UNIDAD

5

DIPLOMATURA EN PROGRAMACION ABAP

MÓDULO 5: OPEN SQL

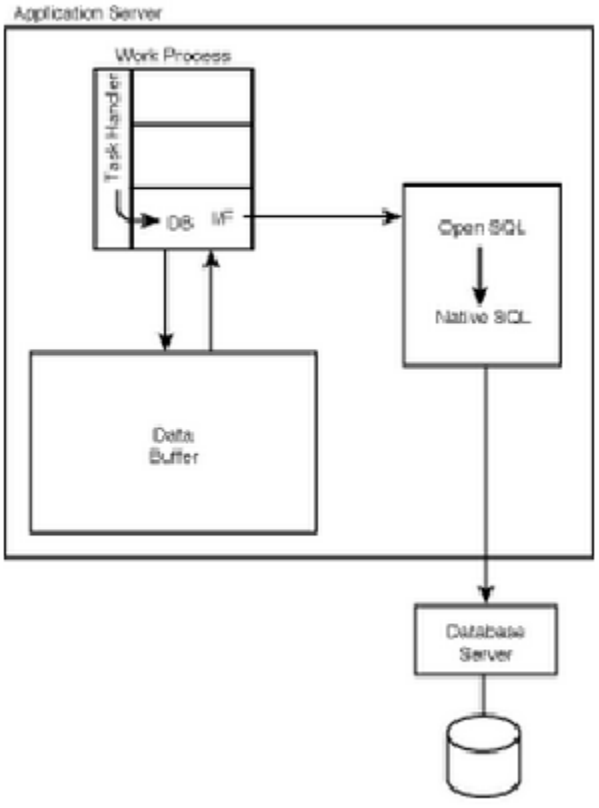
OPEN SQL

OPEN SQL

Este módulo introduce el uso de *Open SQL*, que permite comunicar a los programas *ABAP* con la base de datos física. Asimismo, se introduce el uso básico de la funcionalidad provista por *ABAP* para la depuración de programas, es decir, el *debugger*.

ABAP SQL

El código / 4 es portable entre bases de datos. Para acceder a la base de datos en un programa ABAP / 4 se usa el código de SAP *Open SQL* . Open SQL es un subconjunto y una variación de ANSI SQL. El intérprete ABAP / 4 pasa todas las sentencias Open SQL a la parte de interfaz de base de datos del proceso de trabajo:



Allí, ellos se convierten en SQL nativo de la RDMS instalado. Por ejemplo, si se ejecuta una base de datos Oracle, su código Open SQL de ABAP / 4 sería convertido por la interfaz de base de datos a las declaraciones de Oracle SQL.

Si utiliza Open SQL, las sentencias SQL se pasarán a la interfaz de base de datos. El uso de Open SQL tiene tres ventajas principales. Todas estas ventajas se implementan a través de la interfaz de base de datos y son:

- **Portabilidad:** La primera ventaja es el hecho de que las sentencias SQL serán portable entre bases de datos. Por ejemplo, si por alguna razón su compañía quería pasar de Oracle a una base de datos Informix, podría cambiar la base de datos y su código ABAP / 4 continuaría funcionando sin modificación.

- **Almacenamiento intermedio de datos en el servidor de aplicaciones:** En segundo lugar, la interfaz de base de datos almacena temporalmente la información de la base de datos en el servidor de aplicaciones. Cuando los datos se leen desde la base de datos, puede ser almacenados en buffers en el servidor de aplicaciones. Si se hace entonces una solicitud de acceso a los mismos registros, que ya estarían en el servidor de aplicaciones, la petición se satisface desde el buffer sin tener que ir a la base de datos. Esta técnica de buffering reduce la carga en el servidor de base de datos y en el enlace de red entre los servidores de base de datos y de la aplicación, y puede acelerar los tiempos de acceso a la base de datos por un factor de 10 a 100 veces.

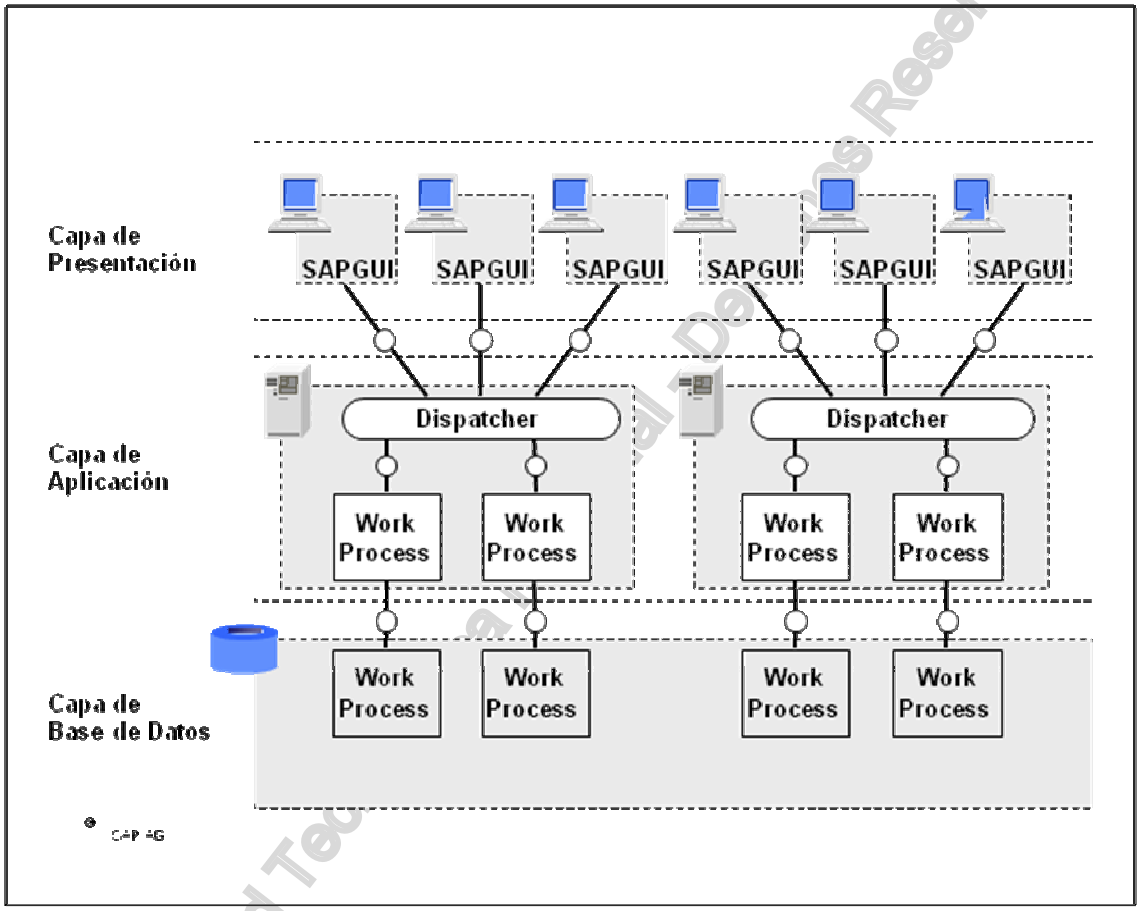
- **Manejo automático de los clientes:** La tercera ventaja de utilizar Open SQL es *la administración automática de cliente (mandante)* . Con Open SQL, el campo de cliente se rellena automáticamente por la interfaz de base de datos. Esto da a sus equipos de desarrollo y prueba muchas ventajas, como la capacidad de realizar pruebas simultáneas múltiples y la formación en una sola base de datos sin interferencias entre sí.

En resumen:

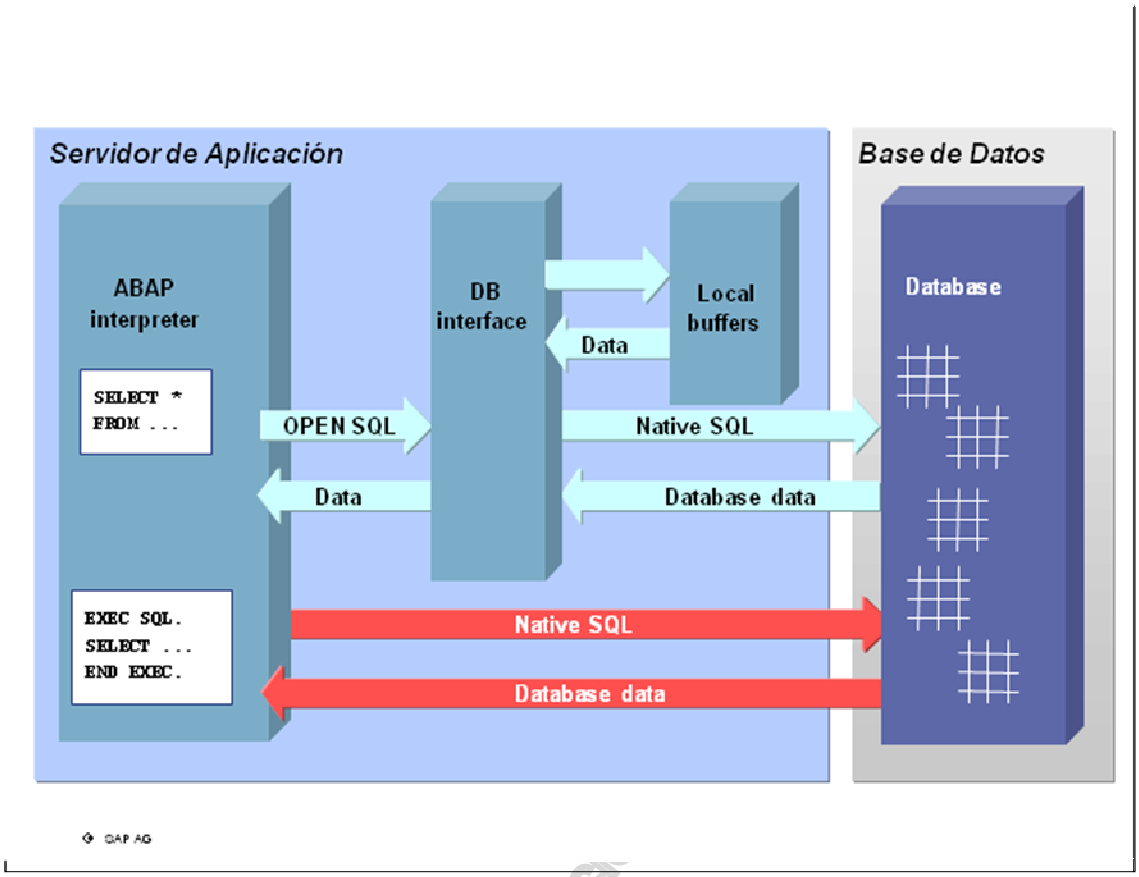
- SAP R / 3 soporta múltiples plataformas de hardware, sistemas operativos y bases de datos.
- Además de permitir SQL nativo, ABAP / 4 proporciona Open SQL. El uso de Open SQL hace que su código sea más portable, más rápido, y proporciona administración automática de cliente.

- El manejo de cliente de inicio de sesión permite que varios grupos independientes almacenen datos en la misma tabla. Los datos que se acceden depende del número de cliente que se introdujo al iniciar sesión.

Si repasamos el esquema de la arquitectura de 3 capas cliente/servidor de SAP, veíamos esto:



Ahora que vimos cómo trabaja Open SQL, podemos agregar este otro esquema:



Sentencias SQL

La sentencia SELECT en ABAP/4 recupera los registros de la base de datos.

El siguiente código muestra la sintaxis simplificada para la la declaración `select` :

```
select * from t1 [into wa] [where f1 op v1 and/or f2 op v2 ...]
[order by f1].
    (other abap/4 statements)
    endselect.
```

donde:

`*` indica que todos los campos de la tabla se deben recuperar.

`t1` es el nombre de una tabla previamente.

`wa` es el nombre del área de trabajo que coincida con la estructura de la tabla.

f1 es el nombre de un campo en la tabla *t1*.

op es uno de los siguientes operadores lógicos: = <> > > = < <= .

v1 es un literal o una variable.

Se necesita utilizar la declaración *tables* para poder acceder al DDIC.

En este ejemplo traemos los registros de la tabla (ver las actividades prácticas de la Unidad para realizar un ejercicio sobre el mismo):

```
1 Report ztx0501.
2 tables spfli.
3 select * from spfli into spfli order by carrid.
4 write: spfli-carrid, spfli-connid.
5 ENDSELECT.
```

Este programa lee todos los registros de la tabla `spfli` (tabla de vuelos provista por SAP) y escribe el contenido de los campos `carrid y connid` (compañía aérea y número de vuelo) en orden ascendente.

- En la línea 1, se requiere una declaración `report` como la primera línea de un informe.
- En la línea 2, la declaración `tables` hace dos cosas. En primer lugar, se asigna un área de memoria (llamada área de trabajo) llamada `spfli` . El área de trabajo tiene el mismo diseño que la definición de la tabla DDIC `spfli` . En segundo lugar, le da al programa acceso a la tabla de base de datos `spfli` .
- En la línea 3, la la declaración `select` comienza un bucle. El `ENDSELECT` en la línea 5 marca el final del bucle. Las líneas de código entre el `select` y `ENDSELECT` se ejecutan una vez para cada fila devuelta por la base de datos.
- En la línea 4, la sentencia `write` se ejecuta una vez para cada fila que se lee de la tabla. El `/` (barra inclinada) después de cada `write` comienza una nueva línea.

Tenga en cuenta que en su programa, usted tiene *dos* cosas nombradas `spfli` : un área de trabajo y una tabla. Ambos tienen el mismo nombre- `spfli` . La posición del nombre `spfli` dentro de una sentencia determina a cuál se refiere. En la línea 3, la primera aparición de `spfli` refiere a la tabla de base de datos. La segunda aparición se refiere a la zona de trabajo.

Podemos simplificar el código anterior así:

```
1 Report ztx0502.
2 tables spfli.
3 select * from spfli order by carrid.
4 write: spfli-carrid, spfli-connid.
5 ENDSELECT.
```

En comparación con el listado anterior, sólo la línea 3 ha cambiado. Este programa funciona exactamente de la misma manera y produce el mismo resultado que la anterior. Dado que no se especifica la cláusula `into` en la declaración `select`, el sistema utiliza por defecto como tabla para el área de trabajo la misma `spfli` . Se denomina área de trabajo implícita.

La declaración `tables` siempre crea un área de trabajo de tabla por defecto, por lo que normalmente no tiene que definir uno propio. En algunos casos, sin embargo, es posible que desee definir un área de trabajo de tabla adicional. Por ejemplo, si desea mantener la versión original de una fila y tener una versión modificada también, usted necesitaría dos áreas de trabajo, la implícita más otra explícita.

Puede definirse áreas adicionales de trabajo de la tabla usando la declaración `data`.

A continuación se muestra la sintaxis simplificada para la declaración `data` .

```
data wa like t1 .
```

donde:

- *wa* es el nombre de un área de trabajo de tabla que desea definir en forma explícita
- *t1* es el nombre de una tabla que será el patrón de su área de trabajo

He aquí un ejemplo:

```
1 report ztx0503.
2 tables spfli.
3 data wa like spfli.
4 select * from spfli into wa order by carrid.
5 write: spfli-carrid, spfli-connid.
6 ENDSELECT.
```

El código de este listado debe producir el mismo resultado que las dos listas anteriores. Tenga en cuenta que en este ejemplo no es necesario definir un nuevo área de trabajo sólo se utiliza para ilustrar el concepto:

- La línea 3 define una nueva área de trabajo llamado `wa` como la estructura DDIC `spfli` .
- La línea 4 lee una fila a la vez de la tabla `spfli` en el área de trabajo `wa`.
- La línea 5, escribe la primera fila desde `wa` lugar de `spfli` .

Para restringir el número de filas devueltas desde la base de datos, una cláusula `where` se puede agregar a la la declaración `select`. Por ejemplo, para recuperar los vuelos de la compañía aérea "AZ" solamente:

```
1 Report ztx0504.
2 tables spfli.
3 select * from spfli where carrid = 'AZ' order by carrid.
4 write: spfli-carrid, spfli-connid.
5 ENDSELECT.
```

El sistema R / 3 hace que *las variables de sistema* estén disponibles dentro de su programa. Usted no tiene que definir nada para acceder a ellas, ellas siempre están disponibles, y se actualizan de forma automática por el sistema como cambios en el entorno de su programa. Todas las variables del sistema comienzan con el prefijo `sy-` . Por ejemplo, la fecha actual del sistema está disponible en el campo del sistema `sy-datum` , el usuario actual está en `sy-uname` y la hora actual en `sy-uzeit` . Por lo general se llaman campos *`sy`*, para abreviar.

Todas las variables del sistema se definen en la estructura DDIC `sist` . No defina `sist` en su programa, pues sus campos están disponibles automáticamente dentro de cada programa.

Dos variables del sistema son de gran utilidad para conocer al codificar la declaración `select`:

- `sy-SUBRC`
- `sy-dbcnt`

Para determinar si la declaración `select` no devuelve ninguna fila, pruebe el valor de la variable del sistema *`SY-SUBRC`* después de la sentencia `ENDSELECT` . Si se encontraron filas, el valor de `SY-SUBRC` será 0. Si no se encuentra ninguna fila, el valor será 4.

Vea el siguiente ejemplo:

```
1 Report ztx0505.
2 tables spfli.
3 select * from spfli where carrid = 'ZZ' order by carrid.
4 write: spfli-carrid, spfli-connid.
5 ENDSELECT.
6 if sy-subrc <> 0.
7 write 'No se encontraron registros'.
8 endif.
```

En este caso, no hay filas coincidentes con los criterios establecidos en el `donde` cláusula, por lo que después de la `ENDSELECT` , el valor de `sy-SUBRC` se establece en 4. La prueba viene siempre después del `ENDSELECT`.

Si cero filas se devuelven desde la base de datos, el código entre `select` y `ENDSELECT` nunca se ejecuta. Por lo tanto, debe codificar la

Para determinar el número de filas devueltas por una declaración `select` , se debe comprobar el valor de *`sy-dbcnt`* después de `ENDSELECT` . También puede utilizarlo como contador del bucle; entre el `select` y `ENDSELECT`, contiene un recuento de la iteración actual. Para la primera fila, `SY-dbcnt` será 1, para la segunda que será 2, y así sucesivamente. Después de `ENDSELECT` , conservará su valor, por lo que contendrá el número de filas seleccionadas. Por ejemplo:

```
1 Report ztx0506.
2 tables spfli.
3 select * from spfli where carrid = 'AZ' order by carrid.
4 write / sy-dbcnt.
6 write: spfli-carrid, spfli-connid.
7 ENDSELECT.
7 write / sy-dbcnt.
8 write ' registros encontrados'.
```

La línea 4, escribe el valor de sy-dbcnt para cada fila que se devuelve por el select . La barra (/) inicia una nueva línea, por lo que cada sy-dbcnt comienza en una nueva línea.

La línea 5 escribe el valor de los campos en la misma línea que sy-dbcnt .

Para mostrar los componentes de la estructura `sist` :

- Desde el Editor ABAP / 4, pantalla de edición de programas, haga doble clic en el nombre de cualquier campo `sy` dentro de su código

Los dos puntos (:) se llaman el *operador cadena* . Se usa para combinar líneas de código que comienzan con la misma palabra o secuencia de palabras. Coloque la parte común al principio de la declaración seguido de dos puntos. A

continuación, coloque las piezas que terminan de las declaraciones después de que, cada uno separado por una coma.

Por ejemplo, para definir dos tablas, se podría codificar esto:

```
tables ztxlfa1.
tables ztxlfb1.
```

O bien, puede utilizar el operador de la cadena, así:

```
tables: ztxlfa1, ztxlfb1.
```

Funcionalmente, los dos anteriores segmentos de código son idénticos, y en tiempo de ejecución no hay diferencia en el rendimiento. Durante la generación de código, el segundo segmento de código se expande en dos estados, por lo que desde un punto de vista funcional, los dos programas son idénticos.

Los operadores de cadena deben ser utilizados para mejorar la legibilidad de su programa.

La instrucción `select single` recupera un solo registro de la base de datos. Si conoce toda la clave principal del registro que desea recuperar, `select single` es mucho más rápido y más eficiente que `select` / `ENDSELECT` .

A continuación se muestra la sintaxis simplificada para la declaración `select single`:

```
select * from t1 [into wa] [where f1 op v1 and/or f2 op v2 ...].
```

donde:

`*` indica que todos los campos de la tabla se deben recuperar.

t1 es el nombre de una tabla previamente definida en con `tables`.

wa es el nombre del área de trabajo que coincide con la estructura de la tabla.

f1 es el nombre de un campo en la tabla *t1* .

op es uno de los siguientes operadores lógicos: `= <> > > = < <=` .

v1 es un literal o una variable.

Los siguientes puntos se aplican:

- `select single` no comienza un bucle, ya que sólo devuelve una fila. Por lo tanto, no codificar `ENDSELECT` . Si lo hace, obtendrá un error de sintaxis.
- Para asegurarse de que una fila única es devuelta, debe especificar todos los campos de clave principal en la cláusula `where`. Si no lo hace, el programa se ejecutará, pero usted recibirá una advertencia.

Veamos un ejemplo de `select single`:

```
1 report ztx0507.
2 tables spfli.
3 select single * from spfli where carrid = 'AZ'.
4 if sy-SUBRC = 0.
5 write: / spfli-carrid, spfli-connid.
6 else.
7 write 'Registro no hallado'.
8 endif.
```

Se devolverá sólo una fila, la primera que cumpla con la condición.

En resumen

- Los informes ABAP / 4 deben comenzar con la sentencia `report` .
- La instrucción `tables` asigna un área de trabajo de tabla por omisión y también le da al programa acceso a la tabla de base de datos del mismo nombre.
- El sentencia `select` recupera filas de una tabla de base de datos. Utilice `select single` para recuperar una sola fila. Use `select` / `ENDSELECT` para recuperar varias filas.
- `select` / `ENDSELECT` forma un bucle. El código en el bucle se ejecuta una vez para cada fila de la tabla que satisface la cláusula `where` . El bucle finaliza automáticamente cuando se han procesado todas las filas. Si no se especifica una cláusula `into` , cada fila se coloca en el área de trabajo de tabla predeterminada y sobrescribe la fila anterior.
- `sy-SUBRC` se establece en 0 si se han seleccionado filas, y se establece a 4 si no se han seleccionado filas.
- `sy-dbcnt` se incrementa en 1 cada vez en el bucle, y después de la sentencia `ENDSELECT`, contiene el número de filas recuperadas.
- El operador de la cadena (`:`) se utiliza para reducir la redundancia en la que dos o más sentencias comienzan con la misma palabra o secuencia de palabras.

Gestión de las instrucciones SQL de ABAP/4

ABAP/4 tiene un subconjunto de sentencias SQL para su aplicación sobre tablas de la base de datos SAP. Estas son:

SELECT, INSERT, UPDATE, MODIFY, DELETE, COMMIT WORK, ROLLBACK WORK.

Además de las variables del sistema:

SY-SUBRC : Código de retorno de una operación.
SY-DBCNT : Cantidad de registros afectados por la operación procesada.

Sentencia SELECT

La sentencia **SELECT** será la instrucción fundamental para leer información de la base de datos.

- **Lectura de un único registro :**

```
SELECT SINGLE * FROM <tab>
WHERE <cond>
```

Como realizamos la búsqueda de un registro, en la condición sólo podremos utilizar la igualdad y el operador AND, ya que especificaremos toda la clave del registro.

Si **SY-SUBRC** = 0 Registro encontrado. Resultado en área de trabajo.

Si **SY-SUBRC** = 4 No existe el registro buscado.

- **• Lectura Iterativa :** Selección de un grupo de registros. Su sintaxis es la siguiente:

```
SELECT [DISTINCT] * FROM <tab> INTO <lugar>

(WHERE <cond>) (GROUP BY <campos>) (ORDER BY <campos>) .

ENDSELECT.
```

Donde:

Distinct Excluye duplicados de líneas.

Into Determina el área (blanco) hacia el cual los datos seleccionados sean colocado para ser leídos posteriormente.

Where Especifica cuáles son las líneas que serán leídas especificando condiciones de selección.

Group by Produce una sola línea de resultado desde grupos de varias líneas. Un grupo de líneas con los mismos datos.

Order by Proporciona un orden en las líneas seleccionadas por los campos **<campos>**

Se seleccionan todos los registros que cumplan la condición de la cláusula WHERE, o todos en caso de no utilizarla. El resultado lo tendremos en el área de trabajo, es decir en cada iteración del bucle SELECT ... ENDSELECT tendremos un registro leído en dicha área.

Si **SY-SUBRC** = 0 Algún registro encontrado.

Si **SY-SUBRC** = 4 No existe ningún registro que cumpla la condición del WHERE.

Si la condición del WHERE se acerca a la clave de la tabla, la búsqueda de registros será más óptima.

Otras posibilidades del WHERE:

SELECT * FROM <tab> WHERE <campo> ...

BETWEEN <var1> AND <var2>.
Si <campo> está entre los valores <var1> y <var2>.

LIKE <literal enmascarado>. Si <campo> cumple la máscara.
Se pueden utilizar :
'_' como carácter cualquiera.
'%' como una cadena de caracteres.

IN (<var1> , <var2> ...)
Si <campo> esta en el conjunto de valores <var1>, <var2> ...

- **• Otras lecturas :**

Podemos leer una tabla de base de datos y simultáneamente llenar una tabla interna con el resultado de la lectura.

SELECT * FROM <tab> INTO TABLE <intab> (WHERE <cond>).

Llena la tabla interna <intab> machacando los registros que pudiera tener esta. Si queremos que respete los registros que tenía la tabla interna antes de realizar el SELECT tendremos que utilizar :

SELECT * FROM <tab> APPENDING TABLE <intab> (WHERE <cond>).

Podemos indicar un orden en el proceso de selección de registros.

SELECT * ... ORDER BY <campo1> <campo2> ...

Si queremos seleccionar un registro para bloquearlo de posibles modificaciones.

SELECT SINGLE FOR UPDATE * FROM <tab>.

Ejemplo para un select *:

Se hará una selección del tabla clientes en donde se obtendrá del select, el nombre del cliente, cuenta y saldo, dependiendo de la cuenta, es decir, se obtendrán las cuentas que terminen con **-456**.

Código (todo lo referente a tabla interna lo veremos más adelante, concentrarse ahora en el *select*):

```
Report ztx0508.
Tables: clientes.
Data: begin of clientes-tab occurs 100,
nombre like clientes-nombre,
num_cuenta like clientes- num_cuenta,
saldo like clientes-saldo,
end of clientes-tab.
Select * from clientes where cuenta like '%-456'
Order by nombre.
Move-corresponding clientes to clientes_tab.
Append clientes_tab.
Clear clientes_tab.
Endselect.
Write:/ 'Sucursal:', clientes_tab-sucursal.
Loop at clientes_tab.
```

```
Write:/ clientes_tab-nombre, clientes_tab-num_cuenta, clientes_tab-saldo.
At last.
Sum.
Write:/ 'Saldo total:', 30 clientes_tab-saldo.
Uline.
Endat.
Endloop.
```

Salida:

Sucursal: 123
Ezequiel Nava 80194776-456 2,589.50
Pedro Gamboa 80000468-456 5,698.21
Sara Rosas 80000236-456 1,500.25
Total de la Sucursal 9,789.96

Sentencia INSERT

La sentencia INSERT permite introducir registros sencillos o el contenido de una tabla interna en una base de datos SAP.

INSERT <tab>

Grabará en la BDD el registro de cabecera. Por tanto previamente a esta instrucción moveremos los valores que queremos introducir sobre el área de trabajo de la tabla de la BDD.

Si **SY-SUBRC** = 0 Registro insertado.
Si **SY-SUBRC** > 0 La clave del registro que queríamos insertar ya existía en la tabla.

También es posible introducir datos desde una tabla interna.

INSERT <tab> FROM TABLE <intab>.

Si **SY-SUBRC** = 0 Registros insertados.

Si existe algún registro en la base de datos con clave igual a algún registro de la tabla interna, se producirá un error de ejecución del programa.
La tabla interna podrá tener la misma estructura que la tabla de base de datos utilizando: **INCLUDE STRUCTURE** en su declaración.

Sentencia UPDATE

La sentencia UPDATE permite modificar el contenido de uno o varios registros.

UPDATE <tab>.

Modifica el registro de la base de datos que está especificado en el registro de cabecera.
Si queremos modificar el contenido de más de un registro a la vez:

UPDATE <tab> SET <campo> = <valor> WHERE <cond>.

Con este UPDATE, todos los registros que cumplan <cond> modificarán el contenido del <campo> por <valor>.
También es posible utilizar la cláusula **SET** con :
<campo> = <campo> + <valor> *o* <campo> = <campo> - <valor>.

En este caso, lo que se hace es aumentar o agregar un valor al valor del campo actual o bien sustraerlo a dicho campo.

Ejemplo del uso de Update:

Considerando la tabla clientes_tab del ejemplo de select *.

Código:

```
UPDATE clientes SET num_cuenta = '8000235-456'
saldo ='2,000.00'
Where clave = '0001053'
Select * ............
```

Salida:

En esta ocasión tendremos en la salida:

Sucursal: 123
Ezequiel Nava 80194776-456 2,589.50
Pedro Gamboa 80000468-456 5,698.21
Sara Rosas 80000235-456 2,000.00
Total de la Sucursal 10,389.96

Es posible modificar registros desde una tabla interna:

UPDATE <tab> FROM TABLE <intab>.

Si el sistema no puede actualizar un registro, el proceso no finalizará sino que continuará con el siguiente registro.
Si **SY-SUBRC** = 0 Todos los registros modificados.
Si **SY-SUBRC** = 4 No todos los registros han sido modificados.
En **SY-DBCNT** Tendremos la cantidad de registros modificados.

Sentencia MODIFY

La sentencia **MODIFY** se utilizará cuando no estemos seguros si utilizar un INSERT o un UPDATE. Es decir, cuando no sepamos con certeza si un registro existe o no, para modificarlo o añadirlo:

MODIFY <tab>.
MODIFY <tab> FROM TABLE <intab>.

En caso de que sepamos si existe o no un registro, por eficacia utilizaremos uno o más INSERT o UPDATE.

Sentencia DELETE

Para realizar borrados de datos se aplica la sentencia DELETE:

DELETE <tab>.

Borrará el registro que especifiquemos en el área de trabajo.
Para borrar más de un registro (todos los que cumplan una cierta condición):

DELETE FROM<tab> WHERE <cond>.

Ejemplo del uso de DELETE:

Usaremos la tabla clientes_tab del ejemplo select *, para borrar al cliente Ezequiel Nava.

Código:

```
DELETE FROM clientes WHERE nombre = 'Ezequiel Nava'.
Select * from clientes_tab.
```

Salida:

Sucursal: 123
Pedro Gamboa 80000468-456 5,698.21
Sara Rosas 80000235-456 2,000.00
Total de la Sucursal 7,698.21

Podemos borrar de la base de datos todos los registros de una tabla interna:

DELETE FROM <tab> FROM TABLE <intab>.

Si **SY-SUBRC** = 0 Todos los registros han sido borrados.
Si **SY-SUBRC** = 4 No todos los registros han sido borrados.
En **SY-DBCNT** Tendremos la cantidad de registros borrados.

Otros aspectos de la programación de base de datos

- El control del mandante es automático. Siempre se procesará el mandante en uso. Si queremos controlar manualmente el mandante en una instrucción de lectura o actualización utilizaremos la cláusula CLIENT SPECIFIED. Es decir, si queremos obtener o modificar datos de un cliente diferente al de entrada.
- Las instrucciones INSERT, DELETE, MODIFY y UPDATE se utilizarán en la medida que sea posible el menor número de veces sobre tablas SAP.
- Siempre se intentará insertar o modificar datos mediante transacciones estándares SAP o vía Batch Input (tema que veremos más adelante). Ya que no siempre es fácil conocer la compleja estructura de toda la base de datos SAP y así nos aseguramos no producir alguna inconsistencia en la base de datos.

El Bloqueo de objetos:

Para bloquear un registro en el momento de una actualización sobre éste utilizaremos

FOR
UPDATE.
SELECT SINGLE FOR UPDATE * FROM <tab>

Para coordinar el acceso de distintos usuarios a los mismos datos, y para evitar posibles inconsistencias en las actualizaciones, SAP dispone de un método

interno de bloqueo, de forma que únicamente un usuario podrá procesar un objeto de datos de la primera operación hasta la última.
Para implementar los bloqueos es necesario utilizar objetos de bloqueo, que pueden ser compuestos de un registro de una cierta tabla o de una colección de registros de una o varias tablas, como se indica en los siguientes pasos:

1. Ir a ABAP4 Dictionary
2. En la pantalla se le debe de colocar en el campo Nombre_del_Objeto la letra **E** seguida del nombre de la tabla que se desea bloquear. Se selecciona el radio button de Objeto_de_bloqueo (Lock Objects).
3. Después de que se ha presionado el botón para Crear, aparecerá una pantalla, en la cual en el primer campo que solicita información (Descripción Breve), donde se introducirá una pequeña descripción del objeto creado.
4. En el segundo campo Tabla Primaria, se introduce el nombre de la tabla que se desea bloquear.
5. Si se requiere bloquear otras tablas, se ponen en tablas secundarias y en ese caso va a Pasar a => Tablas.
6. Indicar cuál es la relación entre ellas (join).
7. Si se va a Pasar a => Argumentos bloqueo, aquí se observaran en la pantalla los campos por los que se va a seleccionar la entrada a bloquear, por defecto toma todos los campos clave de las tablas. Solo se tendrá que ir a esta pantalla en caso de que hayan seleccionado varias tablas y coincidan nombre de campos claves entre ellas o si por otro motivo se quiere cambiar el nombre de los argumentos.
8. Verificar, generar y salvar. Es muy importante GENERAR, de otra forma nunca estará activado el objeto.

Al activarlo, inmediatamente se crean las siguientes funciones:

ENQUEUE_<nombre del objeto de bloqueo> para bloquearlo.
DEQUEUE_<nombre del objeto de bloqueo> para desbloquearlo.

Los parámetros de la función **ENQUEUE** son:

EXPORTING

- arg *y* xarg Estos dos parámetros existen para cada campo arg en el argumento de bloqueo. Se coloca en arg el valor del campo llave. Si no se especifica el valor para el campo, se omite el parámetro arg. Si se desea el

valor inicial del campo como el actual valor, se indica colocando en x_arg una 'X'.

- SCOPE Toma los valores 1,2,3 e indica en qué momento se va a desbloquear la entrada, si en su programa antes de llamar al programa que actualiza las tablas de la base de datos, si lo desbloquea dicho programa, o si hay que desbloquear en los dos sitios en el programa de dialogo (el suyo) y en el programa que actualiza las tablas.
- COLLECT Por defecto tiene un blanco y es lo que debe de tener ya que de otro modo no lo bloquea directamente sino que lo deja en espera hasta que se llama a la función FLUSH_ENQUEUE (más adelante veremos todo lo referido a funciones).
- WAIT Indica si en caso de que esté bloqueada esa entrada; si debe esperar o no.

EXCEPTIONS

Después de llamar a la función ENQUEUE, se deberá checar que el objeto ha sido bloqueado en el programa. Las siguientes excepciones están definidas en el módulo de la función.

- FOREING_LOCK El objeto ha sido bloqueado por otro usuario. La variable del sistema SY_MSGV1 contiene el nombre de este usuario.
- SYSYTEM_FAILURE Error general del sistema.

Para desbloquear el objeto, solo basta con escribir la siguiente sentencia:

CALL FUNCTION DEQUEUE_<nombre del objeto de bloqueo>

NOTA: Si cuando se le da el nombre al objeto de bloqueo, el nombre de la tabla corresponde a una de usuario, todo es inmediato, PERO si es una tabla de SAP, pide una clave de acceso para crear el objeto de bloqueo. Esa clave se tiene que pedir por OSS a SAP.

Actualización de la base de datos o Recuperación:

Para finalizar una unidad de procesamiento lógico (LUW) de base de datos se utiliza un COMMIT WORK, que realiza un UPDATE físico en la base de datos, haciendo irrevocable cualquier modificación en la base de datos. Si deseamos deshacer todas las operaciones realizadas sobre la base de datos desde el último COMMIT WORK, realizaremos un ROLLBACK WORK.

Chequeo de autorizaciones:

Las instrucciones SQL de SAP no realizan ninguna verificación de autorizaciones, lo cual resulta peligroso ya que todo el mundo puede acceder a todos los datos que acceda un report. Es responsabilidad del programador el comprobar si un usuario está autorizado a acceder a esa información.
Para chequear las autorizaciones de un determinado usuario utilizaremos la instrucción **AUTHORITY-CHECK**.

```
AUTHORITY-CHECK OBJECT <objeto_de_autorización>
ID <Campo1> FIELD <f1>
ID <Campo2> FIELD <f2>
ID <Campo3> DUMMY.
...
```

donde <Campo(n)> son los campos de autorización del objeto y <f(n)> es un valor posible de autorización.
El parámetro DUMMY indicará que no hace falta verificar ese campo.

Si **SY-SUBRC** = 0 Usuario autorizado.
Si **SY-SUBRC** <> 0 Usuario NO autorizado.

Ejemplo :

Verificar el objeto de autorización 'Acreedor : Autorizaciones para sociedades' (F_LFA1_BUK), para saber si el usuario puede efectuar la operación Visualizar (01), sobre proveedores de la sociedad 0001:

```
AUTHORITY CHECK OBJECT 'F_LFA1_BUK'
ID 'ACTVT' FIELD '01'
ID 'BUKRS'FIELD '0001'.
```

Sentencias en SQL nativo:

Podemos ejecutar cualquier sentencia de SQL permitida por el gestor de base de datos sobre el que corra el sistema R/3, utilizando EXEC SQL. En este caso las instrucciones de base de datos no están restringidas al subconjunto SAP-SQL que hemos estado estudiando a lo largo de este capítulo. Gracias a la interfaz EXEC SQL también es posible acceder a datos externos a SAP, desde un programa en ABAP/4. La sintaxis es la siguiente:

```
EXEC SQL.
< Instrucciones SQL-Nativas>.
ENDEXEC.
```

Tenemos que tener en cuenta en la utilización de SQL nativo, que no todas las bases de datos SAP pueden ser accedidas con este sistema, ya que no todas tienen una representación física de tabla en el gestor de base de datos. Por ejemplo las tablas de tipo POOL y CLUSTER no son tablas reales de base de datos, aunque sean consideradas como tales y mantenidas por el diccionario de datos.

Recapitulación del proceso de creación de una tabla.

El proceso de creación de una tabla de base de datos en el diccionario de datos consta de una serie de pasos, que incluyen crear el objeto tipo tabla, especificar los campos de la tabla, definir los elementos de datos y dominios de la tabla, especificar los índices, claves externas y parámetros técnicos, y finalmente activar la tabla. Podemos desglosarlos más detalladamente así:

- Entrar en al ABAP/4 Dictionary (SE11), indicar el nombre de la tabla que vamos a crear y seleccionar 'Crear'. (Empezará por Z).
- Introducir una descripción breve de la tabla.
- Introducir una 'clase entrega', que indicará el grado de responsabilidad que tiene SAP o el cliente sobre la tabla. Normalmente utilizaremos el tipo A , que corresponde a la tablas de aplicación (datos maestros o de transacción, es decir que se generan vía transacciones SAP). Podemos ver la utilidad del resto de posibilidades, (C,G,L,E,S,W) pulsando F1 sobre este campo.
- Entrar el nombre de los campos de la tabla en la columna 'Nom. campo'.
- Seleccionar los campos que queremos que formen parte de la clave primaria de la tabla.
- Introducir el elemento de datos de cada campo :
    - Si este ya existe en el Diccionario de datos, la información del mismo, aparecerá automáticamente.
    - Si el elemento de datos no existía en el diccionario de datos, con doble-clic podemos ir al mantenimiento de elementos de datos para crearlo en ese momento.
- Seleccionar el flag '*initial'*, si los campos deben ser inicializados con sus valores iniciales.
- Si queremos permitir el mantener la tabla desde la transacción de mantenimiento de tablas (SM31), lo indicaremos activando el flag '*table manteinance allowed*'.
- Si queremos que la tabla que estamos creando dependa del mandante, incluiremos el campo MANDT como el primer campo de la tabla.

- Es posible especificar parámetros de tipo técnico como el tipo de datos que almacena la tabla, el tamaño aproximado, etc. desde :
  Pasar a -> opciones técnicas.
- • Finalmente, grabaremos los datos introducidos.

Las claves foráneas en la creación de tablas

Si un campo de la tabla tiene un dominio cuyo rango de valores está definido por una tabla de valores, será necesario definir las claves foráneas (externas) con dicha tabla de valores e indicar con un * en la columna 'Check table'.
Para ello :

- Seleccionamos el campo con tabla de chequeo.
- Seleccionamos Pasar a -> Claves externas.
- Introducir :
  - Texto de la clave foránea.
  - Código de la tabla de chequeo.
  - Asignación del campo (Genérico o Constante).
  - Chequeo del campo en las pantallas.
  - Mensaje de error que sustituye al mensaje estándar de error cuando se introduce un valor que no existe en la tabla de chequeo.
  - Cardinalidad. Describe la relación con respecto al número de registros dependientes entre las tablas. (1:C, 1:CN, 1:N, 1:1).
  - Tipo de campo de clave foránea.

Será necesario definir las claves externas si queremos que el chequeo automático de las pantallas funcione correctamente.

Otras posibilidades en la creación de tablas

Si hemos marcado la posibilidad de poder mantener el contenido de la tabla desde la transacción SM31, será necesario que ejecutemos un proceso de generación de dicho mantenimiento. Para hacerlo:

- Deberemos seleccionar : Entorno -> Gen. act. tablas .
- Introduciremos:
  - El grupo de funciones para el mantenimiento de tablas.
  - El grupo de autorizaciones para el mantenimiento de tablas.
  - El tipo de mantenimiento que queremos:
    - De una pantalla.
    - De dos pantallas. Una pantalla con lista de registros y otra con valores

individuales.

- El número de Dynpro(s) que queremos asignar a la(s) pantalla(s).

- El índice primario se crea cuando una tabla es activada. Bajo ciertas circunstancias es posible que sea recomendable ayudar a la selección de datos con índices secundarios.
- Para definir índices secundarios ir a : Pasar a -> Índices.
- Finalmente, antes de poder utilizar una tabla creada por nosotros, será necesario realizar el proceso de activación.

ABAP Debugger. Opciones Principales

Con **/h** activo el debugger.

F5 avanza paso a paso sentencia por sentencia

F6 ejecuta un bloque de sentencias,

F7 se comporta igual que **F6**, pero va hasta una sentencia de corte. En el caso de entrar en una función, **F8** la ejecuta hasta que termine.

Puedo poner un breakpoint, a nivel lógico, o lo puedo poner a nivel código con la sentencia break point (es una sentencia obsoleta), o por usuario con la sentencia break **user** (aquí pongo el nombre de usuario) . Como regla general es que cuando pasamos a testing, procuremos sacar los break del programa.

Un Wathchpoint es un una variable que está dentro de mi programa y puedo decir que mi programa pare ahí, quizás quiero saber en qué momento una variable toma un valor **x** que es el que me da un error en testing, entonces por watchpoint puedo hacer que el programa se frene cuando tome el valor **x**. Sirve para investigar el programa y para analizar lo que pasa en un determinado caso.

El uso general del *debugging* es similar al de cualquier otro lenguaje de programación.