting to the database...");
            Connection conn = DriverManager.getConnection(CONN_URL, USER, PASSWD);
            System.out.println("connected.");

            // Demarrage de la transaction (implicite dans notre cas)
            conn.setAutoCommit(false);
            conn.setTransactionIsolation(conn.TRANSACTION_SERIALIZABLE);

            EasyIn easy = new EasyIn();

            // login de l'utilisateur : connection
            System.out.println("***Entrez votre login:***");
            String login = easy.readString();
            System.out.println("***Entrez votre mot de passe:***");
            String passwd = easy.readString();

            //AUTHENTIFICATION
            PreparedStatement pstmt5 = conn.prepareStatement("select count(*) from CLIENT
            where login='"+login+"' and passwd='"+passwd+"' ");

```

```

ResultSet rset5 = pstmt5.executeQuery();
while(rset5.next()){
    a5=rset5.getInt(1);
}
rset5.close();
conn.commit();
pstmt5.close();
if (a5==0){
    System.out.println(">>L'authentification a échoué");
    return ;
}

// dans le cas ou l'authentification a réussi on récupère l'adresse mail de
l'utilisateur
PreparedStatement pstmt6 = conn.prepareStatement ("select mail from CLIENT where
login = '"+login+"' ");
ResultSet rset6 = pstmt6.executeQuery();

while(rset6.next()){
    mail_livre=rset6.getString(1);
}
conn.commit();
pstmt6.close();
rset6.close();

// on rajoute dans la table visible tous les albums que possède l'utilisateur.
PreparedStatement pstmt8 = conn.prepareStatement ("select codalbum from ALBUM
where mail = '"+mail_livre+"' ");
ResultSet rset8 = pstmt8.executeQuery();
while(rset8.next()){
    PreparedStatement pstmt9 = conn.prepareStatement("select count(*) from
VISIBLE where mail='"+mail_livre+"' and codalbum='"+rset8.getInt(1)+"' ");
    ResultSet rset9 = pstmt9.executeQuery();
    while(rset9.next()){
        a9=rset9.getInt(1);
    }
    rset9.close();
    conn.commit();
    pstmt9.close();
    if (a9==0){
        Statement stmt8=conn.createStatement();
        int rest8 = stmt8.executeUpdate("insert into
VISIBLE(mail,codalbum) values '"+mail_livre+"','"+rset8.getInt(1)+"'");
        conn.commit();
        stmt8.close();
    }
}
conn.commit();
pstmt8.close();
rset8.close();

// on entre les paramètres du livre
debut0:
while(true){
    System.out.println("***code du livre qui va etre crée:***");
    codalbum_livre = easy.readInt();

    // on vérifie qu'un autre album n'a pas déjà le meme codalbum
    PreparedStatement pstmt7 = conn.prepareStatement ("select count(*) from
ALBUM where codalbum = '"+codalbum_livre+"' ");
    ResultSet rset7 = pstmt7.executeQuery();
    while(rset7.next()){
        a7=rset7.getInt(1);
    }
    conn.commit();
    pstmt7.close();
    rset7.close();
    if (a7==0){
        break;
    }else{
        System.out.println(">>ce numero de codalbum existe deja. Veuillez
en choisir un autre");
        continue debut0;
    }
}
}

```

```

System.out.println("***titre du livre qui va etre crée:***");
String titralbum_livre = easy.readString();

System.out.println("***sous-titre du livre qui va etre crée:***");
String stitralbum_livre = easy.readString();

System.out.println("***preface du livre qui va etre crée:***");
String preface_livre = easy.readString();

System.out.println("***postface du livre qui va etre crée:***");
String postface_livre = easy.readString();

debut:
while (test==0){ // on entre les photos dans le livre

// Condition : la photo doit faire partie d'un album visible ou d'une
image possédée

// chemin du fichier image correspondant à une photo qui va etre ajouté à
l'album
System.out.println("***chemin de l'image à insérer dans le livre:***");
String chemin_photo = easy.readString();

// on regarde si ce chemin de fichier existe bien.
PreparedStatement pstmt4 = conn.prepareStatement ("select count(*) from
FILEPICTURE where chemin = '"+chemin_photo+"' ");
ResultSet rset4 = pstmt4.executeQuery();
while(rset4.next()){
    a4=rset4.getInt(1);
}
conn.commit();
pstmt4.close();
rset4.close();
if (a4==0){
    System.out.println(">>ce chemin de fichier image n'existe pas.
Veillez en entrer un autre");
    continue debut;
}

//on récupère l'adresse mail du possesseur de la photo
PreparedStatement pstmt0 = conn.prepareStatement ("select mail from
FILEPICTURE where chemin = '"+chemin_photo+"' ");
ResultSet rset0 = pstmt0.executeQuery();
while(rset0.next()){
    mail_test=rset0.getString(1);
}
conn.commit();
pstmt0.close();
rset0.close();

// dans le cas ou l'utilisateur est possesseur du fichier image , on
ajoute directement la photo au livre.
if (mail_test.compareTo(mail_livre)==0){

    System.out.println("***titre de la photo qui va etre
insérée:***");
    String titrephoto = easy.readString();

    System.out.println("***commentaire de la photo qui va etre
insérée:***");
    String commentaire = easy.readString();

// on crée ensuite la photo dans la table avec le codalbum
correspondant au livre crée
Statement stmt01=conn.createStatement();
int rest01 = stmt01.executeUpdate("insert into
PHOTOS(codalbum,numordre,chemin,titrephoto,commentaire)
values('"+codalbum_livre+"','"+num_photo+"','"+chemin_photo+"','"+titrephoto+"','"+comme
ntaire+"'");

    conn.commit();

```

```

        stmt01.close();

        num_photo=num_photo+1;

        System.out.println("***Tapez 0 pour ajouter une nouvelle photo au
livre ou 1 pour créer le livre:***");
        test = easy.readInt();
        continue debut;
    }

    //on récupère le code de l'album auquel la photo appartient
    PreparedStatement pstmt1 = conn.prepareStatement ("select codalbum from
PHOTOS where chemin = '"+chemin_photo+"' ");
    ResultSet rset1 = pstmt1.executeQuery();
    while(rset1.next()){
        codalbum_photo=rset1.getInt(1);
    }
    conn.commit();
    pstmt1.close();
    rset1.close();

    // on regarde si l'utilisateur a bien visibilité sur l'album dont la
photo fait partie
    PreparedStatement pstmt2 = conn.prepareStatement ("select count(*) from
VISIBLE where mail='"+mail_livre+"' and codalbum='"+codalbum_photo+"' ");
    ResultSet rset2 = pstmt2.executeQuery();
    while(rset2.next()){
        a2=rset2.getInt(1);
    }
    conn.commit();
    pstmt2.close();
    rset2.close();
    if (a2==0){// on regarde si l'utilisateur a visibilité ou non sur l'album
dont la photo fait partie
        System.out.println(">>>la photo que vous cherchez à télécharger
fait partie d'un album sur lequel vous n'avez pas visibilité");
        continue debut; //on invite l'utilisateur à entrer un nouvel
album si celui ci n'existe pas
    }

    // on entre un titre et un commentaire pour la photo

    System.out.println("***titre de la photo qui va etre insérée:***");
    String titrephoto = easy.readString();

    System.out.println("***commentaire de la photo qui va etre insérée:***");
    String commentaire = easy.readString();

    // on crée ensuite la photo dans la table avec le codalbum correspondant
au livre créé
    Statement stmt1=conn.createStatement();
    int rest1 = stmt1.executeUpdate("insert into
PHOTOS(codalbum,numordre,chemin,titrephoto,commentaire)
values('"+codalbum_livre+"','"+num_photo+"','"+chemin_photo+"','"+titrephoto+"','"+comme
ntaire+"'");
    conn.commit();
    stmt1.close();

    num_photo=num_photo+1;

    System.out.println("***Tapez 0 pour ajouter une nouvelle photo au livre
ou 1 pour créer le livre:***");
    test = easy.readInt();

} // sortie de la boucle insérant les images

// on entre le chemin de la photo qui va figurer en couverture.

```

```

debut2:
while(true){

    System.out.println("***chemin de la photo qui va figurer en couverture du
livre:***");
    chemin_livre = easy.readString();

    // on regarde si la photo qui va figuré en couverture est bien contenue
dans le livre
    PreparedStatement pstmt3 = conn.prepareStatement ("select count(*) from
PHOTOS where chemin='"+chemin_livre+"' and codalbum='"+codalbum_livre+"' ");
    ResultSet rset3 = pstmt3.executeQuery();
    while (rset3.next()){
        a3=rset3.getInt(1);
    }
    conn.commit();
    pstmt3.close();
    rset3.close();

    if (a3==0){
        System.out.println(">>la photo que vous voulez placer sur la
couverture n'est pas contenue dans le livre");
        continue debut2;
    }else{
        break;
    }
}

    // création effective du livre
    Statement stmt2=conn.createStatement();
    int rest2 = stmt2.executeUpdate("insert into
ALBUM(codalbum,chemin,mail,titralbum,stitalbum,preface,postface)
values('"+codalbum_livre+"','"+chemin_livre+"','"+mail_livre+"','"+titralbum_livre+"','"+
+stitalbum_livre+"','"+preface_livre+"','"+postface_livre+"')");
    conn.commit();
    stmt2.close();

    // on détruit les fichiers images qui ne sont pas associés à des photos, c'est à
dire les fichiers images qui correspondent à
//des images ne figurant dans aucun album.
    PreparedStatement pstmt10 = conn.prepareStatement ("select chemin from
FILEPICTURE ");
    ResultSet rset10 = pstmt10.executeQuery();
    while(rset10.next()){
        chemin_file=rset10.getString(1);
        PreparedStatement pstmt11 = conn.prepareStatement("select count(*) from
PHOTOS where chemin='"+chemin_file+"' ");
        ResultSet rset11 = pstmt11.executeQuery();
        while(rset11.next()){
            all=rset11.getInt(1);
        }
        rset11.close();
        conn.commit();
        pstmt11.close();
        if (all==0){
            Statement stmt10=conn.createStatement();
            int rest10 = stmt10.executeUpdate("delete from FILEPICTURE where
chemin='"+chemin_file+"'");
            conn.commit();
            stmt10.close();
        }
    }
    conn.commit();
    pstmt10.close();
    rset10.close();

    //fermeture
    conn.close();

}
catch (SQLException e) {
    System.err.println("ERREUR LORS DE LA CREATION DU LIVRE: vérifiez les données
entrées");
    e.printStackTrace();
}

```

```

    }
}

public static void main(String args[]) {
    new Creation_Livres();
}
}

```

Consultation_album.java

```

import java.sql.*;
import java.io.*;

public class Consultation_album {

    static final String CONN_URL = "jdbc:oracle:thin:@ensibm.imag.fr:1521:ensi2";
    static final String USER = "sportoud"; // A remplacer pour votre compte, sinon
    genere une exception
    static final String PASSWD = "toto";

    public Consultation_album() {

        String mail = "init";
        int a0=0;
        int a9=0;

        try{
            // Enregistrement du driver Oracle
            System.out.println("Loading Oracle thin driver...");
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            System.out.println("loaded.");

            // Etablissement de la connexion
            System.out.println("Connecting to the database...");
            Connection conn = DriverManager.getConnection(CONN_URL, USER, PASSWD);
            System.out.println("connected.");

            // Demarrage de la transaction (implicite dans notre cas)
            conn.setAutoCommit(false);
            conn.setTransactionIsolation(conn.TRANSACTION_SERIALIZABLE);

            EasyIn easy=new EasyIn();

            // login de l'utilisateur
            System.out.println("***login de l'utilisateur:***");
            String login = easy.readString();
            System.out.println("***password de l'utilisateur:***");
            String passwd = easy.readString();
            //AUTHENTIFICATION
            PreparedStatement pstmt0 = conn.prepareStatement("select count(*) from CLIENT
where login='"+login+"' and passwd='"+passwd+"' ");
            ResultSet rset0 = pstmt0.executeQuery();
            while(rset0.next()){
                a0=rset0.getInt(1);
            }
            conn.commit();
            pstmt0.close();
            if (a0==0){
                System.out.println(">>>L'authentification a échoué");
                return ;
            }
            rset0.close();

            // dans le cas ou l'authentification a réussi on récupère l'adresse mail de
            l'utilisateur
            PreparedStatement pstmt1 = conn.prepareStatement ("select mail from CLIENT where
login = '"+login+"' ");
            ResultSet rset1 = pstmt1.executeQuery();

            while(rset1.next()){
                mail=rset1.getString(1);
            }
            conn.commit();
            pstmt1.close();
            rset1.close();

            // on remplit la table visible avec tous les albums que l'utilisateur possède
            ( on rajoute uniquement dans la table ceux qui n'y

```

```

//figure pas déjà).
PreparedStatement pstmt8 = conn.prepareStatement ("select codalbum from ALBUM
where mail = '"+mail+"' ");
ResultSet rset8 = pstmt8.executeQuery();
while(rset8.next()){
    PreparedStatement pstmt9 = conn.prepareStatement("select count(*) from
VISIBLE where mail='"+mail+"' and codalbum='"+rset8.getInt(1)+"' ");
    ResultSet rset9 = pstmt9.executeQuery();
    while(rset9.next()){
        a9=rset9.getInt(1);
    }
    rset9.close();
    conn.commit();
    pstmt9.close();
    if (a9==0){
        Statement stmt8=conn.createStatement();
        int rest8 = stmt8.executeUpdate("insert into
VISIBLE(mail,codalbum) values '"+mail+"', '"+rset8.getInt(1)+"'");
        conn.commit();
        stmt8.close();
    }
}
conn.commit();
pstmt8.close();
rset8.close();

System.out.println("***tapez 0 pour consulter un album ou 1 pour voir la liste
des albums sur lesquels vous avez visibilité***");
int entree =easy.readInt();

if (entree==1){
    System.out.println("***Voici la liste des albums qui vous sont
accessibles***");

    PreparedStatement pstmt3 = conn.prepareStatement("select * from VISIBLE
where mail='"+mail+"'");
    ;
    ResultSet rset3 = pstmt3.executeQuery();
    dumpResultSet(rset3);
    rset3.close();
    conn.commit();
    pstmt3.close();

}

}

else{
    System.out.println("***entrez le code de l'album que vous désirez
consulter***");
    int codalbum=easy.readInt();

    PreparedStatement pstmt4 = conn.prepareStatement("select * from PHOTOS
where codalbum='"+codalbum+"'");
    ;
    ResultSet rset4 = pstmt4.executeQuery();
    dumpResultSet(rset4);
    rset4.close();
    conn.commit();
    pstmt4.close();
}

}

conn.close();
}
catch (SQLException e) {
System.err.println("ERREUR LORS DE LA CREATION DU LIVRE");
e.printStackTrace();
}
}

private void dumpResultSet(ResultSet rset) throws SQLException {
    ResultSetMetaData rsetmd = rset.getMetaData();
    int i = rsetmd.getColumnCount();
    for (int k=1;k<=i;k++)
        System.out.print(rsetmd.getColumnName(k) + "\t");
    System.out.println();
    while (rset.next()) {
        for (int j = 1; j <= i; j++) {
            System.out.print(rset.getString(j) + "\t\t");
        }
    }
}

```

```

        System.out.println();
    }
}

public static void main(String args[]) {
    new Consultation_album();
}
}

```

Saisie_commande_album.java

```

import java.sql.*;
import java.io.*;
import java.util.Calendar;

public class Saisie_commande {

    static final String CONN_URL = "jdbc:oracle:thin:@ensibm.imag.fr:1521:ensi2";
    static final String USER = "sportoud"; // A remplacer pour votre compte, sinon
    genere une exception
    static final String PASSWD = "toto";

    public Saisie_commande() {

        String mail="init";
        int a0=0;
        int a9=0;
        int test=0;
        int a4=0;
        int a6=0;
        int a15=0;
        int a16=0;
        int numcommande=1;
        double prixtotal=0;
        double prix_format=0;

        try{
            // Enregistrement du driver Oracle
            System.out.println("Loading Oracle thin driver...");
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            System.out.println("loaded.");

            // Etablissement de la connexion
            System.out.println("Connecting to the database...");
            Connection conn = DriverManager.getConnection(CONN_URL, USER, PASSWD);
            System.out.println("connected.");

            // Demarrage de la transaction (implicite dans notre cas)
            conn.setAutoCommit(false);
            conn.setTransactionIsolation(conn.TRANSACTION_SERIALIZABLE);

            EasyIn easy=new EasyIn();

            // login de l'utilisateur
            System.out.println("***login de l'utilisateur***");
            String login = easy.readString();
            System.out.println("***password de l'utilisateur***");
            String passwd = easy.readString();
            //AUTHENTIFICATION
            PreparedStatement pstmt0 = conn.prepareStatement("select count(*) from CLIENT
where login='"+login+"' and passwd='"+passwd+"' ");
            ResultSet rset0 = pstmt0.executeQuery();
            while(rset0.next()){
                a0=rset0.getInt(1);
            }
            conn.commit();
            pstmt0.close();
            if (a0==0){
                System.out.println(">>>L'authentification a échoué");
                return ;
            }
            rset0.close();
            // dans le cas ou l'authentification a réussi on récupère l'adresse mail de
            l'utilisateur
            PreparedStatement pstmt1 = conn.prepareStatement ("select mail from CLIENT where
login = '"+login+"' ");

```

```

ResultSet rset1 = pstmt1.executeQuery();

while(rset1.next()){
    mail=rset1.getString(1);
}
conn.commit();
pstmt1.close();
rset1.close();

// on remplit la table visible avec tous les albums que l'utilisateur possède
( on rajoute uniquement dans la table ceux qui n'y
//figure pas déjà).
PreparedStatement pstmt8 = conn.prepareStatement ("select codalbum from ALBUM
where mail = '"+mail+"' ");
ResultSet rset8 = pstmt8.executeQuery();
while(rset8.next()){
    PreparedStatement pstmt9 = conn.prepareStatement("select count(*) from
VISIBLE where mail='"+mail+"' and codalbum='"+rset8.getInt(1)+"' ");
    ResultSet rset9 = pstmt9.executeQuery();
    while(rset9.next()){
        a9=rset9.getInt(1);
    }
    rset9.close();
    conn.commit();
    pstmt9.close();
    if (a9==0){
        Statement stmt8=conn.createStatement();
        int rest8 = stmt8.executeUpdate("insert into
VISIBLE(mail,codalbum) values ('"+mail+"', '"+rset8.getInt(1)+"')");
        conn.commit();
        stmt8.close();
    }
}
conn.commit();
pstmt8.close();
rset8.close();

// on affiche la liste des albums qui peuvent etre commandé par l'utilisateur
System.out.println("***Voici liste des albums que vous pouvez commander***");
PreparedStatement pstmt3 = conn.prepareStatement("select * from VISIBLE where
mail='"+mail+"'");
ResultSet rset3 = pstmt3.executeQuery();
dumpResultSet(rset3);
rset3.close();
conn.commit();
pstmt3.close();

// on génère un numéro de commande, en faisant attention au cas ou c'est la
premiere commande qu'on rentre dans la base de
//donnée.
PreparedStatement pstmt4 = conn.prepareStatement("select count(*) from COMMANDE
");
ResultSet rset4 = pstmt4.executeQuery();
while(rset4.next()){
    a4=rset4.getInt(1);
}
rset4.close();
conn.commit();
pstmt4.close();

if (a4==0){
    numcommande=1;
}else{

    PreparedStatement pstmt6 = conn.prepareStatement("select max(numcommande)
from COMMANDE ");
    ResultSet rset6 = pstmt6.executeQuery();
    while(rset6.next()){
        a6=rset6.getInt(1);
    }
    rset6.close();
    conn.commit();
    pstmt6.close();

    numcommande=a6+1;
}

```

```

        Date_Commande dt=new Date_Commande();
        Calendar cal = new java.util.GregorianCalendar(dt.getAnnee(), dt.getMois(),
dt.getJour());
        Date date_sql = new Date(cal.getTime().getTime());

        System.out.println("la date:"+dt.getJour()+"-"+dt.getMois()+"-"+dt.getAnnee());

        String testing = "insert into COMMANDE(numcommande,mail,date_sql,prixtotal)
values('"+numcommande+"','"+mail+"',to_date('"+date_sql+"','YYYY-MM-DD'),'0')";

        //création provisoire de la commande dans la table
        Statement stmt11=conn.createStatement();
        int rest11 = stmt11.executeUpdate(testing);
        conn.commit();
        stmt11.close();

        // on débute ici la commande
        debut:
        while (test==0){

                // on entre les parametres de la commande
                System.out.println("***Entrez le code de l'album/livre que vous désirez
commander***");
                int codalbum = easy.readInt();

                System.out.println("***Entrez le format avec lequel vous désirez imprimer
cet ou ces albums***");
                int codeformat = easy.readInt();

                PreparedStatement pstmt16 = conn.prepareStatement("select count(*) from
FORMAT where codeformat='"+codeformat+"' ");
                ResultSet rset16 = pstmt16.executeQuery();
                while(rset16.next()){
                        a16=rset16.getInt(1);
                }
                conn.commit();
                pstmt16.close();
                if (a16==0){
                        System.out.println(">>>Ce type de format n'existe pas veuillez en
choisir un autre");
                        continue debut ;
                }
                rset16.close();

                System.out.println("***Entrez le nombre d'exemplaires de cet album que
vous désirez commander***");
                int quantite = easy.readInt();

                // on regarde si ce type d'impression pour cet album dans la commande n'a pas
                déjà eu lieu.
                PreparedStatement pstmt15 = conn.prepareStatement("select count(*) from
IMPRIME_AVEC where numcommande='"+numcommande+"' and codeformat='"+codeformat+"' and
codalbum='"+codalbum+"' ");
                ResultSet rset15 = pstmt15.executeQuery();
                while(rset15.next()){
                        a15=rset15.getInt(1);
                }
                conn.commit();
                pstmt15.close();
                if (a15!=0){
                        System.out.println(">>>Cet album a déjà été imprimé avec ce type de
format dans la commande. Veuillez choisir de nouveau paramètres");
                        continue debut ;
                }
                rset15.close();

                Statement stmt2=conn.createStatement();
                int rest2 = stmt2.executeUpdate("insert into
IMPRIME_AVEC(numcommande,codeformat,codalbum)
values('"+numcommande+"','"+codeformat+"','"+codalbum+"')");
                conn.commit();
                stmt2.close();

```

```

        Statement stmt3=conn.createStatement();
        int rest3 = stmt3.executeUpdate("insert into
CONTIENT(codeformat,numcommande) values ('"+codeformat+"', '"+numcommande+"')");
        conn.commit();
        stmt3.close();

        Statement stmt4=conn.createStatement();
        int rest4 = stmt4.executeUpdate("insert into
PORTE_SUR(numcommande,codalbum,quantite)
values ('"+numcommande+"', '"+codalbum+"', '"+quantite+"')");
        conn.commit();
        stmt4.close();

        PreparedStatement pstmt5 = conn.prepareStatement("select prix from FORMAT
where codeformat='"+codeformat+" ");
        ResultSet rset5 = pstmt5.executeQuery();
        while(rset5.next()){
            prix_format=rset5.getDouble(1);
        }
        rset5.close();
        conn.commit();
        pstmt5.close();

        prixtotal=prixtotal+quantite*prix_format;

        System.out.println("***Tapez 0 pour rajouter des éléments à la commande
ou 1 pour vous arreter ici***");
        test=easy.readInt();

        } // fin de la commande ici

        // on modifie ici la ligne correspondant à la commande dans la table pour avoir
la bonne valeur pour prixtotal.
        PreparedStatement pstmt14 = conn.prepareStatement("Update Commande set prixtotal
= "+prixtotal+" where numcommande = "+numcommande+"");
        ResultSet rset14 = pstmt14.executeQuery();
        rset14.close();
        conn.commit();
        pstmt14.close();

        conn.close();
    } catch (SQLException e) {
        System.err.println("ERREUR LORS DE LA VALIDATION DE LA COMMANDE");
        e.printStackTrace();
    }
}

private void dumpResultSet(ResultSet rset) throws SQLException {
    ResultSetMetaData rsetmd = rset.getMetaData();
    int i = rsetmd.getColumnCount();
    for (int k=1;k<=i;k++)
        System.out.print(rsetmd.getColumnName(k) + "\t\t");
    System.out.println();
    while (rset.next()) {
        for (int j = 1; j <= i; j++) {
            System.out.print(rset.getString(j) + "\t\t");
        }
        System.out.println();
    }
}

public static void main(String args[]) {
    new Saisie_commande();
}
}

```

Date_test.java

```
import java.util.*;
import java.io.*;

// but : récupérer la date actuelle, on la convertit ensuite au format date_sql

class Date_Commande {
    Date rightNow;

    public Date_Commande() { // constructeur

        rightNow=new java.util.Date();

    }

    // méthodes
    public int getAnnee(){
        return (rightNow.getYear()+1900);
    }

    public int getMois(){
        return (rightNow.getMonth());
    }

    public int getJour(){
        return rightNow.getDate();
    }

}

public class Date_test{
    public static void main(String args[] ) {
        Date_Commande d= new Date_Commande();
        System.out.println("la
date:"+d.getJour()+"-"+d.getMois()+"-"+d.getAnnee());
    }
}
```

Telechargement_photo.java

```
import java.io.*;

public class Telechargement_photos {

    static final String CONN_URL = "jdbc:oracle:thin:@ensibm.imag.fr:1521:ensi2";
    static final String USER = "sportoud"; // A remplacer pour votre compte, sinon
    genere une exception
    static final String PASSWD = "toto";

    public Telechargement_photos() { // le téléchargement consiste en le dépôt d'un
    fichier image sur le serveur en rentrant tous les
    //paramètres.

        String mail="tata";
        int a=0;

        try {
            // Enregistrement du driver Oracle
            System.out.println("Loading Oracle thin driver...");
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            System.out.println("loaded.");

            // Etablissement de la connexion
            System.out.println("Connecting to the database...");
            Connection conn = DriverManager.getConnection(CONN_URL, USER, PASSWD);
            System.out.println("connected.");

            // Demarrage de la transaction (implicite dans notre cas)
            conn.setAutoCommit(false);
            conn.setTransactionIsolation(conn.TRANSACTION_SERIALIZABLE);
        }
    }
}
```

```

EasyIn easy = new EasyIn();

// login de l'utilisateur
System.out.println("***login de l'utilisateur:***");
String login = easy.readString();
System.out.println("***password de l'utilisateur:***");
String passwd = easy.readString();
//AUTHENTIFICATION
PreparedStatement pstmt0 = conn.prepareStatement("select count(*) from
CLIENT where login='"+login+"' and passwd='"+passwd+"' ");
ResultSet rset0 = pstmt0.executeQuery();
while(rset0.next()){
    a=rset0.getInt(1);
}
conn.commit();
pstmt0.close();
if (a==0){
    System.out.println(">>L'authentification a échoué");
    return ;
}
rset0.close();

// dans le cas ou l'authentification a réussi on récupère l'adresse mail
de l'utilisateur
PreparedStatement pstmt1 = conn.prepareStatement ("select mail from
CLIENT where login = '"+login+"' ");
ResultSet rset1 = pstmt1.executeQuery();

while(rset1.next()){
    mail=rset1.getString(1);
}
conn.commit();
pstmt1.close();
rset1.close();

// chemin de la photo à déposer
System.out.println("***chemin de la photo à déposer:***");
String chemin = easy.readString();

System.out.println("***appareil utilisé:***");
String appareil = easy.readString();

System.out.println("***objectif utilisé:***");
String objectif = easy.readString();

System.out.println("***distance focale:***");
double distancefoc = easy.readDouble();

System.out.println("***sensibilité ISO:***");
double sensibilite = easy.readDouble();

System.out.println("***ouverture:***");
double ouverture = easy.readDouble();

System.out.println("***vitesse d'obturation:***");
double vitessobtu = easy.readDouble();

System.out.println("***résolution de l'image:***");
double resolution = easy.readDouble();

// modification de la base de donnée : rajout du fichier image dans la
table FILEPICTURE
Statement stmt=conn.createStatement();
int rest = stmt.executeUpdate("insert into
FILEPICTURE(chemin,mail,appareil,objectif,distancefoc,sensibilite,ouverture,vitessobtu,r
esolution) values
('"+chemin+"','"+mail+"','"+appareil+"','"+objectif+"','"+distancefoc+"','"+sensibilite+
"','"+ouverture+"','"+vitessobtu+"','"+resolution+"')");

rset1.close();
// terminaison de la modification
conn.commit();

```

```
        //fermeture
        stmt.close();
        conn.close();
    }
    catch (SQLException e) {
        System.err.println("ERREUR LORS DU TELECHARGEMENT:vérifiez les données entrées");
        e.printStackTrace();
    }
}

public static void main(String args[]) {
    new Telechargement_photos();
}
}
```

