COBOL

(COmmon Business Oriented Language)

- C'est vers la fin des années 1950 que le gouvernement américain, devant la diversité et l'incompatibilité des ordinateurs, langages machines et assembleurs, a demandé à des spécialistes de bâtir un langage commun, indépendant des machines, orienté vers les applications de gestion et compréhensible pour tous, informaticiens ou non.

- C'est ainsi que naquit le langage de programmation COBOL, basé sur l'anglais et structuré un peu comme une oeuvre littéraire sous forme de chapitres, sections, paragraphes et phrases avec verbes, mots et ponctuation.

- A l'origine, les ambitions pour COBOL étaient assez limitées. Il s'agissait essentiellement de pouvoir traiter de grands fichiers séquentiels, de les mettre à jour, de réaliser des calculs relativement simples afin d'éditer et d'imprimer des milliers de lignes d'états de paie, comptabilité, stock, etc.

- Le langage a évolué, COBOL est aujourd'hui capable de traiter des bases de données en accès direct, d'échanger des informations via des lignes de transmission, de générer automatiquement des états.

- Le langage COBOL était de loin le langage le plus employé des années 1960 à 80, et reste toujours en utilisation dans des grandes entreprises, notamment dans les institutions financières qui disposent d'une vaste bibliothèque d'applications COBOL

- En 2005, le Gartner Group estimait que **75% des données** du monde des affaires étaient traitées par des programmes en COBOL et que 15% des nouveaux programmes développés le seront dans ce langage.

  - **Gartner**, Inc., fondée en 1979, est une firme américaine de consulting et de recherche dans le domaine de la technologie. Elle mène des recherches, fournit des services de consultation, tient à jour différentes statistiques et maintient un service de nouvelles spécialisées.

Histoire des standards COBOL

- Le langage fut développé en moins de six mois de travail, et il est encore en utilisation quarante ans plus tard, après plusieurs révisions standardisées par l'ANSI (American National Standards Institute), dont
  - COBOL-68 (1968)
  - COBOL-74 (1974)
  - COBOL-85 (1985) qui témoigne d'un grand pas vers l'adoption de la programmation structurée par l'industrie informatique
  - COBOL 2002 introduit la programmation objet, le XML, etc.

- COBOL permet d'effectuer des traitements comptables du fait de ses capacités arithmétiques en virgule fixe, notamment pour les traitements par lot où il présente d'excellentes performances, à condition que les calculs soient très basiques.

- Mais, même si les évolutions de COBOL l'ont aujourd'hui doté de certains des outils fournis par les langages modernes (récursivité, allocation dynamique, objets, etc.), son usage reste dédié aux applications de gestion.

Structure d'un programme en COBOL

- Un programme comporte quatre divisions. La norme COBOL-85 ne rend obligatoire que la première.
- IDENTIFICATION DIVISION.
  - Contient des informations générales sur le programme (dont le nom).
- ENVIRONMENT DIVISION.
  - Contient des informations sur l'environnement (matériel et logiciel) dans lequel le programme s'exécute.
- DATA DIVISION.
  - Contient les descriptions de données (variables, fichiers, paramètres et parfois description d'écran).
- PROCEDURE DIVISION.
  - Contient la description des traitements effectués.
  - Chaque division est composée de 'sections', formées de 'paragraphes' composés de 'phrases' qui peuvent être des phrases impératives ou des clauses. Chaque phrase doit être terminée par un point.
- Les six premières colonnes de chaque ligne de programme sont considérées comme une zone de commentaire, servant autrefois à numéroter les cartes perforées (en cas de chute du paquet, il suffisait de les passer sur une trieuse pour reconstituer la version correcte du programme). La septième colonne contient un caractère de contrôle : espace pour les lignes actives, étoile pour les commentaires.
- La huitième colonne est le début des titres de paragraphes.

COBOL

1 - Les fichiers séquentiels

Introduction

- Un fichier est un ensemble de données qui peut être vu :
  - du point de vue informationnel : **fichier logique**
  - du point de vue de sa mémorisation et de sa gestion par le SGF : **fichier physique**

- Un **fichier logique** est un ensemble d'enregistrement le plus souvent de même type en général une *structure*

- Le SGF gère les **fichiers physiques** répertoriés dans un catalogue des fichiers.

Les fichiers séquentiels

- Les enregistrements :
  - de même taille
  - sont consécutifs sur le support
  - chaque enregistrement possède un unique enregistrement prédécesseur et un unique enregistrement successeur
  - dans l'ordre où les enregistrements ont été ajoutés dans le fichier.

- L'**accès séquentiel** consiste, pour accéder à un enregistrement, à accéder à tous ses prédécesseurs dans l'ordre où ils sont dans le fichier.

Trois organisations différentes

- **record sequential** : organisation séquentielle par **enregistrements** (données structurées) : l'organisation par défaut

- **line sequential** : organisation séquentielle par lignes qui est celle des fichiers texte des éditeurs

- **printer sequential** : organisation séquentielle directement adaptée à l'impression

La déclaration d'un fichier est constituée :

- d'une **phrase select** :
  caractéristiques du fichier physique
  - Assign
    - Exemple

- d'une **déclaration de niveau fd** :
  caractéristiques du fichier logique
  - FD
    - Exemple

Les Instructions globales sur un fichier

- *ouverture et création si nécessaire*
  - **open**
    - **input**
    - **output :** permet de remplacer l'ancien contenu du fichier
    - **i-o**
    - **Extend :** ouverture pour ajout d'info

- *fermeture du fichier*
  - **close**

    - *Exemple*

Instructions élémentaires sur un fichier

- *Lecture d'un article (ouverture en **input** ou **i-o**)*

read *id-fs* [**into** *id-var*]
[**end** inst-imper-1]
[**not end** inst-imper-2]
[**end-read**]

  - la clause 'end' détecte l'adresse de fin de fichier lors d'une tentative de lecture.
  - la clause 'into' affecte un exemplaire de l'article lu à *id-var*.
    - Exemple

Instructions élémentaires sur un fichier

- *Ecriture (séquentielle) d'un article (en **output** ou **extend**)*

  write*id-var-art*
  [**from** *id-var*]
  [**end-write**]

- *Réécriture (séquentielle) d'un article (en **i-o**)*

  rewrite *id-var-art*
  [**from** *id-var*]
  [**end-rewrite**]

  - La réécriture doit être précédée d'une lecture réussie de l'article.

```
program-id. pg-consult-f-produit.

file-control.
   select f-produit assign 'f-prod.dat' organization record
   sequential.

data division.

file section.
fd f-produit.
1 produit.
   2 code-produit pic x(7).
   2 designation pic x(40).
   2 prix-unitaire pic 999v99.
   2 stock pic 9(4).

working-storage section.
1 v-fin-f-produit pic x value 'N'.
88 fin-f-produit value 'O' false 'N'.

procedure division.
debut.
   open input f-produit
   read f-produit end set fin-f-produit to true end-read

perform until fin-f-produit
   perform mod-trait-produit
   read f-produit end set fin-f-produit to true end-read
end-perform
close f-produit
goback.

mod-trait-produit.
.....
end program pg-consult-f-produit.
```

Assignation du fichier logique au fichier physique et indication de l'organisation

Description d'un enregistrement

Déclaration d'un booléen permettant de gérer la fin de fichier

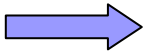
Ouverture du fichier

Une lecture en avance

Boucle permettant de traiter tous les enregistrements jusquà la fin de fichier

fermeture du fichier

Description du module de traitement d'un enregistrement



COBOL

2 - Les fichiers directs

Les fichiers directs : le fichier relatif

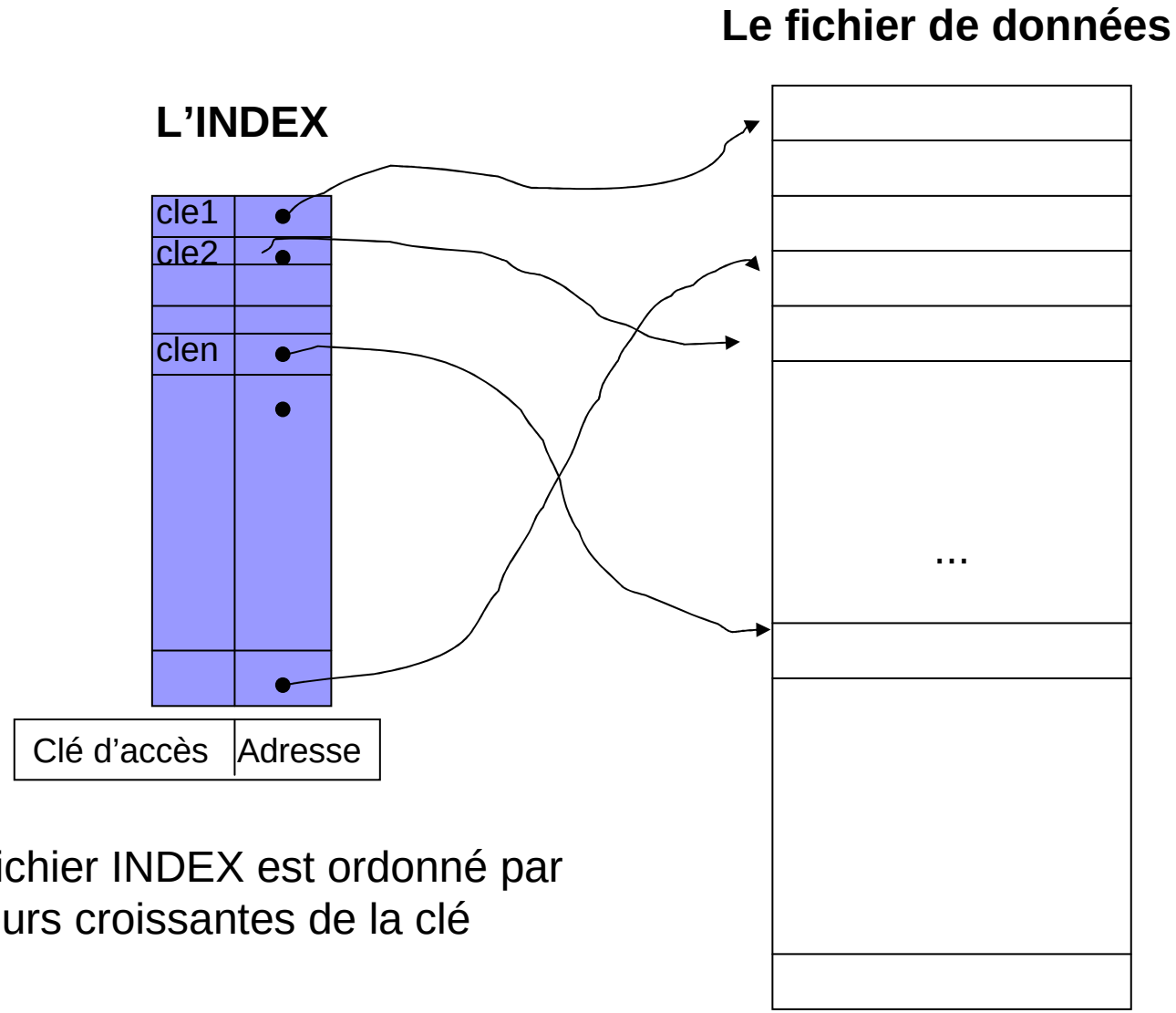
- Un fichier relatif est d'une suite d'emplacements ayant une adresse relative qui correspond à son rang (de 1 à n) dans le fichier : **La clé d'accès du fichier**
- Chaque emplacement est susceptible de contenir un enregistrement
  - Il existe un bit de chargement par emplacement
    - **chargé :** un enregistrement y a été écrit
    - '**non chargé**' : aucun enregistrement n'y a été écrit ou son contenu a fait l'objet d'un effacement.
- L'accès dans un fichier relatif :
  - l'***accès séquentiel*** est assuré dans l'ordre croissant des valeurs de la clé d'accès du fichier aux seuls emplacements chargés,
  - ***l'accès direct*** à un emplacement pour une valeur donnée de la clé d'accès du fichier et donc accès à l'enregistrement contenu s'il en existe un.

Les fichiers directs : le fichier indexé

- La clé d'accès du fichier est constituée d'un ou de plusieurs champs.

- Un fichier indexé est constitué de deux fichiers physiques :
  - le fichier des données, les enregistrements effectifs du fichier.
  - le fichier 'index' dont chaque élément est constitué :
    - d'une valeur de la clé d'accès
    - de l'adresse, dans le fichier de données, de l'article associé à cette valeur de clé d'accès

Le fichier indexé



Le fichier INDEX est ordonné par valeurs croissantes de la clé

Accès à un fichier indexé

- ***l'accès séquentiel***
  - C'est l'accès aux enregistrements dans l'ordre croissant des valeurs de la clé d'accès du fichier.

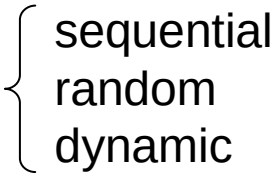
- ***l'accès direct***
  - C'est l'accès **direct** à un enregistrement étant donnée la valeur de clé d'accès qui le caractérise.

- Remarque

  Les opérations sont identiques pour les deux types de fichiers directs, à ceci près que :
  - pour un **fichier relatif** la clé d'accès est représentée par une **variable entière** associée au fichier, au moment de sa déclaration dans un programme,
  - pour un **fichier indexé**, c'est un ou plusieurs **champs de l'article** qui jouent le rôle de clé d'accès.

Exemple de déclaration de fichier indexé

```
file-control.
select F-VOITURES
assign to “voitures.dat"
organization indexed
access random
record key PLAQUE.
...
file section.
fd F-VOITURES.
1 ENR-VOITURE.
    2 PLAQUE pic X(10).
    2 MARQUE pic X(10).
    2 MODELE pic X(10).
    2 COULEUR pic X(10).
```



Les Instructions globales sur un fichier indexé

- Ouverture et fermeture du fichier

  - **open**

    input
    output *id-fich-dir*
    i-o

  - **close** *id-fich-dir*

Instructions élémentaires : LECTURE

- **Lecture directe**
  - **read** *id-fich-dir* [**into** *id-var*]
    [**invalid** inst-imper1]
    [**not invalid** inst-imper2]
    end-read

- Contextes de mise en œuvre
  - 'access random' ou 'access dynamic'
  - 'open input' ou 'open i-o'

- La clé doit être valorisée avant toute ordre de lecture

Exemple de lecture

```
move "123 ABC 45" to PLAQUE
read F-VOITURES
     invalid key
          display "pas de voiture dans le fichier"
     not invalid key
          display "modele = " MODELE
end-read
```

Instructions élémentaires : ECRITURE

write *id-article* [**from** *id-var*]
[**invalid** inst-imper1]
[**not invalid** inst-imper2]
end-write

- Contextes de mise en œuvre
  - **'open output'** ou **'open i-o'**
  - **'access random' ou 'access dynamic'**
  - **L'option 'invalid' permet de prendre le contrôle :**

-pour un fichier relatif l'emplacement concerné est déjà chargé

-pour un fichier indexé il existe déjà un article ayant même valeur de clé

Exemple d'écriture

```
move "123 ABC 54" to PLAQUE
move “FORD" to MARQUE
move "1000" to MODELE
 ...
write ENR-VOITURE
      invalid key  display "voiture deja enregistree."
      not invalid key  display "ok."
end-write
```

Ré-écriture

rewrite *id-article* [**from** *id-var*]
[**invalid** inst-imper1]
[**not invalid** inst-imper2]
end-rewrite

- *Contextes de mise en oeuvre*
  - 'open i-o'
  - Dans le cas de ''access random' ou 'access dynamic' la réécriture est directe (pas de lecture préalable indispensable)
  - l'option '(not) invalid' permet de prendre le contrôle en cas d'échec de la réécriture.

Suppression

delete *id-fich-dir*
[**invalid** inst-imper1]
[**not invalid** inst-imper2]
end-delete

- *Contextes de mise en œuvre*
  - 'open i-o'
  - Dans le cas de ''access random' ou 'access dynamic' l'effacement est direct (pas de lecture préalable indispensable)
  - l'option '(not) invalid' permet de prendre le contrôle en cas d'échec de l'effacement.

Exemple suppression

```
move "123 ABC 54" to PLAQUE
delete F-VOITURES
     invalid key     display "voiture absente"
     not invalid key     display "ok"
 end-delete
```

Recherche de l'existence d'un enregistrement

start *id-fich-dir* [**key**= *id-clé*
<=
>
>=
[**invalid** inst-imper1]
[**not invalid** inst-imper2]
end-start

- La recherche est effectuée par rapport à la valeur de clé indiquée dans *id-clé*.
- L'instruction **start** ne délivre aucune donnée mais, si elle s'exécute correctement, place un pointeur à une adresse précise d'un article dans le fichier.
- *Contextes de mise en oeuvre:*
  - 'open input' ou 'open i-o'
  - 'access dynamic'

Exemple recherche

```
move "123 XY 89" to PLAQUE
 start F-VOITURES
     key = PLAQUE
     invalid key
          set false to TROUVE
     not invalid key
          set true to TROUVE
end-start
```

Fichier direct traité en séquentiel

read *id-fich-dir* [**next/previous**][**into** *id-var*]
[**end** inst-imper1]
[**not end** inst-imper2]
end-read

- *Contextes de mise en oeuvre*
  - 'access sequential' ou 'access dynamic'
  - 'open input' ou 'open i-o'
  - Les options 'next' (lecture séquentielle en avant) et 'previous' (lecture séquentielle en arrière) sont restreintes à 'access dynamic'.

Exemple

```
read F-VOITURES next
    end          ....
    not end          ....
end-read
```

Écriture séquentielle

- *Syntaxe : idem à l'écriture directe*

- *Contextes de mise en oeuvre*
  - 'open output'
  - 'access sequential'
    - Pour un fichier relatif le SGF charge automatiquement l'emplacement suivant dans le fichier.
    - Pour un fichier indexé le programme doit préciser avant chaque exécution du 'write' la valeur de la clé pour l'article à écrire
  - L'option 'invalid' sert pour un fichier indexé à prendre le contrôle s'il n'y a pas respect de l'ordre croissant des valeurs de la clé à chaque écriture.

Réecriture séquentielle

- *Syntaxe : idem à la ré-écriture directe*

- *Contextes de mise en oeuvre*
  - 'open i-o'
  - Dans le cas de 'access sequential' l'instruction 'rewrite' doit être précédée d'une lecture réussie
  - la valeur de la clé doit être la même que pour la lecture, de plus l'option '(not) invalid' n'est pas autorisée.

Suppression en séquentiel

- *Syntaxe : idem à la suppression directe*

- *Contextes de mise en œuvre*
  - 'open i-o'
  - Dans le cas de 'access sequential' l'instruction 'delete' doit être précédée d'une lecture réussie
  - la valeur de la clé doit être la même que pour la lecture, de plus l'option '(not) invalid' n'est pas autorisée.

Les fichiers indexés multi-clés

- Un **fichier indexé** possède nécessairement

  - Une clé primaire (**record key**) simple ou composée

Mais

  - peut aussi posséder des clés secondaires (**alternate** key) simples ou composées pour chacune desquelles est géré également automatiquement un fichier index spécifique

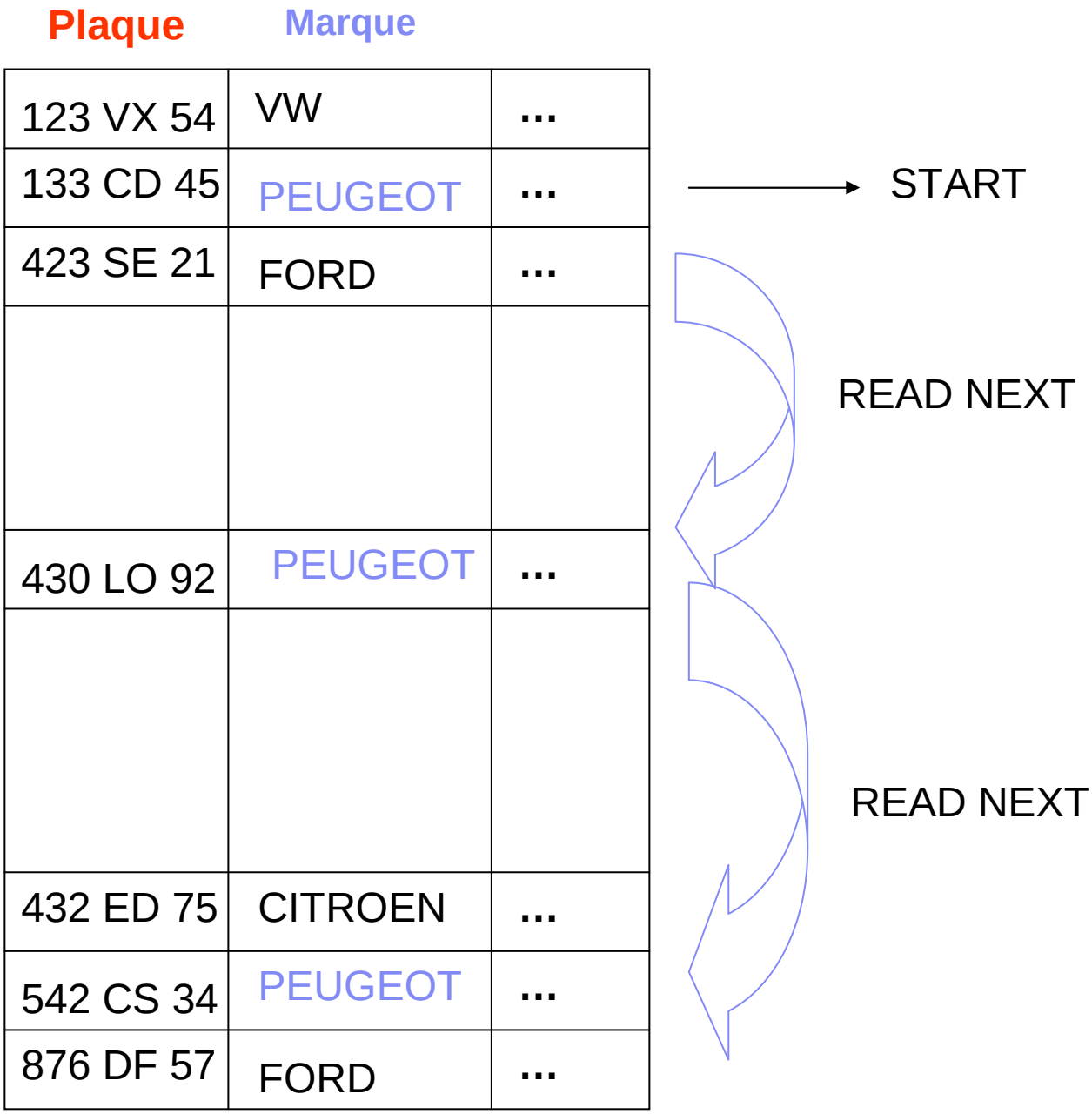
  - Pendant une période d'accessibilité au fichier multi-clés **la clé courante**, par défaut la clé primaire peut être changée grâce aux instructions '**read**' ou '**start**'.

Déclaration d'un fichier indexé multi-clés

```
file-control.
 select F-VOITURES
    assign to "voitures.dat"
    organization indexed  access random
    record key PLAQUE
    alternate record key MARQUE duplicates.
```

Affichage de toutes les voitures de marque PEUGEOT 403

```
move 'PEUGEOT' to MARQUE
start F-VOITURES key = MARQUE
    invalid key  display 'Pas de Peugeot'
    not invalid key  set  FIN-FICHIER  to FALSE
        perform until FIN-FICHIER
        read F-VOITURES next
                end   set  FIN-FICHIER to TRUE
                not  end
                  if MARQUE ='PEUGEOT'
                       and MODELE = '403'
                     then   display PLAQUE, MODELE
                  end-if
       end-read
    end-perform
end-start
```


Plaque
Marque
123 VX 54
VW
...
133 CD 45
PEUGEOT
...
START
423 SE 21
FORD
...
READ NEXT
430 LO 92
PEUGEOT
...
READ NEXT
432 ED 75
CITROEN
...
542 CS 34
PEUGEOT
...
876 DF 57
FORD
...