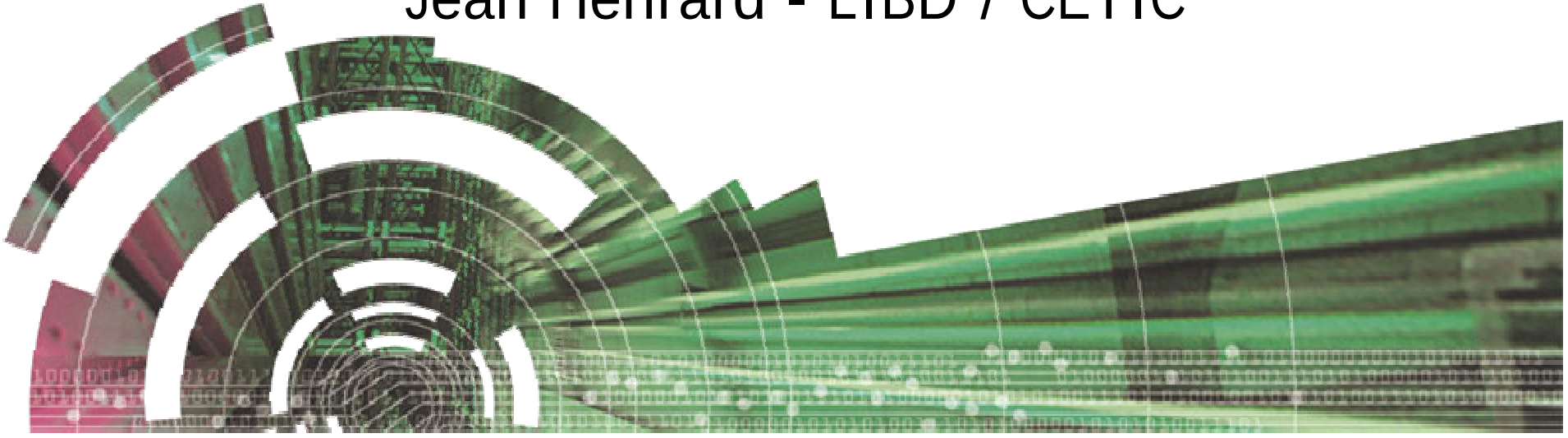
cetic

Your connection to ICT research

# Retrouvez la maîtrise de votre système d'information

Aspects techniques

Jean Henrard - LIBD / CETIC



cetic
Your connection to ICT research

# Plan

- Rétro-ingénierie
- Ré-ingénierie
- Gestion de projets

cetic
Your connection to ICT research

# Rétro-ingénierie

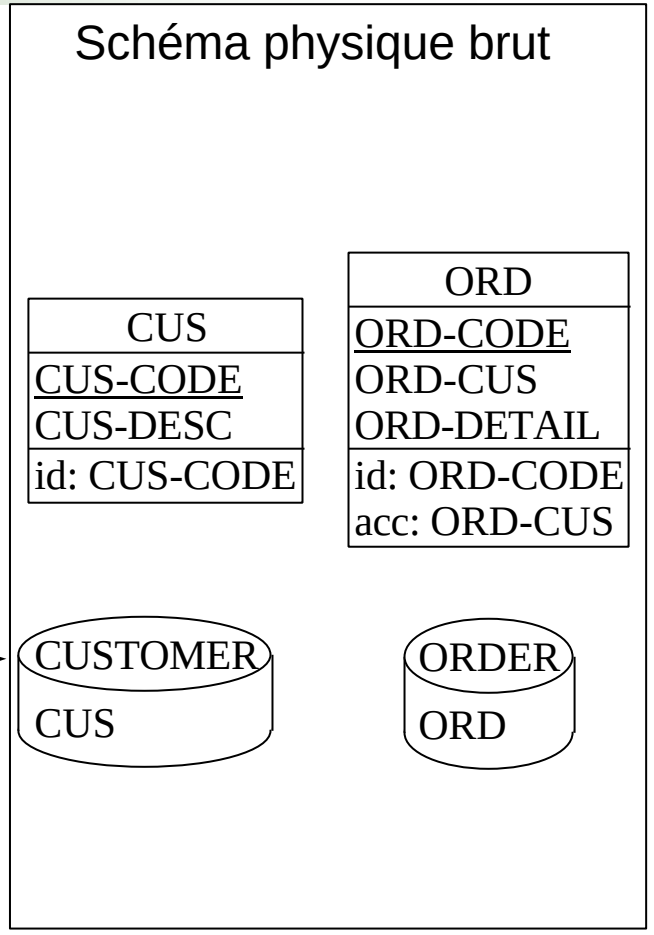
- Exemple
- Introduction
- Outils
  - Extracteurs
  - Compréhension de programmes
    - graphe de dépendance
    - fragmentation de programmes
  - Analyse de données
- Automatisation

# Un exemple

## Déclaration des structures de données

```
select CUSTOMER assign to "cust.dat"
          organisation is indexed
          record key is CUS-CODE.
select ORDER assign to "ord.dat"
          organisation is indexed
          record key is ORD-CODE
          alternate record key is ORD-CUS
          with duplicates.
...
FD CUSTOMER.
01 CUS.
          CUS-CODE pic X(12).
          CUS-DESC pic X(80).
FD ORDER.
01 ORD.
          02 ORD-CODE PIC 9(10).
          02 ORD-CUS PIC X(12).
          02 ORD-DETAIL PIC X(200).
```

extraction des déclarations



# Un exemple

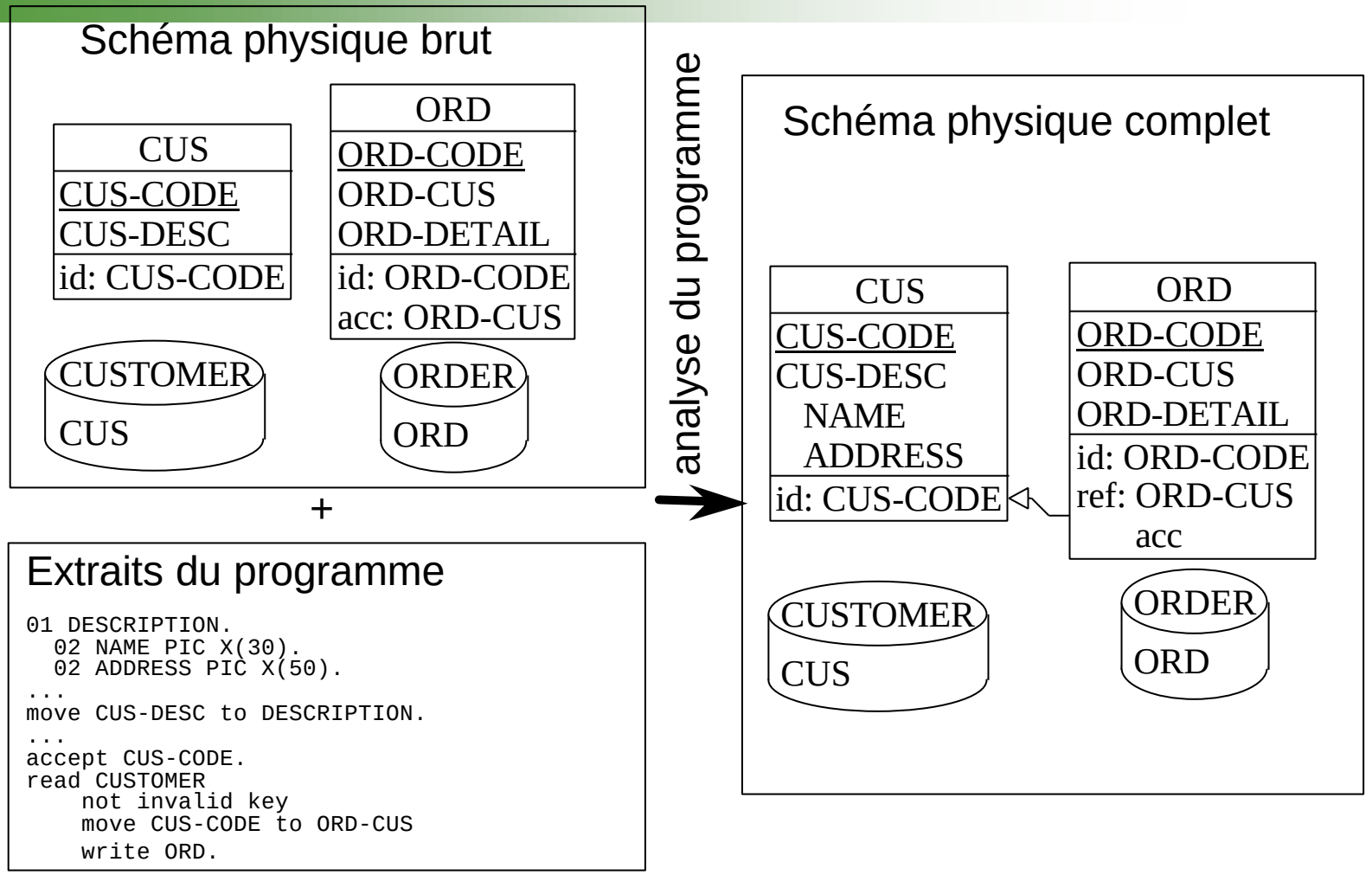


Schéma physique brut
CUS
CUS-CODE
CUS-DESC
id: CUS-CODE
ORD
ORD-CODE
ORD-CUS
ORD-DETAIL
id: ORD-CODE
acc: ORD-CUS
CUSTOMER
CUS
ORDER
ORD
+
Extraits du programme
01 DESCRIPTION.
02 NAME PIC X(30).
02 ADDRESS PIC X(50).
...
move CUS-DESC to DESCRIPTION.
...
accept CUS-CODE.
read CUSTOMER
not invalid key
move CUS-CODE to ORD-CUS
write ORD.
analyse du programme
Schéma physique complet
CUS
CUS-CODE
CUS-DESC
NAME
ADDRESS
id: CUS-CODE
ORD
ORD-CODE
ORD-CUS
ORD-DETAIL
id: ORD-CODE
ref: ORD-CUS
acc
CUSTOMER
CUS
ORDER
ORD

# Un exemple

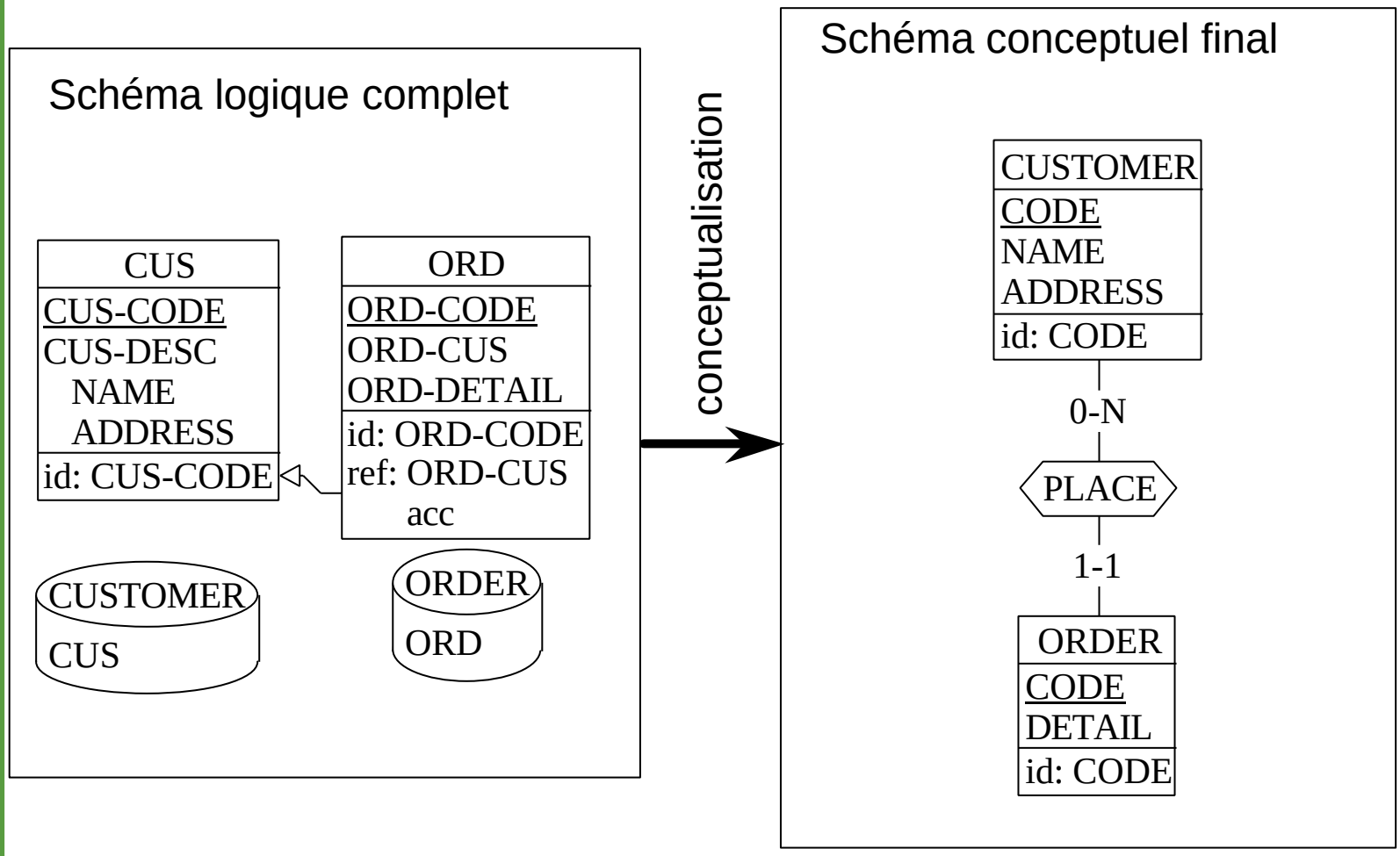


Schéma logique complet
CUS
CUS-CODE
CUS-DESC
NAME
ADDRESS
id: CUS-CODE
ORD
ORD-CODE
ORD-CUS
ORD-DETAIL
id: ORD-CODE
ref: ORD-CUS
acc
CUSTOMER
CUS
ORDER
ORD
conceptualisation
Schéma conceptuel final
CUSTOMER
CODE
NAME
ADDRESS
id: CODE
0-N
PLACE
1-1
ORDER
CODE
DETAIL
id: CODE

cetic
Your connection to ICT research

# Introduction

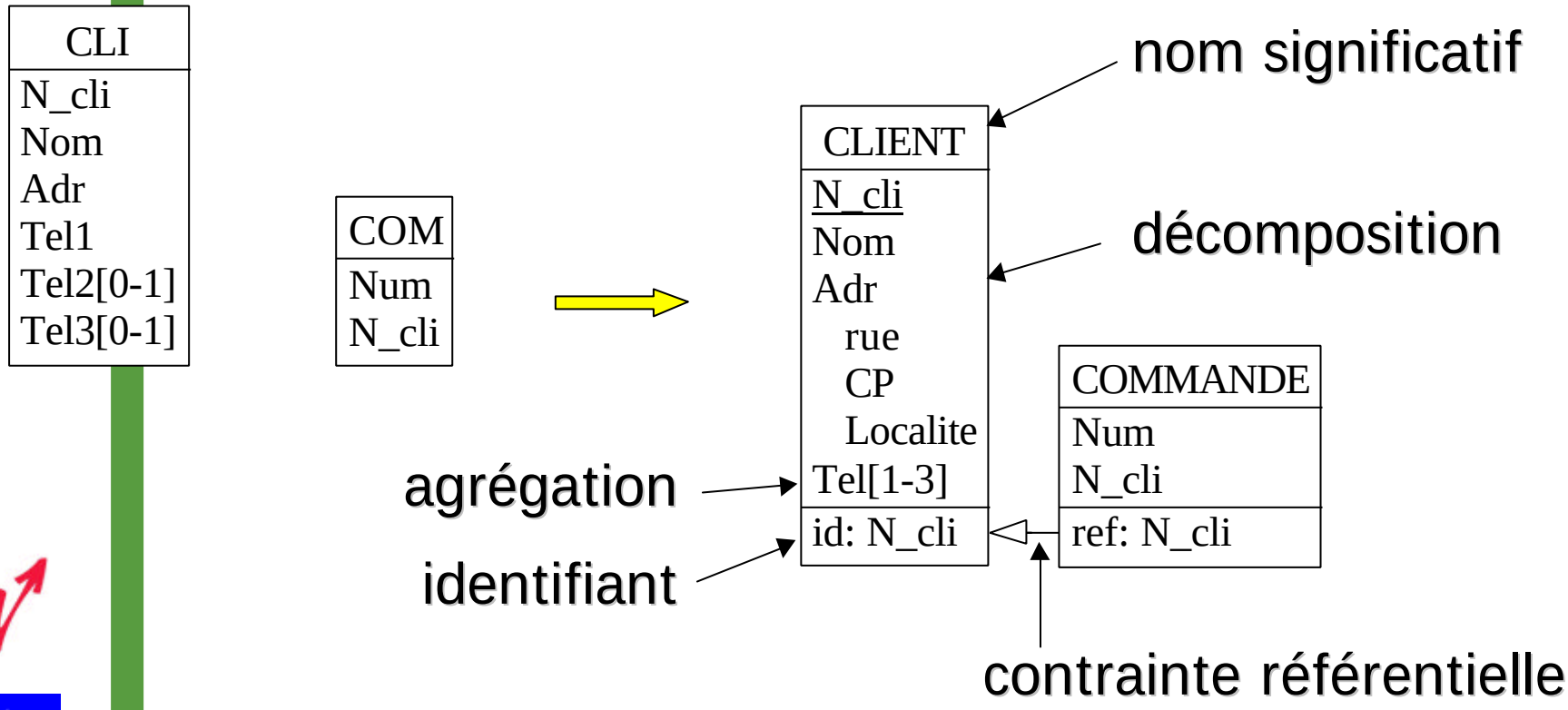
- La partie déclarative du code est facilement analysable de façon automatique
  = *structures explicites*
- Une part plus ou moins importante des contraintes est codée dans la partie procédurale du code (entre autres)
  = structures implicites

# Introduction

- La recherche des structures implicites :
  - fastidieuse, difficile et non déterministe
  - les contraintes sont disséminées dans tout le programme
  - chaque programmeur à son (ses) propre(s) style(s) de programmation
  - dépend de l'expertise de l'analyste et des outils disponibles
- Nécessité de techniques et d'outils de compréhension de programmes

# Introduction

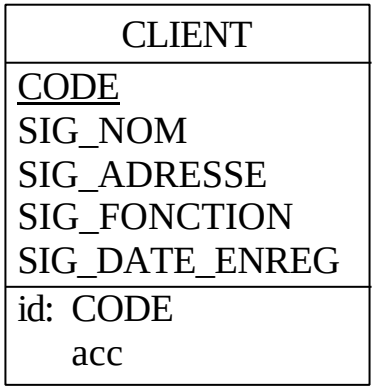
- Structures implicites recherchées :



# Outils : les extracteurs

- Analyse automatique du code DDL (Data Description Language)
- Produit un premier schéma physique

```
create table CLIENT (
     CODE char(12) not null ,
     SIG_NOM char(20) not null ,
     SIG_ADRESSE char(40) not null ,
     SIG_FONCTION char(10) not null ,
     SIG_DATE_ENREG char(10) not null ,
     primary key (CODE))
     in CLI_SPACE;
```



# Outils : compréhension de programmes

- Techniques de compréhension de programmes
  - calcul du graphe de dépendance des variables
  - fragmentation de programmes (program slicing)

# Outils : Graphe de dépendance

- Le *graphe de dépendance* est un graphe où chaque variable du programme est représentée par un nœud et dont les arcs représentent une relation entre deux variables
- S'il existe un chemin entre deux variables, il est possible que certaines propriétés structurelles ou sémantiques soient communes aux deux variables.

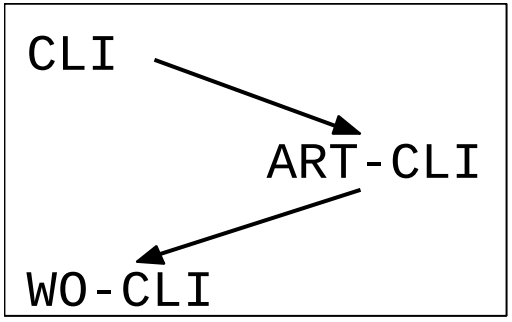
# Outils : graphe de dépendance

- Exemple

```
FD CLIENT.
01 CLI PIC X(100)
...
01 ART-CLI.
 02 NOM PIC X(50).
 02 FILLER PIC X(50).
...
01 WO-CLI.
 02 NOM PIC X(30).
 02 PRE PIC X(20).
 02 ADRESSE PIC X(50).
...
MOVE CLI TO ART-CLI.
      ...
MOVE ART-CLI TO WO-CLI.
```

Le pattern définissant la relation

```
move::="MOVE" - @var_1
       - "TO" - @var_2;
```



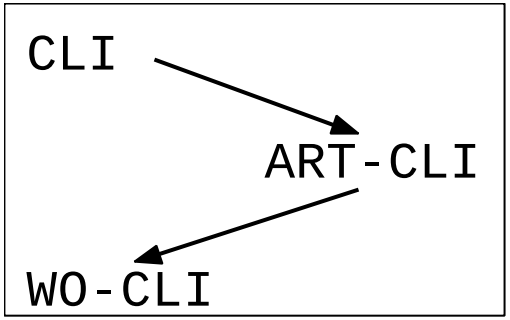
# Outils : graphe de dépendance

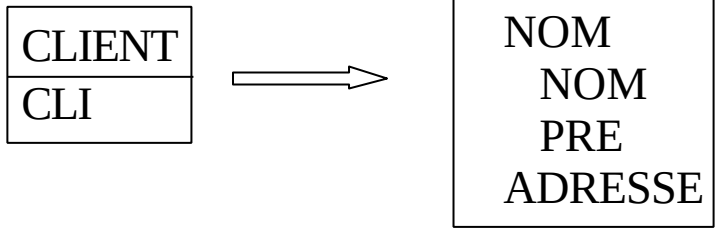
- Utilisation : composition, agrégation, recherche de clés étrangères
- Les relations entre les variables sont définies grâce à des patterns qui contiennent deux variables
- Visualisation du graphe dans le texte source

# Outils : graphe de dépendance

- Exemple

```
FD CLIENT.
01 CLI PIC X(100)
...
01 ART-CLI.
 02 NOM PIC X(50).
 02 FILLER PIC X(50).
...
01 WO-CLI.
 02 NOM PIC X(30).
 02 PRE PIC X(20).
 02 ADRESSE PIC X(50).
...
MOVE CLI TO ART-CLI.
      ...
MOVE ART-CLI TO WO-CLI.
```



# Outils : fragmentation de programmes

- Le *fragment de programme* (program slice) par rapport au point p et à la variable x est constitué de toutes les instructions du programme qui peuvent affecter la valeur de x au point p (M. Weiser 1984)
- Extrait le fragment nécessaire et suffisant pour comprendre et expliquer le comportement du programme en un point déterminé.

# Outils : fragmentation de programmes

```
64   NOUV-COM.
65     DISPLAY "NUM COMMANDE:"
                 WITH NO ADVANCING.
66     ACCEPT COM-CODE.
67
68     MOVE 1 TO FIN-FICHIER.
69     PERFORM LECT-CODE-CLI
                 UNTIL FIN-FICHIER=0.
70
71      DISPLAY CLI-NOM.
72
73     MOVE CLI-CODE TO COM-CLIENT.
74     SET IND-DET TO 1.
75     MOVE 1 TO FIN-FICHIER.
76     PERFORM LECT-DETAIL
         UNTIL FIN-FICHIER=0
         OR IND-DET = 21.
77     MOVE LISTE-DETAIL
                 TO COM-DETAIL.
78
79     WRITE COM
80      INVALID KEY
                 DISPLAY "ERREUR".
82   LECT-CODE-CLI.
83     DISPLAY "NUM DU CLIENT:"
                 WITH NO ADVANCING.
84     ACCEPT CLI-CODE.
85     MOVE 0 TO FIN-FICHIER.
86     READ CLIENT INVALID KEY
87       DISPLAY "CLIENT INEXITANT"
88       MOVE 1 TO FIN-FICHIER
89     END-READ.
```

Fragment par rapport à la ligne 71 et à CLI-NOM

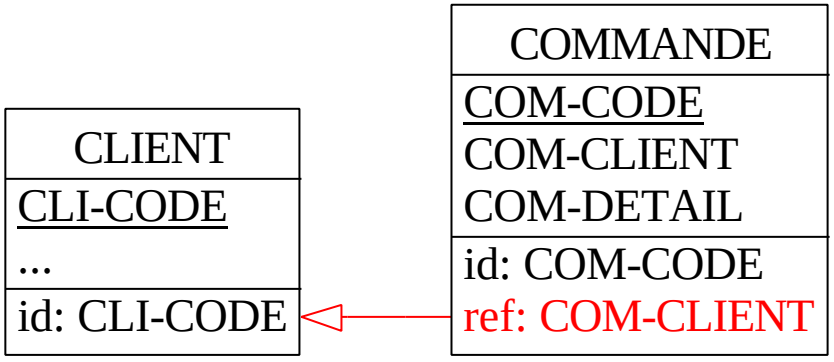
# Outils : fragmentation de programmes

- Utilisation : recherche de clés étrangères et de dépendances fonctionnelles
- Spécifique à un language
- Précis car connaît la sémantique du language
- Visualisation du fragment dans le texte source

# Outils : fragmentation de programmes

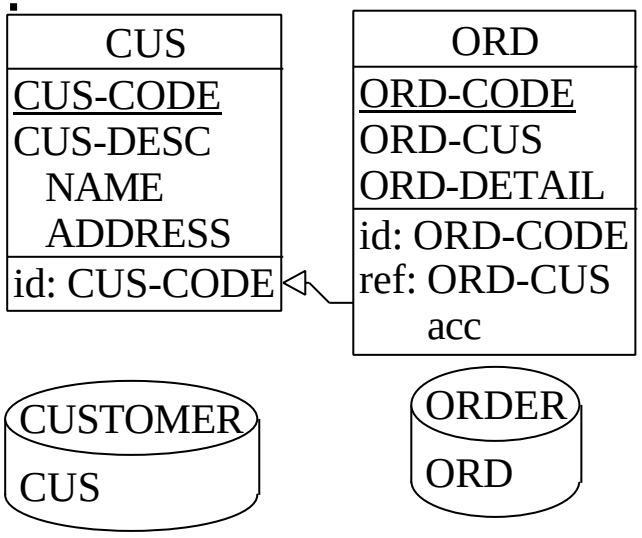
- Exemple

```
171  NOUV-COM.
176    MOVE 1 TO END-FILE.
177    PERFORM READ-CODE-CLI UNTIL END-FILE = 0.
178    MOVE CLI-SIGNAL TO SIGNALETIQUE.
180    MOVE CLI-CODE TO COM-CLIENT.
...
188    WRITE COM
189      INVALID KEY DISPLAY "ERREUR".
...
195  READ-CODE-CLI.
196    DISPLAY "NUM DU CLIENT : " WITH NO ADVANCING.
197    ACCEPT CLI-CODE.
198    MOVE 0 TO END-FILE.
199    READ CLIENT INVALID KEY
200      DISPLAY "CLIENT INEXITANT"
201      MOVE 1 TO END-FILE
202    END-READ.
```



# Outils : analyse de données

- Surtout utilisé pour valider un schéma
- Exécution de requêtes qui vérifient si les données respectent les contraintes du schéma
- Ex :




```
Select count(*)
from ORD
where ORD-CUS not in
    (select CUS-CODE
     from CUS)
```

# Automatisation (1/2)

- Extraction de DDL : des extracteurs analysent le code DDL pour produire (automatiquement) le schéma physique
- Analyse de données : génération automatique des requêtes (SQL) ou des programmes (COBOL) qui vérifient si les données sont conformes au schéma

# Automatisation (2/2)

- La fragmentation de programme est utilisée pour générer un tableau contenant les couples de champs qui sont en relations (suspicion de clés étrangères ou de dépendance fonctionnelles).

| Cible | | Origine | |
|---|---|---|---|
| CLIENT | CLI-CODE, | COMMANDE | COM-CLIENT, |
| CLIENT | CLI-CODE, | CLIENT | CLI-CODE, |
| CLIENT | CLI-HISTORIQUE, | STOCK | STK-CODE, |
| COMMANDE | COM-CLIENT, | CLIENT | CLI-CODE, |
| COMMANDE | COM-DETAIL, | STOCK | STK-CODE, |
| STOCK | STK-CODE, | CLIENT | CLI-HISTORIQUE, |
| STOCK | STK-CODE, | COMMANDE | COM-DETAIL, |

# Ré-ingénierie

- Problem statement
- Stratégies
- Stratégie « Statement rewritting »
- Stratégie « Wrapper »

# Problem statement

- Ré-ingénierie des données =

  dériver une nouvelle base de données d'une base de données ancienne et adapter les composants logiciels
  - Les fonctionnalités du système ne changent pas
- Trois étapes :
  - conversion du schéma
  - conversion des données
  - modification des programmes

# Problem statement

- Conversion du schéma
  - traduire l'ancien schéma en un schéma équivalent dans une nouvelle technologie
  - Rétro-ingénierie + conception de la nouvelle BD
- Conversion des données
  - migration des données de l'ancien système vers le nouveau
  - dépend de la conversion de schéma

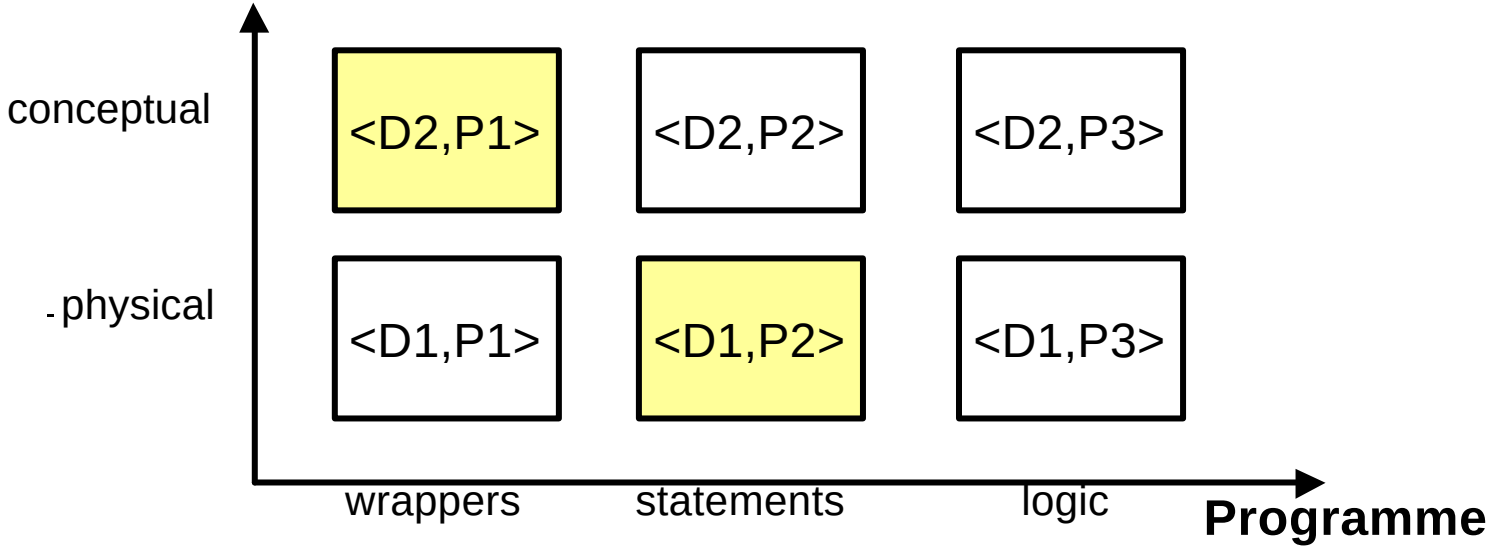
# Problem statement

- Modification des programmes
  - modification des programmes pour qu'ils accèdent à la nouvelle BD au lieu de l'ancienne
  - fonctionnalités, langage de programmation, interface utilisateur inchangés
  - processus complexe qui repose sur la conversion du schéma

# stratégies

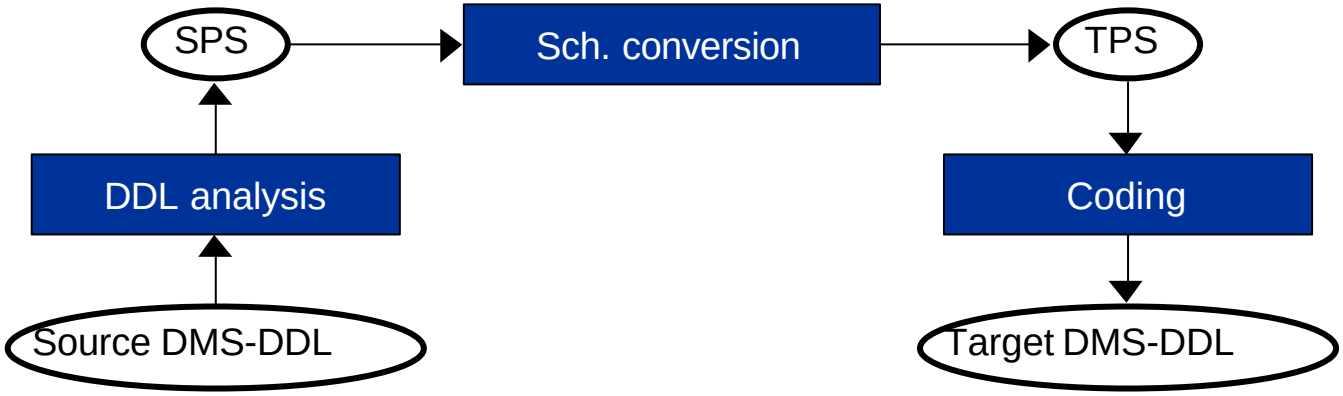
Six stratégies

**Base de données (schéma)**



# Statement rewriting strategy <D1,P2>

- *Physical conversion (D1):*
  - *traduit vers la structure la plus proche dans le SGBD cible*
    *(ex. 1 fichier Þ 1 table)*
  - *la sémantique de la BD n'est pas retrouvée*
  - *"facile" mais BD de mauvaise qualité*



# Statement rewriting strategy <D1,P2>

- Statement rewriting
  - DMS-DML ancien ⇒ DMS-DML cible
    ex: remplace l'accès aux fichiers COBOL par des instructions SQL
  - la modification du code est minimale
- Remarques
  - chaque instruction d'accès doit être localisée et remplacée par une (ou plusieurs) instructions équivalentes
  - possiblité d'automatiser le processus
  - Quick and dirty....

# Statement rewriting strategy <D1,P2>

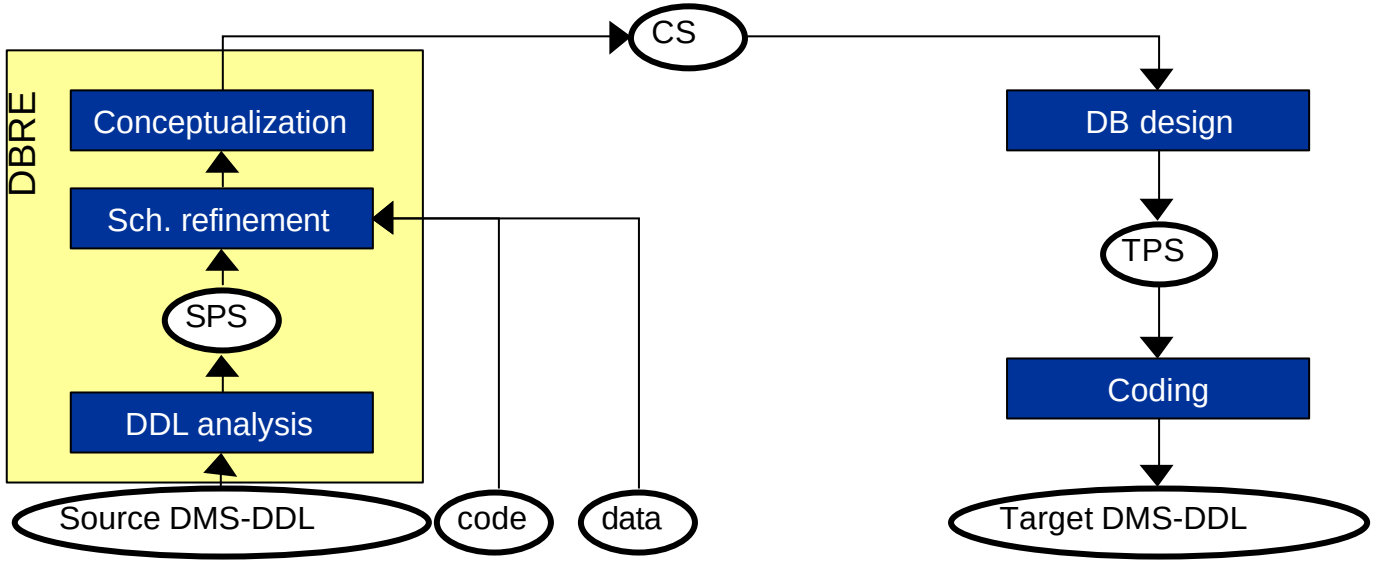
- Illustration ⇒

```
  MOVE CUS-CODE TO ORD-CUSTOMER.
  START ORDER KEY >= ORD-CUSTOMER.
  MOVE 0 TO END-FILE.
  PERFORM READ-ORD UNTIL END-FILE = 1.
READ-ORD SECTION.
BEG-ORD.
  READ ORDER NEXT

    AT END MOVE 1 TO END-FILE
           GO TO EXIT-ORD.
  <<processing current ORD record>>
EXIT-ORD.
  EXIT.
```

```
EXEC SQL declare cursor ORD_GE_K1 for
  select ORD_CODE,ORD_CUSTOMER,ORD_DETAIL
  from ORDER where ORD_CODE >= :ORD-CODE
  order by ORD_CODE END-EXEC.
  . . .
  EXEC SQL declare cursor ORD_GE_K2 for
  select ORD_CODE,ORD_CUSTOMER,ORD_DETAIL
  from ORDER where ORD_CUSTOMER >= :ORD-CUSTOMER
  ORDER BY ORD_CUSTOMER END-EXEC.
  ...
  MOVE CUS-CODE TO ORD-CUSTOMER.
  EXEC SQL open ORD_GE_K2 END-EXEC.
  MOVE "ORD_GE_K2" to ORD-SEQ.

  IF ORD-SEQ = "ORD_GE_K1"
    EXEC SQL fetch ORD_GE_K1 into :ORD-CODE,
         :ORD-CUSTOMER,:ORD-DETAIL END-EXEC
    ELSE IF ORD-SEQ = "ORD_GE_K2"
      EXEC SQL fetch ORD_GE_K2 into :ORD-CODE,
           :ORD-CUSTOMER,:ORD-DETAIL END-EXEC
     ELSE IF ...
     END-IF.
  IF SQLCODE NOT = 0
     MOVE 1 TO END-FILE GO TO EXIT-ORD.
  <<processing current ORD record>>
```

# Wrapper strategy <D2,P1>

- *Conceptual conversion (D2)*
  - retrouver la sémantique (schéma conceptuel)
  - concevoir une nouvelle BD à partir du sch conceptuel
  - BD de bonne qualité et documentée mais coûteux



# Wrapper strategy <D2,P1>

- Wrapper
  - wrapper de données=
    - conversion du modèle de données
    - conversion de la sémantique
    - simulation des fonctionnalités
  - wrapper « inverse »: simule l'ancienne interface avec la nouvelle BD
  - ex: utilise read et write COBOL pour accéder à des données SQL
  - la modification du code est minimale,
  - bonne solution :
    la BD est d'abord migrée, ensuite ce sont les programmes...

# Wrapper strategy <D2,P1>

- Remarques
  - Projet InterDB : production automatique de wrappers
- Illustration ⇒

```
READ PRODUCT
     KEY IS PROD-CODE
     INVALID KEY
          GO TO ERR-123.
```

```
CALL WR-ORD-MNGMT
  USING "READKEY","PRODUCT",
        "PROD-CODE",
        PRODUCT,WR-STATE.
IF STATUS OF WR-STATE NOT= 0
    GO TO ERR-123.
```

```
DELETE PRODUCT
  END-DELETE.
```

```
CALL WR-ORD-MNGMT
  USING "DELETE ","PRODUCT",
        "",PRODUCT,WR-STATE.
```

# Gestion de projets

- Sensibilisation
  - Peu de personnes connaissent le terme « rétro-ingénierie » et les difficultés potentielles
  - Un projet de ré-ingénierie doit être supporté par la direction générale **et** l'équipe technique
    - Direction générale
      - projet stratégique qui ne rapporte pas directement
      - implique aussi une modification de l'organisation du travail
    - Equipe technique
      - participation active au projet
      - détient la connaissance du système actuel
      - devra maintenir le nouveau système

# Gestion de projets

- Coût
  - Projet long et coûteux
  - Beaucoup de travail manuel et développement d'outils spécifiques
  - Nécessite du personnel qualifié
    - environnement de développement de l'ancien système
    - domaine d 'application
    - expérience de développement (comment a-t-on développé le système)
    - méthodologie de ré-ingénierie
    - nouvel environnement de développement

# Gestion de projets

- Risque
  - Difficile de garantir le résultat
  - Dépend fortement de l'ancien système
  - Projet à haut risque
- Evaluation
  - Difficile d'évaluer le résultat de la rétro-ingénierie
    - Pour évaluer un schéma, il faut développer l'application qui utilise ce schéma !
    - Pour évaluer un développement classique, il « suffit » de tester le programme
  - Il faut évaluer le processus