Etudes de cas

Projet de rétro-ingénierie - Conversion de fichiers COBOL

Jean-Luc Hainaut

en collaboration avec Jean Henrard

23 juillet 2005

1 Introduction

Il existe pratiquement autant d'approches de la rétro-ingénierie qu'il y a de projets, de sorte que l'idée de développer une application pilote qui pourrait servir de modèle aux analystes de terrain serait illusoire. Notre objectif est donc plus modeste : proposer le traitement complet de la conversion des structures de données d'une collection de fichiers standards (COBOL en l'occurrence) en structures de base de données relationnelle, essentiellement par analyse du code source d'une petite application. Nous ignorerons l'exploitation des autres sources d'information qui sont typiquement utilisées lors de la conduite de projets de rétro-ingénierie en vraie grandeur. Le lecteur se tournera alors vers les études de cas, qui abordent le traitement de ces sources.

1.1 Démarche de rétro-ingénierie

Nous rappellerons brièvement les phases principales de la rétro-ingénierie sous la forme des scénarios des Figures 1 et 3.

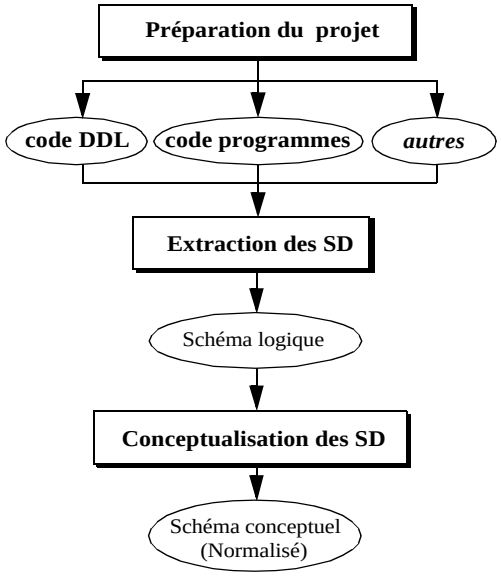

Figure 1 - Les phases du processus de rétro-ingénierie.

a) Phase de préparation du projet

La portée du projet est délimitée. En particulier, on fixe les objectifs, on identifie les sources d'information, les ressources nécessaires, on établit le planning et les techniques de conduite du projet. En particulier, on fixe les critères de validation des produits et de terminaison des processus.

L'un des documents particulièrement utiles dans cette tâche de préparation est l'architecture générale de l'application, qu'il est en général assez aisé de reconstituer.

b) Phase d'extraction des structures de données

On produit, à partir des sources disponibles, un *schéma logique* incluant les constructions explicites et les constructions implicites. Les premières sont déclarées dans le code DDL définissant les structures de données tandis que les secondes doivent être découvertes par l'analyse de sources telles que les données et le code source des programmes.

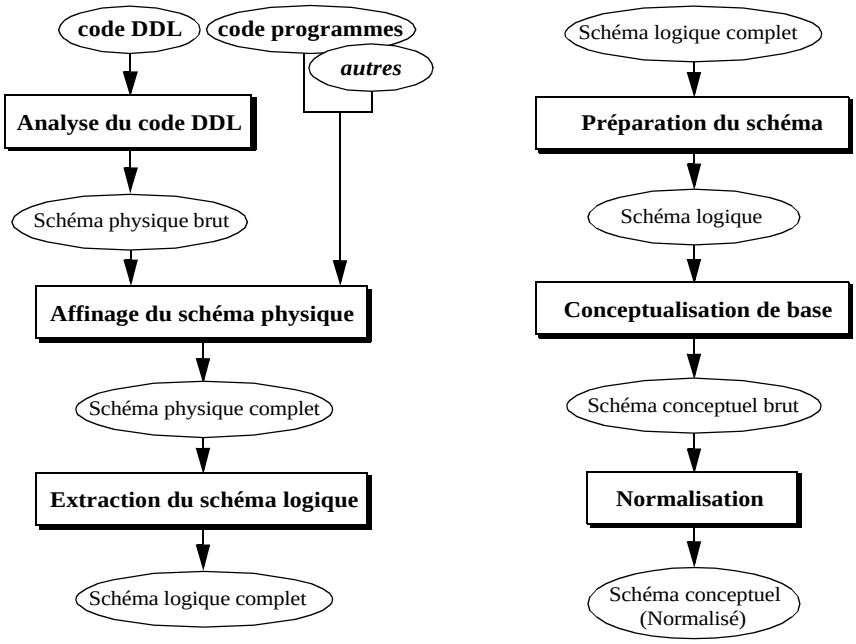

Figure 3 - Extraction (gauche) et conceptualisation (droite) des structures de données.

Cette phase comporte généralement trois processus :

1. l'*analyse du code DDL*, qui fournit un *schéma physique brut* (processus le plus souvent automatique);
2. l'*affinage du schéma physique*, qui enrichit celui-ci des constructions implicites découvertes par l'analyse des autres sources d'information, telles que le code des programmes d'application utilisant la base de données;
3. l'*extraction du schéma logique*, qui produit le *schéma logique complet* à partir du schéma physique en retirant de ce dernier les constructions physiques qui ne seraient pas pertinentes au niveau logique[1].

On notera que ce schéma logique final est en général plus riche que ce que permettrait le modèle du SGD pris au sens strict. On trouvera par exemple des clés étrangères exprimant les liens de référence implicites dans une collection de fichiers COBOL.

d) Phase de conceptualisation des structures de données

Cette phase produit un schéma conceptuel qui constitue l'interprétation sémantique la plus probable du schéma logique. Elle comprend trois processus essentiels :

1. la *préparation du schéma*, par laquelle on conditionne le schéma pour en améliorer la lisibilité et l'interprétation; en particulier, on supprimera les constructions techniques qui subsisteraient à ce stade;
2. la *conceptualisation du schéma* proprement dite, qui produit par transformations successives un premier schéma conceptuel brut; il comprend deux types de transformations :
   a) *désoptimisation* : on élimine ou on remplace par transformation les structures destinées à optimiser la gestion et l'exploitation des données;
   b) *détraduction* : on remplace les constructions propres au modèle du système de gestion de données par leur interprétation conceptuelle;
3. la *normalisation*, qui transforme le schéma conceptuel brut pour en éliminer les anomalies de représentation et pour en améliorer les qualités de lisibilité et d'expressivité, ou encore pour le rendre conforme à un standard méthodologique déterminé.

2 Description du projet

Le projet que nous allons réaliser consiste à convertir les structures de données d'un ensemble de fichiers COBOL en une base de données relationnelle équivalente.

On dispose d'une seule source d'information : le texte source d'un programme COBOL qui gère et exploite le contenu de ces fichiers. Ce programme comprend des sections de définition des fichiers et des types d'enregistrements, ce que nous désignerons par le terme de *code DDL*[2]. Il comprend aussi la définition des variables locales et le code procédural.

Nous construirons un schéma logique aussi complet que possible à partir du code du programme, nous le transformerons en un schéma conceptuel plausible, puis nous traduirons ce dernier en un schéma relationnel exprimé en SQL-DDL.

1. Rappelons que certaines constructions techniques sont de niveau logique pour certains modèles de données anciens (COBOL, CODASYL, IMS, SQL 89 par exemple).
2. DDL = Data Definition Language.

3 Préparation du projet

Ce processus consiste essentiellement en l'identification des sources d'information, ce qui se réduit ici au texte source du programme retenu. Le code complet est repris à la Section 7. Le lecteur non familier avec le langage COBOL trouvera à l'annexe 5 un résumé des principales constructions de ce langage.

La structure générale du programme est donnée à la Figure 5. Elle permet d'identifier les portions de programmes qui accèdent aux fichiers.

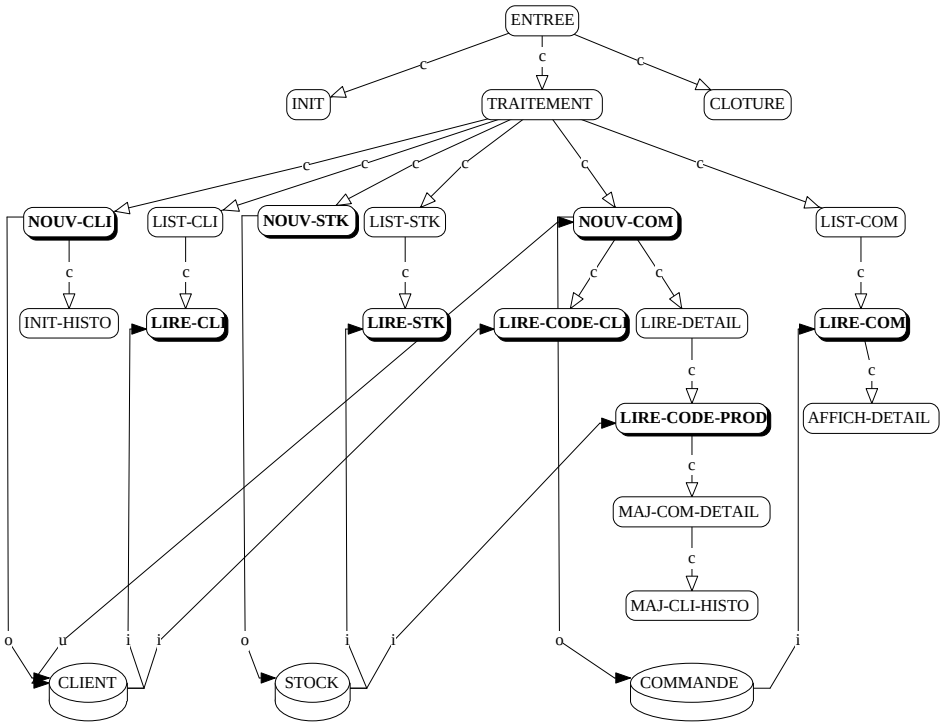

Figure 5 - Architecture générale du programme. On y représente les liens d'invocation (arcs "c" du graphe des appels, ou *call graph*) entre procédures ainsi que les liens d'interaction avec les fichiers ("i" pour *input*, "o" pour *output* et "u" pour *update*).

4 Extraction des structures de données

Nous analyserons d'abord les sections DDL du programme pour en extraire le schéma physique brut, puis nous affinerons ce schéma pour y incorporer les constructions implicites que nous découvrirons par l'analyse du reste du programme.

4.1 Extraction du schéma physique brut des textes DDL

Le code DDL d'un programme COBOL comprend deux parties utiles.

- le FILE-CONTROL de l'INPUT-OUTPUT SECTION, qui définit les fichiers utilisés par le programme, leur organisation, leurs index et le caractère identifiant de ces derniers [006-019].
- les FD de la FILE SECTION, qui énumèrent les types d'enregistrements de chaque fichier, leur décomposition en champs et le type de valeurs de chacun d'eux [022-036].

Le résultat de l'interprétation de ce code est présenté à la Figure 6.

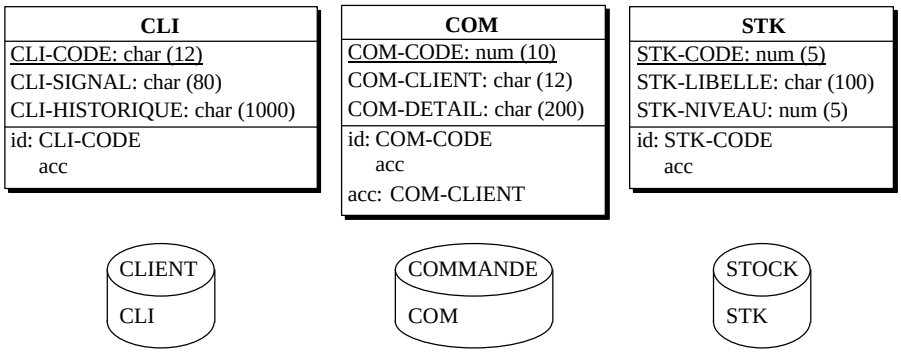

Figure 6 - Schéma physique brut extrait du code DDL du programme COBOL.

4.2 Affinage du schéma physique

Nous analyserons ensuite le code procédural pour y découvrir les éventuelles constructions implicites. L'usage de l'organisation séquentielle indexée nous autorise raisonnablement à considérer que tous les identifiants sont déclarés explicitement, de sorte qu'il n'est pas nécessaire de rechercher la présence d'autres identifiants dans cette phase d'affinage.

Nous retiendrons un nombre limité d'objectifs : la *décomposition* précise des champs, les *clés étrangères*, les *identifiants des champs multivalués* et les *cardinalités* précises des champs multivalués. Dans un projet réel, il conviendrait sans doute de rechercher d'autres constructions et contraintes implicites.

g) Décomposition précise des champs

La présence de champs de grande taille suggère que ceux-ci pourraient être dotés d'une structure implicite que nous allons essayer de découvrir. Nous nous intéresserons de plus près aux quatre champs CLI-SIGNAL, CLI-HISTORIQUE, COM-DETAIL et STK-LIBELLE.

L'analyse du diagramme des flux du programme montre que le champ CLI-SIGNAL est en relation avec la variable locale SIGNALETIQUE. On trouve en effet deux instructions, en [105] et [152] (Code 1), qui indiquent que, lors de l'exécution

du programme, ce champ et cette variable contiennent les mêmes valeurs. On peut donc légitimement penser que la structure de ces valeurs est identique.

```
MOVE SIGNALETIQUE TO CLI-SIGNAL.                [105]
MOVE CLI-SIGNAL TO SIGNALETIQUE.                [152]
```

Code 1 - Instructions d'assignation définissant le graphe de flux concernant le champ CLI-SIGNAL.

Nous allons donc assigner au champ CLI-SIGNAL la structure de la variable SIGNALISATION (Code 2), ce qu'on consigne dans le schéma physique (Figure 8).

```
01 SIGNALETIQUE.                                [038]
   02 NOM PIC X(20).                            [039]
   02 ADRESSE PIC X(40).                        [040]
   02 FONCTION PIC X(10).                       [041]
   02 DATE-ENREG PIC X(10).                     [042]
```

Code 2 - Définition de la variable SIGNALETIQUE.

Selon une technique similaire, nous pouvons affiner la structure des champs CLI-HISTORIQUE et COM-DETAIL. En revanche, l'examen de STK-LIBELLE ne fournit pas d'indice d'une décomposition pertinente, ce qui suggère qu'il s'agit d'un champ atomique. Nous obtenons alors le schéma physique de la Figure 8.

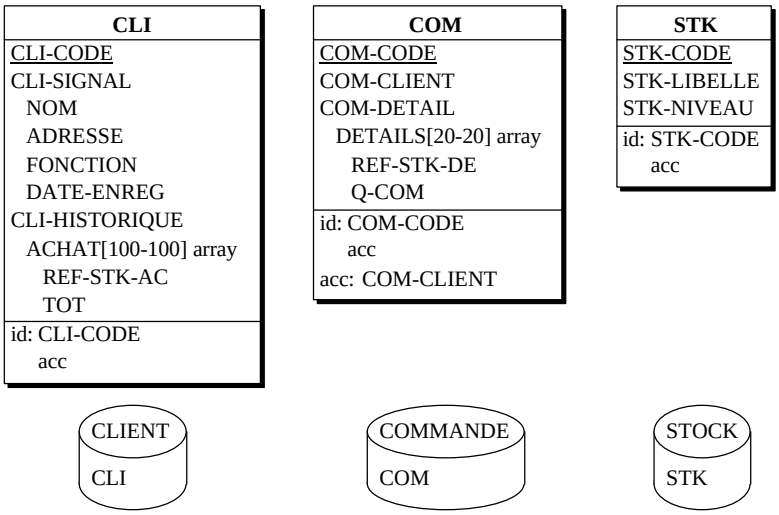

Figure 8 - Schéma physique : expression des champs composés.

i) Recherche des clés étrangères

Il existe certainement des liens entre ces trois fichers qui s'expriment sous la forme de clés étrangères implicites. Nous allons rechercher ces liens à l'aide de plusieurs techniques coordonnées.

L'examen du nom des champs peut donner une première indication. En effet le nom d'un champ de référence contient souvent, sous une forme plus ou moins explicite, l'identification du type d'enregistrements cible. Cette propriété nous fait repérer, par exemple, les noms REF-**STK**-AC (contient le nom d'un type d'enregistrement), REF-**STK**-DE (idem) et COM-**CLIENT** (contient le nom d'un fichier). Examinons de plus près le champ COM-**CLIENT**.

1. Comme nous venons de l'observer, son nom fait référence à un fichier contenant des enregistrements CLI.
2. Son type et sa longueur sont ceux de l'identifiant du type d'enregistrements CLI.
3. Il constitue un index, ce qui est fréquent pour les identifiants et les clés étrangères.
4. Il apparaît dans le diagramme de flux du programme en association avec la *record key* (index identifiant) du fichier CLIENT (Code 3).
5. L'analyse de l'instruction WRITE COM en [160] montre qu'un enregistrement COM n'est écrit dans le fichier COMMANDE qu'après qu'on ait vérifié que la valeur de COM-CLIENT correspond à un enregistrement existant dans le fichier CLIENT (Code 4).

```
MOVE CLI-CODE TO COM-CLIENT.                    [154]
```

Code 3 - Instruction d'assignation définissant le graphe de flux concernant le champ COM-CLIENT.

```
NOUV-COM.                                       [146]
   MOVE 1 TO END-FILE.                          [150]
   PERFORM LIRE-CODE-CLI UNTIL END-FILE = 0.    [151]
   MOVE CLI-CODE TO COM-CLIENT.                 [154]
   WRITE COM                                    [160]
     INVALID KEY DISPLAY "ERREUR".              [161]

LIRE-CODE-CLI.                                  [165]
   ACCEPT CLI-CODE.                             [167]
   MOVE 0 TO END-FILE.                          [168]
   READ CLIENT INVALID KEY                      [169]
      DISPLAY "CLIENT INEXITANT"                [170]
      MOVE 1 TO END-FILE                        [171]
   END-READ.                                    [172]
```

Code 4 - Fragment de programme (*program slice*) relatif à l'objet COM.COM-CLIENT au point [160]. Ce code illustre une validation suggérant une contrainte référentielle.

Sur base de ces cinq indices, nous pouvons en confiance conclure que COM-CLIENT doit être une clé étrangère vers CLI. Par des raisonnements similaires, nous ferons de COM-DETAIL.DETAILS.REF-STK-DE et CLI-HISTORIQUE.ACHAT.REF-STK-AC des clés étrangères vers STK, ce qui conduit au schéma physique augmenté de la Figure 10.

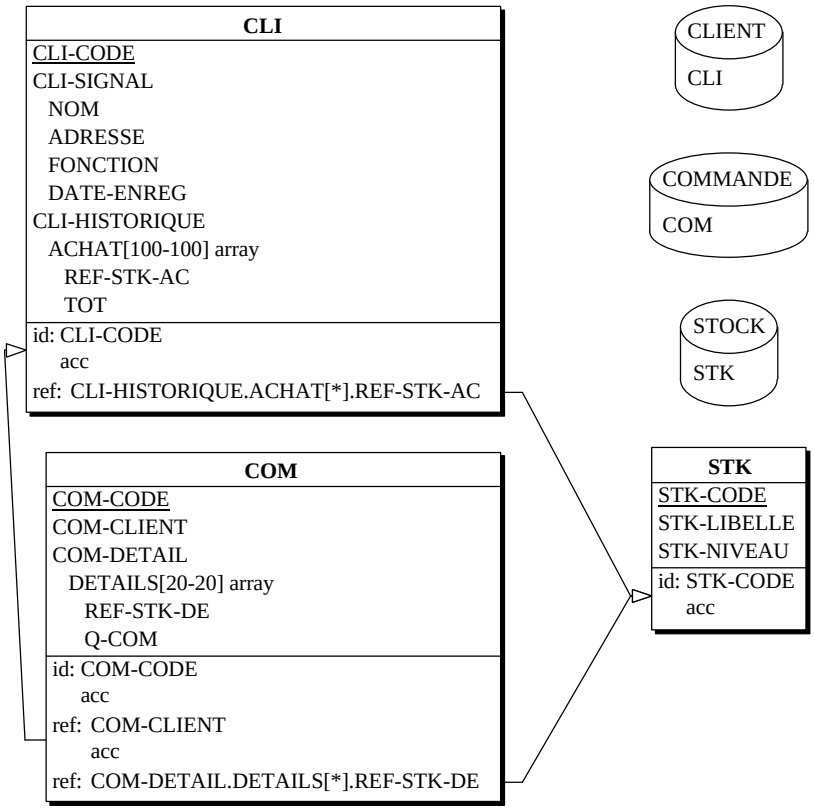

Figure 10 - Schéma physique : expression des clés étrangères.

k) Recherche des identifiants des champs multivalués

Le schéma physique contient deux champs multivalués composés, DETAILS et ACHAT. Il est fréquent que les valeurs de tels champs soient soumises à une contrainte d'unicité portant sur un de leurs composants.

En examinant la manière dont le programme lit et gère les valeurs de ces champs, nous pouvons tirer des renseignements utiles concernant la présence éventuelle d'identifiants locaux. Considérons par exemple le champ multivalué DETAILS et calculons le fragment de programme pour ce champ au point WRITE COM [160] (Code 5), qui devrait nous apprendre comment ce champ est garni préalablement à l'écriture de l'enregistrement dans le fichier.

Ce fragment montre clairement ([190] à [204]) qu'il ne peut exister deux éléments DETAILS de même valeur de REF-STK-DE. On en conclut donc que ce champ est un identifiant de COM-DETAIL.DETAILS.

On montrerait de la même manière que REF-STK-AC est un identifiant du champ multivalué CLI-HISTORIQUE.ACHAT. Ces découvertes sont incorporées au schéma physique courant (Figure 12).

```
ENTREE.                                          [059]
   PERFORM TRAITEMENT UNTIL CHOIX = 0.           [061]
TRAITEMENT.                                      [068]
   IF CHOIX = 3                                  [081]
      PERFORM NOUV-COM.                          [082]
NOUV-COM.                                        [146]
   SET IND-DET TO 1.                             [156]
   MOVE 1 TO END-FILE.                           [157]
   PERFORM LIRE-DETAIL
           UNTIL END-FILE = 0 OR IND-DET = 21.   [158]
   MOVE LIST-DETAIL TO COM-DETAIL.               [159]
   WRITE COM                                     [160]
LIRE-DETAIL.                                     [173]
   ACCEPT CODE-PROD.                             [175]
   IF CODE-PROD = "0"                            [176]
      MOVE 0 TO END-FILE                         [177]
      MOVE 0 TO REF-STK-DE(IND-DET)              [178]
   ELSE                                          [179]
      PERFORM LIRE-CODE-PROD.                    [180]
LIRE-CODE-PROD.                                  [181]
   MOVE 1 TO EXIST-PROD.                         [182]
   MOVE CODE-PROD TO STK-CODE.                   [183]
   READ STOCK INVALID KEY                        [184]
      MOVE 0 TO EXIST-PROD.                      [185]
   IF EXIST-PROD = 0                             [186]
      DISPLAY "PRODUIT INEXISTANT"               [187]
   ELSE                                          [188]
      PERFORM MAJ-COM-DETAIL.                    [189]
MAJ-COM-DETAIL.                                  [190]
   MOVE 1 TO NEXT-DET.                           [191]
   ACCEPT Q-COM(IND-DET).                        [193]
   PERFORM UNTIL                                 [194]
            REF-STK-DE(NEXT-DET) = CODE-PROD     [195]
         OR IND-DET = NEXT-DET                   [196]
      ADD 1 TO NEXT-DET                          [197]
   END-PERFORM.                                  [198]
   IF IND-DET = NEXT-DET                         [199]
      MOVE CODE-PROD TO REF-STK-DE(IND-DET)      [200]
      SET IND-DET UP BY 1                        [202]
   ELSE                                          [203]
      DISPLAY "ERREUR : PRODUIT DEJA COMMANDE".  [204]
```

Code 5 - Fragment de programme (*program slice*) relatif à l'objet COM.COM-DETAIL.DETAILS au point [160]. Ce code illustre une validation contrôlant l'unicité des valeurs de REF-STK-DE.

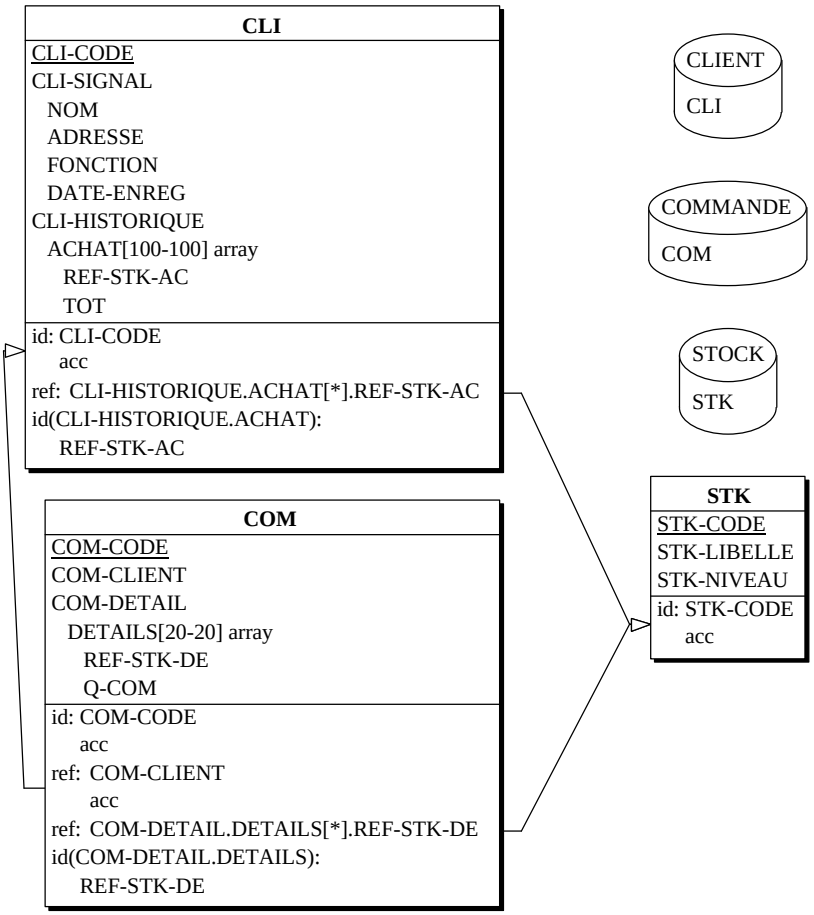

Figure 12 - Schéma physique : expression des identifiants des champs multivalués.

m)Détermination des cardinalités exactes des champs multivalués

Les champs multivalués ont été interprétés comme des tableaux[3] contenant un nombre fixe de valeurs, respectivement [20-20] et [100-100]. Ces bornes sont-elles effectivement d'application ?

Le fragment de programme Code 5 répond à cette question pour le champ DETAILS. On observe en effet que le tableau peut contenir jusqu'à 20 valeurs, mais que le programme accepte que le tableau soit vide. On en conclut que la cardinalité exacte de DETAILS est [0-20]. On montrerait de manière semblable que celle de ACHAT

3. En réalité, il s'agit de *unique arrays* (`u-array`), étant donné que chacun est doté d'un identifiant. Nous ignorerons cette spécification qui n'apporte rien du point de vue sémantique.

est [0-100]. Ces précisions sont ajoutées au schéma physique de la Figure 14, qui devient ainsi le schéma physique complet, malgré les réserves qu'on pourrait formuler sur la notion de *complétude*.

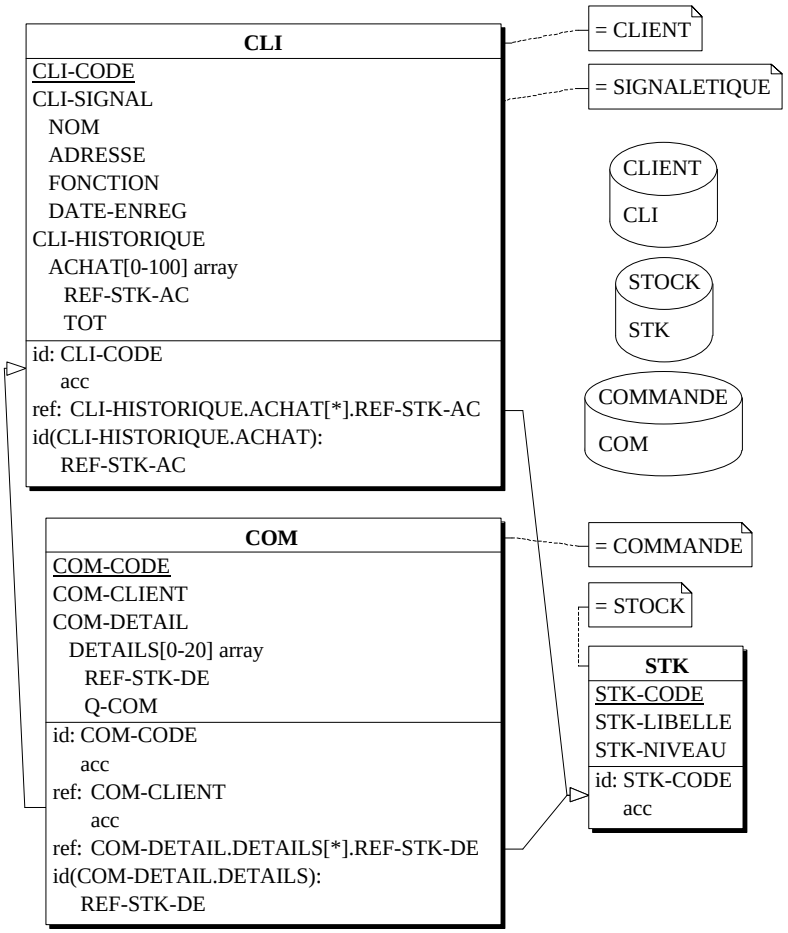

Figure 14 - Schéma physique affiné et schéma logique COBOL - Expression des cardinalités exactes et noms significatifs.

o) Recherche de noms significatifs

L'analyse des programmes et celle du schéma nous donne l'occasion d'affecter aux objets de celui-ci des noms plus évocateurs que leur dénimation d'origine. C'est ainsi que :

1. le nom de chaque fichier est plus significatif que celui du type des enregistrements qu'il contient : CLIENT pour CLI, STOCK pour STK et COMMANDE pour COM;

2. une variable qui reçoit, ou qui émet des valeurs d'un champ peut avoir un nom plus clair que celui du champ en question; tel est le cas de la variable SIGNALETIQUE, qui est en relation avec le champ CLI-SIGNAL.

Nous éviterons à ce stade d'effectuer la substitution, qui rendrait inopérant le schéma logique. Celui-ci peut en effet être utilisé pour maintenir les programmes et pour en développer d'autres. Il est donc important de conserver les noms d'origine. Nous nous contenterons pour l'instant d'*annoter* le schéma, laissant à la phase de conceptualisation le soin d'effectuer la modification des noms.

4.3 Extraction du schéma logique complet

En toute généralité, le schéma logique se déduit du schéma physique par l'élimination des constructions physiques non pertinentes au niveau logique. Cette notion dépend cependant du modèle selon lequel le schéma physique est exprimé. Dans notre cas le schéma relève du modèle COBOL, qui inclut les concepts d'index et de fichier au niveau logique[4]. Dans le cas d'une base de données SQL, il faudrait supprimer ces constructions pour obtenir le schéma logique.

Le schéma physique de la Figure 14 est donc aussi le schéma logique.

5 Conceptualisation des structures de données

Le schéma logique de la Figure 14 va être retravaillé pour produire un schéma conceptuel raisonnable.

5.1 Préparation du schéma logique

Le schéma logique comporte des traces des opérations d'extraction et d'affinage qui sont sans intérêt pour le recouvrement des structures conceptuelles et qui risquent d'obscurcir le travail ultérieur. Nous les éliminerons de manière à disposer d'un schéma clair et expressif[5]. Nous appliquerons trois techniques.

p) Traitement des noms

Préfixes non significatifs. Nous observons que les noms des champs de niveau supérieur sont systématiquement préfixés par celui de leur type d'enregistrement (**CLI**-SIGNAL). Il s'agit d'une pratique courante de formation de noms uniques en COBOL qui évite de devoir qualifier le nom des composants

4. Rappelons qu'on qualifie de logique un schéma qui contient les éléments nécessaires et suffisants pour que le programmeur puisse rédiger les programmes d'application. En COBOL, la connaissance des fichiers et des index est nécessaire, tandis qu'en SQL elle ne l'est pas.
5. Ce processus n'est pas étranger à celui de normalisation.

lorsqu'il apparaît dans un programme[6]. Ces préfixes n'apportant aucune information, on propose de les éliminer.

Noms significatifs. On effectue la substitutions des noms significatifs mis en évidence sous la forme d'annotations.

q) Transformation des structures anormales

La technique d'affinage de la structure des champs longs a créé deux champs composés qui ne possèdent qu'un seul composant, soient CLI-HISTORIQUE et COM-DETAIL . On propose de remplacer ces champs composés par cet unique composant.

r) Suppression des constructions physiques

Les clés d'accès (index) et les fichiers n'ont plus d'utilité à ce stade, et peuvent être supprimés du schéma logique.

Nous obtenons de la sorte un schéma logique préparé qui peut être conceptualisé (Figure 19).

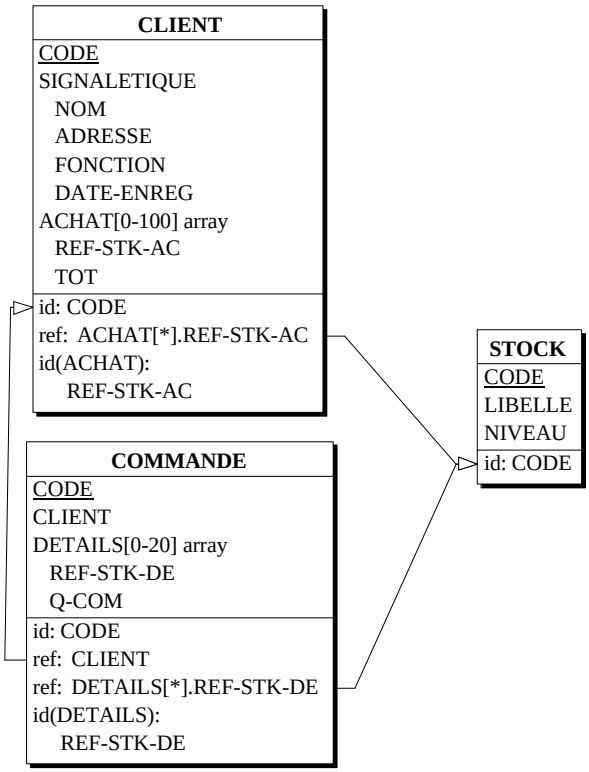

Figure 19 - Schéma logique nettoyé.

6. On peut écrire CLI-SIGNAL au lieu de CLI-SIGNAL of CLI si ce nom est unique, ce qui rend plus concises les instructions d'un langage généralement verbeux.

5.2 *Désoptimisation* et *détraduction* du schéma logique

Les processus de *désoptimisation* et de *détraduction* sont en général intimement liés, il est inutile de chercher à les distinguer sous la forme de processus séquentiels indépendants.

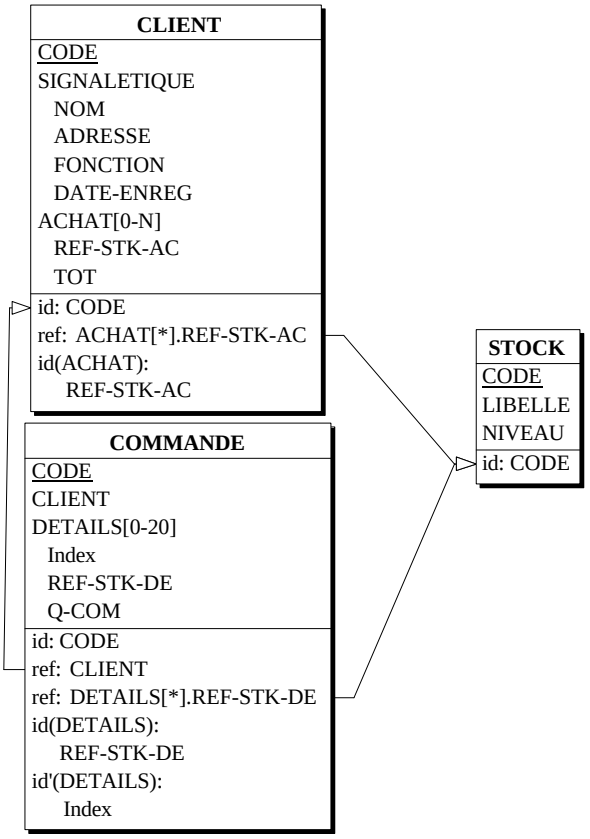

Figure 20 - Schéma partiellement conceptualisé : traitement des tableaux et des cardinalités.

Les tableaux. Le langage COBOL, à l'instar de la plupart des langages procéduraux, ne dispose que de la structure de *tableau* pour représenter explicitement des attributs multivalués. Il importe de préciser l'interprétation réelle des deux tableaux détectés dans les types d'enregistrements CLIENT et COMMANDE.

- ACHAT[0-100] : on admet que la structure d'ordre et le concept de *cellule vide*[7] sont sans signification; il s'agit donc d'un simple ensemble;

7. On rappelle qu'un tableau est une collection de cellules indexées pouvant contenir chacune une valeur. Le tableau a donc pour caractéristiques, par rapport à l'ensemble de valeurs : pas d'unicité, ordre et possibilité de *trous* dans la séquence (cellules vides).

- DETAILS[0-20] : la notion d'ordre semble importante mais pas celle de *cellule vide*, ce qui suggère de remplacer la structure de tableau par celle de liste. Cette dernière est exprimée sous une forme ensembliste par l'introduction d'un composant Index (Figure 20).

Les cardinalités. La cardinalité maximum de 100 de ACHAT n'a pas d'autre justification qu'une limite d'occupation d'espace dans la structure des enregistrements. On peut donc la considérer comme résultant d'une décision d'optimisation. On la remplace par N. En revanche, il apparaît que la limite de 20 de DETAILS a une justification organisationnelle (on évite les commandes trop longues, considérées comme anormales). On propose donc de la conserver (Figure 20).

Les attributs multivalués complexes. Les deux attributs complexes DETAIL et ACHAT correspondent à une implémentation typique de types d'entités dépendants. Cette structure permet de diminuer le nombre d'accès lorsqu'on accède à une commande et à tous ses détails. Il s'agit de structures d'optimisation qu'on traite par transformation en types d'entités DETAILS et ACHAT (Figure 21).

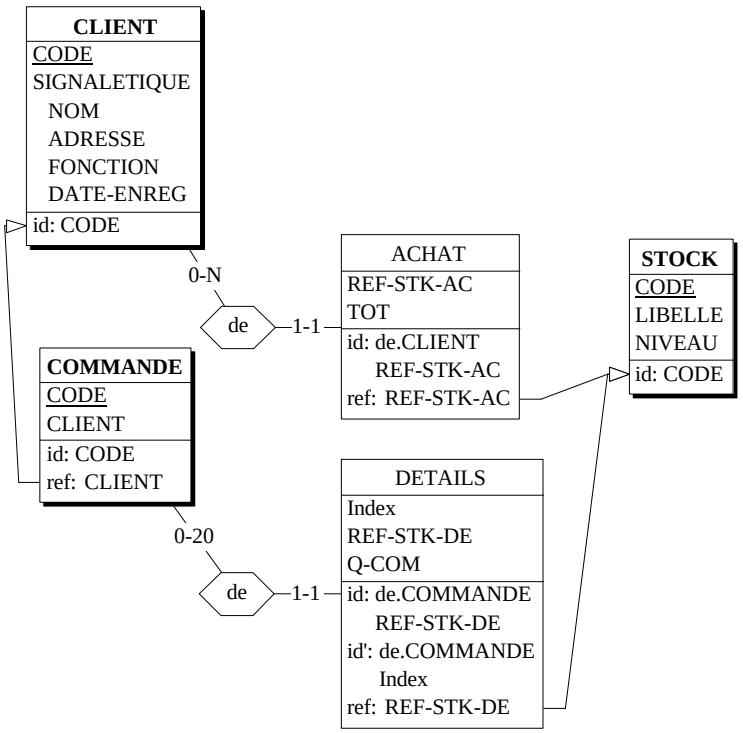

Figure 21 - Schéma partiellement conceptualisé : les attributs multivalués complexes sont transformés en types d'entités dépendants.

Les clés étrangères. Les clés étrangères sont remplacées par le type d'associations dont elle est l'implémentation. Ces clés étant mono-valuées et non identifiantes, les types d'associations résultants sont *un-à-plusieurs* (Figure 22).

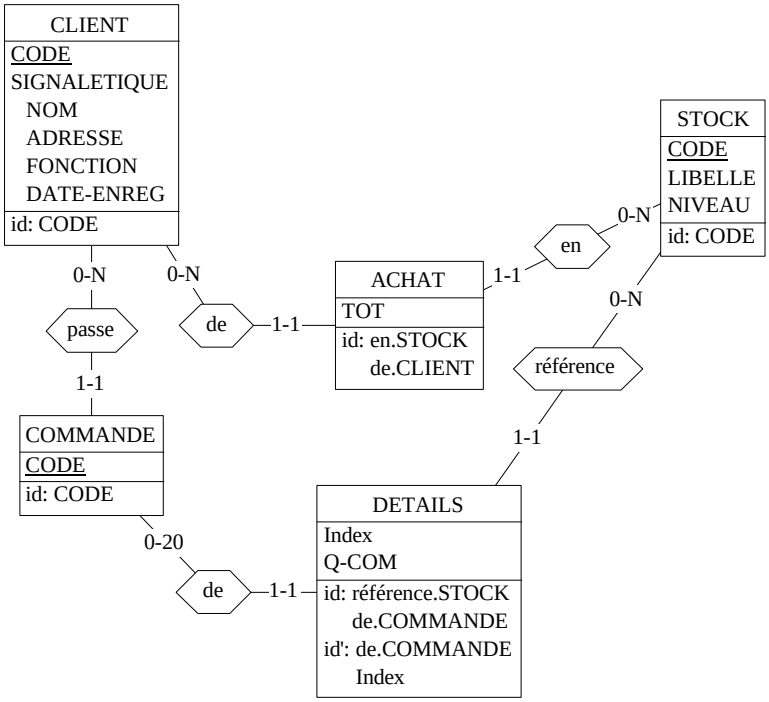

Figure 22 - Schéma conceptuel brut. Les clés étrangères sont transformées en types d'associations *un-à-plusieurs*.

5.3 Normalisation du schéma conceptuel

Nous tenterons à présent de rendre ce schéma plus lisible, de lui donner des qualités de concision, de minimalité, et d'expressivité, et d'éliminer les éventuelles constructions maladroites. Nous appliquerons quelques transformations qui nous semblent pertinentes.

Types d'entités associations. Les entités ACHAT et DETAILS pourraient être considérées comme jouant un rôle d'associations entre les entités CLIENT et STOCK d'une part et entre COMMANDE et STOCK d'autre part. On peut donc envisager de les réduire à de simples types d'associations. Nous le ferons pour ACHAT, mais pas pour DETAILS, qui nous semble plus complexe, et mieux exprimé sous sa forme actuelle.

Traitement des noms. Certains noms mériteraient d'être rendus plus expressifs. On pense en particulier à l'attribut Index, de nature technique, qu'on renomme en Num-Détail.

De même, l'attribut TOT peut être complété en TOTAL.

Enfin, les différents identifiants dénommés CODE pourraient recevoir des noms plus discriminants, tels que CODE-CLIENT, CODE-COMMANDE et CODE-STOCK.

Le terme DETAILS au pluriel est en contradiction avec l'usage généralisé du singulier. On le remplace par DETAIL.

Structures maladroites. L'attribut monovalué SIGNALETIQUE regroupe les informations signalétiques des clients. Ces informations étant, en dehors de l'identifiant, les seules présentes, ce regroupement ne se justifie pas. On désagrège cet attribut.

L'identifiant primaire de DETAIL est porteur de plus d'information, et donc moins stable, que l'identifiant secondaire. On suggère d'échanger leur statut.

Standards méthodologique et documentaire. Nous adopterons nos conventions habituelles de formation des noms, qui veulent qu'un nom d'attribut apparaisse en lettres capitalisé. Le schéma final normalisé est celui de la Figure 23.

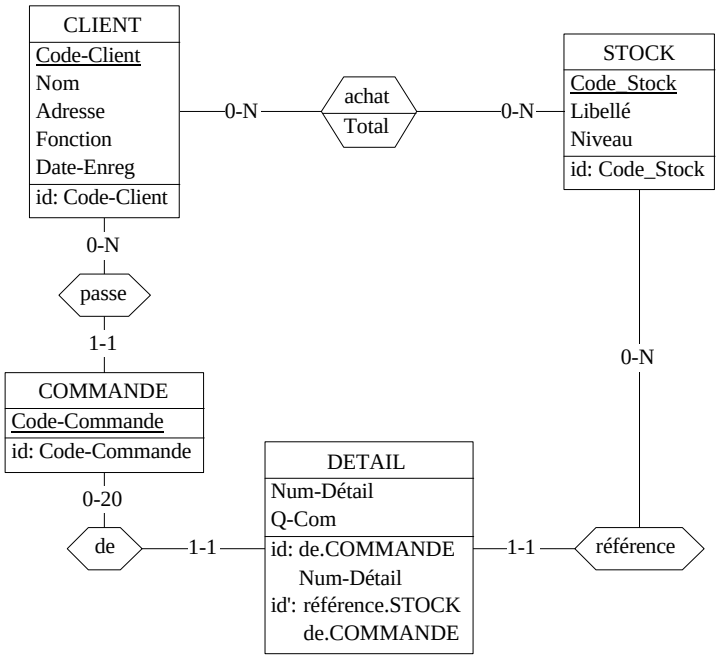

Figure 23 - Schéma conceptuel normalisé.

6 Production du schéma relationnel

Nous développerons une base de données de structure simple, dérivant le plus directement possible du schéma conceptuel. Nous veillerons cependant à produire un schéma physique qui respecte toutes les spécifications exprimées dans le schéma de la Figure 23.

6.1 Production du schéma logique relationnel

Le schéma conceptuel ne pose aucune problèmes particuliers, et se transforme aisément dans le schéma logique de la Figure 24. La contrainte de cardinalité [0-20] est représentée par une annotation.

Les noms sont convertis de la manière suivante :

- les accents sont enlevés,
- les minuscules sont converties en majuscule,
- les espaces et tirets sont remplacés par le signe '_',
- les noms correspondant à un mot réservé sont modifiés.

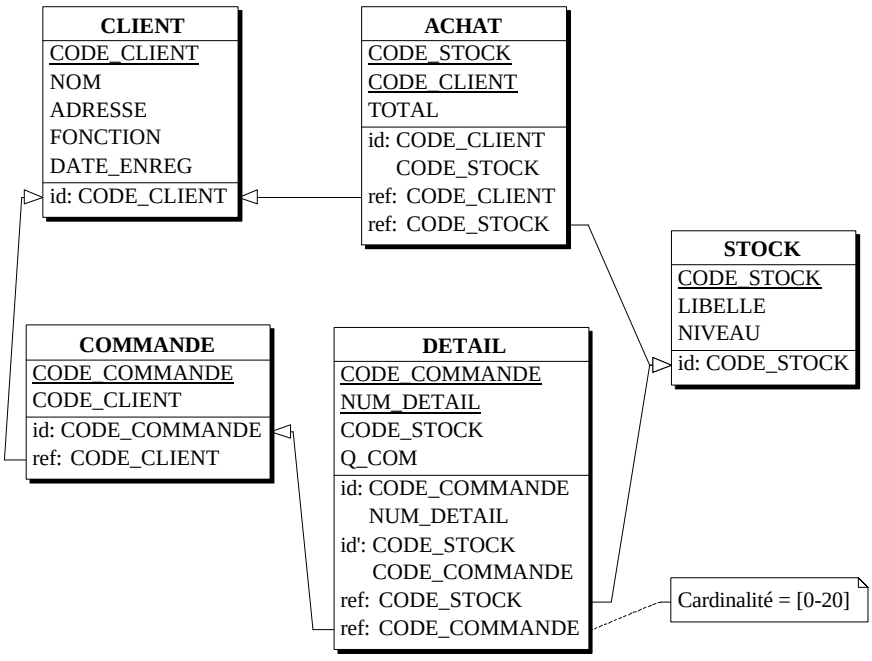

Figure 24 - Schéma logique relationnel.

6.2 Production du schéma physique relationnel

Cette phase se limitera à la définition des index, des espaces de stockage et des *triggers*. Les mécanismes d'accès doivent être au moins équivalents à ceux des structures des fichiers sources. Cependant, on tiendra également compte de besoins nouveaux. Nous veillerons à ne pas déclarer d'index inutiles.

y) Les index

On observera d'abord que les fichiers de données que nous avons analysés offrent des mécanismes d'accès qui correspondraient, dans le schéma relationnel, aux index suivants :

- CLIENT.CODE_CLIENT, correspondant à la *record key* du fichier CLIENT,
- COMMANDE.CODE_COMMANDE, correspondant à la *record key* du fichier COMMANDE,
- COMMANDE.CODE_CLIENT, correspondant à l'*alternate record key* du fichier COMMANDE,
- STOCK.CODE_STOCK, correspondant à la *record key* du fichier STOCK,
- DETAIL.CODE_COMMANDE, correspondant à l'inclusion de DETAILS dans COM,
- ACHAT.CODE_CLIENT, correspondant à l'inclusion de ACHAT dans CLI.

Ces index sont représentés à la Figure 26.

Pour compléter ce schéma, nous ferons l'hypothèse que les achats seront progressivement utilisés pour analyser les produits en stock, de sorte qu'un index sur ACHAT.CODE_STOCK devient également nécessaire. Nous admettrons également que l'accès à DETAIL à partir de STOCK prendra de plus en plus d'importance, ce qui se traduit par un index sur DETAIL.CODE_STOCK.

Outre ces index, nous définirons ceux qui sont nécessaires à la validation des identifiants. Il faudrait en principe définir trois index supplémentaires pour les identifiants d'ACHAT et de DETAIL. C'est ce que nous ferons pour les deux identifiants primaires. Quant à l'identifiant secondaire de DETAIL, nous vérifierons la propriété d'unicité qu'il induit via un système de *triggers*, plus léger qu'un index.

Il reste à éliminer les index préfixes, c'est-à-dire ceux dont les composants apparaissent en premier dans d'autres index. Ces index sont en effet inutiles, les index qui les englobent pouvant être utilisés lors de l'exécution des requêtes[8]. Ceci concerne ACHAT.CODE_CLIENT et DETAIL.CODE_COMMANDE que nous retirons du schéma.

Le jeu d'index proposé est illustré à la Figure 29.

8. Pour autant que l'index englobant soit implémenté par une technique de *B-tree*, qui définit un ordre trié des valeurs de l'index, et non par *hashing*.

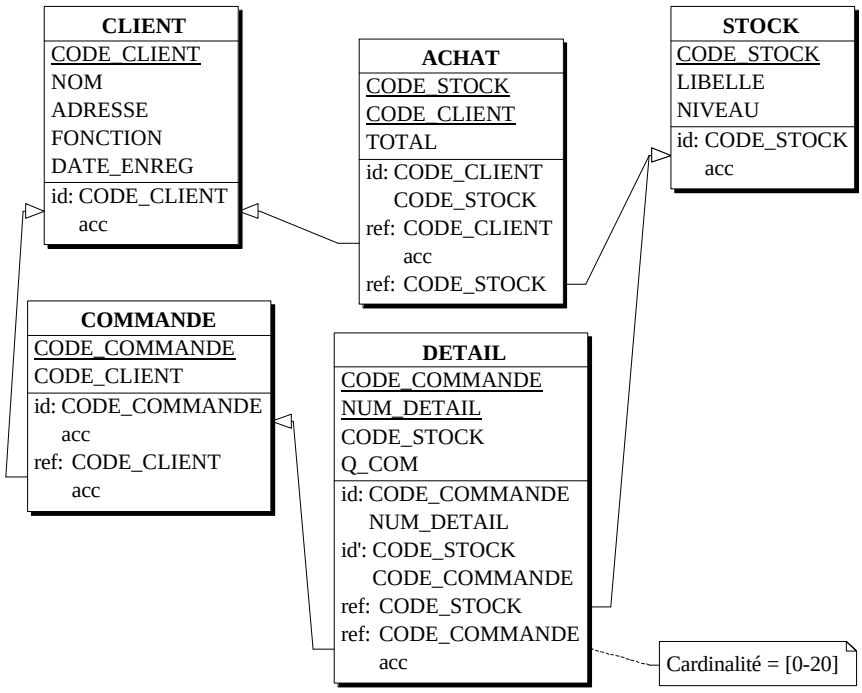

Figure 26 - Schéma physique - Première version (accès identiques à ceux des fichiers COBOL).

aa)Les espaces de stockage

Nous définirons trois espaces de stockage[9] : SP_CLIENT réservé à la table CLIENT, SP_STOCK contenant les tables STOCK et ACHAT, SP_COMMANDE rassemblant les tables COMMANDE et DETAIL. Ces décisions obéissent aux raisonnements suivants :

- la table CLIENT est isolée des autres en raison d'un profil d'exploitation spécifique (backup, profil de mise à jour, type de consultation, etc.);
- les tables COMMANDE et DETAIL sont naturellement fortement couplées, et peuvent faire l'objet d'une politique de stockage entrelacé (*clustering*) qui favorisera leur jointure;
- la table ACHAT a été placée dans l'espace de la table PRODUIT pour favoriser l'analyse des achats.

9. Terme générique désignant les conteneurs tels que les *table-spaces*, *db-spaces* et autres *spaces*.

ab)Les contraintes d'intégrité additionnelles

Le contrôle de l'identifiant secondaire de DETAIL et la contrainte de cardinalité [0-20] seront pris en charge par des *triggers*.

Identifiant DETAIL.{CODE_STOCK,CODE_COMMANDE}. Deux événements sont susceptibles de perturber cette contrainte d'unicité, l'insertion d'une ligne de DETAIL et la modification des colonnes CODE_STOCK ou CODE _COMMANDE.

Cardinalité [0-20] de la clé étrangère DETAIL.CODE_COMMANDE. Les deux événements concernés sont l'insertion d'une ligne de DETAIL et la modification de la colonne CODE_COMMANDE.

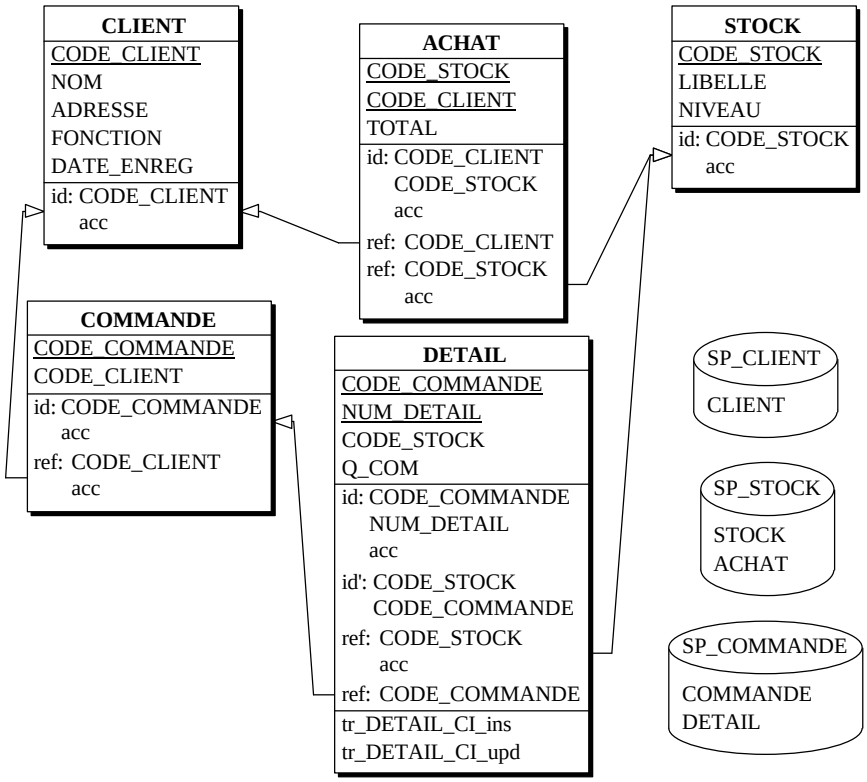

Figure 29 - Schéma physique relationnel final.

Ces contraintes peuvent être prises en charge par deux jeux de *triggers* indépendants, chacun dédié à une contrainte, ou par un seul jeu assurant le respect des deux contraintes simultanément. Nous choisirons la seconde solution, qui minimise le nombre de *triggers*. Nous définissons donc les deux *triggers* tr_DETAIL_CI_ins et

tr_DETAIL_CI_upd attachés à la table DETAIL et déclenchés respectivement par les événements insert et update.

Le schéma final est présenté à la Figure 29.

6.3 Génération du code SQL

Nous choisirons un stype de codage simple correspondant approximativement au standard SQL2.

ad)Définition des identifiants

Les identifiants primaires sont déclarés primary keys. L'identifiant secondaire de DETAIL sera géré par les *triggers* de contrôle des contraintes d'intégrité.

ae)Définition des clés étrangères

Afin d'éviter toute référence en avant, nous déclarerons les clés étrangères par des instructions de modification des tables sources (alter table add constraint).

af)Définition des autres contraintes

Comme décidé dans l'élaboration du schéma physique, nous coderons deux triggers gérant les deux contraintes d'intégrité additionnelles affectant la table DETAIL.

ag)Définition des index

En ce qui concerne les identifiants primaires, la plupart des SGBD exigent qu'ils soient supportés par des index, ce qui a été acté dans le schéma physique. Cependant, ils se différencient par le mode de déclaration. Pour certains, un index est automatiquement construit pour chaque identifiant primaire. Pour les autres, il est nécessaire de déclarer ces index explicitement. Nous ferons l'hypothèse que les index primaires sont construits automatiquement par le SGBD, comme c'est le cas pour Oracle par exemple.

Nous ne coderons donc pas les index identifiants assignés aux *primary keys*.

ah)Le code SQL-DDL

Le texte intégral du code est reporté à la Section 8.

7 *Complément* - Le code source du programme COBOL

```
IDENTIFICATION DIVISION.                                        001
PROGRAM-ID. CLIENT-COMMANDE.                                    002
ENVIRONMENT DIVISION.                                           003
INPUT-OUTPUT SECTION.                                           004

FILE-CONTROL.                                                   005
    SELECT CLIENT ASSIGN TO "CLIENT.DAT"                        006
        ORGANIZATION IS INDEXED                                 007
        ACCESS MODE IS DYNAMIC                                  008
        RECORD KEY IS CLI-CODE.                                 009

    SELECT COMMANDE ASSIGN TO "COMMANDE.DAT"                    010
        ORGANIZATION IS INDEXED                                 011
        ACCESS MODE IS DYNAMIC                                  012
        RECORD KEY IS COM-CODE                                  013
        ALTERNATE RECORD KEY IS COM-CLIENT                      014
        WITH DUPLICATES.                                        015

    SELECT STOCK ASSIGN TO "STOCK.DAT"                          016
        ORGANIZATION IS INDEXED                                 017
        ACCESS MODE IS DYNAMIC                                  018
        RECORD KEY IS STK-CODE.                                 019

DATA DIVISION.                                                  020
FILE SECTION.                                                   021
FD CLIENT.                                                      022
01 CLI.                                                         023
   02 CLI-CODE PIC X(12).                                       024
   02 CLI-SIGNAL PIC X(80).                                     025
   02 CLI-HISTORIQUE PIC X(1000).                               026

FD COMMANDE.                                                    027
01 COM.                                                         028
   02 COM-CODE PIC 9(10).                                       029
   02 COM-CLIENT PIC X(12).                                     030
   02 COM-DETAIL PIC X(200).                                    031

FD STOCK.                                                       032
01 STK.                                                         033
   02 STK-CODE PIC 9(5).                                        034
   02 STK-LIBELLE PIC X(100).                                   035
   02 STK-NIVEAU PIC 9(5).                                      036
```

```
WORKING-STORAGE SECTION.                                    037
01 SIGNALETIQUE.                                            038
   02 NOM PIC X(20).                                        039
   02 ADRESSE PIC X(40).                                    040
   02 FONCTION PIC X(10).                                   041
   02 DATE-ENREG PIC X(10).                                 042

01 LIST-ACHAT.                                              043
   02 ACHAT OCCURS 100 TIMES INDEXED BY IND.                 044
      03 REF-STK-AC PIC 9(5).                                045
      03 TOT PIC 9(5).                                       046

01 LIST-DETAIL.                                             047
   02 DETAILS OCCURS 20 TIMES INDEXED BY IND-DET.            048
      03 REF-STK-DE PIC 9(5).                                049
      03 Q-COM PIC 9(5).                                     050

01 CHOIX PIC X.                                             051
01 END-FILE PIC 9.                                          052
01 END-DETAIL PIC 9.                                        053
01 EXIST-PROD PIC 9.                                        054
01 CODE-PROD PIC 9(5).                                      055
01 QTE PIC 9(5) COMP.                                       056
01 NEXT-DET PIC 99.                                         057

PROCEDURE DIVISION.                                         058
ENTREE.                                                     059
   PERFORM INIT.                                            060
   PERFORM TRAITEMENT UNTIL CHOIX = 0.                      061
   PERFORM CLOTURE.                                         062
   STOP RUN.                                                063

INIT.                                                       064
   OPEN I-O CLIENT.                                         065
   OPEN I-O COMMANDE.                                       066
   OPEN I-O STOCK.                                          067

TRAITEMENT.                                                 068
   DISPLAY "1 NOUVEAU CLIENT".                              069
   DISPLAY "2 NOUVEAU STOCK".                               070
   DISPLAY "3 NOUVELLE COMMANDE".                           071
   DISPLAY "4 LISTE DES CLIENTS".                           072
   DISPLAY "5 LISTE DU STOCK".                              073
   DISPLAY "6 LISTE DES COMMANDES".                         074
   DISPLAY "0 FIN".                                         075
   ACCEPT CHOIX.                                            076

   IF CHOIX = 1                                             077
      PERFORM NOUV-CLI.                                     078
   IF CHOIX = 2                                             079
      PERFORM NOUV-STK.                                     080
   IF CHOIX = 3                                             081
      PERFORM NOUV-COM.                                     082
   IF CHOIX = 4                                             083
      PERFORM LIST-CLI.                                     084
   IF CHOIX = 5                                             085
      PERFORM LIST-STK.                                     086
   IF CHOIX = 6                                             087
      PERFORM LIST-COM.                                     088
```

```
CLOTURE.                                                        089
   CLOSE CLIENT.                                                090
   CLOSE COMMANDE.                                              091
   CLOSE STOCK.                                                 092

NOUV-CLI.                                                       093
   DISPLAY "NOUVEAU CLIENT :".                                  094
   DISPLAY "CODE DU CLIENT ?"  WITH NO ADVANCING.               095
   ACCEPT CLI-CODE.                                             096

   DISPLAY "NOM DU CLIENT : " WITH NO ADVANCING.                097
   ACCEPT NOM.                                                  098
   DISPLAY "ADRESSE DU CLIENT : " WITH NO ADVANCING.            099
   ACCEPT ADRESSE.                                              100
   DISPLAY "FONCTION DU CLIENT : " WITH NO ADVANCING.           101
   ACCEPT FONCTION.                                             102
   DISPLAY "DATE : " WITH NO ADVANCING.                         103
   ACCEPT DATE-ENREG.                                           104
   MOVE SIGNALETIQUE TO CLI-SIGNAL.                             105
   DISPLAY CLI-SIGNAL.                                          106
   PERFORM INIT-HISTO.                                          107
   WRITE CLI                                                    108
     INVALID KEY DISPLAY "ERREUR".                              109

LIST-CLI.                                                       110
   DISPLAY "LISTE DES CLIENTS".                                 111
   CLOSE CLIENT.                                                112
   OPEN I-O CLIENT.                                             113
   MOVE 1 TO END-FILE.                                          114
   PERFORM LIRE-CLI UNTIL (END-FILE = 0).                       115

LIRE-CLI.                                                       116
   READ CLIENT NEXT                                             117
      AT END MOVE 0 TO END-FILE                                 118
      NOT AT END                                                119
         DISPLAY CLI-CODE                                       120
         DISPLAY CLI-SIGNAL                                     121
         DISPLAY CLI-HISTORIQUE.                                122

NOUV-STK.                                                       123
   DISPLAY "NOUVEAU STOCK".                                     124
   DISPLAY "NUM PRODUIT : " WITH NO ADVANCING.                  125
   ACCEPT STK-CODE.                                             126

   DISPLAY "LIBELLE : " WITH NO ADVANCING.                      127
   ACCEPT STK-LIBELLE.                                          128

   DISPLAY "NIVEAU : " WITH NO ADVANCING.                       129
   ACCEPT STK-NIVEAU.                                           130

   WRITE STK                                                    131
     INVALID KEY DISPLAY "ERREUR ".                             132
```

```
LIST-STK.                                                          133
   DISPLAY "LISTE DU STOCK ".                                      134

   CLOSE STOCK.                                                    135
   OPEN I-O STOCK.                                                 136

   MOVE 1 TO END-FILE.                                             137
   PERFORM LIRE-STK UNTIL END-FILE = 0.                            138

LIRE-STK.                                                          139
   READ STOCK NEXT                                                 140
      AT END MOVE 0 TO END-FILE                                    141
      NOT AT END                                                   142
         DISPLAY STK-CODE                                          143
         DISPLAY STK-LIBELLE                                       144
         DISPLAY STK-NIVEAU.                                       145

NOUV-COM.                                                          146
   DISPLAY "NOUVELLE COMMANDE".                                    147
   DISPLAY "NUM COMMANDE : " WITH NO ADVANCING.                    148
   ACCEPT COM-CODE.                                                149

   MOVE 1 TO END-FILE.                                             150
   PERFORM LIRE-CODE-CLI UNTIL END-FILE = 0.                       151
   MOVE CLI-SIGNAL TO SIGNALETIQUE.                                152
   DISPLAY NOM.                                                    153
   MOVE CLI-CODE TO COM-CLIENT.                                    154
   MOVE CLI-HISTORIQUE TO LIST-ACHAT.                              155

   SET IND-DET TO 1.                                               156
   MOVE 1 TO END-FILE.                                             157
   PERFORM LIRE-DETAIL UNTIL END-FILE = 0 OR IND-DET = 21.         158
   MOVE LIST-DETAIL TO COM-DETAIL.                                 159

   WRITE COM                                                       160
     INVALID KEY DISPLAY "ERREUR".                                 161

   MOVE LIST-ACHAT TO CLI-HISTORIQUE.                              162
   REWRITE CLI                                                     163
     INVALID KEY DISPLAY "ERREUR CLI".                             164

LIRE-CODE-CLI.                                                     165
   DISPLAY "NUM DU CLIENT : " WITH NO ADVANCING.                   166
   ACCEPT CLI-CODE.                                                167
   MOVE 0 TO END-FILE.                                             168
   READ CLIENT INVALID KEY                                         169
      DISPLAY "CLIENT INEXITANT"                                   170
      MOVE 1 TO END-FILE                                           171
   END-READ.                                                       172

LIRE-DETAIL.                                                       173
   DISPLAY "CODE DU PRODUIT (0 = FIN) : ".                         174
   ACCEPT CODE-PROD.                                               175
   IF CODE-PROD = "0"                                              176
      MOVE 0 TO END-FILE                                           177
      MOVE 0 TO REF-STK-DE(IND-DET)                                178
   ELSE                                                            179
      PERFORM LIRE-CODE-PROD.                                      180
```

```
LIRE-CODE-PROD.                                                  181
   MOVE 1 TO EXIST-PROD.                                         182
   MOVE CODE-PROD TO STK-CODE.                                   183
   READ STOCK INVALID KEY                                        184
      MOVE 0 TO EXIST-PROD.                                      185
   IF EXIST-PROD = 0                                             186
      DISPLAY "PRODUIT INEXISTANT"                               187
   ELSE                                                          188
      PERFORM MAJ-COM-DETAIL.                                    189

MAJ-COM-DETAIL.                                                  190
   MOVE 1 TO NEXT-DET.                                           191
   DISPLAY "QUANTITE COMMANDEE : " WITH NO ADVANCING.            192
   ACCEPT Q-COM(IND-DET).                                        193
   PERFORM UNTIL                                                 194
            REF-STK-DE(NEXT-DET) = CODE-PROD                     195
         OR IND-DET = NEXT-DET                                   196
      ADD 1 TO NEXT-DET                                          197
   END-PERFORM.                                                  198
   IF IND-DET = NEXT-DET                                         199
      MOVE CODE-PROD TO REF-STK-DE(IND-DET)                      200
      PERFORM MAJ-CLI-HISTO                                      201
      SET IND-DET UP BY 1                                        202
   ELSE                                                          203
      DISPLAY "ERREUR : PRODUIT DEJA COMMANDE".                  204

MAJ-CLI-HISTO.                                                   205
   SET IND TO 1.                                                 206
   PERFORM UNTIL                                                 207
           REF-STK-AC(IND) = CODE-PROD                           208
        OR REF-STK-AC(IND) = 0 OR IND = 101                      209
      SET IND UP BY 1                                            210
   END-PERFORM.                                                  211
   IF IND = 101                                                  212
      DISPLAY "ERREUR : HISTORIQUE TROP PETIT"                   213
   EXIT.                                                         214
   IF REF-STK-AC(IND) = CODE-PROD                                215
      ADD Q-COM(IND-DET) TO TOT(IND)                             216
   ELSE                                                          217
      MOVE CODE-PROD TO REF-STK-AC(IND)                          218
      MOVE Q-COM(IND-DET) TO TOT(IND).                           219

LIST-COM.                                                        220
   DISPLAY "LISTE DES COMMANDES ".                               221
   CLOSE COMMANDE.                                               222
   OPEN I-O COMMANDE.                                            223
   MOVE 1 TO END-FILE.                                           224
   PERFORM LIRE-COM UNTIL END-FILE = 0.                          225
```

```
LIRE-COM.                                                          226
   READ COMMANDE NEXT                                              227
      AT END MOVE 0 TO END-FILE                                    228
      NOT AT END                                                   229
         DISPLAY "COM-CODE " WITH NO ADVANCING                     230
         DISPLAY COM-CODE                                          231
         DISPLAY "COM-CLIENT " WITH NO ADVANCING                   232
         DISPLAY COM-CLIENT                                        233
         DISPLAY "COM-DETAIL "                                     234
         MOVE COM-DETAIL TO LIST-DETAIL                            235
         SET IND-DET TO 1                                          236
         MOVE 1 TO END-DETAIL                                      237
         PERFORM AFFICH-DETAIL.                                    238

INIT-HISTO.                                                        239
   SET IND TO 1.                                                   240
   PERFORM UNTIL IND = 100                                         241
      MOVE 0 TO REF-STK-AC(IND)                                    242
      MOVE 0 TO TOT(IND)                                           243
      SET IND UP BY 1                                              244
   END-PERFORM.                                                    245
   MOVE LIST-ACHAT TO CLI-HISTORIQUE.                              246

AFFICH-DETAIL.                                                     247
   MOVE 0 TO IND-DET.                                              248
   IF IND-DET = 21                                                 249
      MOVE 0 TO END-DETAIL                                         250
      EXIT.                                                        251
   IF REF-STK-DE(IND-DET) = 0                                      252
      MOVE 0 TO END-DETAIL                                         253
   ELSE                                                            254
      DISPLAY REF-STK-DE(IND-DET)                                  255
      DISPLAY Q-COM(IND-DET)                                       256
      SET IND-DET UP BY 1.                                         257
   MOVE IND-DET TO IND.                                            258
   DISPLAY IND.                                                    259
```

8 Complément - Le code SQL-DDL de la base de données relationnelle

```
create database CLICOM;


create dbspace SP_COMMANDE;
create dbspace SP_STOCK;
create dbspace SP_CLIENT;


create table CLIENT (
     CODE_CLIENT   char(12) not null,
     NOM           char(20) not null,
     ADRESSE       char(40) not null,
     FONCTION      char(10) not null,
     DATE_ENREG    char(10) not null,
     primary key (CODE_CLIENT))
     in SP_CLIENT;

create table STOCK (
     CODE_STOCK    numeric(5) not null,
     LIBELLE       char(100)  not null,
     NIVEAU        numeric(5) not null,
     primary key (CODE_STOCK))
     in SP_STOCK;

create table COMMANDE (
     CODE_COMMANDE numeric(10) not null,
     CODE_CLIENT   char(12)    not null,
     primary key (CODE_COMMANDE))
     in SP_COMMANDE;

create table DETAIL (
     CODE_COMMANDE numeric(10) not null,
     NUM_DETAIL    numeric(2)  not null,
     CODE_STOCK    numeric(5)  not null,
     Q_COM         numeric(5)  not null,
     primary key (CODE_COMMANDE, NUM_DETAIL))
     in SP_COMMANDE;

create table ACHAT (
     CODE_STOCK   numeric(5) not null,
     CODE_CLIENT  char(12)   not null,
     TOTAL        numeric(5) not null,
     primary key (CODE_CLIENT, CODE_STOCK))
     in SP_STOCK;


alter table ACHAT add constraint FKACH_CLI
     foreign key (CODE_CLIENT) references CLIENT;

alter table ACHAT add constraint FKACH_STO
     foreign key (CODE_STOCK) references STOCK;
```

```
alter table COMMANDE add constraint FKPASSE
     foreign key (CODE_CLIENT) references CLIENT;

alter table DETAIL add constraint FKREFERENCE
     foreign key (CODE_STOCK) references STOCK;

alter table DETAIL add constraint FKDE|1
     foreign key (CODE_COMMANDE) references COMMANDE;


create index FKACH_STO
     on ACHAT (CODE_STOCK);

create index FKPASSE
     on COMMANDE (CODE_CLIENT);

create index FKREFERENCE
     on DETAIL (CODE_STOCK);

-- index implicite
-- create unique index CLI_CODE
--     on CLIENT (CODE_CLIENT);

-- index implicite
-- create unique index STK_CODE
--      on STOCK (CODE_STOCK);

-- index implicite
-- create unique index COM_CODE
--      on COMMANDE (CODE_COMMANDE);

-- index implicite
-- create unique index IDSEC
--      on DETAIL (CODE_COMMANDE, NUM_DETAIL);

-- index implicite
-- create unique index IDACHAT
--     on ACHAT (CODE_CLIENT, CODE_STOCK);


create exception DETAIL_CARD_20 "Erreur: pas plus de 20 DETAILs
       par COMMANDE";
create exception DETAIL_SEC_ID  "Erreur: {CODE_COMMANDE,
       CODE_STOCK} identifiant de DETAIL";
```

```
create trigger tr_DETAIL_CI_ins
before insert on DETAIL
for each row
declare N integer;
begin
   -- contrainte de cardinalite [0-20]
   select count(*) into N from DETAIL
   where  CODE_COMMANDE = new.CODE_COMMANDE;
   if (N = 20) then exception DETAIL_CARD_20;
   -- identifiant secondaire
   select count(*) into N from DETAIL
   where  CODE_COMMANDE = new.CODE_COMMANDE
   and    CODE_STOCK    = new.CODE_STOCK;
   if (N = 1)  then exception DETAIL_SEC_ID;
end;

create trigger tr_DETAIL_CI_upd
before update of CODE_COMMANDE,NUM_DETAIL,CODE_STOCK on DETAIL
for each row
declare N integer;
begin
   -- contrainte de cardinalite [0-20]
   if (new.CODE_COMMANDE <> old.CODE_COMMANDE)
   then begin
      select count(*) into N from DETAIL
      where  CODE_COMMANDE = new.CODE_COMMANDE;
      if (N = 20) then exception DETAIL_CARD_20;
   end;
   -- identifiant secondaire
   if (new.CODE_COMMANDE <> old.CODE_COMMANDE
       or new.CODE_STOCK <> old.CODE_STOCK)
   then begin
      select count(*) into N from DETAIL
      where  CODE_COMMANDE = new.CODE_COMMANDE
      and    CODE_STOCK    = new.CODE_STOCK;
      if (N = 1)  then exception DETAIL_SEC_ID;
   end;
end;
```