DEBRECENI EGYETEM

SZÉCHENYI 2020

Bevezetés az ABAP programozásba

Vágner Anikó

Debreceni Egyetem

Informatikai Kar

TÁMOP 4.1.1.C-12/1/KONV-2012-0013

Integrált szervezeti és komplex felsőoktatási szolgáltatások, valamint képzések fejlesztése a versenyképes Debreceni Egyetemért

Tartalomjegyzék

Előszó

A tananyagot azoknak az olvasóknak ajánlom, akik rendelkeznek némi előismerettel, azaz olvasták és alkalmazni tudják a Bevezetés az ERP rendszerekbe és a navigáció alapjai az SAP-ban című tananyagot, ismernek egy magasszintű programozási nyelvet (C, Java, C#, PL/SQL) és tudnak SQL utasításokat írni.

A tananyag nem ad teljes ismeretet az ABAP programozásról, néhol egyszerű példákon át vezeti be az olvasót egy adott eszköz használatába, illetve nem tartalmazza az ABAP nyelv minden eszközét. További ismeretek megszerzéséhez az ABAP dokumentációjának részletes tanulmányozását javaslom.

Az ABAP programozási nyelv

Az ABAP programozási nyelv segítségével üzleti alkalmazásokat hozhatunk létre SAP környezetben. Egy ABAP program komponenseit feladatuktól függően elhelyezhetjük a háromrétegű kliens-szerver architektúra egyik rétegébe, azaz a prezentációs, az alkalmazás vagy az adatbázis rétegbe.

Az ABAP egy 4GL nyelv, típusos, többnyelvű alkalmazások készítését teszi lehetővé, SQL-t használhatunk benne, objektumorientált kiterjesztése van, platform független.

Bejelentkezés a rendszerbe

Az ABAP programok írásához egy speciális felhasználóval kell bejelentkezni a rendszerbe. A felhasználónak rendelkeznie kell fejlesztői kulccsal. A cégeknek minden egyes fejlesztői kulcsért fizetniük kell. Egy fejlesztői kulcs csak egy SAP felhasználóhoz tartozhat, de egy SAP felhasználóval egyidejűleg több programozó is bejelentkezhet és programozhat a rendszerben. Emiatt gyakori, hogy a cégek csak néhány fejlesztői kulcsot vásárolnak, amelyet egyszerre több fejlesztő használ.

Az egyik ilyen speciális felhasználó a BCUSER, amelyet a legtöbb rendszer ismer. Ha egy központi rendszerre jelentkezünk be, akkor kérjünk az adminisztrátortól felhasználói nevet és jelszót. Ha saját SAP szervert használunk, akkor meg kell vizsgálnunk, hogy a BCUSER létezik-e a rendszerünkben.

Jelentkezzünk be a DDIC vagy a SAP* felhasználói névvel. Mindkét felhasználóval rendszeradminisztációs feladatokat tudunk elvégezni. Csak akkor használjuk ezeket a felhasználói neveket, ha ismerjük a rendszeradminisztrációs utasításokat és azok hatását. Ha ellenőrizni akarjuk, hogy a BCUSER létezik-e, az egyikkel be kell jelentkeznünk. Erre a feladatra a DDIC felhasználói név jobb.

A bejelentkezéshez válasszuk a 001-es Client-et és írjuk be a "DDIC"-t a User mezőbe. Az SU01 tranzakció meghívja a User Maintenance tranzakciót. Írjuk a User mezőbe a "BCUSER"-t, és nyomjuk meg a Display gombot (F7).

Ha a rendszer megtalálja a felhasználót, akkor megváltoztathatjuk a jelszavát a Logon Data fülön.

Ha nem találjuk meg a felhasználót, akkor hozzuk létre. Írjuk a User mezőbe a "BCUSER"-t és nyomjuk meg a Create gombot (F8). Töltsünk ki minden kötelező mezőt az Address fülön, a Profile fülön a Profile mezőbe írjuk be az SAP_ALL-t, és állítsuk be a kezdő jelszót a Logon Data fülön (általában az "initial"-t használják, ha az első bejelentkezés után megváltoztatják). Ha készen vagyunk, mentsük el a munkánkat a Save gombbal (Ctlr+s) és lépjünk ki. Amikor az első programunkat írjuk, a rendszer kérni fogja a fejlesztői kulcsot.

A felhasználók karbantartásáról az SAP Adminisztráció című tananyagban találhatunk több információt.

Jelentkezzünk be a BCUSER felhasználói névvel. Ha beállítottuk, akkor a rendszer kérni fogja, hogy változtassuk meg a jelszavunkat.

Dokumentáció

Az ABAP dokumentációt megtalálhatjuk a rendszerben, ha futtatjuk az ABAPDOCU tranzakciót vagy navigálunk az SAP menüben: Tools, ABAP Workbench, Utilities és Example Library. Más hasznos tranzakciókat is találhatunk a Utility mappában, mint a Keyword Documentation (ABAPHELP) és a Demos (DWDM).

Amikor az ABAP programunkat írjuk az ABAP Editor-ban, egy ABAP utasításró kérhetünk információt. Álljunk a kurzorral az utasításra nyomjuk meg az F1-et vagy a kattintsunk a "Help on …" gombra (Ctrl+F8).

Az első ABAP program

Az ABAP Editorban írhatjuk meg az ABAP programjainkat. Az ABAP Editort az SE38 tranzakcióval hívhatjuk meg, vagy megnyithatjuk az SAP menüből is: Tools, ABAP Workbench, Development, és ABAP Editor. Az első program neve legyen Z_FIRST_ABAP, amelyet a Program nevű mezőbe írjunk. Jegyezzük meg, hogy minden programnévnek Z vagy Y betűvel kell kezdődnie (a Z-t használják többször). Válasszuk a Source Code szöveget a Subobjects területen, és kattintsunk a Create gombra.

A következő ablakban a Title mezőben a programunk címét kell megadnunk. Ez a cím fog megjelenni a program ablakának a címeként. A program típusaként (Type mező) válasszuk az Executable program-ot (futtatható programot). Az ablak többi mezőjét ugyan nem kötelező kitölteni, de a Status mezőbe a listából válasszuk ki a Test program-ot, az Application mezőben pedig a Basis-t. Kattintsunk a Save gombra, majd a következő ablakban a Local Object gombra, és az újonnan nyíló ablakba írhatjuk a kódunkat.

A SAP generál egy pár sort előre. Az első sorban kommentek vannak, aztán a REPORT utasítás a program nevével, amelyet egy pont zár.

A futtatható utasítások mindig a REPORT utasítással kezdődnek, amelyet a program neve követ.

Az utasításokat ponttal (.) zárjuk.

Kommentet kétféleképp lehet írni. A teljes sor komment lesz, ha a sor első karaktere egy * jel. Azonban a *-nak más jelentése van, ha bárhol máshol szerepel a sorban. A másik lehetőség, hogy idézőjelet (“) használunk, aminek eredményeként a sor idézőjeltől jobbra lévő része komment lesz.

Gépeljük be a következő utasítást:

```
write 'Hello '.
```

a `REPORT  Z_FIRST_ABAP.` utasítás után. A WRITE utasítás a rákövetkező sztringet kiírja a képernyőre.

ABAP utasítások nem érzékenyek a kis és nagybetűk közötti különbségre.

A kódunkat könnyedén formázhatjuk a Pretty Printer gomb (Shift+F1) segítségével. A Pretty Printer beállításait a Utility menüben, a Settings, Pretty Printer fülön módosíthatjuk.

A Save gombbal (Ctrl+S) menthetjük el a programunkat.

A program szintaktikáját a Check gombbal (Ctrl+F2) ellenőrizhetjük. A Check gomb hatására a rendszer lefordítja a programot. Ha a rendszer hibát talál, akkor a hibaüzenetek a képernyő alján jelennek meg. Ha nincs hiba, akkor a "Program Z_FIRST_ABAP is syntactically correct" üzenet jelenik meg.

A Direct processing gomb (F8) megnyomásával kipróbálhatjuk a programunkat.

Ha nem aktiváljuk a programunkat, akkor más felhasználók nem látják, így nem is használhatják. Az Activate gombbal (CTRL+F3) aktiválhatjuk a programunkat.

Változódeklaráció és előre definiált egyszerű típusok

A következő utasítással deklarálhatunk változót:

```
DATA változó_név TYPE típus_név.
```

A változó neve

- csak angol betűket, számjegyeket és aláhúzásjelet (_) tartalmazhat,
- betűvel vagy aláhúzásjellel (_) kell kezdődnie,
- maximum 30 karakter hosszú lehet,
- nem tartalmazhat előre definiált ABAP típusneveket vagy előre definiált adatobjektumneveket,
- nem javasolt változónévként foglalt ABAP szót használni.

Az előre definiált egyszerű ABAP típusok 3 kategóriába sorolhatóak: karakterszerű, numerikus és bájtszerű típusok.

A **numerikus adattípusokat** numerikus számításokhoz használjuk. A numerikus típusok a következőek:

- i, egész típus 4-bájton. A rendszer az ilyen típusú számok osztásakor nem csonkolást, hanem kerekítést használ. Számlálók, mennyiségek, indexek, offset-ek és időperiódusok tárolására használják.

- p, pakolt szám típus. A hossza 1 és 16 között lehet. Egy bájton két számjegyet tárol. Az utolsó bájt egy számjegyet és egy előjelet tárol. A tizedes elválasztó után maximum 14 tizedesjegy állhat. A hosszát a LENGTH kulcsszó, míg a tizedejegyek számát a DECIMAL kulcsszó segítségével határozhatjuk meg. Az alapértelmezett hosszúsága 8. Hosszúságok, súlyok, pénzösszegek tárolására használják.
- decfloat16, decimális lebegőpontos szám típus, mely 16 decimális számjegyet tud tárolni,
- decfloat34, decimális lebegőpontos szám típus, mely 34 decimális számjegyet tud tárolni.
- f, bináris lebegőpontos szám típus 8 bájton. A nagy számokat kerekíti, emiatt két ilyen típusú szám egyenlőségét úgy vizsgáljuk meg, hogy vesszük a relatív különbségüket és ha az egy előre definiált korláton belül van, akkor egyenlőek, egyébként nem. Olyan teljesítménykritikus alkalmazásokban használjuk, ahol a pontosság nem fontos.

```
DATA integer01 TYPE i.
DATA packed01 TYPE p LENGTH 4.
DATA packed02 TYPE p LENGTH 4 DECIMALS 1.
```

Az **előredefiniált egyszerű karakterszerű** típusok a következőek:

- c, karakter típus, a hosszúságát, mely maximum 262 143 karakter lehet, megadhatjuk,
- n, numerikus karakter típus. Csak számjegyeket tárol. Megadhatjuk a hosszúságát, amely maximum 262 143 karakter hosszú lehet. Számításokat ne végezzünk az ilyen típusú értékekkel. Például számlaszámok, cikkszámok, irányítószámok tárolására alkalmas.
- d, dátum mező, a formátuma ‘yyyymmdd’.
- t, idő mező, a formátuma ‘hhmmss’
- string, változó hosszúságú karaktersorozatok tárolására.

```
DATA text01 TYPE c LENGTH 30 .
DATA date01 TYPE d.
```

Az SAP javasolja, hogy egy program input adatát c (karakter) típusú változóban tároljuk, mivel ennek fix hosszúsága van.

Az **előredefiniált bájtszerű** típusok a következőek:

- x, hexadecimális érték egy bájton. Megadhatjuk a hosszát, mely maximum 524 287 bájt lehet.
- xstring, bájtok változó hosszúságú sorozata.

A változódeklarációban a típust egy másik változó típusával is megadhatjuk a LIKE kulcsszó segítségével:

```
DATA integer02 LIKE integer01.
```

A VALUE kulcsszó kezdőértéket rendel a változóhoz:

```
DATA integer03 TYPE i VALUE 5.
```

A deklarációban más típusokat is megadhatunk, mint referenciatípus, összetett típus (struktúra, belső tábla), ABAP Dictionary típus vagy felhasználó által definiált típus.

Konstansdeklaráció

A CONSTANTS kulcsszóval lehet konstansokat deklarálni. A VALUE kulcsszó kötelező, ezzel adhatunk a konstansnak kezdőértéket.

```
CONSTANTS const1 TYPE p DECIMALS 1 VALUE '6.6'.
```

Literálok

A **numerikus literálok** számjegyekből és egy opcionális előjelből állnak. Nincs törtrészük. Ha törtszámokra van szükségünk, akkor használjunk szövegmező literálokat. A következő példákban szereplő szöveges literálokat a rendszer a megfelelő numerikus típusúvá konvertálja: '123'; '+122'; '-045'; '-34.023'; '12.2E21'; '-45E-2'.

A **szövegmező literál** előtt és után aposztróf (') áll, a típusa c (karakter), maximum 255 karakter hosszú, és legalább egy karaktert tartalmaz, amely miatt a '' literál megegyezik a ' ' literállal. Ha a literálon belül aposztrófot szeretnénk tenni, akkor használjunk dupla aposztrófot, pl: 'It''s raining.'

A **sztringliterál** hasonló a szövegmező literálhoz, de a nyitó és záró karaktere egy másik fajta aposztróf (`), a típusa sztring, és létezik üres sztringliterál (``).

Értékadás és inicializáció

A MOVE utasítással és az = operátorral rendelhetünk értéket egy változóhoz. Az = operátor előtt és után szóköz szerepel.

```
MOVE 2 TO integer01.
integer02 = 3.
packed01 = 2.
packed02 = '+23.3'.
MOVE '2015.01.01' TO date01.
```

A CLEAR utasítás hozzárendeli egy változóhoz a kezdőértékét.

```
CLEAR integer03.
```

Kifejezés, operátor és precedencia táblázat

Négyféle típusú kifejezés van az ABAP-ban: logikai, aritmetikai, sztring és bit kifejezés. Az aritmetikai, a sztring és a bit kifejezéseket nem lehet keverni. A kifejezésekben lehet kerek zárójeleket használni.

Aritmetikai kifejezések

Az *1. táblázat* az aritmetikai operátorok precedencia táblázatát mutatja. A harmadik oszlopban a műveletek kötési iránya van feltüntetve. A ** operátor a legmagasabb prioritású. Az aritmetikai operátorok előtt és után szóközök állnak.

1. táblázat – Precedencia táblázat

Operátor	**Művelet**	**Kötési irány**
**	hatványozás	jobbról balra
*, /, DIV, MOD	szorzás, osztás, osztás egész része, osztás maradéka	balról jobbra
+, -	összeadás, kivonás	balról jobbra

Sztringkifejezések

A && operátor két karakter típusú operandust fűz össze.

Logikai kifejezések

A logikai kifejezésekben relációs operátorokat (=, <>, <, >, <=, >=, etc.), logikai operátorokat (AND, OR, NOT, EQUIV), BETWEEN, IN, IS kulcsszavakat használhatunk. A relációs operátorok előtt és után szóköz szerepel.

Bit kifejezések

Az ABAP programokban használhatunk bit operandusokat (pl. BIT-AND, BIT-OR, BIT-XOR).

További információkat találhatunk a logikai és bit kifejezésekről a dokumentációban.

Az első selection screen

A PARAMETERS utasítás teszi lehetővé, hogy a programunkban input paramétereket használjunk. Ha a programban szerepel ez az utasítás és futtatjuk a programot, akkor a rendszer automatikusan egy selection screen-t hoz létre, ahová a felhasználó begépelheti az input paraméterek értékeit. A program outputja egy listára kerül, amelyet a rendszer automatikusan generál, ha a program legalább egy WRITE utasítást tartalmaz.

A PARAMETERS utasítás szintaktikája hasonló a DATA utasításéhoz.

```
PARAMETERS pint01 TYPE i.
PARAMETERS pchar02 TYPE c LENGTH 20.
PARAMETERS pdate03 TYPE d.
PARAMETERS pchar04 LIKE pchar02.
```

A paraméter neve maximum 8 karakter hosszú lehet.

A PARAMETERS utasítás egy változót deklarál a programban. Amikor a programot futtatjuk, a rendszer létrehozza a selection screen-t és minden deklarált változóhoz egy input paramétermezőt helyez el rajta, melynek a típusa és a neve megegyezik a változó típusával és nevével.

A paraméternek adhatunk alapértelmezett értéket:

```
PARAMETERS pdate03 TYPE d DEFAULT '2015.04.01'.
```

Az alapértelmezett érték megjelelenik az input mezőben a selection screen-en, amelyet a felhasználó változtathat, ha akar.

HA a PARAMETERS utasításban használjuk az OBLIGATORY kulcsszót, akkor a mezőt kötelező kitölteni.

```
PARAMETERS pint01 TYPE i OBLIGATORY.
```

Hozhatunk létre checkbox-okat a selection screen-eken:

```
PARAMETERS pch01 AS CHECKBOX.
```

Ha a paraméter értéke 'X' vagy 'x', a checkbox ki van pipálva, egyébként nincs kipipálva. A paraméter típusa c, hossza 1.

Használhatunk rádiógombokat is:

```
PARAMETERS ryes    RADIOBUTTON GROUP r1.
PARAMETERS rno     RADIOBUTTON GROUP r1 DEFAULT 'X'.
PARAMETERS rdontno RADIOBUTTON GROUP r1.
```

A rádiógomb ki van választva, ha az értéke 'X' vagy 'x', egyébként nincs kiválasztva. Ugyanazon csoportnevű rádiógombok egy rádiógombcsoporthoz tartoznak. A csoportnév maximum 4 karakter hosszú lehet. A paraméterek típusa c, hossza 1. Egy rádiógombcsoportban csak egy paraméternek lehet alapértelmezett értéke, amelynek 'X'-nek kell lennie.

A következő példa egy egyszerű programot mutat be, amelyben bekérünk két számot, mint input paramétert és kiírjuk a képernyőre az összegüket és a szorzatukat:

```
REPORT  Z_SUM_PRODUCT.
PARAMETERS: a type i,
            b type i.
data s type i.
data p type i.
s = a + b.
p = a * b.
write: 'Sum: ' , s.
write: 'Product: ' , p.
```

Amikor futtatjuk és kitöltöttük az input mezőket, nyomjuk meg az Execute (F8) gombot, hogy a rendszer kiírja a képernyőre az eredményt.

A példában láncolt utasításokat használtunk. Ha több olyan ABAP utasításra van szükségünk, amelyeknek ugyanaz a kezdőutasítása, akkor a sok egyedi utasítás helyett használhatunk egy láncolt utasítást. A kezdő rész után tegyünk kettőspontot (:) és a maradék részeket vesszővel válasszuk el. Például:

```
write: 'Product: ' , p.
```

A program szöveges elemei

Ha futtatjuk az előző példát, akkor láthatjuk, hogy a mezőnevek a változók neveivel egyeznek meg. Az input mezőhöz adhatunk több információt, miután aktiváltuk a programunkat. Ehhez menjünk a Goto menü, Text elements és Selection texts pontba. Ott találjuk a name oszlopban a programunk paramétereinek a neveit, amelyekhez a text oszlopban informatív neveket adhatunk meg vagy információt szerezhetünk a mezőről az adatszótárból. Miután megadtuk a szöveget, aktiválnunk kell a szöveges elemeket.

A szöveges elemeket könnyen lehet változtatni, amikor le kell fordítani őket más nyelvre. Azonban a programunkat ilyen esetekben nem kell újraírni.

Lista (list)

A lista (list) tartalmazza az ABAP programjaink outputját. Az outputot ABAP utasításokkal hozhatjuk létre. A példáinkban ezt az egyszerű technikát használjuk, bár az SAP dokumentáció nem javasolja a használatukat. Helyette azt ajánlja, hogy használjunk ALV technikákat, a Browser Control vagy a Textedit Controls wrappereket.

A következő listautasításokat használjuk a tananyagban:

`WRITE 'apple'.` utasítás a képernyőre (a list-re) írja az 'apple' sztringet. Más adattípusokat is hasonlóan tud kiírni.

`WRITE 'apple' QUICKINFO 'is a kind of fruit'.` utasítás az outputhoz egy gyorsinfót rendel. Ha futtatjuk a programot és az egérrel ráállunk az 'apple' szövegre, akkor a megadott infó megjelenik.

`ULINE` utasítás egy vízszintes vonalat hoz létre a listán.

`WRITE /.` utasítás egy új sort hoz létre.

`WRITE / 'apple tree'.` utasítás új sorba írja az 'apple tree' sztringet.

`NEW-LINE` utasítás egy új sort hoz létre.

Szöveges szimbólumok

A több nyelvet támogató programok literáljait kicserélhetjük. A Goto menü, Text elements, Text symbols pontban szöveges szimbólumokat (text symbols) deklarálhatunk. A programkódban a Sym-beli értéket használjuk, mely egy 3 jegyű szám (pl. 001), míg a képernyőn a Text (pl. 'banán') jelenik meg, mely maximum 132 karakter hosszú lehet. A programkódból a szöveges szimbólumra a text-001-gyel hivatkozhatunk, azaz a szöveges szimbólumokat a rendszer egy text nevű struktúrában tárolja, amelynek a mezői 3 jegyű számok.

`WRITE text-001.` utasítás megjeleníti a képernyőn a 'banán' sztringet.

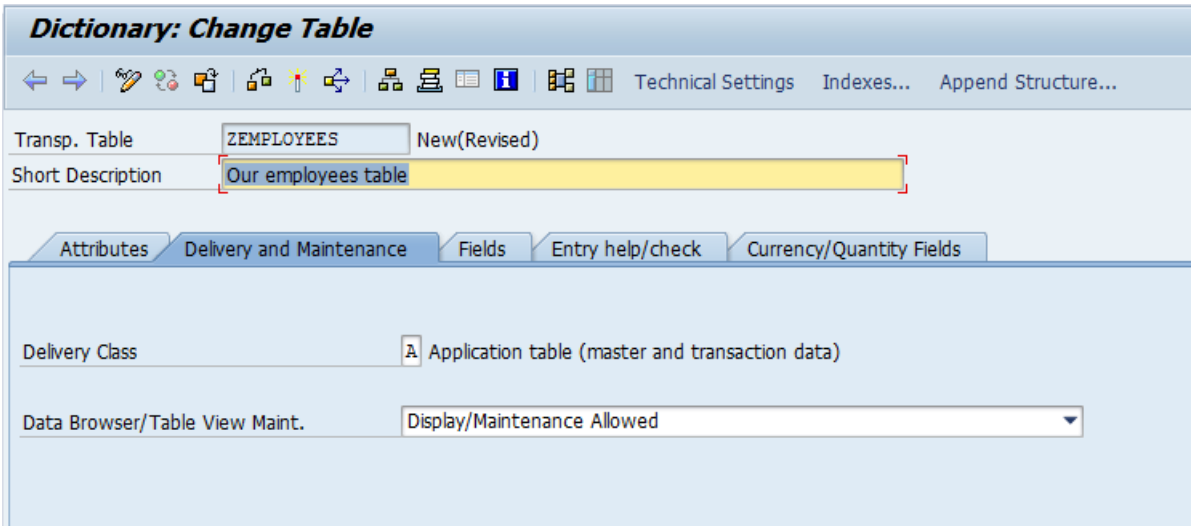
Az adatszótár (Data Dictionary)

Az adatszótárban kereshetünk, létrehozhatunk, módosíthatunk és eldobhatunk táblákat, nézeteket, adattípusokat és más objektumokat. Sok beépített objektumot tartalmaz az adatszótár, ezeket, ha szükséges kibővíthetjük. Az adatszótárat a SE11 tranzakcióval vagy az SAP menü segítségével nyithatjuk meg: Tools, ABAP Workbench, Development, ABAP Dictionary.

Tábla létrehozása

Ha táblát akarunk létrehozni, akkor írjuk a nevét a Database table mezőbe és nyomjuk meg a Create gombot. A tábla nevének Z-vel vagy Y-nal kell kezdődnie (a Z-t használják gyakrabban). Az első példánk a ZEMPLOYEES tábla.

Az *1. ábra* a “Dictionary: Change Table” ablak Delivery and Maintenance fülét mutatja, ahol ki kell választanunk a Delivery Class-t. Az egyszerű példáinkhoz válasszuk az Application table (master and transaction data) lehetőséget. Ez a táblatípus más néven a transparent tábla, amelyre tekinthetünk úgy, mint egy relációs adatbázis táblára.



1. ábra – Dictionary: Change Table ablak

A Data Browser/Table View Maint. mezőben 3 érték közül választhatunk Ha a "Display/Maintenance Allowed"-t választjuk, akkor a felhasználók (ha van joguk) a Data Browser (SE16), Maintain Table Views (SM30 and SM31) és a Generate Table Maintenance Dialog (SE54) eszközökkel megnézhetik és karbantarthatják a táblát. Ha a "Display/Maintenance Allowed with Restrictions"-t választjuk, akkor a felhasználók megnézhetik a táblát a Data Browser (SE16) eszközzel vagy az SE54 tranzakcióval karbantartó dialógust generálhatnak a táblához. Ha a "Display/Maintenance Not Allowed"-t választjuk, akkor a felhasználók egyik tranzakcióval sem nézhetik meg vagy tarthatják karban a táblát.

A ZEMPLOYEES táblához válasszuk a "Display/Maintenance Allowed"-t a Data Browser/Table View Maint. mezőben.

A táblához megadhatunk rövid leírást (short description).

Ugyan nem vagyunk készen, de most elmenthetjük a táblát, ezért nyomjuk meg a Save (Ctrl+S) gombot. A következő ablakban kattintsunk a Local Object gombra, amely azt fogja jelenteni, hogy a tábla csak a fejlesztői rendszerben fog létezni, és nem fogjuk transzportálni.

A Fields fülön megadhatjuk a tábla mezőit vagy oszlopait. Az első mezőnek a "Client" mezőnek kell lennie, amelynek az adattípusát "MANDT" kell választani, és az elsődleges kulcs részének kell lennie. Ezt a mezőt a rendszer automatikusan kezelni fogja. Ebben a mezőben a rendszer annak a kliensnek a számát tárolja, amelyiket a táblát használó

felhasználó épp használja. A transparent táblának mindig kliensspecifikusnak (vagy más néven mandantfüggőnek) kell lennie.

A táblához meg kell határoznunk az elsődleges kulcsát, amely a legtöbb esetben összetett. Az elsődleges kulcs részét alkotó mezőknél pipáljuk be a kulcs mezőt. A kulcsmezőknek a mezőlista elején kell állniuk.

Egy mezőhöz meg kell adnunk az adatelemét az adatelem oszlopban vagy egy előre definiált adattípust hosszúsággal a Data Type, Length és a Decimal Places oszlopokban. A kétféle adattípus megadás között "Data Element"/"Predefined Type" gombbal lehet váltani.

Az előre definiált típusnak beépített típusnak kell lennie, amelyet az F4 megnyomásával egy listából választhatunk ki. Hasonlóan az adataelemet is kiválaszthatjuk listából az F4 megnyomásával, de új adatelemet is hozhatunk létre, ha beírjuk a nevet a megfelelő mezőbe és kétszer kattintunk rajta.

A mezőkhöz adhatunk rövid leírást.

Mielőtt aktiváljuk a táblát, karban kell tartani a technikai beállításokat a "Technical settings" gombbal. Ki kell választani egy adatosztályt, a példánkhoz ez legyen a "Master Data, Transparent Tables", és egy méretkategóráit, ahol megbecsüljük, hogy a táblánknak kb. hány sora lesz.

Ha készen vagyunk a tábladefinícióval, akkor mentsük el, (Ctrl+S), ellenőrizzük (Ctrl+F2), és ha nincs probléma, aktiválhatjuk (Ctrl+F3). Az aktiválás azt jelenti, hogy a rendszer létrehozza a táblát az adatbázisban és ezután azt lehet használni, azaz lehet beszúrni új sort, módosítani, törölni sorokat, vagy hivatkozni rá egy programból.

Adatelem, domain és adattípus

A tábla mezőjéhez adattípust kell valahogyan rendelnünk. A legegyszerűbb mód az, amikor egy előre definiált típust használunk, azonban ezt nem javasoljuk, mert a mezőhöz így nem tudunk több információt rendelni. Jobb megoldás, ha egy adatelemet hozunk létre, amelyhez megadhatunk mezőcímkéket, amelyek a programjainkban megjelennek. Az adatelemekhez kereső információkat is megadhatunk. Ha egy adattípust és hosszúságot többször használunk vagy tartományt szeretnénk megadni az adattípusunkhoz, akkor érdemes domain-t létrehozni. Ha a domain-t változtatjuk, akkor minden olyan adatelem típusa, amelyhez a domain-t rendeltük, változni fog.

Az adatelem létrehozásához nyissuk meg a Data Dictionary-t az SE11 tranzakcióval, válasszuk a Data type mezőt, írjuk be a nevét és nyomjuk meg a Create gombot. A másik lehetőség, hogy amikor a táblát hozzuk létre, az új adatelem nevét beírjuk az adatelem oszlopba és kétszer kattintunk rajta. Válasszuk az "Data element" opciót, majd nyomjunk entert.

A "Dictionary: Change Data Element" ablakban meg kell adnunk az adatelem rövid leírását a Short description mezőben, majd az adatelemünkhöz válasszuk ki a domain-t vagy az előre definiált típust. Az előre definiált típust egy beépített listából kell kiválasztani, míg a domain lehet beépített vagy létrehozhatjuk mi is. A "Further Characteristics" fülön az adatelemhez keresési segítséget adhatunk meg, míg a "Field Label" fülön mezőcímkéket, amelyek az adatelemet használó képernyőkön fognak megjelenni.

Ha készen vagyunk, mentsük el (Ctrl+S), válasszuk a "Local Object" gombot, ellenőrizzük (Ctrl+F2), és aktiváljuk (Ctrl+F3), hogy a rendszerben használni lehessen.

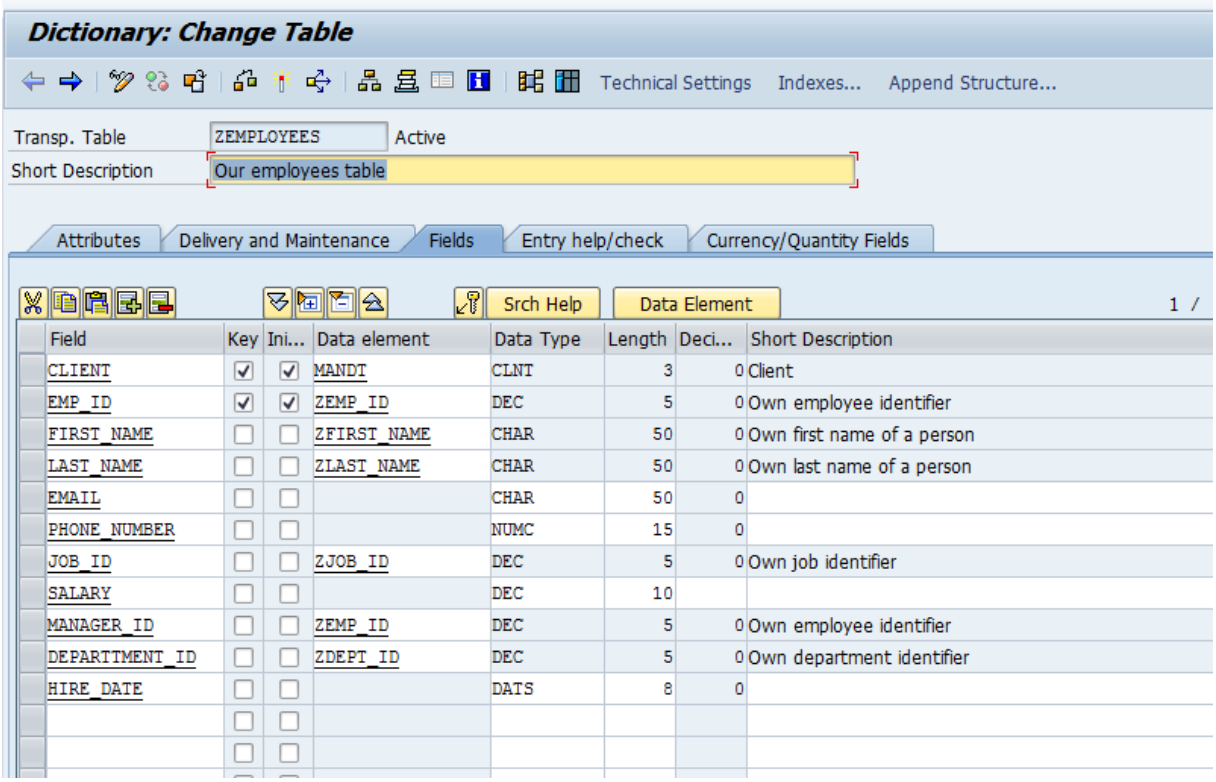
Domain létrehozásához nyissuk meg a Data Dictionary-t az SE11 tranzakcióval, válasszuk a Domain mezőt, írjuk be a nevét, és nyomjuk meg a Create gombot. A másik lehetőség, hogy amikor adatelemet hozunk létre, írjuk be az új domain nevét a domain mezőbe és kattintsunk kétszer rajta.

A "Dictionary: Change Domain" ablakban meg kell adni a domain rövid leírását a Short description mezőben, aztán ki kell választani egy adattípust és megadni a hosszúságot. A "Value Range" fülön az értékekhez megszorításokat rendelhetünk egyszerű értékek, intervallumok vagy egy értéktábla segítségével.

Ha készen vagyunk, mentsük el (Ctrl+S), válasszuk a "Local Object" gombot, ellenőrizzük, (Ctrl+F2), és aktiváljuk (Ctrl+F3), hogy a rendszerben használni lehessen.

A ZEMPLOYEES tábla

A tanagyagban a ZEMPLOYEES táblát használjuk. A *2. ábrán* láthatjuk a definícióját.



2. ábra – A ZEMPLOYEES tábla létrehozása

A MANDT egy beépített adatelem, ZEMP_ID, ZJOB_ID, ZDEPT_ID, ZFIRST_NAME és ZLAST_NAME oszlopok adatelemei saját adatelemek, melyeknek saját domain-uk van.

Rekord beszúrása táblába

A “Dictionary: Change Table” ablakban válasszuk a Utilities menü, Table Contents és Create Entries pontot. Ha kitöltjük a táblamezők technikai nevével szereplő mezőket és megnyomjuk a Save (Ctrl+S) gombot, akkor beszúrtunk egy sort a táblába. Ha a mezők címkéit szeretnénk látni a mezők nevei helyett, akkor a Settings menü User Parameters menüpontjában válasszuk a Keyword-nél a Field label-t a Field name helyett.

Tábla tartalmának a megjelenítése

A “Dictionary: Change Table” ablakban válasszuk a Utilities menü, Table Contents és Display pontot. Egy selection screen nyílik meg, ahol szűrőket lehet beállítani. Nyomjuk meg az Execute (F8) gombot, hogy a szűrőnek megfelelő táblatartalmat listázza a rendszer. Természetesen nem szükséges szűrőt beállítani, ekkor a rendszer minden sort meg fog mutatni.

Ha duplán kattintunk egy soron, akkor a sor minden adatát láthatjuk. Ha más műveletekre is van igényünk, érdemes kipróbálni az eszköztár gombjait.

A Data Browser (SE16 tranzakció) is megmutatja a táblák tartalmát.

Tábla, adattípus vagy domain másolása és törlése

Először meg kell találnunk az objektumot az adatszótárban (Data Dictionary) (SE11), azaz írjuk be a nevét a megfelelő mezőbe.

Az eszköztár a Copy (Ctrl+F5) gombja segítségével másolhatjuk az objektumot. Ha táblát másolunk, a rendszer egy üres táblastruktúrát fog létrehozni adatok nélkül.

Csak a felhasználók által létrehozott objektumokat lehet törölni, a beépített objektumokat nem. Mielőtt törlünk egy objektumot, nyomjuk meg a "Where-used-list" (Ctlr+Shift+F3) gombot, hogy megtaláljuk azokat az objektumokat (programokat, táblákat), amelyek hivatkoznak a törölni kívánt oszlopra. Ha valóban törölni akarunk egy objektumot, akkor nyomjuk meg a Delete (Shift+F2) gombot.

Tábla oszlopainak módosítása

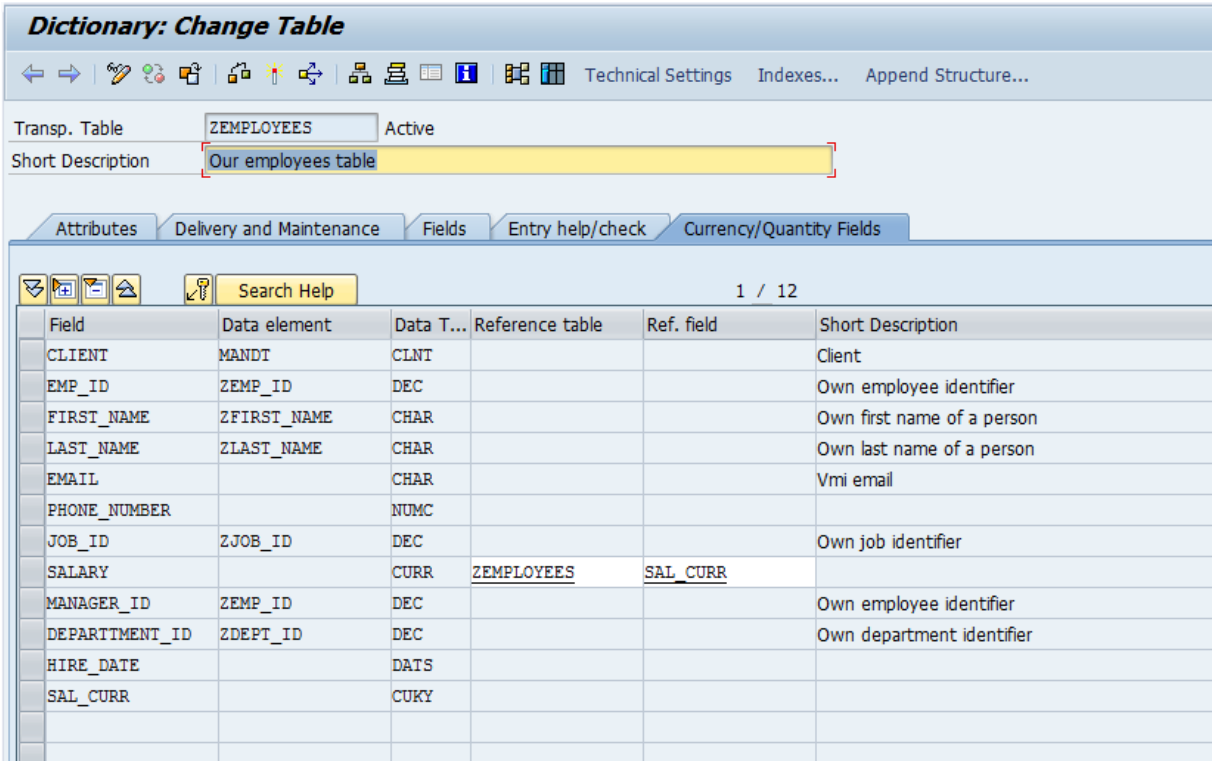
Nyissuk meg a Data Dictionary-t (SE11), írjuk be a módosítandó tábla nevét a Database table mezőbe és nyomjuk meg a Change gombot.

Ha törölni akarjuk a tábla egy oszlopát, akkor kattintsunk jobb egérgombbal a megfelelő soron, és válasszuk a törlést.

Ha új oszlopot akarunk adni a táblához, akkor írjuk a mezőlista végére azt, vagy az egyik oszlopon kattintsunk a jobb egérgomb, és válasszuk a beszúrást.

Pénznem (Currency)

A SALARY mezőhöz meg kell határoznunk a pénznemét, ezért a típusa CURR legyen. Változtassuk meg és adjuk meg a hosszát. Így az oszlop csak az összeget határozza meg, de a pénznemet nem. A pénznem meghatározásához szükségünk van egy másik mezőre SAL_CURR néven, amelynek a típusa CUKY legyen, és a SALARY mezőnek hivatkoznia kell rá. Kattintsunk a "Currency/Quantity Fields" fülre és állítsuk be Reference table és a Ref. field oszlopok értékét a SALARY mező sorában, ahogy a *4. ábra* mutatja.

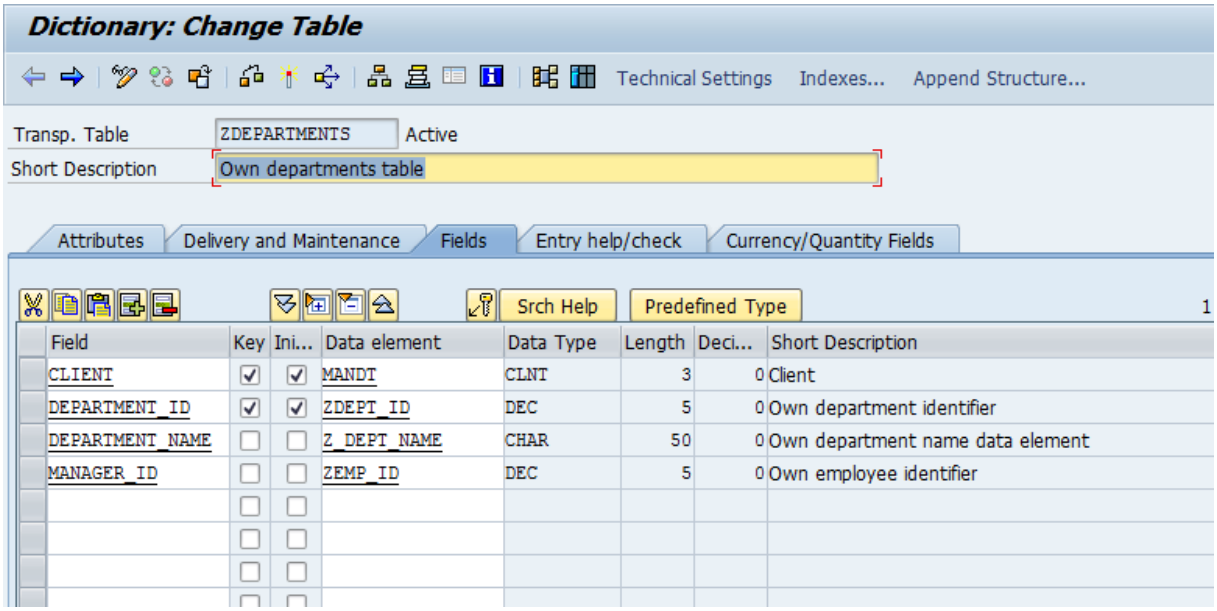


4. ábra – Hivatkozás a pénznemre

Ha aktiváljuk a táblát, a rendszer lehet, hogy megkér, hogy konvertáljuk a táblát az SE14 tranzakcióval. Nyissuk meg a Utilities menü, Database Objects, Database utility pontot (vagy futtassuk a SE14 tranzakciót). Az új ablakban nyomjuk meg az activate and adjust database gombot.

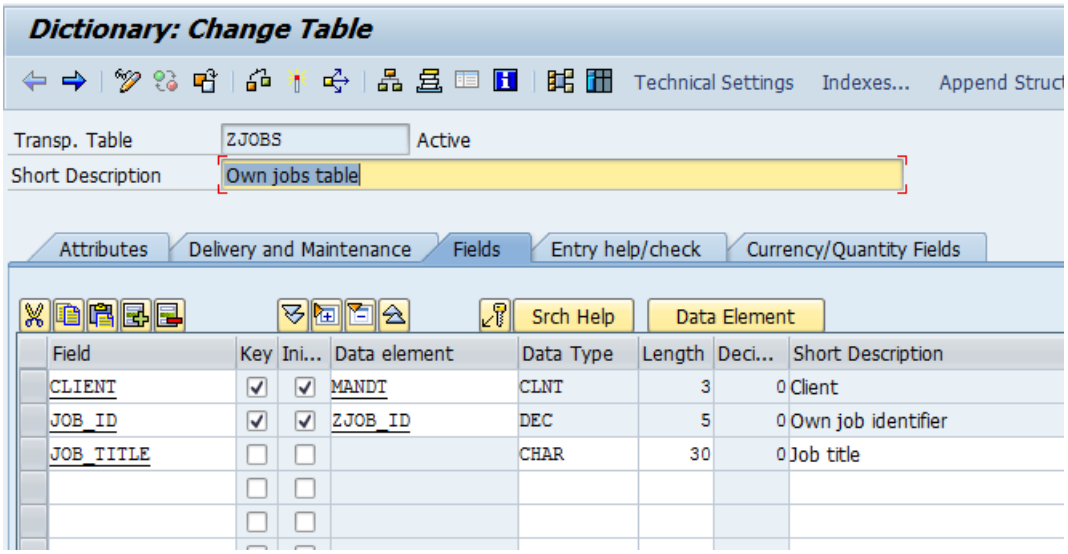
A ZDEPARTMENTS és a ZJOBS táblák

Az *5. ábrán* a ZDEPARTMENTS tábla definícióját, míg a *6. ábrán* a ZJOBS tábla definícióját láthatjuk.



5. ábra – ZDEPARTMENTS tábla definíciója

A DEPARTMENT_ID adatelemének ugyanaz a típusa, mint a ZEMPLOYEES tábla DEPARTMENT_ID oszlopának adattípusa, míg a MANAGER_ID adattípusa ugyanaz, mint a ZEMPLOYEES tábla EMP_ID mezőjének adattípusa. A Z_DEPT_NAME egy felhasználó által definiált adatelem.



6. ábra – ZJOBS tábla definíciója

A JOB_ID adateleme ugyanolyan típusú, mint a ZEMPLOYEES tábla JOB_ID adattípusa.

Külső kulcs

Az SAP nem javasolja, hogy adatbázisszinten hozzunk létre külső kulcsokat, mivel a rendszer sokféle adatbázis-kezelő rendszert használhat. Helyette az Entry help/check fülön kattintsunk a foreign keys-re és töltsük ki a Check table, Origin of the input help és Srch Help mezőket.

Elágaztató utasítások

Az ABAP programozási nyelvben két feltételes elágaztató utasítás van, az IF és a CASE utasítás.

IF utasítás

```
IF logikai_kifejezés_1.
  [utasítások_1]
[ELSEIF logikai_kifejezés_2.
  [utasítások_2]]…
[ELSE.
  [utasítások_n]]
ENDIF.
```

Az első igaz logikai kifejezés mögött álló utasítássorozat fog lefutni. Ha egyik logikai kifejezés sem igaz, akkor az ELSE kulcsszó után álló utasítások futnak le, ha az ELSE rész létezik, különben az IF utasítás nem csinál semmit.

```
REPORT  Z_IF.
parameters sales type i.
data bonus  type i value 0.

IF sales > 50000.
   bonus = 1500.
 ELSEIF sales > 35000.
   bonus = 500.
 ELSE.
   bonus = 100.
ENDIF.

WRITE: 'Sales = ' , sales, ', bonus = ' , bonus.
```

CASE utasítás

```
CASE operandus.
  [WHEN operandus1 [OR operandus2 [OR operandus3 [...]]].
    [utasítások_1]]
    ...
  [WHEN OTHERS.
    [utasítások_n]]
ENDCASE.
```

Ha a CASE kulcsszó után álló operandus egyenlő az első WHEN után álló operandusok egyikével, akkor az utasítások_1 utasítássorozat fog lefutni, különben a következő WHEN ágat vizsgálja. Ha nem talál egyetlen olyan operandust sem, amellyel egyenlő lenne, akkor a WHEN OTHERS utáni utasítások futnak le, ha létezik. Csak egy WHEN rész utáni utasítássorozat fut le.

A következő példa bekér egy érdemjegyet, és kiírja a hozzá tartozó szöveget a képernyőre.

```
REPORT  Z_CASE_STATEMENT.
parameters grade type i.
case grade.
  when 1.
    write 'Fail'.
    write /'You should learn it again'.
  when 2.
    write 'Pass'.
    write /'You can learn the next subjects'.
  when 3.
    write 'Satisfactory'.
    write /'You can learn the next subjects'.
  when 4.
    write 'Good'.
    write /'You can learn the next subjects'.
  when 5.
    write 'Excellent'.
    write /'You can learn the next subjects'.
  when others.
    write 'Grade is not known'.
endcase.
```

Ciklusok

Az ABAP programozási nyelvben a következő ciklusokat használhatjuk:

- DO – ENDDO,
- WHILE – ENDWHILE,
- LOOP – ENDLOOP (belső táblából olvas sorokat)
- PROVIDE – ENDPROVIDE (belső táblákkal dolgozik)
- SELECT – ENDSELECT (egy adatbáziselérés eredményét dolgozza fel)

DO – ENDDO ciklus

```
DO [n TIMES].
  [utasítás_blokk]
ENDDO.
```

Az n TIMES meghatározhatja, hányszor fusson a ciklusmagbeli utasítássorozat. Az n értékét a rendszer csak egyszer, a futás elején értékeli ki. Ha nem adjuk meg az n TIMES kiegészítést, akkor végtelen ciklust írtunk. A ciklusból kiléphetünk az EXIT utasítással.

A ciklus magjában használhatjuk az sy-index rendszermezőt, amely az ismétlődések számát tartalmazza. Beágyazott ciklusok esetén az sy-index mindig az aktuális ciklusra hivatkozik.

```
REPORT  Z_DO.
do 3 times.
 write sy-index.
enddo.
```

WHILE – ENDWHILE ciklus

```
WHILE logikai_kifejezés.
  [utasítások_blokk]
ENDWHILE.
```

A ciklus magja addig ismétlődik, amíg a logikai kifejezés igaz, vagy amíg egy EXIT utasítással ki nem lépünk. Ebben a ciklusban is használhatjuk az sy-index rendszermezőt.

```
REPORT  Z_WHILE.
parameters sz type i.

if sz = 0.
  write 0.
  else.
  if sz < 0.
    write -1.
    sz = sz * ( -1 ).
  endif.

  data c type i value 2.
  while sz <> 1.
    if sz mod c = 0.
        sz = sz / c.
        write c.
      else. c = c + 1.
    endif.

  endwhile.
endif.
```

EXIT utasítás

```
EXIT.
```

A cikluson belüli EXIT utasítás befejezi az aktuális ciklus futását. A program futása a ciklus záró utasítása után folytatódik.

```
REPORT  Z_EXIT.
parameters x type i.
data: c type i value 1,
      sz type i value 0.

if x <= 0.
  write 'Only positive numbers'.
else.

  do.
    sz = sz + c.
    if sz + c + 1 > x.
      exit.
    endif.
    c = c + 1.
  enddo.
  write c.
endif.
```

CHECK utasítás

```
CHECK logikai_kifejezés.
```

Ha a cikluson belüli CHECK után álló logikai kifejezés értéke nem igaz, akkor megszakítja a ciklus futását, és a következő ciklusfutással folytatódik.

CONTINUE utasítás

```
CONTINUE.
```

A cikluson belüli CONTINUE megszakítja a ciklus futását, és a következő ciklusfutással folytatódik.

```
REPORT  Z_CONTINUE.
parameters p type i.
do p times.
  if sy-index mod 2 <> 0.
    continue.
  endif.
  check sy-index mod 3 = 0.
  write sy-index.
enddo.
```

Struktúrák

Struktúrát a következőképp deklarálhatunk:

```
TYPES: BEGIN OF countries,
  c_id type i,
  c_name type c length 40,
  c_continent type c length 30,
 END OF countries.
```

A mezők deklarációja hasonló a változódeklarációhoz, azaz meg lehet adni hosszúságot vagy használhatjuk a LIKE kulcsszót. Egy struktúrának legalább egy mezőjének lennie kell. A struktúrák egymásba ágyazhatóak.

```
TYPES: BEGIN OF address_type,
         name TYPE c LENGTH 30,
         street TYPE street_type,
         BEGIN OF city,
           zipcode TYPE n LENGTH 5,
           name TYPE c LENGTH 40,
         END OF city,
       END OF address_type.ó
```

Ha a TYPES kulcsszó helyett a DATA kulcsszót használjuk, akkor közvetlenül deklarálunk struktúratípusú változót.

Struktúratípusú változót nem csak lokálisan definiált típussal deklarálhatunk, hanem adatszótárbeli struktúra vagy adatbázisbeli tábla segítségével. A következő példában a zemployees egy adatbázisbeli tábla (amelyet korábban létrehoztunk), míg a zname egy adatszótárbeli struktúra (amelyet a SE11 tranzakció segítségével hozhatunk létre).

```
data: vc1 type countries,
      vc2 type countries,
      ve type zemployees,
      vn type zname.
```

A struktúra mezőit kötőjel segítségével hivatkozhatjuk:

```
vc1-c_id = 1.
vc1-c_name = 'Hungary'.
vc1-c_continent = 'Europe'.
```

A MOVE-CORRESPONDING utasítás a forrásstruktúra tartalmát a célstruktúrába másolja. A két struktúrának nem feltétlenül ugyanaz a típusa.

```
MOVE-CORRESPONDING struc1 TO struc2.
```

A rendszer a struct1 elemeit a struct2 azonos nevű elemeibe másolja, míg a többi mezőt nem változtatja.

```
move-corresponding vc1 to vc2.

write: vc2-c_id, vc2-c_name, vc2-c_continent.
```

Belső táblák

A belső táblák dinamikus objektumok, amelyek ugyanolyan típusú elemek sorozatát tartalmazzák. A belső tábla elemei bármilyen típusúak lehetnek. A belső táblákat a programokban adatbázisbeli táblák adatainak tárolására és formázására használjuk. Csak addig léteznek, amíg a program fut.

A sortípus a belső tábla adatelemeinek típusa. Egy elemnek vannak mezői, így a mezők értékeire hivatkozhatunk oszlopként.

Egy beágyazott tábla kulcsa kulcsmezőkből áll. A kulcsok szerint lehet rendezni a belső táblákat. A kulcs lehet egyedi vagy nemegyedi.

3 féle belső táblát használhatunk az ABAP nyelvben: standard, rendezett és hash.

A standard táblát a rendszer index segítségével adminisztrálja. Rendelkezik egy nemegyedi kulccsal. Az elemeit kulcs szerint vagy index szerint érhetjük el. A kulcs szerinti elérés válaszideje a tábla elemszámával arányos. Több kulcsot is definiálhatunk egy táblához, hogy hatékonyabban tudjuk kulcs szerint kinyerni az adatokat. Ha nem definiálunk kulcsot a standard belső táblához, akkor a rendszer automatikusan generál egy nemegyedi kulcsot. Gyorsan lehet új adatokat felvinni a táblába, mert a rendszernek nem kell ellenőriznie a duplikátumokat.

A rendezett táblát (sorted tábla) a rendszer index segítségével adminisztrálja. Rendelkezik egy egyedi vagy egy nemegyedi kulccsal. Az elemek a kulcs szerinti növekvő sorrendben vannak tárolva. Az elemeket kulcs és index szerint érhetjük el. A kulcs szerinti elérés válaszideje logaritmikusan arányos a tábla elemeinek számával, mivel bináris keresést használ. Egy rendezett tábla tartalmának a módosítása után a rendszer automatikusan rendezi azt.

A hash táblát a rendszer egy hash függvény segítségével adminisztrálja. Rendelkezik egy egyedi kulccsal. Kulcs szerint érhetjük el az elemeit. A kulcs szerinti elérés válaszideje

konstans, független a tábla elemeinek a számától. Ha egy táblában gyorsan akarunk kulcs szerint keresni, akkor válasszuk a hash belső táblát.

A belső tábla típusokat deklarálhatunk lokálisan a programban vagy az adatszótárban.

```
data st_tab1 type standard table of countries with non-unique key c_id.
data st_tab2 type standard table of countries.
data so_tab1 type sorted table of countries with non-unique key c_id.
data so_tab2 type sorted table of countries with unique key c_id.
data ht_tab1 type hashed table of countries with unique key c_id.
```

Műveletek a belső táblákkal

Belső táblák feltöltése

INSERT

INSERT utasítás egy vagy több sort szúr be egy belső táblába. Meghatározhatjuk az új elem pozícióját a belső táblában.

```
REPORT  Z_INTERNAL_INSERT.

DATA: BEGIN OF line,
            col1 TYPE i,
            col2 TYPE i,
          END OF line.

DATA: itab LIKE STANDARD TABLE OF line,
      jtab LIKE itab,

      itab1 LIKE TABLE OF line,
      jtab1 LIKE itab,
      itab2 LIKE STANDARD TABLE OF line,
      jtab2 LIKE SORTED TABLE OF line
            WITH NON-UNIQUE KEY col1 col2.

* Fill table

DO 3 TIMES.
  line-col1 = sy-index.
  line-col2 = sy-index ** 2.
  APPEND line TO itab.
  line-col1 = sy-index.
  line-col2 = sy-index ** 3.
  APPEND line TO jtab.
ENDDO.

* Insert a single line into an index table

MOVE itab TO itab1.

line-col1 = 11. line-col2 = 22.
INSERT line INTO itab1 INDEX 2.
```

```
*   INSERT INITIAL LINE INTO itab1 INDEX 1.

LOOP AT itab1 into line.
  WRITE: / line-col1, line-col2.
endloop.

* Insert lines into an index table with LOOP

MOVE itab TO itab1.

LOOP AT itab1 INTO line.
  line-col1 = 3 * sy-tabix. line-col2 = 5 * sy-tabix.
  INSERT line INTO itab1.
ENDLOOP.

LOOP AT itab1 into line.
  WRITE: / line-col1, line-col2.
endloop.

* Insert lines into an index table

MOVE itab TO itab1.
MOVE jtab TO jtab1.

INSERT LINES OF itab1 INTO jtab1 INDEX 1.

LOOP AT jtab1 into line.
  WRITE: / line-col1, line-col2.
endloop.
* Insert lines into a sorted table

MOVE itab TO itab2.
MOVE jtab TO jtab2.

INSERT LINES OF itab2 INTO TABLE jtab2.

LOOP AT jtab2 into line.
  WRITE: / line-col1, line-col2.
endloop.
```

APPEND

Az APPEND utasítás egy vagy több sort fűz egy belső standard vagy rendezett táblához. Az új sorok index szerint az utolsóak lesznek. Az APPEND utasítás beállítja a sy-tabix rendszermezőt, amely az utoljára beszúrt sor indexbeli sorszámát adja meg.

COLLECT

A COLLECT utasítás egy változó tartalmát beszúrja egy belső táblába vagy egy egyedi sorként, vagy ha léteznek sorok, amelyeknek az elsődleges kulcsa megegyezik az új sor megfelelő mezőivel, akkor a beszúrandó sor numerikus komponenseinek az értékeit a megfelelő a sorokhoz adja.

```
REPORT  Z_INTERNAL_COLLECT.
DATA: BEGIN OF dept,
        dept_id   TYPE zdepartments-department_id,
        name    TYPE zdepartments-department_name,
        number_of_emp TYPE i,
      END OF dept.

DATA dept_tab LIKE HASHED TABLE OF dept
               WITH UNIQUE KEY dept_id name.

SELECT department_id department_name
       FROM zdepartments
       INTO dept.
  COLLECT dept INTO dept_tab.
ENDSELECT.

LOOP AT dept_tab into dept.
  WRITE: / dept-dept_id, dept-name, dept-number_of_emp.
endloop.
```

Belső táblák olvasása

READ TABLE

A READ TABLE utasítás a belső táblából egy sort olvas. A sort kulcs vagy index segítségével határozhatjuk meg. Ha több megfelelő sor létezik, akkor az elsőt olvassa fel. Azt is meg kell határoznunk, hogy a felolvasott sort hol és hogyan tároljuk.

```
REPORT  Z_INTERNAL_READ.

DATA: m   TYPE i,
      itab TYPE STANDARD TABLE
           OF i
           WITH NON-UNIQUE KEY table_line.

DO 10 TIMES.
  m = 2 * sy-index.
  INSERT m INTO TABLE itab.

ENDDO.

LOOP AT ITAB into m.
  WRITE: / m.
endloop.

read table itab index 3 into m.
write: / m.
```

Az itab belső tábla elemei nem struktúrák, az egyetlen oszlopára a table_line-nal lehet hivatkozni.

```
REPORT  Z_INTERNAL_READ_KEY.
DATA: BEGIN OF dept,
        dept_id   TYPE zdepartments-department_id,
        name    TYPE zdepartments-department_name,
        number_of_emp TYPE i,
      END OF dept.

DATA dept_tab LIKE HASHED TABLE OF dept
               WITH UNIQUE KEY dept_id name.

SELECT department_id department_name
       FROM zdepartments
       INTO dept.
  COLLECT dept INTO dept_tab.
ENDSELECT.

read table dept_tab with key name = 'IT' into dept.
write: / dept-dept_id, dept-name, dept-number_of_emp.
```

LOOP AT

A LOOP utasítás egy belső tábla sorait sorban felolvassa. Meghatározhatunk logikai feltételt, hogy korlátozzuk a felolvasandó sorokat.

Belső táblák tartalmának módosítása

MODIFY

A MODIFY utasítás egy belső tábla meghatározott sorainak a tartalmát módosítja.

```
REPORT  Z_INTERNAL_MODIFY2.

DATA: BEGIN OF line,
            col1 TYPE i ,
            col2 TYPE i ,
          END OF line.

DATA itab like standard TABLE OF line.

DO 10 TIMES.
  line-col1 = 2 * sy-index.
  line-col2 = 5 * sy-index.
  append line TO itab.
ENDDO.

LOOP AT itab into line.
  if line-col1 mod 3 = 0.
    line-col2 = line-col1 / 3.
    modify itab from line.
  endif.
ENDLOOP.

loop at itab into line.
  write: / line-col1, line-col2.
endloop.
```

DELETE

A DELETE utasítás egy belső tábla meghatározott sorait törli. A törlendő sorokat többféleképp is meghatározhatjuk, mint pl. index-szel, kulccsal, egy ciklus kulcsával vagy ciklusfeltétellel. A következő példákban index szerinti törlésre és ciklusfeltétel szerinti törlésre láthatunk példát.

```
delete dept_tab index 2 .
```

```
REPORT  Z_INTERNAL_DELETE.
DATA: BEGIN OF line,
            col1 TYPE i ,
            col2 TYPE i ,
          END OF line.

DATA itab like standard TABLE OF line.

DO 10 TIMES.
  line-col1 = 2 * sy-index.
  line-col2 = 5 * sy-index.
  append line TO itab.
ENDDO.

LOOP AT itab INTO line where col1 > 15 .
  DELETE itab.
ENDLOOP.

loop at itab into line.
   write: / line-col1, line-col2.
  endloop.
```

SORT

A SORT utasítás egy belső táblát növekvő vagy csökkenő sorba rendez. A BY kulcsszó után a sor egy vagy több komponensét adhatjuk meg.

```
sort itab by col2 descending col1 ascending.
```

Keresés a belső táblában

FIND IN TABLE és REPLACE IN TABLE

Mindkét utasítás egy mintában megadott sztringet keres egy sztringelemű standard belső táblában. A FIND IN TABLE csak keres, míg a REPLACE IN TABLE a keresett sztringeket kicseréli.

```
REPORT  Z_INTERNAL_FIND.

DATA itab TYPE TABLE OF string.
data v type string.

APPEND 'banana - kg' TO itab.
APPEND 'apple - kg' TO itab.
APPEND 'bread - pc' TO itab.
APPEND 'yoghurt - pc' TO itab.

REPLACE ALL OCCURRENCES OF 'pc'
  IN TABLE itab WITH 'piece'
  RESPECTING CASE.

loop at itab into v.
  write: / v.
endloop.

data  results TYPE match_result_tab.
data rl like line of results.

FIND ALL OCCURRENCES OF 'kg'
  IN TABLE itab
  RESPECTING CASE
  RESULTS results.

LOOP AT results INTO rl.
  READ TABLE itab INTO v index rl-line.
  IF sy-subrc = 0.
    WRITE:/ 'Line:', rl-line, 'Offset:', rl-offset.
    WRITE: 'Value is:',  v+rl-offset.
  ENDIF.
ENDLOOP.
```

Belső táblák intervallumösszekapcsolása

PROVIDE – ENDPROVIDE

Ez a ciklus több belső táblán dolgozik egyszerre. A belső táblákban intervallumok vannak megadva. A PROVIDE utasítás megtalálja, hogy az intervallumok között hol van átfedés.

```
REPORT  Z_INTERNAL_PROVIDE.
DATA: BEGIN OF v1,
        lower_bound TYPE i,
        upper_bound TYPE i,
        element TYPE string,
      END OF v1.

DATA: BEGIN OF v2,
        lower_bound TYPE i,
        upper_bound TYPE i,
        element TYPE string,
      END OF v2.
      DATA: BEGIN OF r,
        lower_bound TYPE i,
        upper_bound TYPE i,
```

```
        element TYPE string,
      END OF r.

DATA: itab1 LIKE STANDARD TABLE OF v1,
      itab2 LIKE STANDARD TABLE OF v2,
      itab_r like standard table of r.

DATA: flag1 TYPE c length 1,
      flag2 TYPE c length 1.

v1-lower_bound = 1.
v1-upper_bound = 4.
v1-element = 'table1 interval1'.
APPEND v1 TO itab1.

v1-lower_bound = 10.
v1-upper_bound = 12.
v1-element = 'table1 interval2'.
APPEND v1 TO itab1.

v2-lower_bound = 3.
v2-upper_bound = 6.
v2-element = 'table2 interval1'.
APPEND v2 TO itab2.

v2-lower_bound = 9.
v2-upper_bound = 14.
v2-element = 'table2 interval2'.
APPEND v2 TO itab2.

v2-lower_bound = 16.
v2-upper_bound = 19.
v2-element = 'table2 interval3'.
APPEND v2 TO itab2.

do 20 times.
  write / sy-index.
  loop at itab1 into v1.
    if sy-index between v1-lower_bound and v1-upper_bound.
      write v1-element.
      else.
        write '                '.
    endif.
  endloop.
  loop at itab2 into v2.
    if sy-index between v2-lower_bound and v2-upper_bound.
      write v2-element.
      else.
        write '                '.
    endif.
  endloop.
enddo.


PROVIDE FIELDS element FROM itab1 INTO v1
                              VALID flag1
                              BOUNDS lower_bound AND upper_bound
        FIELDS element FROM itab2 INTO v2
                              VALID flag2
```

```
                                BOUNDS lower_bound AND upper_bound
       BETWEEN 2 AND 20
  including gaps.
  WRITE: / SY-INDEX.
  WRITE: / v1-lower_bound, v1-upper_bound, v1-element, flag1.
  WRITE: / v2-lower_bound, v2-upper_bound, v2-element, flag2.
  SKIP.
  r-lower_bound = v1-lower_bound.
  r-upper_bound = v1-upper_bound.
  r-element = sy-index .
  append r to itab_r.
ENDPROVIDE.

do 20 times.
  write / sy-index.
  loop at itab1 into v1.
    if sy-index between v1-lower_bound and v1-upper_bound.
      write v1-element.
      else.
        write '                '.
    endif.
  endloop.
  loop at itab2 into v2.
    if sy-index between v2-lower_bound and v2-upper_bound.
      write v2-element.
      else.
        write '                '.
    endif.
  endloop.
   loop at itab_r into r.
    if sy-index between r-lower_bound and r-upper_bound.
      write r-element.
    endif.
  endloop.
enddo.
```

A belső tábláknak teljesen típusos indextáblaknak kell lenniük és az elemeiknek rendelkezniük kell két speciális oszloppal, amelyeknek a típusa minden táblában megegyezik. A BOUNDS részben kell megadni ezeket az oszlopokat. Egy tábla két oszlopának értékei egy zárt intervallumot határoznak meg. Egy tábla intervallumai nem lehetnek átfedőek és növekvő sorrendben kell állniuk.

Az INTO részben egy olyan változót adunk meg, amelynek a típusa megegyezik a belső tábla sortípusával. Az utasítás ezt a változót használja az eredménysorok tárolására. A VALID részben egy karakter típusú, egy hosszúságú változót adunk meg.

A BETWEEN rész két értéke egy zárt intervallumot határoz meg. A két értéknek ugyanolyan típusúnak kell lennie, mint a BOUNDS részben álló változóknak.

Az INCLUDING GAPS záradék opcionális.

CLEAR és FREE

Mindkét utasítás törli a belső tábla összes sorát.

Open SQL

Az SAP az adatok tárolására relációs adatbáziskezelő-rendszert használ. Különböző SAP rendszerek különböző adatbázis-kezelő rendszereket használhatnak. Minden relációs adatbázis-kezelő rendszer a saját SQL utasításait használja.

Az Open SQL egy SAP által definiált, adatbázis-független SQL szabvány, mely az ABAP nyelvhez készült. Az utasításait a rendszer a használt adatbázis-kezelő rendszer natív SQL utasításaira konvertálja.

Az Open SQL automatikusan kezeli a klienseket (az adatbázistáblák kliens oszlopait). Az aktuális kliensazonosítót a sy-mandt rendszermező tartalmazza.

Az adatok lekérdezése

A SELECT utasítás hasonló, de nem ugyanaz, mint azt az Adatbázisrendszerek című tárgyból megtanultuk. A fő struktúrája ugyanaz, azaz a SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY részeket ugyanazzal a jelentéssel használhatjuk. A FROM részbe használhatunk INNER JOIN-t és LEFT OUTER JOIN-t ON kulcsszóval az összekapcsolás feltételének megadásához.

Az AS kulcsszót használni kell, ha egy táblához vagy oszlophoz másodlagos nevet szeretnénk megadni. A táblák és az oszlopok nevei között szóköznek kell szerepelni a megszokott vessző helyett. Ha a táblához használunk másodlagos nevet, akkor a tábla oszlopaira a ~ jel segítségével hivatkozhatunk. Az aggregáló függvények és a nyitó zárójeleik (() között nincs szóköz. A COUNT függvénybe * vagy DISTINCT oszlopnév kerülhet.

A rendszernek a SELECT eredményét valahol tárolnia kell. A SELECT utasítás két lehetőség ajánl. Egyrészt az eredményt tárolhatjuk változókban, amelyek lehetnek belső táblák (több soros eredmény esetén), vagy egyszerű változók (egy soros eredmény esetén). Ha az eredményt belső táblákban tároljuk, akkor az APPENDING TABLE és a INTO TABLE kulcsszavak közül választhatunk. Az első hozzáfűzi az eredményt a táblához, a második

először inicializálja, majd beszúrja az eredményeket a táblába. Mindkét esetben használhatjuk a CORRESPONDING FIELDS OF záradékot.

```
REPORT  Z_SELECT_ONE_ROW.
data  nof type i.
SELECT count( * )
  from zdepartments as dept left outer join zemployees as emp on dept~depar
tment_id = emp~departtment_id
  into nof
 where department_name = 'IT'.

write: nof.
```

```
REPORT  Z_SELECT.
DATA: BEGIN OF dept,
        department_id   TYPE zdepartments-department_id,
        department_name   TYPE zdepartments-department_name,
      END OF dept.

DATA dept_tab LIKE HASHED TABLE OF dept
               WITH UNIQUE KEY department_id department_name.

SELECT department_id department_name
       FROM zdepartments
       INTO CORRESPONDING FIELDS OF table dept_tab.

SELECT department_id department_name
     FROM zdepartments
     INTO CORRESPONDING FIELDS OF table dept_tab
  WHERE department_name = 'IT'.

SELECT department_id department_name
    FROM zdepartments
    APPENDING CORRESPONDING FIELDS OF table dept_tab
  WHERE department_name = 'SALES'.


LOOP AT dept_tab into dept.
  WRITE: / dept-department_id, dept-department_name.
endloop.
```

A másik lehetőség, hogy a SELECT utasítást ciklusként használjuk, és az eredményt sorról sorra dolgozzuk fel. Ekkor az INTO részben adjuk meg, hogy egy sort melyik változókban tároljuk, és az utasításunkat az ENDSELECT-tel kell zárnunk.

```
REPORT  Z_SELECT_LOOP.
data wa type  zemployees.

select *
  from zemployees
  into wa
  where job_id = 3.
  write: / wa-first_name, wa-last_name, wa-departtment_id.
endselect.
```

Adatok lekérdezése – Kurzor

Az ABAP nyelv az adatok felolvasására más lehetőséget is ajánl: a kurzort. A fő utasításai: az OPEN SELECT, a FETCH NEXT CURSOR, és a CLOSE CURSOR.

Adatbázisbeli adatok módosítása

Ahhoz hasonlóan, ahogy azt az Adatbázisrendszerek című tárgyból elsajátítottuk, az ABAP programozási nyelvben is használhatunk DML utasításokat, azaz INSERT-et, UPDATE-ot, DELETE-t. Az ABAP-ban ezeken felül még használhatjuk a MODIFY utasítást is, mint DML utasítás. Ehhez a témához még sok más fontos témakör tartozik, mint tranzakciókezelés, zárolás, adatkonzisztencia és jogosultságkezelés.

Tranzakció

Az Adatbázisrendszerek című tárgyból tanultunk a tranzakciókról. Az adatbázisban egy tranzakció egy vagy több DML utasítást tartalmaz, és véglegesíteni a COMMIT utasítással lehet, vagy visszavonhatjuk a ROLLBACK utasítással. Az SAP az adatbázisbeli tranzakciókat adatbázis LUW-nak (Logical Unit of Work) hívja.

Azonban az SAP LUW és az adatbázis LUW nem ugyanaz. A felhasználó több programegységen keresztül navigálhat, azaz a programegységek meghívhatják egymást. A felhasználó a munkáját egyben szeretné véglegesíteni. Így egy alkalmazásprogram több munkafolyamatot tartalmazhat, amely munkafolyamatokban történő módosítást a rendszer implicit adatbázisbeli véglegesítéssel zár.

Amikor a felhasználó futtat egy programot, akkor elindít egy SAP LUW-t. A program hívhat más programegységeket. Az SAP LUW-t nekünk kell befejezni egy COMMIT WORK (véglegesítés) vagy egy ROLLBACK WORK (visszavonás) utasítással.

INSERT

Egy táblába új sort vagy sorokat helyez el. Egy sor értékeit megadhatjuk változók segítségével, több sort belső tábla segítségével adhatunk meg.

```
REPORT  Z_SELECT_DML.

data line type zjobs.

line-job_id = 10.
line-job_title = 'Administrator'.
insert into zjobs values line.

data: job_tab type table of zjobs.
line-job_id = 20.
line-job_title = 'Manager'.
append line to job_tab.
line-job_id = 30.
line-job_title = 'Cleaner'.
append line to job_tab.

insert zjobs from table job_tab.

select * from zjobs
  into line.
  write: / line-job_id, line-job_title.
endselect.
```

UPDATE

Egy tábla egy vagy több sorát módosítja.

```
select *
  from zjobs
  into line
  where job_id = 10.
  line-job_title = 'IT Administration'.
 endselect.

update zjobs from line.

select *
  from zjobs
  into table job_tab.

loop at job_tab into line.
  line-job_title = line-job_title && '!'.
  modify job_tab from line.
endloop.

update zjobs from table job_tab.
```

DELETE

Egy vagy több sort töröl egy táblából. A sorokat feltétellel vagy adatelemek segítségével határozhatjuk meg.

```
delete from zjobs where job_id = 10.
```

```
delete zjobs from line.

delete zjobs from table job_tab.
```

MODIFY

Az utasítás egy vagy több sort beszúr vagy módosít egy táblában.

```
line-job_id = 40.
line-job_title = 'Tester'.
modify zjobs from line.

select *
  from zjobs
  into table job_tab.

loop at job_tab into line.
  line-job_title = line-job_title && '+'.
  modify job_tab from line.
endloop.

line-job_id = 50.
line-job_title = 'Programmer'.
append line to job_tab.

modify zjobs from table job_tab.
```

Üzenetek (Message)

A MESSAGE utasítás megszakítja a program futását és egy rövid üzenetet jelenít meg. Az üzenet a T100-as táblában megadott üzenet vagy bármilyen szöveg lehet. Az üzenet típusát kötelező megadni. Az üzenet típusa meghatározza az üzenet viselkedését.

```
MESSAGE text TYPE message_type
```

Az üzenet típusa a következő lehet:

- "A" megszakítás üzenet (termination message): egy dialógusablak jelenik meg és a program leáll.
- "E" hibaüzenet (error message): a program környezetétől függően egy hibaüzenet jelenik meg vagy a program leáll.
- "I" információs üzenet (information message): Egy dialógus ablakban jelenik meg. Ha a felhasználó tudomásul veszi, akkor a program folytatja a működését.
- "S" státusz üzenet (status message): az üzenet a status bar-on jelenik meg és utána a program normálisan folytatja a működését.

- "W" figyelmeztetés (warning): a program környezetétől függően egy hibaüzenet jelenik meg vagy a program leáll.
- "X" kilépési üzenet (exit message): A program leáll, nem jelenik meg üzenet.

```
REPORT  Z_MESSAGE.
parameters a type i.
if a <= 0.
  message 'Give positive number!' type 'A'.
endif.
```

Az ABAP Dictionary-ben (SE11) keressük ki a T100-as tábla üzeneteit.

Az SE91 tranzakcióval (Message Maintenance) új üzeneteket adhatunk a táblához. Először hozzunk létre egy ZMES1 nevű üzenetosztályt (Message Class). Majd miután a Create gombra kattintottunk, töltsünk ki minden mezőt az Attributes fülön. Aztán a Messages fülön írjuk be az üzeneteinket, majd mentsük el őket. A programkódunkból a következőképp használhatjuk őket:

```
REPORT  Z_MESSAGE.
parameters a type i.
if a <= 0.
  message e000(ZMES1).
endif.
```

A szintaktikában az 'e' a hibaüzenet (error message) típusra utal, míg a 000 az üzenetosztályban az üzenethez rendelt számra. ZMES1 az üzenetosztály neve.

Események és blokkok

Ha használjuk a PARAMETERS kulcsszót, akkor selection screen-t hozunk létre, ahol a felhasználó input értékeket adhat meg. Ha a WRITE utasítást használjuk, akkor listát készítünk, amely a program outputjának megjelenítésére szolgál. Egy program tartalmazhat selection screen-t és listát is. A program eseményblokkokat tartalmazhat, amelyeket események váltanak ki.

Egy blokk egy kezdő kulcsszóval kezdődik, azonban a végét a következő blokk kezdő kulcsszava határozza meg.

A jegyzet csak néhány főbb eseményt és blokkot mutat be.

Az eseményblokkok előtt globális deklarációk szerepelhetnek. A blokkok sorrendje a programlogikát követi. Minden blokk opcionális.

A lista eseményei és blokkjai

Többféle blokkot lehet egy listában használni, de a jegyzetben csak az AT LINE-SELECTION eseményblokkal foglalkozunk. Az esemény akkor váltódik ki, ha a felhasználó egy soron kétszer kattint.

```
REPORT  Z_SCREEN_EVENT.
write 'Click on me'.

AT LINE-SELECTION.
  WRITE: / 'You clicked on the previous list'.
```

Inicializációs blokk

Ebben a blokkban lehet a selection screen input mezőit vagy más változókat inicializálni. A program betöltése után fog lefutni. A kulcsszava az INITIALIZATION.

A selection screen eseményei és blokkjai

AT SELECTION-SCREEN ON paraméter

Akkor váltódik ki, amikor az ABAP programban a paraméter értéket kap.

AT SELECTION-SCREEN ON VALUE-REQUEST FOR paraméter

Akkor váltódik ki, amikor a felhasználó meghívja az input help-et (F4).

AT SELECTION-SCREEN ON HELP-REQUEST FOR paraméter

Akkor váltódik ki, amikor a felhasználó meghívja a help-et (F1).

AT SELECTION-SCREEN

A selection screen feldolgozásának utolsó eseményeként akkor váltódik ki, amikor a program minden input paramétere értéket kapott.

START-OF-SELECTION

Ez a kulcsszó definiálja egy futtatható programnak a standard feldolgozó blokkját. A hozzátartozó esemény a selection screen feldolgozása után váltódik ki.

```
REPORT  Z_EVENT.
parameters: a type i,
            b type i.
data: c type i,
      d type i,
      e type i,
      f type decfloat16.

data itab type table of i.
data line like line of itab.

INITIALIZATION.

  do 10 times.
    append sy-index to itab.
  enddo.

at selection-screen on b.
  if b = 0.
    message 'The second value must not be 0' type 'E'.
  endif.

at selection-screen.
  c = a + b.
  d = a - b.
  e = a * b.
  f = a / b.

at line-selection.

  loop at itab into line.
    line = line * c.
    modify itab from line.
  endloop.

start-of-selection.
  write: c, d, e, f.
  write: / 'Click'.

at line-selection.
  loop at itab into line.
    write: / line.
  endloop.
```

Modularizáció

Egy komplex ABAP kódot feldarabolhatunk több, kisebb, egyszerűbb modulra. Az SAP rendszer több lehetőséget ajánl fel egy program kisebb, könnyebben kezelhető kódrészletekre való felbontására.

Alrutinok

Az alrutinok (subroutines) segítik az aktuális programkódon belüli modularizációt. A `Z_SUBROUTINE_1` riport háromszor tartalmazza a zjobs tábla lekérdezését.

```
REPORT  Z_SUBROUTINE_1.
write 'ZJobs'.

data line type zjobs.

select * from zjobs
  into line.
  write: / line-job_id, line-job_title.
endselect.
write /.

line-job_id = 100.
line-job_title = 'IT'.
insert into zjobs values line.

select * from zjobs
  into line.
  write: / line-job_id, line-job_title.
endselect.
write /.

delete from zjobs where job_id = 100.

select * from zjobs
  into line.
  write: / line-job_id, line-job_title.
endselect.
write /.
```

Létrehozhatunk egy alrutint, amely a zjobs tábla lekérdezését tartalmazza, hogy a lekérdezést csak egyszer kelljen implementálni és a programból többször ugyanazt a kódot hívjuk meg. Ennek kivitelezéséhez írjuk a `perform select_zjobs.` utasítást a kódba. Ez az utasítás hívja meg a select_zjobs nevű alrutint. Aztán kattintsunk kétszer a select_zjobs-on. A megnyíló új ablakban válasszuk azt, hogy egy objektumot akarunk létrehozni, és a másik új ablakban válasszuk a főprogramunkat, ami most a z_include_1, azaz kattintsunk a megfelelő sor elején lévő téglalapra. A kódunkban egy új programrész jelenik meg. A `form select_zjobs.` után írhatjuk a select utasítást. Eredményképp a következő kódot kapjuk:

```
REPORT  Z_SUBROUTINE_1.
write 'ZJobs'.

data line type zjobs.

perform select_zjobs.

line-job_id = 100.
line-job_title = 'IT'.
insert into zjobs values line.

perform select_zjobs.

delete from zjobs where job_id = 100.
```

```
perform select_zjobs.
*&---------------------------------------------------------------------*
*&      Form  SELECT_ZJOBS
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM SELECT_ZJOBS .
select * from zjobs
  into line.
  write: / line-job_id, line-job_title.
endselect.
write /.

ENDFORM.                    " SELECT_ZJOBS
```

Az alrutinnak adhatunk át paramétert a `using` kulcsszóval. A paraméter lehet tábla típusú is.

```
REPORT  Z_SUBROUTINE_2.

parameters gvt_max type i.
data gvt_count type i.
data gvt_st_primes type standard table of i.

perform f_prime tables gvt_st_primes using gvt_max gvt_count.

perform f_writeout tables gvt_st_primes.

*&---------------------------------------------------------------------*
*&      Form  prim
*&---------------------------------------------------------------------*
*       It generated the prime numbers between 1 and v_max
*----------------------------------------------------------------------*
*      -->T_PRIMEK   it stored the prime numbers
*      -->V_MAX      it contains the upper bound of the range
*      -->V_DB       it contains how many prime numbers are generated.
*----------------------------------------------------------------------*
form f_prime tables t_primes using v_max v_count.
  data i type i value 2.
  data prime type i.
  while i <= v_max.
    data line type i.
    prime = 1.
    LOOP AT t_primes into line.
      if i mod line = 0.
        prime = 0.
        exit.
      endif.
    endloop.

    if prime = 1.
      append i to t_primes.
    endif.
    i = i + 1.
  endwhile.

endform.                    "prim
```

```
*&---------------------------------------------------------------------*
*&      Form
*&---------------------------------------------------------------------*
*      It writes the integer typed element of the table
*        got as a parameter to the screen.
*---------------------------------------------------------------------*
form f_writeout tables t_numbers.
  data line type i.
  LOOP AT t_numbers into line.
    WRITE: / line.
  endloop.

endform.                    "f_writeout
```

Más programból is meg lehet hívni egy program alrutinját.

```
REPORT  Z_SUBROUTINE_3.
data i type i value 2.
data gvt_st_numbers type standard table of i.
parameters gvt_max type i.

while i <= gvt_max.
  append i to gvt_st_numbers.
  i = i + 1.
endwhile.

perform f_writeout in program z_subroutine_2 tables gvt_st_numbers.
```

Include

Az alrutinokat include-okban is elhelyezhetjük. Kétféleképp lehet include-ot létrehozni. Az egyik lehetőség az, hogy amikor beírjuk a `perform select_zjobs.` utasítást a kódba, amely utasítás a select_zjobs nevű alrutint fogja hívni, és kétszer kattintunk rajta, akkor ahelyett, hogy az új ablakban a főprogramot választanánk, az új include-ot választjuk (esetleg egy meglévő include-ot). A másik lehetőség, hogy az új include létrehozását úgy kezdjük el, mintha új programot hoznánk létre, de a program típusának az INCLUDE-ot választjuk.

Az első esetben a riportba az `INCLUDE include_name.` utasítást generálja a rendszer. A második esetben nekünk kell ezt beírni a kódba. Ha a riportban az include nevén kétszer kattintunk, akkor a rendszer megnyitja az include-ot.

Az include-ot nem lehet közvetlenül lefuttatni. Mielőtt a kódot futtatnánk, az include-ot aktiválni kell.

```
*---------------------------------------------------------------------*
***INCLUDE Z_INCLUDE_2_INC_SELECT01 .
*---------------------------------------------------------------------*
*&---------------------------------------------------------------------*
*&      Form  SELECT_ZJOBS
*&---------------------------------------------------------------------*
*       text
```

```
*---------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*---------------------------------------------------------------------*
FORM SELECT_ZJOBS .
  select * from zjobs
  into line.
  write: / line-job_id, line-job_title.
endselect.
write /.

ENDFORM.                    " SELECT_ZJOBS
*&---------------------------------------------------------------------*
*&      Form  DELETE_ZJOB
*&---------------------------------------------------------------------*
*       text
*---------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*---------------------------------------------------------------------*
FORM DELETE_ZJOB using v_job_id.
delete from zjobs where job_id = v_job_id.
ENDFORM.                    " DELETE_ZJOB
```

A Z_INLCUDE_2 riport a Z_INCLUDE_2_INC_SELECT01 include-ot használja.

```
REPORT  Z_INCLUDE_2.
write 'ZJobs'.

data line type zjobs.
data v_job_id type zjobs-job_id value 100.

perform select_zjobs.

line-job_id = 100.
line-job_title = 'IT'.
insert into zjobs values line.

perform select_zjobs.

perform delete_zjob using v_job_id.

perform select_zjobs.

INCLUDE Z_INCLUDE_2_INC_SELECT01.
```

Egy include tartalmazhat más include-okat. Egy ABAP program forráskódjának modularizálásához használhatunk include-okat. Nem javasolt egy include-ot több ABAP programban használni.

Függvénymodulok

A függvénymodulok alkotják egy SAP rendszer fő részét, mivel az SAP évek óta modularizálja őket a kódok újrafelhasználása miatt. A **függvénycsoport (function group)** a

logikailag összetartozó **függvénymodulok (function modules)** egyfajta tárolója. A függvénymodulokat bármely ABAP program meghívhatja. Az SAP rendszerek több ezer függvénymodullal rendelkeznek.

Az Object Navigatorban (SAP menu: Tools, ABAP Workbench, Overview, and Object Navigator - SE80) listázhatjuk a függvénycsoportokat. Kattintsunk a repository browserre, válasszuk ki a Function Group-ot, kattintsunk az Input Help gombon, az új ablakban kattintsunk Information System gombon, és végül az új ablakban kattintsunk az execute gombon. Eredményképp a függvénycsoportok listáját kapjuk meg, igaz, csak az első 200-at. Megvizsgálhatjuk az F017 függvénycsoportot.

A **függvénycsoportok** (**Function groups** vagy function pools) speciális típusú ABAP programok. Ez az egy program típus tartalmazhat függvénymodulokat. A függvénymodulok publikus interfésszel rendelkező eljárások, amelyeket arra terveztek, hogy más programok használják. A függvénycsoportok a csoport minden moduljában elérhető globális deklarációkat és alrutinokat tartalmazhatnak. A függvénycsoportok komponensként dynprokat tartalmazhatnak. Ha dynprokkal dolgozunk, akkor függvénycsoportokat fogunk használni.

A rendszer minden függvénycsoporthoz generál egy főprogramot, amelyet `SAPLfunctiongroupname`-nek hívnak. A főprogram a következő programokhoz tartalmaz include-okat:

- `LfunctiongroupnameTOP`: Ez az első include program; a függvénycsoport globális adatdefinícióit tartalmazza.
- `LfunctiongroupnameUxx`: Ezek az include-ok függvénymodulokat tartalmaznak. Az xx számozás a függvénymodulok létrehozásának sorrendjét tükrözik. Például `LfunctiongroupnameU01` és `LfunctiongroupnameU02` tartalmazza a függvénycsoport első két függvénymodulját.
- `LfunctiongroupnameF01`, `LfunctiongroupnameF02`, ... `LfunctiongroupnameFxx`: Ezeket az include-okat olyan alrutinok (form) írására használhatjuk, amelyeket a csoport függvénymoduljai belső formként hívhatnak meg.

Ezeket az include programokat a Function Builder generálja, nem módosíthatjuk őket.

Az include-okhoz kapcsolódó névkonvenciók a következőek (xx kétjegyű számot jelöl):

- "Exx" utótag: Metódusok és osztályok implementációja

- „Oxx” utótag: PBO modulok implementációja (A PBO a Process Before Output képernyőesemény. Azelőtt váltódik ki, mielőtt a rendszer egy képernyőt a megjelenítő réteghez küld.)
- “Ixx” utótag: PAI modulok implementációja (A PAI a Process After Input képernyőesemény. a GUI-n történő felhasználói tevékenység következtében váltódik ki.)
- “Exx” utótag: Eseményblokkok implementációja
- “Fxx” utótag: Helyi alrutinok implementációja

A top include-hoz “Dxx” előtagú include programokat adhatunk, amelyek helyi osztályok deklarációs részeit tartalmazhatják.

A Function Builderben (SE37) kereshetünk függvénymodulokat. A Function Buildert (SE37) az SAP menüben találhatjuk: Tools, ABAP Workbench, Development, és Function Builder. Válasszuk a Utilities menü Find menüpontját, és keressük az *amount* függvénymodult. Válasszuk a Spell_amount függvénymodult az F017 függvénycsoportból, és nézzük meg.

A függvénymodulok paraméterei

A függvénymodulok paramétereket használnak arra, hogy a hívó programmal kommunikáljanak. A paraméterek típusai a következőek:

- Import: A paraméter értékét a hívó program átadja a függvénymodulnak. Az import paraméterek tartalmát nem írhatjuk felül. Ha egy paraméterhez nem szükséges értéket rendelni, akkor válasszuk ki az Optional checkbox-ot a Function Builder Import fülén.
- Export: A paraméter értékét a függvénymodul adja át a hívó programnak. Mindig opcionális.
- Changing: A paraméter úgy viselkedik, mintha egyszerre import és export paraméter is lenne. A changing paraméter eredeti értékét a hívó program átadja a függvénymodulnak. A függvénymodul megváltoztathatja az eredeti értéket, majd visszaküldi azt a hívó programnak.
- Tables: A paraméter egy importálható és exportálható belső táblát tartalmaz. A belső tábla tartalmát a hívó program átadja a függvénymodulnak. A függvénymodul megváltoztathatja a belső tábla tartalmát, majd visszaküldi a hívó programhoz. Mindig referencia szerinti összerendelés történik. Elavult.

- Exceptions: A hívó program kivételeket használhat, hogy megtalálja a függvénymodulban keletkező hibákat.

Egy formális paraméter adattípusát úgy is megadhatjuk, hogy egy type pool adattípusához kapcsoljuk. A type pool ABAP Dictionary objektum, amely lehetővé teszi, hogy saját globális típusokat definiáljunk. Ha a formális paraméter típusaként type pool-beli típust akarunk használni, akkor a type poolt a függvénycsoport TOP include-jában deklarálni kell.

A függvénymodulok meghívása

Hívjuk meg az SE38 tranzakciót, és hozzunk létre egy új programot `Z_FUNC_MOD_CALL` néven.

```
REPORT  Z_FUNC_MOD_CALL.
parameter v type i.
```

Kattintsunk a Pattern gombra (CRTL+F6), az új ablakban válasszuk a CALL FUNCTION lehetőséget, írjuk be a spell_amount-t, ami egy függvénymodul neve, és kattintsunk a continue gombra. A rendszer automatikusan egy ABAP kódot generál. Az opcionális mezők ki vannak kommentelve, a kötelező mezőket ki kellene tölteni.

```
REPORT  Z_FUNC_MOD_CALL.
parameter v type i.
data result like spell.
CALL FUNCTION 'SPELL_AMOUNT'
                     EXPORTING
                      AMOUNT           = v
*                       CURRENCY         = ' '
*                       FILLER           = ' '
*                       LANGUAGE         = SY-LANGU
                     IMPORTING
                       IN_WORDS         = result
*                     EXCEPTIONS
*                       NOT_FOUND        = 1
*                       TOO_LARGE        = 2
*                       OTHERS           = 3
          .
IF SY-SUBRC <> 0.
  write: 'Returned value:',sy-subrc.
else.
  write: 'The amount in words is:',result-word.
ENDIF.
```

A CALL FUNCTION utasítás szintaktikája a következő:

```
CALL FUNCTION <function module name>
  [EXPORTING  f1 = a1.... fn = an]
  [IMPORTING  f1 = a1.... fn = an]
  [CHANGING  f1 = a1.... fn = an]
  [TABLES  t1 = itab1.... tn = itabn]
```

```
[EXCEPTIONS e1 = r1.... en = rn]
[ERROR_MESSAGE = rE]
[OTHERS = ro]].
```

Az EXPORTING, IMPORTING, CHANGING, és TABLES részekben adhatunk át paramétereket. Explicit rendeljük hozzá a formális paraméterekhez az aktuális paramétereket, azaz <formal parameter> = <actual parameter>. Ha egy részben több paraméterhez rendelünk értékeket, tegyünk közéjük szóköz, vagy kezdjünk új sort.

- EXPORTING: A formális paraméterek a függvénymodul import paramétereiként vannak deklarálva.
- IMPORTING: A formális paraméterek a függvénymodul export paramétereiként vannak deklarálva.
- CHANGING: A formális paraméterek a függvénymodul changing paramétereiként vannak deklarálva.
- TABLES: Belső táblát rendel egy táblaparaméternek. Elavult.
- EXCEPTIONS: Lehetővé teszi, hogy a függvénymodul hibáira reagáljunk. Ha kiváltódik egy kivétel, a függvénymodul működése befejeződik és nem ad vissza egyetlen paraméterértéket sem. A hívó program fogadja a kivételt, azaz a SY-SUBRC mezőhöz rendeli az rE értéket. Egy függvénymodul kivétel kezelését lecserélhetjük, ha az EXCEPTIONS listában egy ERROR_MESSAGE-et adunk meg.
- ERROR_MESSAGE: A rendszer a függvénymodulban meghívott hibaüzeneteket a következőképp kezeli:
    - Az S, I, és W üzenetosztályokat figyelmen kívül hagyja (de rögzíti a naplóban, ha a programot háttérfolyamatként futtatjuk).
    - E és A üzenetosztályok a függvénymodulok befejezését okozzák, mivel az ERROR_MESSAGE kivétel váltódik ki (SY-SUBRC megkapja rE értékét).

  Ha az EXCEPTION listába beírjuk az OTHERS-öt, akkor minden kivételt lehetővé teszünk.

Függvénymodul létrehozása

A függvénymodulok olyan feladatokat hajtanak végre, amelyek iránt más programozók is érdeklődnek. Ilyen feladatok lehetnek például az adó kiszámítása, a gyári naptári napok meghatározása, gyakran használt dialog-ok meghívása, stb.

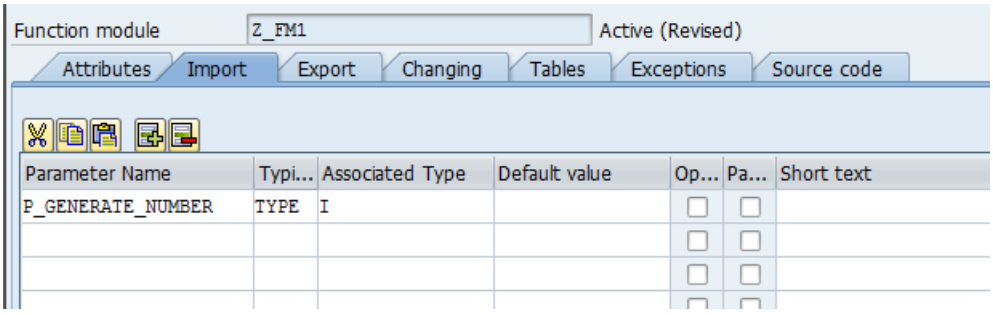
Mielőtt egy függvénymodult létrehoznánk, bizonyosodjunk meg róla, hogy a megfelelő függvénymodul létezik-e már vagy sem. Ha létezik, akkor használjuk azt. Egyébként válasszunk egy megfelelő függvénycsoportot, ha létezik. Ha nem létezik, akkor hozzunk létre egyet.

Miután létrehoztuk a függvénymodult, aktiváljuk, teszteljük (F8) és dokumentáljuk más felhasználók számára. A dokumentáció térjen ki a paraméterek használatára.

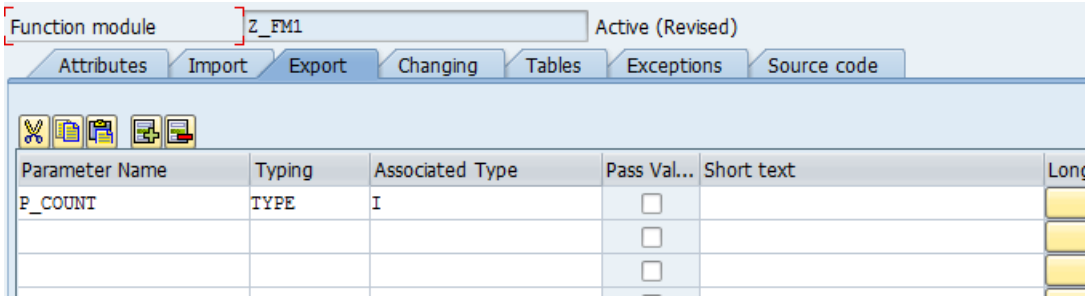
Függvénycsoportot kétféleképp **hozhatunk létre**. Az egyik lehetőség, hogy az Object Navigator-ban (SE80) kiválasztjuk a Function Group, írunk egy új nevet, és rákattintunk az Enterre. A másik lehetőség, hogy a Function Builderben (SE37) a Goto menüben, Function Groups, Create Group menüpontban egy új csoportnevet adunk meg. A példában a Z_FG1 új függvénycsoportnevet használjuk.

Függvénymodult a Function Builderrel (SE37) vagy az Object Navigatorral (SE80) **hozhatunk létre**. Az Object Navigatorban keressük meg a Z_FG1 függvénycsoportot, kattintsunk rajta a hierarchiában a jobb egér gombbal, és adjunk hozzá egy új függvénymodult. Írjuk be az új nevet és nyomjuk meg a Create gombot. A példában az új függvénymodul neveként a Z_FM1 nevet használjuk. Az új ablakban adjuk meg a függvénycsoport nevét és a rövid leírást. A következő új ablakban az import, export, changing, exceptions füleken megadhatjuk a függvénymodul paramétereit. A source code fülre kattintva írhatjuk meg a kódot.

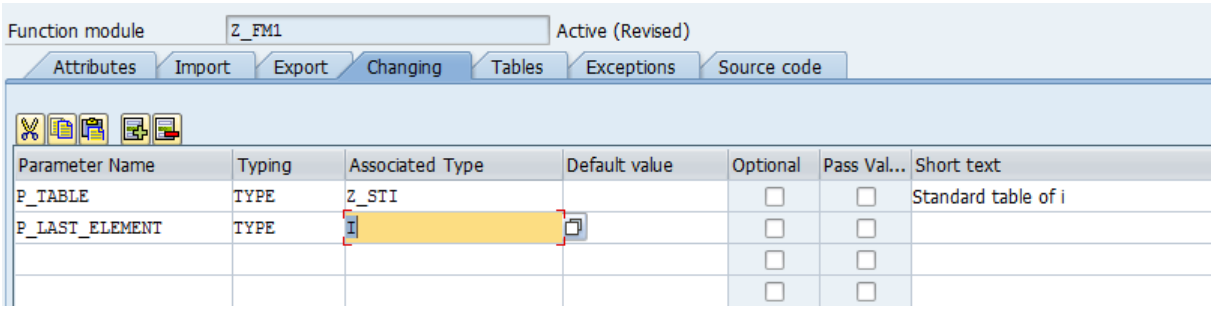
A 7. ábra, a 8. ábra és a 9. ábra Function Builder füleit mutatja, amikor a Z_FM1 függvénymodult hozzuk létre.



7. ábra – Import fül a Function Builderben



8. ábra – Export fül a Function Builderben



9. ábra – Changing fül a Function Builderben

Z_STI típus az adatszótárban definiált egészelemű standard tábla.

```
FUNCTION Z_FM1.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(P_GENERATE_NUMBER) TYPE  I
*"  EXPORTING
*"     REFERENCE(P_COUNT) TYPE  I
*"  CHANGING
*"     REFERENCE(P_TABLE) TYPE  Z_STI
*"     REFERENCE(P_LAST_ELEMENT) TYPE  I
*"----------------------------------------------------------------------
  data i type i value 1.
  data e type i .

  while i <= p_generate_number.
    e = i + p_last_element.
    append e to p_table.
    i = i + 1.
  endwhile.

  p_count = 0.
  data line type i.
  LOOP AT p_table into line.
    WRITE: / line.
    p_count = p_count + 1.
  endloop.
ENDFUNCTION.
```

A függvénymodul dokumentálásához először a paraméterekhez adjunk információt a Function Builderben vagy az Object Navigatorban a Long Text gombbal. Ezen kívül adjunk leírást a függvénymodulról az Object Navigatorban, a Function Module Documentation gombra kattintva. A gomb az SAPscript Editor-t nyitja meg.

A Z_FG1 nevű függvénycsoporthoz adhatunk több függvénymodult, mint például a Z_FMD-t.

```
FUNCTION Z_FMD.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  CHANGING
*"     REFERENCE(P_TABLE) TYPE  Z_STI
*"----------------------------------------------------------------------
data line type i.
LOOP AT p_table into line .
  DELETE p_table.
ENDLOOP.
ENDFUNCTION.
```

Egy függvénycsoport függvénymoduljai meghívhatják egymást.

```
FUNCTION Z_FMCE.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(P_GENERATE_NUM) TYPE  I DEFAULT 10
*"  CHANGING
*"     REFERENCE(P_TABLE_CH) TYPE  Z_STI
*"----------------------------------------------------------------------
CALL FUNCTION 'Z_FMD'
  CHANGING
    P_TABLE       = p_table_ch
          .
data l type i.
CALL FUNCTION 'Z_FM1'
  EXPORTING
    P_GENERATE_NUMBER       = p_generate_num
* IMPORTING
*   P_COUNT                 =
  CHANGING
    P_TABLE                 = p_table_ch
    P_LAST_ELEMENT          = l
          .
ENDFUNCTION.
```

User Dialog

A dynpro vagy képernyő (screen) egy dinamikus program (dynamic program). Egy képernyőből és a programlogikából áll (dynpro flow logic). Ezenkívül dynpro mezőket

tartalmaz. A Screen Painter (SE51) segítségével hozhatunk létre bonyolult képernyővel rendelkező dynprokat.

A user dialog-ok dynpro-kon alapulnak. A jegyzetben bemutatott selection screen-ek és a listák speciális dynpro-k.

A jegyzet azonban a dynpro-k grafikus felhasználói felületének csak a töredékét mutatta be. Egy dynpro-n több típusú képernyőelemet lehet használni, mint menü, eszköztár, szöveges mező, input mező, output mező, rádiógomb, checkbox, nyomógomb, keret, táblavezérlő, alképernyő, státusz kijelző. Az SAP-ban találhatunk beépített függvényeket, amelyet segítik, hogy a lekérdezések eredményét táblázatos formában jelenítsük meg (pl. az ALV_GRID_DISPLAY függvény).

A Screen Painter (SE51), a Menu Painter (SE41), vagy az Object navigator (SE80) tranzakciók használatával hozhatunk létre olyan képernyőket, amelyet tartalmazzák ezeket a képernyőelemeket.

A dynprok futtathatóak más dynpro-k vagy más típusú ABAP programok részeként.

ABAP programtípusok

Az ABAP programok típusai meghatározzák, hogy a programok milyen részekből állhatnak és az SAP hogyan hívhatja meg őket. Egy új ABAP program készítésekor egy listából választhatjuk ki az ABAP program típusát.

Futtatható (Executable) program

Az első utasítása a REPORT utasítás. Ilyen típusú ABAP programokat használunk ebben a jegyzetben példaként. Van saját képernyőjük. A következőképp futtathatjuk őket:

- egy másik programból a SUBMIT utasítással,
- meghívunk egy dynpro-t tranzakciós kóddal,
- meghívunk egy selection screen-t tranzakciós kóddal.

Class Pool

Osztálykönyvtár, lokális interfészek és osztályok, TYPES és CONSTANTS utasítások globális osztálya. A látható metódusait a CALL METHOD vagy egy tranzakciós kód segítségével hívhatjuk meg. Nem támogatja a saját képernyőt.

Function group vagy function pool

Függvény modulokat tartalmaz. Támogatja a saját képernyőt.

Interface pool

Osztálykönyvtárak globális interfésze. Nem futtatható. Nem támogatja a saját képernyőt.

Module pool

Képernyőket és dialog modulokat tartalmaz. Tranzakciókóddal történő dynpro hívásával futtatható.

Subroutine pool

Más ABAP programokból hívható alprogramokat tartalmaz.

Type group vagy type pool

TYPES és CONSTANTS utasításokat tartalmaz. Nem futtatható.

A tananyagban említett tranzakciókódok

ABAPDOCU – Example Library. SAP menu: Tools, ABAP Workbench, Utilities.

ABAPHELP – Keyword Documentation. SAP menu: Tools, ABAP Workbench, Utilities.

DWDM – Demos. SAP menu: Tools, ABAP Workbench, Utilities.

SE11 – ABAP Dictionary. SAP menu: Tools, ABAP Workbench, Development.

SE14 - Utilities for Dictionary Tables

SE16 – Data Browser. SAP menu: Tools, ABAP Workbench, Overview.

SE37 – Function Builder. SAP menu: Tools, ABAP Workbench, Development.

SE38 – ABAP Editor. SAP menu: Tools, ABAP Workbench, Development.

SE41 – Menu Painter. SAP menu: Tools, ABAP Workbench, Development, User Interface.

SE51 – Screen Painter. SAP menu: Tools, ABAP Workbench, Development, User Interface.

SE54 – General Table Maintenance Dialog. SAP menu: Tools, ABAP Workbench, Development, Other Tools.

SE80 – Object Navigator. SAP menu: Tools, ABAP Workbench, Overview.

SE91 – Messages. SAP menu: Tools, ABAP Workbench, Development, Programming Environment.

SM30 - Call View Maintenance

SM31 - Call View Maintenance

SU01 – Users. SAP menu: Tools, Administration, User Maintenance.

Referenciák

1. Online ABAP documentation: http://help.sap.com/abapdocu_731/en. (May 2015.)
2. Moxon, P. (2012): Beginner's Guide to SAP ABAP. SAPPROUK, 274 p.

Debrecen, 2015. május 17.