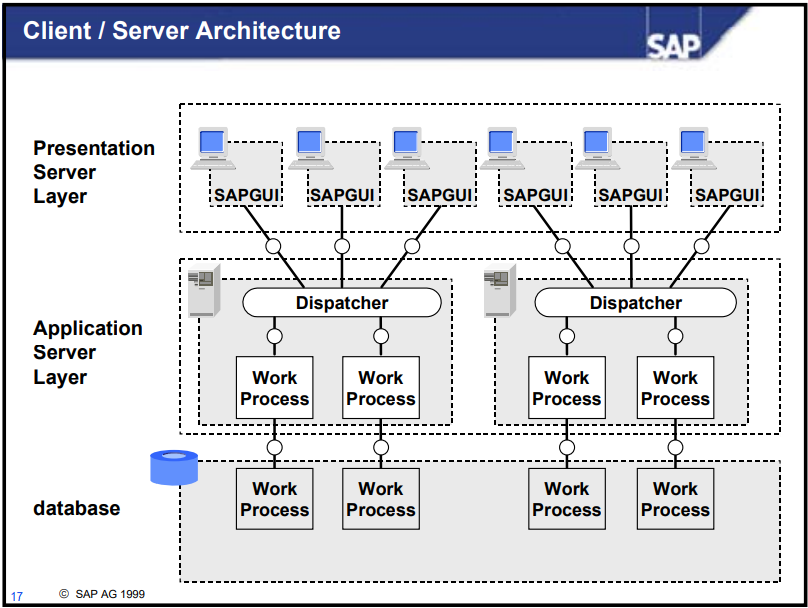
# SAP théorie

SAP offre 3 services de Cloud :

* SaaS : c’est un logiciel en ligne, comme google doc. C’est la même version pour tous les utilisateurs.
* PaaS : Permet d’étendre des applications déjà existantes.
* Private Managed cloud : En ligne mais privé, on peut faire les modifications que l’on souhaite.

## Unit 2

Client server architecture.



Le Sap Gui qu’on installe sur notre Pc se connecte au serveur d’application SAP. Les requêtes que l’on à l’intérieur du client lourd arrive sur un **dispatcher** qui va gérer les requêtes grâce à une file d’attente s’il y a beaucoup trop de demande. Ensuite la requête est envoyée au « Work process » libre.

Mais pour savoir quelle dispatcher choisir et pour que tout le monde n’aille pas sur le même dispatcher il faut faire du « load balacing » lors de la connexion avec le SAP logon. Le Massage Server qui est relié à tous les serveurs d’applications va faire ce boulot. Il va permettre de distribuer la charge. Une connexion arrive sur le Message Server, ce serveur va attribuer la connexion à un serveur d’application. La connexion sera en permanence lié à ce serveur, il ne faudra donc plus passé par le Message Server car les serveurs sont statefull.

C’est dans les « **Work process** » que s’exécute les programmes ABAP. La commande SN50 permet d’afficher les Work process

**Dialog Step** : C’est la requête envoyée au dispatcher qui est ensuite exécuter dans un work process libre. Une tâche (programme) peut avoir un ou plusieurs « dialog step ». Si la tâche à plusieurs « dialog step » celle-ci peuvent être exécuter sur différents work process.

Dans l’ERP il y a une multitude de transactions. Une **transaction** permet d’exécuter un programme ABAP.

KNA1 représente la table des clients dans le système SAP.

Différents types de programmes :

* Report : Programme exécutable qui permet de traiter et afficher des données.
* Dynpro : Application avec une interface graphique, permet de gérer les interactions avec le client. (Fait grâce à un outil)
* Business Server Pages : Création de page HTML avec ABAP.
* Web DynPro
* SAP UI5 :

Dans un programme il y a 3 types d’écrans :

* L’écran de sélection (Parameters et select – option).
* L’écran (Dynpro : call screen XXX).
* Liste (qui permet d’afficher des données grâce à l’instruction WRITE)

Sur les écrans de sélection, on peut sauvegarder les critères de recherche contrairement dans un DYNPRO.

## Unit 3

**ABAP workbench** est l’IDE pour le langage ABAP, c’est là où les programmes ABAP vont être développer. Pour accéder au ABAP workbench il suffit de taper la transaction **s001**. ABAP fonctionne avec **Open SQL**, qui est un SQL light à la sauce SAP.

*Qu’est-ce qu’un mandant (=client)* ?

C’est un numéro qui sert à découper l’ERP, notamment de mélanger les données de différentes « entreprises » dans une table. Le mandant à accès aux données uniquement pour le mandant en question, il n’a pas accès aux données des autres mandant.

Les données du système sont soit :

* Dépendant du mandant
* Indépendant du mandant (Cross-client)

Les programmes ABAP ainsi que les tables ne sont pas dépendant du mandant (=multi mandant).

Table mono mandant : Le mandant est un champ et fait partie de la clé.

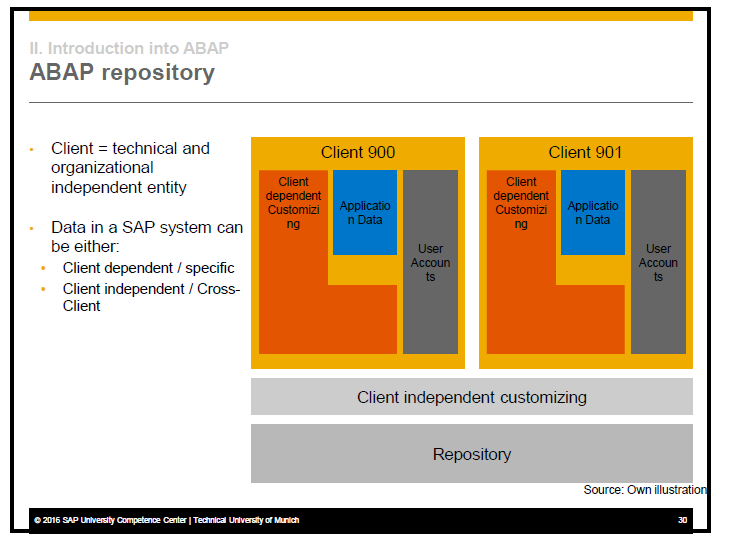
Table multi mandant : tous les mandants on accès à toutes les données de la table.

L’ERP peut réagir d’une manière différente d’un mandant à l’autre. L’exécution du programme est différente d’un mandant à l’autre mais le programme est le même, juste les données qu’il traite sont différentes. Il y a aussi de la paramétrisation qui change d’un mandant à l’autre, ex : un écran peut être appelé d’un mandant à l’autre.

Le Repository est multi mandant.

Les utilisateurs sont mono mandants

Les programmes ABAP sont portable. La paramétrisation se fait par mandant.



### Namespace

Pour tout programmes custom le namespace commence soit par Z soit par Y car SAP utilise les namespaces allant de A à X pour tous leurs programmes et implémentations. Les entreprises peuvent enregistrer leur propre namespace. Un namespace est unique et ne peut être assigné qu’a une seule compagnie (au sein d’un même système ? mais un même namespace peut etre le même sur un serveur d’application différent).

### Repository

Le Repository => table multi mandants qui contient les objets du développement (programme, groupes de fonctions, structure de table, domaine, data, element)

Le Repository est structuré par modules dans lequel il y a une découpe en package (=classe de développement). A l’intérieur d’un package, on retrouve des programmes, des modules de fonctions, des tables (multi mandant uniquement ?), …

Le module contient donc tous les objets de développeur.

L’organisation en package est une organisation pour le développement. Mais attention un package est différent d’une application. Une application peut utiliser plusieurs packages. Un objet abap appartient à 1 et unique package.

Les programmes ABAP sont stocké dans la base de données.

Le Repository peut être vu comme un ensemble de table car en SAP tout est stocké dans des tables.

### ABAP Workbench

L’ABAP Workbench est composé de nombreux outils comme :

* Debugger
* Screen Painter
* Object Navigator (se80 – pour tous les objets de développement) – Repository
* Menu Painter
* ABAP editor
* Fonction Builder
* Class Builder
* ABAP Dictionary

Dans un écran de sélection, on peut faire : **System -> Status**, pour connaitre le nom du programme.

Un programme ABAP est diviser en 4 :

1. Déclaration des données : Parameters => déclare une variable d’un certain type et affiche un champ de saisie dans l’écran de sélection du type déclarer.
2. Dialogue avec la base de données : SELECT…
3. Traiter les écrans : Call screen 100 => Process before output.
4. Création d’une liste : Write.

Dans un programme ABAP il faut toujours tester sy-subrc = 0. Si c’est égal à 0, ça veut dire que tout s’est bien passé.

### Ordre de Transport

Le mécanisme de l’ordre de transport sert à faire passer un ensemble d’objets (du repository) de l’environnement de développement à l’environnement de production. Pour cela, il faut que tous les objets soit défini dans un package transportable.

Un développeur peut avoir plusieurs tâches mais une tâche ne peut qu’avoir un unique développeur. Cette tâche à plusieurs lignes dans laquelle chaque ligne est un pointeur vers un objet. Après avoir fini sa tâche le développeur va rattacher ses lignes de sa tâche, donc les pointeurs vers les objets à **l’ordre de transport**.

Pour la release et l’export, on va prendre directement les « lignes » des tâches finies des développeurs, donc celle qui sont lié à l’ordre de transport. Ensuite on export de l’environnement de développement pour importer dans l’environnement de production. Quand on import on recrée les objets.

Un Objet peut être un :

* Package
* Programme
* Structure de table
* Domain
* Data Element
* …

Chaque objet peut se retrouver dans un et unique package. Mais un objet peut se retrouver dans plusieurs lignes dans des tâches différentes. Par exemple si deux développeurs ont travaillé sur le même objet chacun deux auront dans sa tâche un pointeur vers cet objet.

# Unit 4

### ABAP Dictionnary (Data Dictionnary)

Le dictionnaire ABAP est utiliser pour créer et manager des méta données. Il permet une description centrale de toutes les données utilisées dans le système. Il supporte la définition de type fait par l’utilisateur. Donc le dictionnaire représente tout ce qu’il se trouve dans la base de données, c’est pour cela qu’on parle de méta donné car ce sont des données de données. Dans l’abap dictionnaire on retrouve les tables, data elements, domains, types, …

### Types

Le type défini l’attribut des :

* Champs Input/ouput
* Data objects
* Paramètre d’interfaces pour les sous routines
* Paramètre d’interfaces pour les fonctions/méthodes

Les types peut être défini localement ou bien dans l’ABAP dictionary.

Les types pré défini :

* Character
* Numeric text
* Date (YYYYMMDD)
* Time (HHMMSS)
* Hexadecimal
* Character string
* String of bytes
* Integer
* Packed number
* Floating point number

N -> numérique, mais n’est pas fais pour calculer. Il y a parfois une différence quand on utilise le type N ou le type I.

On définit un type grâce au mot clé : TYPE

### Data Object

Un data object est une variable dans un programme ABAP. Elle peut prendre trois types différents :

* Un type défini par l’utilisateur
* Un type pré defini
* Un type de l’ABAP dictionary (data element, domain, table)

Si on fait : ma\_variable LIKE <data-object>, ma\_variable sera du même type que l’objet référencé.

Ex : DATA : ma\_var LIKE ma\_var2, ma\_var sera du même type que ma\_var2

Sinon : ma\_variable TYPE <type >

On définit une variable grâce au mot clé : DATA

Les 3 objets pour définir des données (c.à.d. la structure qu’une donnée peut avoir)[[1]](#footnote-1) dans l’ABAP Dictionnary sont :

* Data element
* Domain
* Table

Ces objets font aussi parti de l’OT, si on le dit bien évidemment.

### DATA ELEMENT

Dans une table, nous avons des champs, ces champs sont d’un certain type, qu’on appelle **data element**. Le data element est utilisé principalement pour définir des champs dans une table, composant de structure, … Le data element représente la partie sémantique d’un « domain » dans lequel on définit les libellés (court, moyen, long, en tête de liste), aide (f1), description. Un data element représente toujours un unique domain. Les libellés sont affichés dans les écrans de sélection en autre.

Ex : Pays de production, pays de naissance, pays de destination => ils font tous référence au même domaine qui est ici pays.

### DOMAIN

Le domain défini la partie technique d’un champ dans une table dans lequel on définit le type élémentaire, la longueur, les valeurs permises. Un domain peut être utilisé par plusieurs data element.

Ex : Pays : char (3)

(Mettre le schéma ici)

### Elementary Data Objects

Le mot clé « Compute » n’est pas nécessaire pour faire des calculs.

Le mot clé « Clear » réinitialise une variable en fonction du type :

- 0 pour un numérique/integer.

- ‘ ’ pour un alphanumérique

Le mot clé « Case … when» correspond à « switch… case » en java.

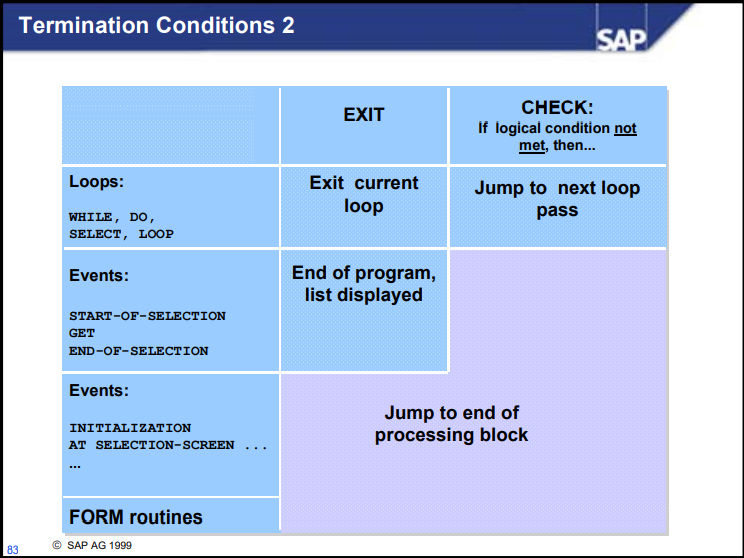
Le mot clé « Initial » regarde si la variable à une valeur initial c’est-à-dire 0 ou ‘ ‘.

Le mot clé « Table » défini une working area qui permet de travailler avec une ligne (tuple) de la table référencée. La WA à la même structure que le table.

Le mot clé « Sy-Index » permet de savoir le nombre de tour de boucle que nous avons déjà fait.

Le mot clé « Exit » permet de quitter une boucle.

Le mot clé « Check » vérifie une condition si celle-ci n’est pas respecté alors on va à la fin du block d’instruction.



### Structure.

Une structure est similaire à une work area, la structure peut être dans l’abap dictionnary tandis que la work area pas. (Demandez au prof pour être sûr ?)

Elles sont similaires aux tables, la seule différence c’est qu’elles ne contiennent qu’une seule ligne.

L’instruction « Move-Corresponding <src> TO <destination> » fait la conversion de type si les champs ayant le même nom sont de type différent.

### Table interne

La durée de vie d’une table interne est équivalent à la durée de vie du programme.

Une table est définie en trois niveaux :

* Structure de ligne => colonne
* Clé => quels champs font partie de la clé ? unique ? multiple ? (Dépend tu type de table)
* Type de table => accéder via un index (je veux la 4ème ligne) ? accéder via une clé ? (Dépend de l’accès qu’on a envie de faire)

Les différents types de table :

* Table indexée :
  + Table standard
  + Table triée
* Table hashé

#### Table standard

* Accès via un index (je veux la 2ème ligne)
* Accès via une clé
* La clé n’est pas forcément unique
* On accède principalement avec l’index

#### Table triée

* Accès via un index (je veux la 2ème ligne)
* Accès via une clé
* Les clés peuvent être unique ou ne pas l’être
* On accède principalement par la clé (recherche dichotomique)

#### Table hashée

* Accès via une clé
* Clé unique
* On accède uniquement par la clé
* C’est une table volumineuse

Instructions sur les tables interne :

* Append : ajout en fin de table
* Insert : ajout n’importe où dans la table
* Read : lecture d’une entrée de table dans une work area
* Modify : modifié une entrée
* Delete : supprimer une entrée
* Clear : Pour la table sans header line, la table est effacée avec l’instruction **clear**. La table avec header line est effacé avec l’instruction **refresh** et la header line est effacé avec l’instruction **clear**

Pour déclarer une table il faut faire :

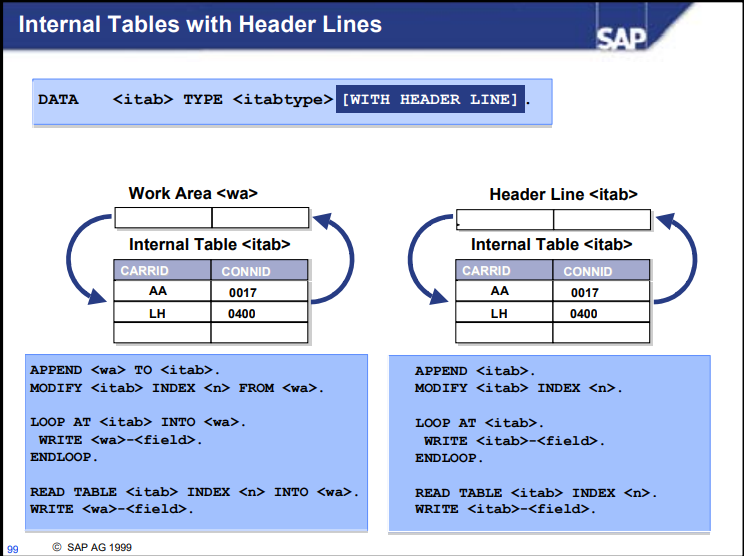
DATA : <nom\_variable> TYPE <nom\_table>

Pour définir une table interne il faut que ça commence par **begin of** … pour définir la structure.

L’instruction append rajoute une ligne dans la table interne.

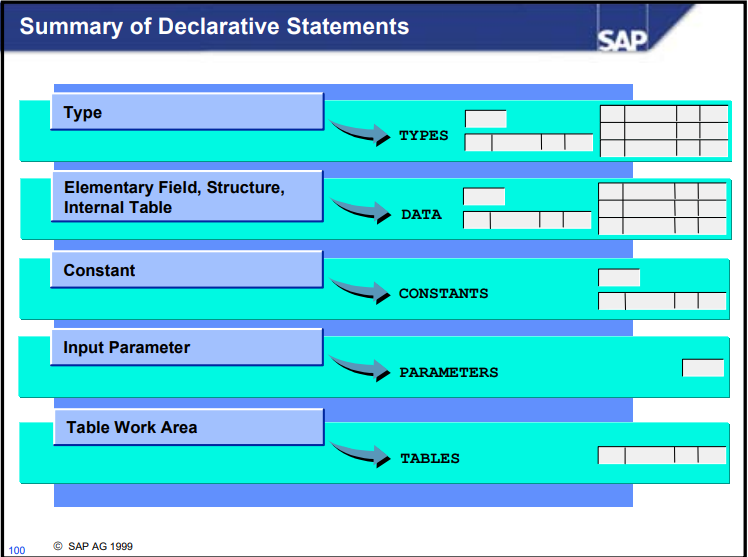
Pour manipuler une table, il faut toujours utiliser des work area que ça soit en lecture ou en écriture.

Il existe une alternative à la work area, elle s’appelle « Header Line ». On peut définir donc une table interne avec un header line qui a le même nom que la table et qui correspond à la ligne courante.

Quand on utilise un header line, on ne doit pas mettre la ligne en court de traitement dans une work area, il sait que quand on va faire appel à la donnée ça sera la ligne courante.

Refresh permet de nettoyer toute la table quand on utilise un header line et clear permet de nettoyer le header line.

### Résumer des déclarations dans un programme ABAP



## ABAP Statement

Après un statement il faut tester sy-subrc. S’il est différent de 0 alors quelque chose c’est mal passé. Dans ce cas il faut afficher un message d’erreur pour avertir l’utilisateur. Nous pouvons utiliser l’instruction message qui va chercher dans la table t100 le message d’erreur avec le bon id. Nous pouvons même personnaliser le message en mettant des « & » dans le message et en donnant la donnée avec with.

Les messages de succès s’affichent sur l’écran d’après.

I : information.

W : Warning

A, X : Plantage du programme

E : Erreur

S : succès

# UNIT 5 : Dialogue avec la base de données

* Le data dictionary est le répertoire global pour les types de données.
* Attribution de textes d'aide et d'explications pour les types de données dans différentes langues
* Le modèle entité-relation peut être affiché automatiquement sous forme de chiffre
* Les objets les plus importants y sont : structure, table, data element, domain

**Le data dictionnary** est un répertoire qui va permettre de décrire tous les types de données qui vont être utilisé dans l’environnement ABAP. On y défini les domain, data element, table externe, structure. Tous ceci sont les éléments (objets) principaux.

Une Structure, c’est un certain nombre de champs. Une table a la possibilité de créer des entrées. Une table à une structure de ligne. La structure de ligne est définie dans l’abap dictionnary de la même manière que la structure d’une table.

**Data Dictionnary** : transaction se11 – créer les éléments de l’abap dictionnary.

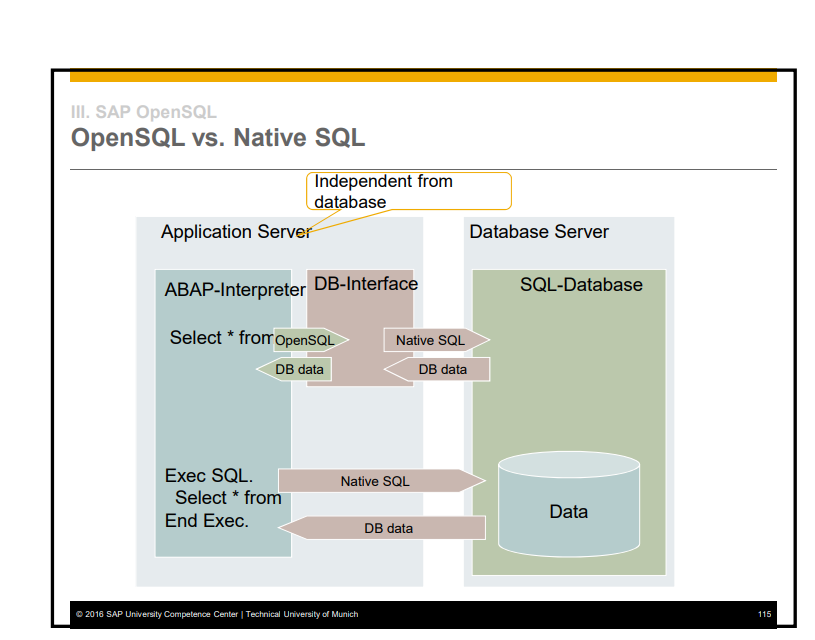
Une table peut changer de structure sans perdre les données. (database utility). Les tables standard peuvent être étendu par l’instruction APPEND.

Tous les objets (data element, domain, …) créer avec l’abap dictionnary vont être stocké dans le repository. Le repository est toujours multi mandant. Tout le monde y aura donc accès. Impossible qu’un domain existe dans un mandant et n’existe pas dans un autre mandant. Quand on crée un domain il est disponible pour tous les mandants. Un data element fait partie de data type.

Objet de blocage permet de bloquer un objet quand on le modifie pour pas que deux personnes ne le modifie en même temps.

**Data browser** : se16 : voir le contenu des tables. SN30 : dialogue de maintenance

## OpenSQL vs. Native SQL



Avec OpenSQL, le programme n’accède jamais directement à une table, il passe par une interface DB qui elle fait les requêtes à la base de données. Il y a une interface DB pour chaque type de DB. L’interface DB convertie la requête en une requête native.

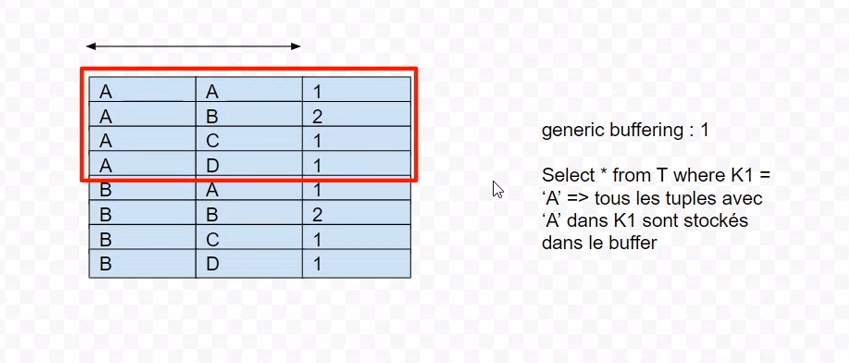
Dans le cas où on utilise le SQL native on perd la portabilité car on est lié au dbms. Pour faire du Sql native il faut placer les requêtes entre le bloc Exec SQL et END Exec. Ça peut être utiliser si parfois on veut bénéficier de certaine fonctionnalité du dbms.

OpenSql est du Sql spécifique à ABAP. En utilisant OpenSQL on est indépendant du dbms (**Database Management System**) sur lequel on travail.

Un autre avantage pour OpenSQL est qu’on a de la bufferisation.

Il y a 3 types de bufferisation pour une table :

* Si la table est « **Fully** **buffered** » alors toute la table sera bufferiser dans le serveur d’application lors de la première requête sur cette table fait par n’importe quelle programme (ce n’est pas lié à un programme en particulier). L’interface Db va donc récupérer les données dans le buffer au lieu de la base de données.
* La table peut être aussi « **Single record buff** », qui contient un tuple d’une table. Chaque fois qu’un tuple est lu, il est placé dans le buffer. Si quelqu’un veut lire un tuple, il va d’abords regarder dans le buffer, si le tuple est présent dans le buffer on renvoie la valeur dans le buffer, sinon on fait une requête à la Db.
* La table peut être « **Generic area buffered** ». Il y a un paramètre qui intervient dans la généricité du buffer qui va dire sur quel champ de la clé on va bufferiser. Ex : Ici on dit qu’il y aura une bufferisation sur le premier champ de la clé, donc tous les tuples ayant la même valeur pour ce champ sera bufferiser. Le cadre rouge représente ce qui sera bufferiser.



C’est pratique quand il faut bufferiser des libellés pour un code langue. On ne va pas bufferiser tous les libellés de toutes les langues alors que l’utilisateur est connecté en anglais par exemple.

La bufferisation est indépendante de l’utilisateur, si un autre utilisateur fait une requête sur une table bufferiser alors il va bénéficier du buffer.

Les buffers sont définis au niveau du serveur d’application et sont dépendant de celui-ci. Si on a un système SAP qui a plusieurs serveurs d’application il y a un buffer par serveur mais SAP synchronise les buffers entre serveurs d’application. Ne pas oublier que notre programme ABAP tourne sur un serveur d’application et pas dans notre client lourd.

La bufferisation fonctionne qu’en lecture, s’il y a une modification dans la table... le buffer sera invalidé. Donc lors de la prochaine requête, il y aura à nouveau une requête pour bufferiser la table.

### SELECT

Il y a deux variantes pour le select :

* Select single (on connait tous les champs de la clé de la table) :
  + Oui j’ai trouvé : sy-subrc = 0
  + Non j’ai pas trouvé : sy-subrc != 0
* Select loop=> est une boucle :
  + Je suis rentrée dans la boucle : sy-subrc = 0 -> après le endselect
  + Je ne suis pas rentré dans la boucle : sy-subrc != 0

Quand on fait un select loop, ce qu’il se passe c’est quand va charger dans l’interface Db tout le résultat de la requête (1) et ensuite ça va boucler sur le buffer de la table dans l’interface et renvoyer au programme appelant les tuples un par un (2).

Une variante au select loop c’est de mettre tout le résultat dans une table interne.

Un select single ne renvoie qu’un seul tuple.

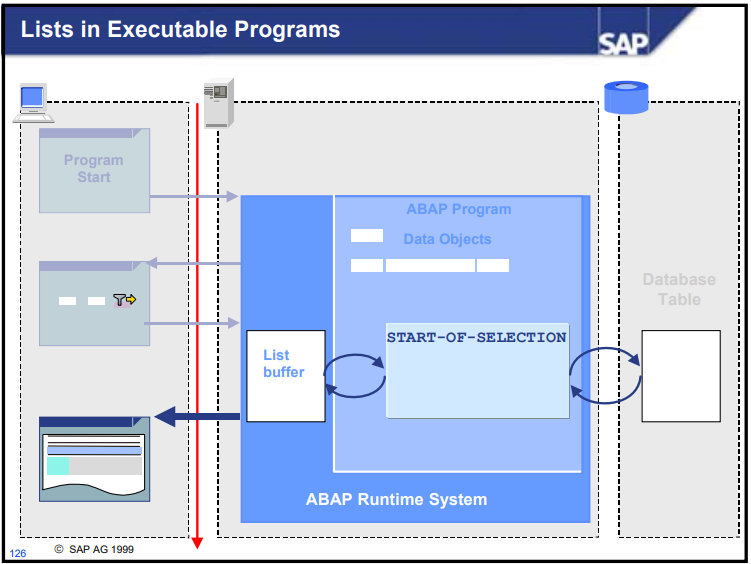
# Unit 7 : User Dialogs : list

A partir d’une liste on peut :

* Afficher la liste à l’écran
* L’imprimer
* Capacité d’être multi lingue

La liste est déclencher avec l’instruction **write.** Quand on voit la liste à l’écran l’exécution du programme est terminée.

La liste est placée dans un buffer après l’instruction Write. Le buffer existe après que le programme a terminé son exécution.



Une liste peut être multi lingue grâce aux text elements

# Unit 8 : User Dialogs : Selection screen

L’écran de selection est difinie par parameters ou select options. C’est un écran qui permet de sélectionner des choses dans la db avant de les traiter

On peut faire plusieurs choses avec les écrans de sélection :

* Capacité multi lingue
* Contrôle de type à la saisie (ex : date, numérique, …)
* Sauvegarder les inputs de l’écran de sélection grâce aux **Variants**
* L’aide à la recherche (f1)
* Critère de sélection sophistiqué (grâce au select options : ex : je veux toutes les entrées avec le nom qui commence par B)

Quand on définit un select options, on définit une table interne qui contient toujours 4 colonnes (Sign, Option, low, high).

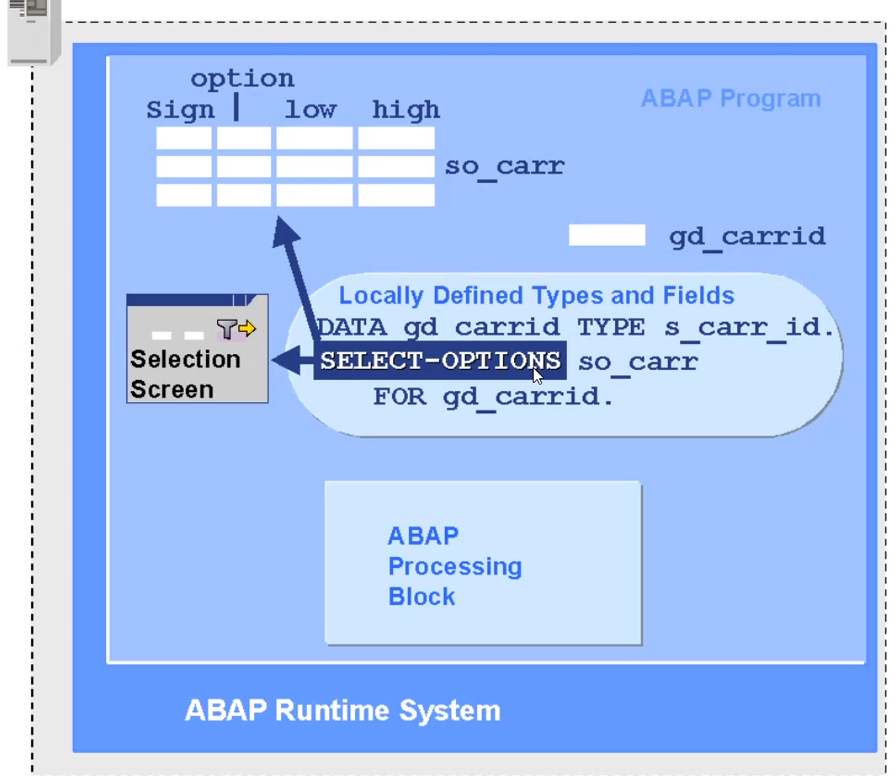
**Sign** sert à définir si on prend les valeurs ou si on les exclu. (**I**nclude ou **E**xclude)

**Option** contient toutes les valeurs classiques d’un opérateur logique (Equal, Not Equal, greater than, greather or equal, less than, less or equal), Between, contain pattern **(CP)** qui est semblable à clause like (ex : je veux tout ce qui commence par B ou je veux tout ce qui contient ‘aze’ au milieu de la chaine). Ils sont utilisés sous forme de signe ou sur un code à 2 caractères.

Le type de Low et High sont du même type que la variable après le FOR (on utilise normalement low et high quand il s’agit de between).

Un select option porte que sur un seul champ.

Quand on utilise un select option on utilise la clause IN.



# Unit 9 : User Dialogs : Screen

Le Dyn Pro (écran) fait 4 choses :

* Contrôle de type à la saisie
* Aide à la recherche (f1)
* Contrôle de consistance lors de la saisie (en rapport avec ce qui est défini dans l’abap dictionnary)
* Flux de navigation (l’utilisateur gère où il veut aller)

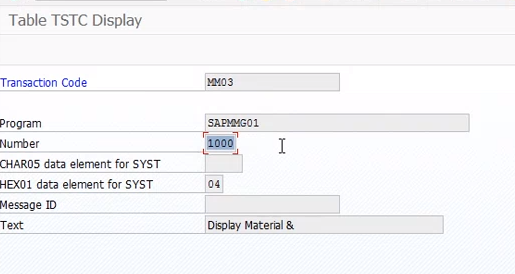
Ecran Dyn Pro : pour gérer la navigation entre les écrans. En fonction d’où l’utilisateur clique un écran différent sera appelé.

Les dyn pros sont créées à partir du Screen Painter.

Il y a 3 niveaux :

* Code Transaction
* Module Pool : contient tout ce qui est programmation ABAP (contrôle au niveau des écrans)
* Ecrans défini avec le screen painter.

**TSTC** est la table qui fait le lien entre le code transaction, le module pool et le premier écran. Dans la table TSTC on peut voir le programme (module pool) qui sera charger pour un certain code de transaction et le premier écran qui sera appelé.

Ici le code transaction MM03 charge le module pool SAPMMG01 et le premier écran qui sera appelé sera l’écran 1000. Donc ici l’appel au premier écran est défini dans un paramètre de table, il n’est pas défini dans le programme lui-même.  
Le code transaction permet donc deux choses, chargé le programme et afficher le premier écran.

Si un programme n’a pas de code transaction, on peut appeler le premier écran dans le programme lui-même (Call Screen xxx).

Dans un écran du screen painter on a 3 éléments :

* La partie visible (ce que l’utilisateur voit)
* Avant l’écran (Process before output) – appelé avant l’affichage
* Après l’écran (Process after input) – après une action

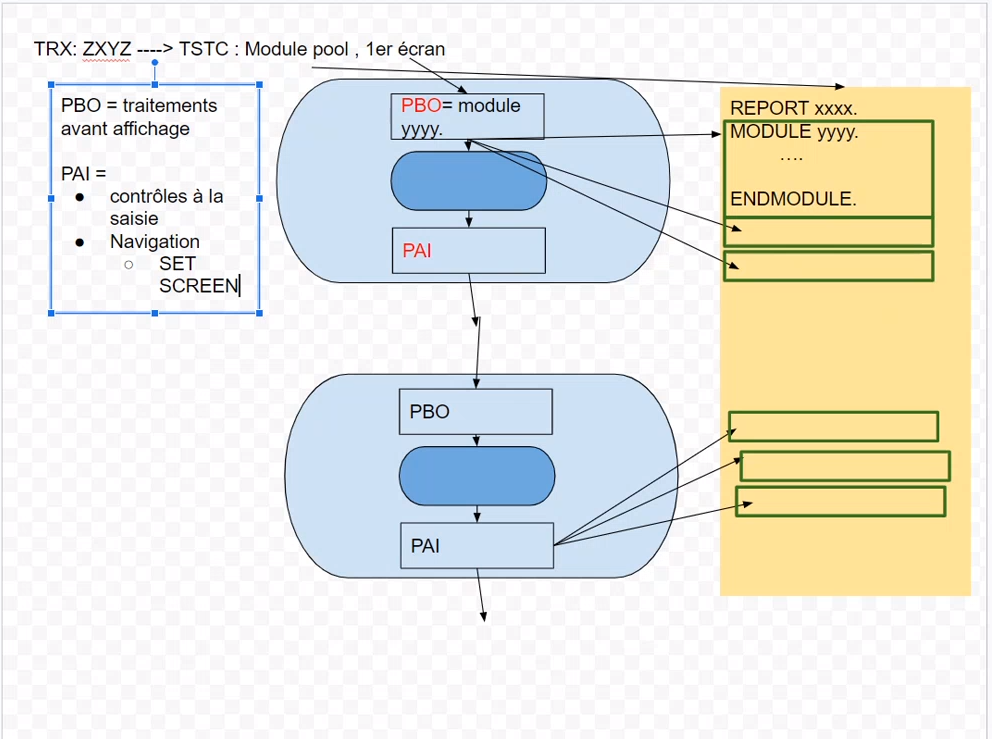
**1 seul module pool (programme) utilisé par tous les écrans.**

Ce qu’il y a entre module et endmodule ce sont des instructions ABAP qui vont être déclenché par le PBO. Donc un PBO va faire appel à un ou plusieurs modules dans le programme avant l’affichage de l’écran. LE PAI appellera aussi les modules.

Un **Module pool** est un ensemble de définition de module.

PBO : Tous les traitements avant l’affichage (ex : aller chercher un libellé par rapport à un code).

PAI : Contrôle à la saisie dans un module (vérifier ce que l’utilisateur à entrer) et gérer la navigation (appelé l’écran suivant) en utilisant set screen pour indiquer quel est le numéro de l’écran suivant quand on clique sur un bouton.



#### Module de fonction

A ne pas confondre avec un module qui lui est appelé exclusivement par le screen painter (PBO,PAI).

Les modules de fonction sont les seuls éléments qui peuvent être appelé au départ de programmes extérieur (python, java, …). Elles seront appelées en remote fonction call

Permet d’accéder aux fonctionnalités de l’ERP au départ d’un programme qui est non-SAP.

Code transaction Se37 pour voir les modules de fonctions. Elles commencent par bapi les modules de fonctions standards de SAP.

Lexiques :

**SAP HANA** est une base de données en mémoire.

**SAP GUI** : client lourd qui est connecté à un serveur d’application.

**SAP LOGON** : interface qui demande à quelle ERP on veut se connecter.

**SAP** **NetWeaver** : nom commercial pour représenter les éléments technologiques.

**SAP ECC CORE** : C’est là où on va retrouver les clients finaux.

**SAP ERP :** C’est l’ensemble des produits technologiques (bleu) et des applications (jaune).

**ABAP** : c’est le langage de programmation, il fait partie de NetWeaver, donc il dépend de ce dernier. Pour mettre à jour ABAP il faut mettre le NetWeaver

**SAP S/4 HANA** : est composer de SAP Fiori et de SAP Hana. Grâce à SAP Fiori, il tourne uniquement sur le WEB.

Liens utiles :

<https://www.saponlinetutorials.com/what-is-sap-abap-workbench/>

<https://answers.sap.com/questions/3969906/dialog-step.html>

<http://sap.technique.free.fr/static.php?op=glossaire_SAP.txt&npds=1>

<https://www.tutorialspoint.com/sap_abap/sap_abap_dictionary.htm>

<https://www.saponlinetutorials.com/what-is-abap-dictionary-data-dictionary-overview/>

<https://help.sap.com/doc/saphelp_nw73ehp1/7.31.19/en-US/cf/21ea0b446011d189700000e8322d00/frameset.htm>

<https://www.saponlinetutorials.com/sap-abap-data-types-data-objects/>

Questions :

Plusieurs entreprises peuvent être attaché à un même ERP (même serveur d’application) ? c’est-a-dire que Carrefour et Delhaize par exemple sont dans le même système. Ils ont donc accès aux mêmes tables et tout.

Peut on utiliser read table sur des tables externes ?

Combien de temps les données sont bufferiser ? quel est la limite ? et si on fait plusieurs fois des requêtes sur la même table mais que toutes les données ne sont pas bufferiser. Dans le cas où on doit récupérer des données de cette même table mais qui ne sont pas bufferiser est ce qu’on écrase les anciennes données de cette table ou pas ?

1. La définition de données (data definition) permet de définir la structure qu’une donnée peut avoir. Par exemple un acteur, comment serra-il représenté ? par une table avec X champs. En Sql, il y a le DDL (data definition language) qui sont toutes les opérations (CREATE – ALTER – DROP – RENAME - …) qui permet de manipuler la structure de données et non la donnée elle-même. En SAP, l’ABAP Dictionnary gère la partie DDL, c’est-à-dire elle définit comment les données sont stocké dans la base de données SAP. Donc le dictionnaire est utilisé pour créer et gérer des définitions de données [↑](#footnote-ref-1)