



Lenguaje SQL en ABAP IV

En este capítulo vamos a conocer las distintas **sentencias de consulta y actualización** de los datos de la base de datos.

Objetivos

- ▶ Aprender el uso de las sentencia de consulta "**SELECT**".
- ▶ Conocer las sentencias de actualización de datos.

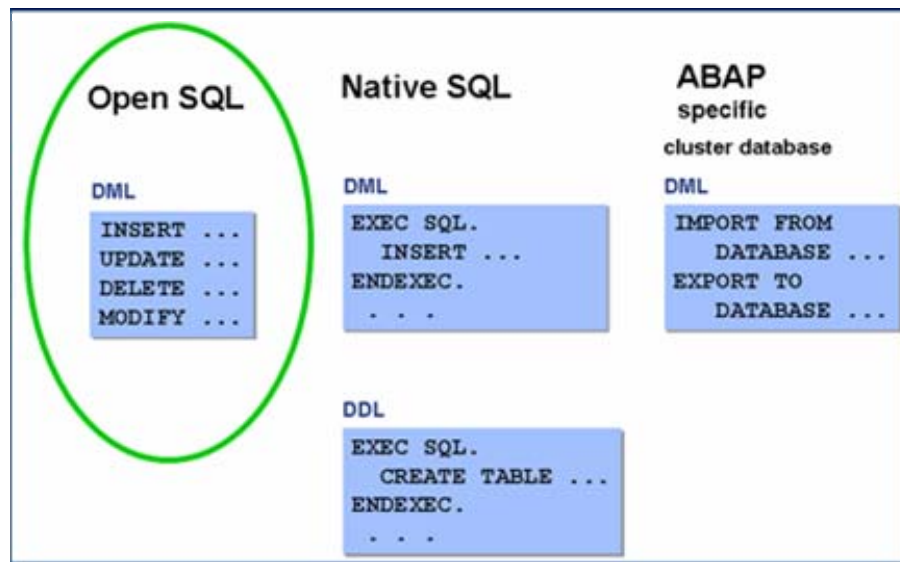
Espacio reservado para los elementos de
identificación visual de la Entidad

Lección 1

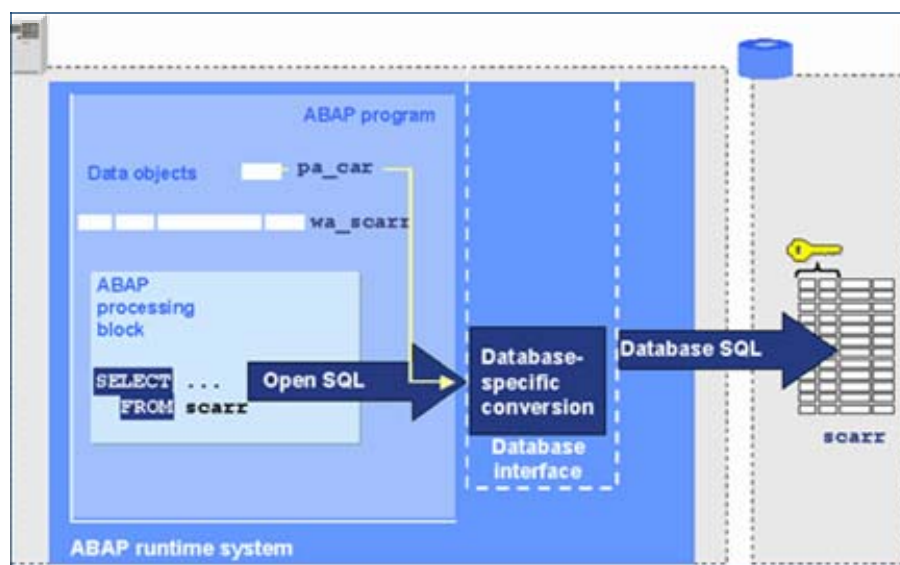
Lenguaje SQL en ABAP IV

En Abap tenemos las sentencias llamadas **OPEN SQL** que son el subconjunto del **STANDARD SQL** integrado en el lenguaje ABAP. De esta forma podemos acceder a la base de datos de forma uniforme. Las sentencias OPEN SQL son convertidas en sentencias específicas del standard SQL mediante la interfaz de la base de datos.

En este capítulo, vamos a hablar de las sentencias **OPEN SQL**.



El acceso a la base de datos se realiza de la siguiente forma:



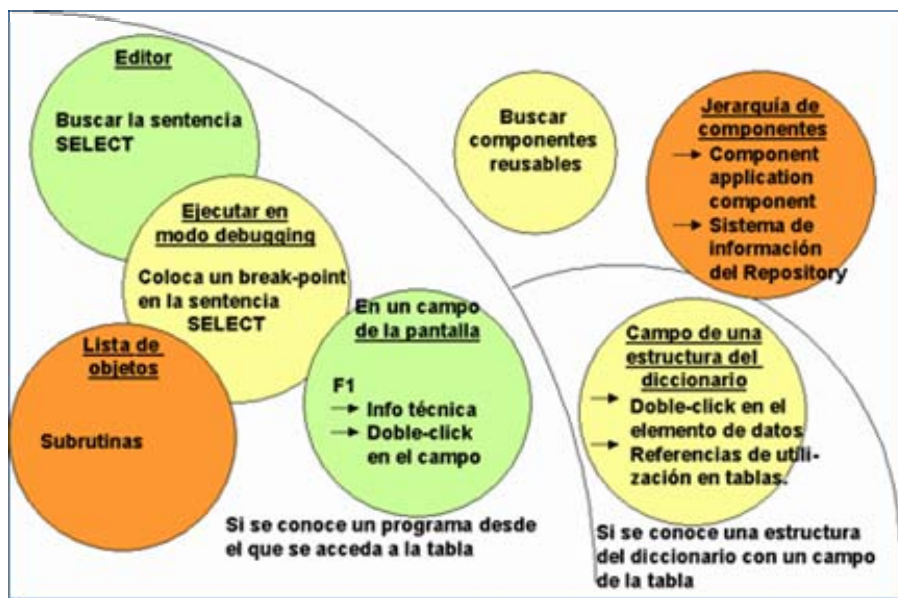
Espacio reservado para los elementos de identificación visual de la Entidad

Podemos buscar tablas de la Base de Datos de diferentes maneras:

En la Jerarquía de Aplicación y el sistema de Información Repository. Ahí podemos buscar las tablas de la base de datos de acuerdo a sus textos cortos (y otros criterios).

Si se tiene el nombre del programa que accede a la tabla:

- Mediante los **campos de entrada** en una pantalla: Si se sabe que un programa con una pantalla que tiene campos de entrada conectados a la tabla, buscamos en *F1-> Technical info*. Luego navegamos por el Diccionario ABAP haciendo doble click en el nombre técnico del campo de la pantalla. Haciendo doble click en el **elemento de datos** y luego usando la función **where-used list** para buscar la tabla transparente de acuerdo con el tipo de campo.
- **Debugger**: Si sabemos el nombre del programa que accede a la tabla de la base de datos que estamos buscando, podemos ejecutar el programa en modo debugging y colocar un breakpoint y un SELECT statement.
- **Editor**: Buscar la sentencia SELECT
- Lista de Objetos en el **Navegador de Objetos**



Lección 2

Sentencia SELECT

La selección de registros en una base de datos se obtiene mediante la sentencia **SELECT**.

- Usamos la sentencia open SQL **SELECT** para leer datos de la base de datos. Esta sentencia tiene varias cláusulas con diferentes tareas.

Cláusulas en una sentencia **SELECT**:

- **SELECT**
- **INTO**
- **FROM**
- **WHERE**
- **GROUP BY**
- **HAVING**
- **ORDER BY**

- La Cláusula **SELECT** especifica:

- Los campos que van a incluirse en el resultado.
- Si el resultado será de una o varias líneas.
- Si el resultado puede tener dos o más líneas idénticas.

- La cláusula **INTO** especifica la variable interna del programa en el que vamos a almacenar el resultado. Podemos almacenar un único campo en una variable, los registros seleccionados en una tabla interna, una única línea en una estructura etc...

- La cláusula **FROM** especifica la fuente de la que se obtiene el resultado (puede ser una o varias tablas de la base de datos o vistas).

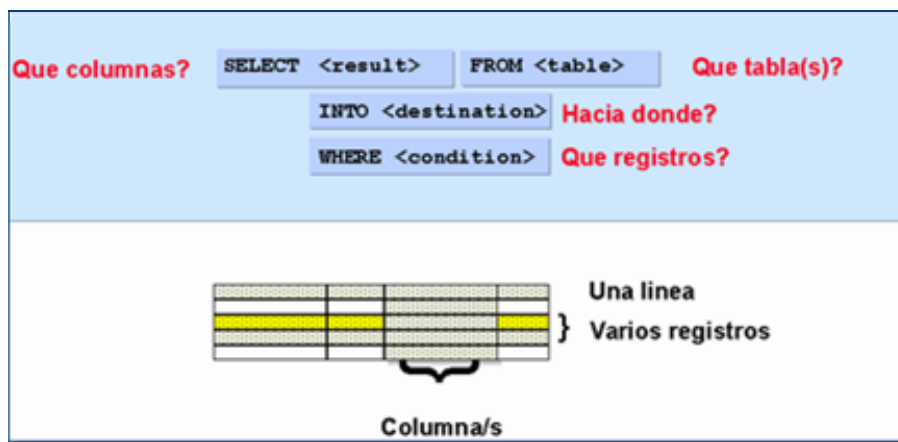
- La cláusula **WHERE** especifica las condiciones que deben cumplir los resultados de la selección.

- La cláusula **GROUP BY** especifica por qué campos van a agruparse los resultados de la selección.

- La cláusula **HAVING** especifica las condiciones que deben cumplir los registros después de ser agrupados mediante **GROUP BY**.

- La cláusula **ORDER BY** especifica por qué campos va a ser ordenado el resultado.

Espacio reservado para los elementos de identificación visual de la Entidad



Selección de un único registro:

Mediante `SELECT SINGLE *` podemos leer un solo registro de una tabla o vista de la base de datos. Para asegurarnos de que la entrada que se lee es única, todos los campos clave deben ser especificados en la cláusula `WHERE`. De esta forma, el filtro nos dejará un único registro que cumpla estas características dentro de la tabla. El `*` indica que deben mostrarse todos los campos de la tabla. El `SINGLE` indica que sólo se necesita leer una línea.

El nombre de la estructura o área de trabajo en la que queremos que se copie el registro se inserta después mediante `INTO`. Esta estructura debe ser declarada de forma idéntica a las columnas de la tabla que estamos leyendo.

Si usamos `CORRESPONDING FIELDS OF` en la cláusula `INTO`, podemos usar estructuras o áreas de trabajo no idénticas ya que el sistema solo rellenará en nuestra estructura los campos que tengan nombres idénticos a las columnas de la tabla. Si no incluimos `CORRESPONDING FIELDS`, el sistema rellena la estructura de izquierda a derecha.

Una vez ejecutada la sentencia `SELECT` podemos comprobar si el sistema ha encontrado alguna entrada en la tabla comprobando el campo `SY-SUBRC`. En el caso de que encuentre una entrada, este campo tendrá el valor 0.



Espacio reservado para los elementos de identificación visual de la Entidad

Selección de varios registros:

Si no usamos **SINGLE** en la cláusula **SELECT**, el número de registros que se lean será restringido por las condiciones de la cláusula **WHERE**. Estas condiciones lógicas pueden añadirse usando **AND** o **OR**.

Mediante la cláusula **FROM** seleccionamos la tabla/vista (pueden ser varias) de la que obtenemos los datos y mediante **WHERE**, indicamos qué registros de esa tabla/vista nos interesan.

Podemos usar la sentencia select de 2 formas distintas:

- **SELECT** mediante **ARRAY FETCH** para leer registros e introducirlos en la variable indicada en **INTO**.

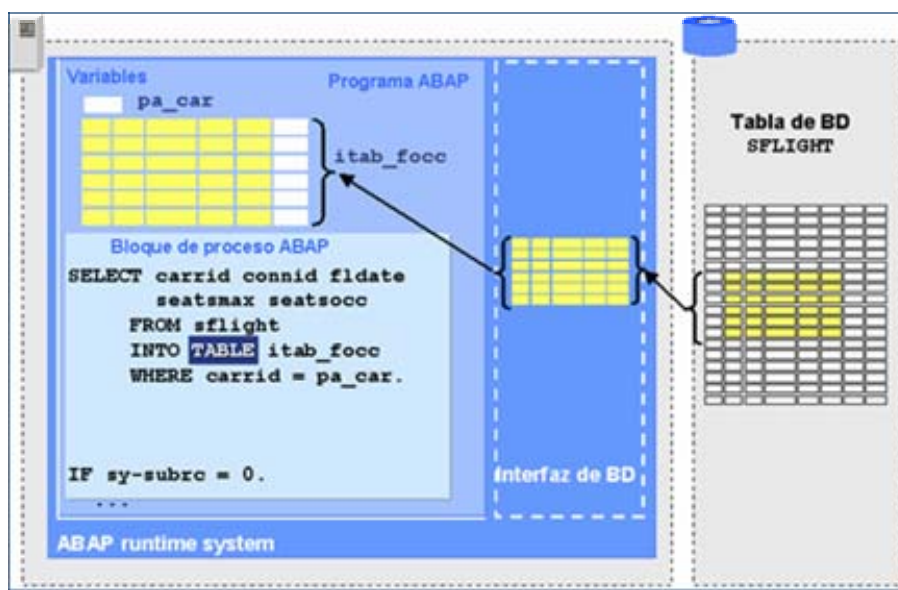
Ejemplo:

```
SELECT airpfrom FROM spfli INTO TABLE itab_airpfrom
WHERE carrid = '001'.
```

Esta instrucción nos añadiría a la tabla itab_airpfrom todos los aeropuertos de salida de la tabla de vuelos de la compañía aérea '001'

INTO TABLE <itab> copia el resultado de la selección en la tabla interna itab. Si luego queremos añadir registros a esta tabla podemos hacerlo mediante **APPENDING TABLE** <itab>

Después de ejecutar la sentencia **SELECT** podemos comprobar si se ha obtenido algún resultado mediante el campo **SY-SUBRC**. El campo sy-subrc tendrá valor 0 si se han seleccionado registros.



**Espacio reservado para los elementos de
identificación visual de la Entidad**

- **SELECT - - - END SELECT.** De esta forma podemos añadir las instrucciones para cada uno de los registros leídos.

Ejemplo:

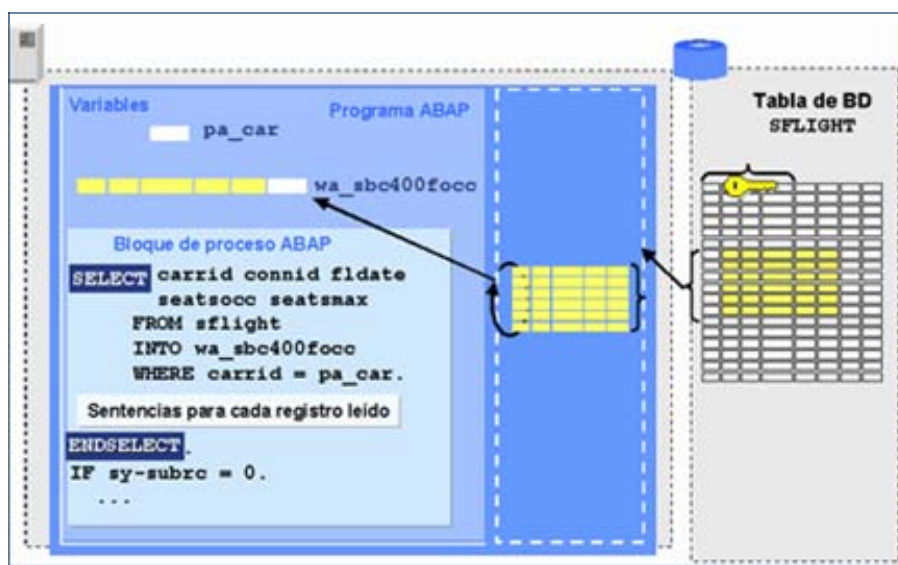
```
SELECT airpfrom FROM spfli INTO wa_airpf where carrid = '001'.
```

<Instrucciones con wa_airpfrom>

```
END SELECT.
```

En el caso de **SELECT- END SELECT** el campo sy-subrc tendrá valor 0 después de **END SELECT** si se ha seleccionado al menos una entrada.

Dentro del loop **SELECT – END SELECT**, podemos comprobar detrás de cada **SELECT** el número de registros leídos mediante **SY-DBCNT**.



**Espacio reservado para los elementos de
identificación visual de la Entidad**

Asignación del resultado a la variable:

Como hemos visto, mediante la cláusula **INTO** especificamos la variable en la que vamos a introducir el dato seleccionado de la base de datos.

Podemos hacerlo de 2 formas:

- Mediante **variables individuales**: Definimos una serie de variables individuales (tantas como campos tenemos en la cláusula SELECT). En la cláusula **INTO** indicamos estas variables en el mismo orden en el que aparecen en el **SELECT**.

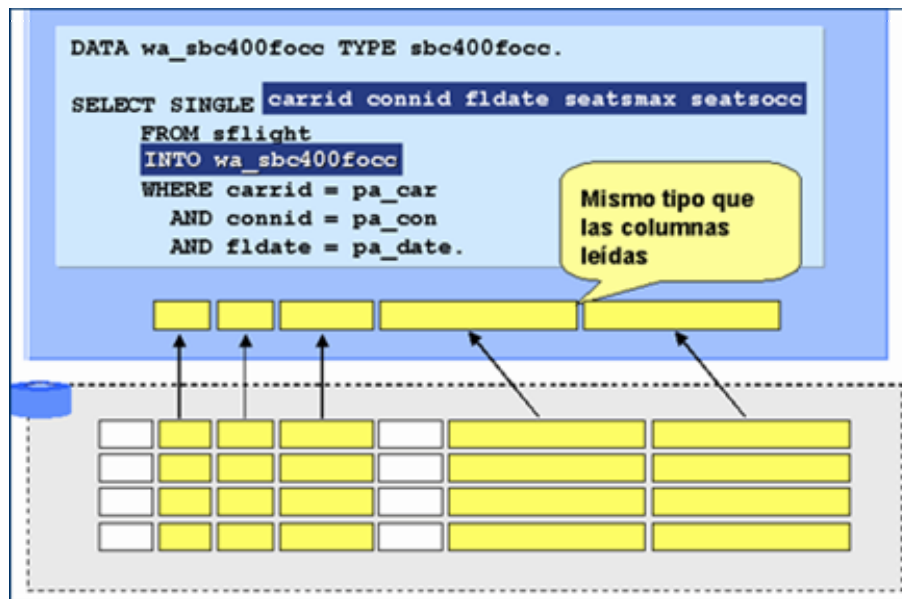
Ejemplo:

```
DATA: gv_carrid    TYPE sflight-carrid,  
      gv_connid    TYPE sflight-connid,  
      gv_fdate     TYPE sflight-fdate,  
      gv_seatsmax   TYPE sflight-seatsmax.  
  
START-OF-SELECTION.  
  
SELECT carrid connid fdate seatsmax  
  FROM sflight  
  INTO (gv_carrid, gv_connid, gv_fdate, gv_seatsmax)  
  WHERE...
```

- Mediante una **estructura** definida previamente: Definimos una estructura en el programa con la misma secuencia de campos que aparecen en la cláusula SELECT. En la cláusula **INTO** indicamos el nombre de esta estructura.

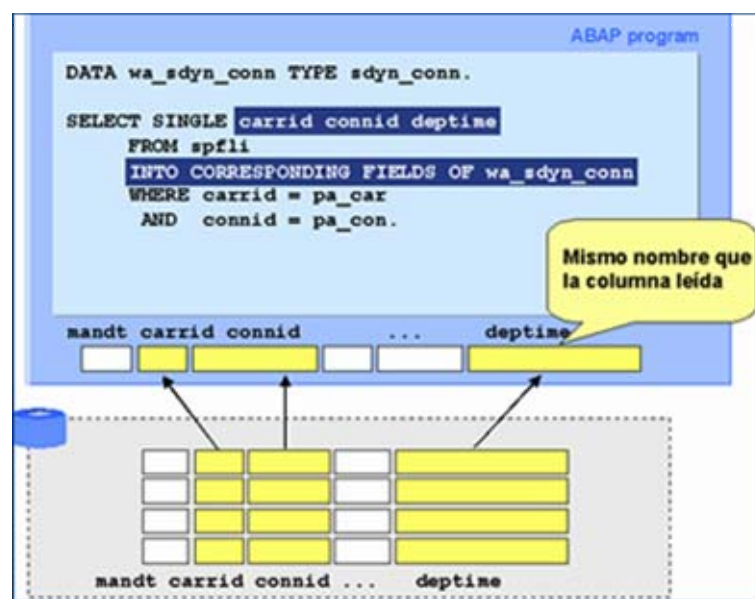
Ejemplo:

Espacio reservado para los elementos de identificación visual de la Entidad



Podemos usar **INTO CORRESPONDING FIELDS** para que los datos se asignen en los campos de la estructura con el mismo nombre. De esta forma no es necesario que la estructura tenga el mismo orden de campos que la lista de los campos en el SELECT. También es más fácil de mantener si hay cambios en el programa ya que, por ejemplo, si se añaden campos a la estructura esto no afectará en la asignación de campos en el SELECT. Por el contrario, INTO CORRESPONDING FIELDS es más lento que INTO.

Para asignar los campos en las columnas de una tabla interna usando un array fetch usamos **INTO CORRESPONDING FIELDS OF TABLE** <itab>.



Agrupación de campos

Cuando hacemos una selección de varios registros, es posible que nos devuelva registros duplicados. Esto puede ocurrir por ejemplo, cuando seleccionamos campos de una tabla que no forman la clave.

En el ejemplo que hemos visto anteriormente:

```
SELECT airpfrom FROM spfli INTO TABLE itab_airpfrom WHERE carrid = '001'.
```

El campo airpfrom no es un campo clave en la tabla spfli por lo que es posible que el mismo aeropuerto aparezca varias veces en los distintos registros de la tabla, es decir, es muy probable que para la compañía aérea que hemos seleccionado existan varios vuelos que salgan desde el mismo aeropuerto. Por lo que en este ejemplo nuestra tabla itab_airpfrom tendría registros duplicados. En función del uso que vayamos a darle a esta tabla, es posible que solo nos interesen ver los distintos aeropuertos que cumplen esta condición por lo que podemos usar la cláusula **GROUP BY**.

```
SELECT airpfrom FROM spfli INTO TABLE itab_airpfrom  
WHERE carrid = '001' GROUP BY airpfrom.
```

De esta forma solo seleccionamos los distintos aeropuertos de salida para la compañía aérea indicada.

En este ejemplo en concreto podemos usar **DISTINCT** en la cláusula **SELECT** y obtendríamos el mismo resultado:

```
SELECT DISTINCT(airpfrom) FROM spfli INTO TABLE itab_airpfrom  
WHERE carrid = '001'.
```

Cuando usamos la cláusula **GROUP BY** todos los campos que aparecen en el select deben aparecer en la cláusula **GROUP BY** excepto si están dentro de una función de agregado:

Las **funciones de agregado** que existen son:

MAX (campo): Devuelve el valor máximo del campo especificado.

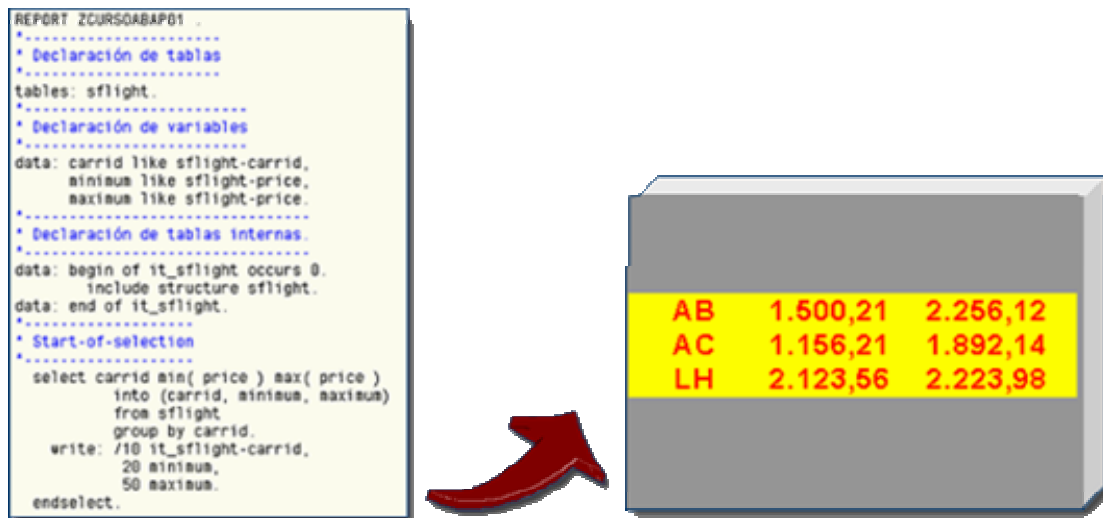
MIN (campo): Devuelve el valor mínimo del campo especificado.

AVG (campo): Devuelve el valor medio del campo especificado.

SUM (campo): Devuelve la suma de todos los registros del campo especificado.

COUNT (campo): Devuelve el número de valores diferentes para el campo especificado.

**Espacio reservado para los elementos de
identificación visual de la Entidad**



Ejemplo:

SELECT airpfrom max(flttime) **FROM** spfli **INTO TABLE** itab_from
WHERE carrid = '001' **GROUP BY** airpfrom.

De esta forma sabemos para cada aeropuerto de salida de una compañía la duración del vuelo más largo.

Espacio reservado para los elementos de identificación visual de la Entidad

Estas funciones de agregado también pueden ser usadas en sentencias select sin GROUP BY.

Ejemplo:

```

REPORT ZCURSABAP02 .
* Declaración de tablas
tables: spfli.
* Declaración de variables
data: maxfield like spfli-distance,
      minfield like spfli-distance,
      counter type i.
* Start-of-selection
select max( distance )
      min( distance )
      count(*)
into (maxfield, minfield, counter)
from spfli.
write: /10 maxfield,
       20 minfield,
       50 counter.

```

CARRID	CONNID	DISTANCE
AB	0350	2.400
AB	0400	520
AC	0350	1.250
AC	0400	3.100
DD	0300	4.800
LH	0400	1.500
LH	0450	2.100

MIN
MAX

Unida a la cláusula GROUP BY tenemos la cláusula **HAVING**. Podemos usar la cláusula HAVING cuando queremos filtrar por condiciones obtenidas tras la agrupación del GROUP BY.

Ejemplo:

```

SELECT airpfrom count(connid) FROM spfli INTO TABLE itab_from
WHERE carrid = '001' GROUP BY airpfrom HAVING count(connid) > 5.

```

Esto nos mostraría para cada aeropuerto de salida de una compañía aérea cuantos vuelos existen siempre que el número de vuelos sea mayor que 5.


Ordenación de resultado:

Para ordenar el resultado obtenido en la sentencia select usaremos la cláusula **ORDER BY**. Podemos ordenar por cualquier campo.

**Espacio reservado para los elementos de
identificación visual de la Entidad**

Sintaxis : SELECT * FROM <nombre_tabla> ORDER BY
<campo1> <campo2> {PRIMARY KEY}.

```
REPORT ZCURSABAP01 .
*.....
* Declaración de tablas
*.....
tables: spfli.
*.....
* Declaración de tablas internas.
*.....
data: begin of it_spfli occurs 0.
      include structure spfli.
data: end of it_spfli.
*.....
* Start-of-selection
*.....
select * from spfli
      order by cityfrom.
endselect.
*.....
* End-of-selection
*.....
loop at it_spfli.
  write: /10 it_spfli-carriid,
        20 it_spfli-connid,
        30 it_spfli-cityfrom,
        60 it_spfli-cityto.
endloop.
```

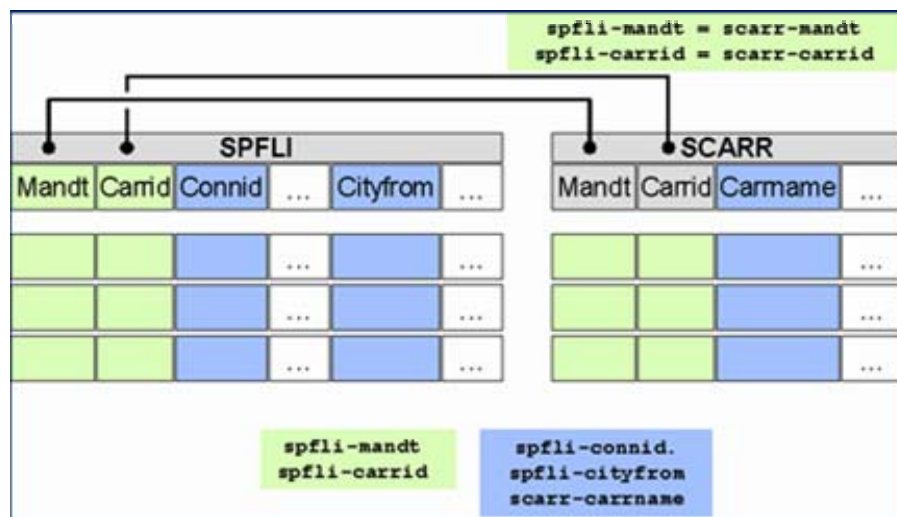


AB	0400	BARNA
AC	0400	BILBAO
DD	0300	BILBAO
LH	0450	CADIZ
AC	0350	MADRID
LH	0400	MÁLAGA
AB	0350	PALMA

Unión de varias tablas

Si queremos unir varias tablas en nuestra sentencia select debemos definir la siguiente información:

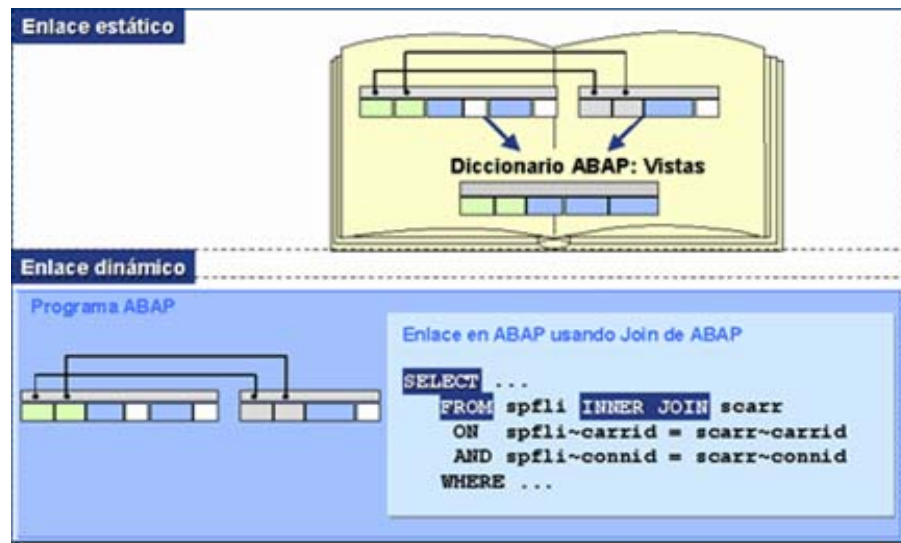
- ¿Qué tablas de la base de datos queremos unir?
- ¿Qué condiciones de enlace queremos entre estas tablas?
- ¿Qué columnas de las tablas queremos mostrar?



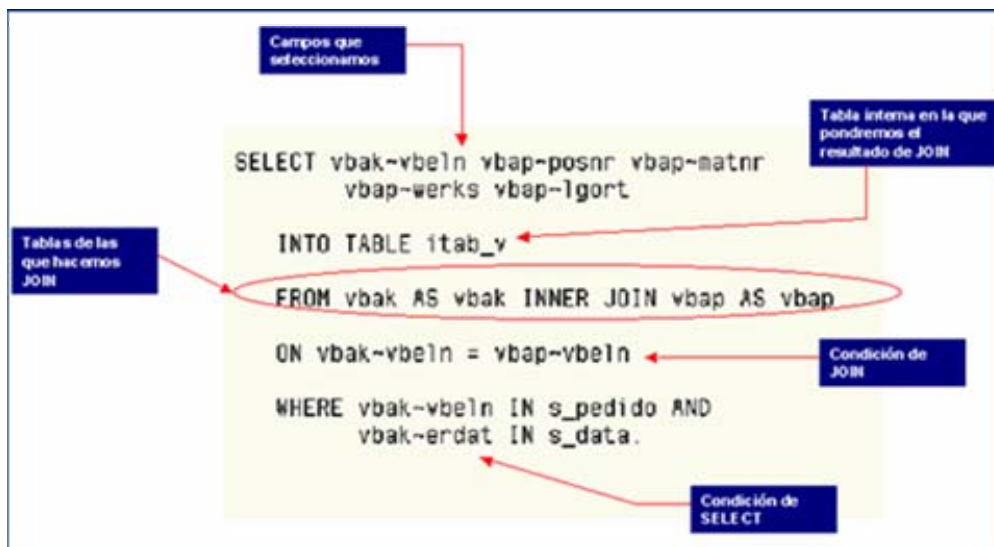
Podemos definir las condiciones de enlace entre estas tablas de forma estática o dinámica:

- **Estática:** Si definimos una vista entre varias tablas estamos haciendo un enlace estático.
- **Dinámico:** Si utilizamos una sentencia select dentro de nuestro programa (ABAP join) la interfaz de la base de datos se genera en tiempo de ejecución.

Espacio reservado para los elementos de identificación visual de la Entidad



Para crear un **Abap Join** debemos saber qué campos son los que vamos a usar de enlace entre las tablas (condición del **JOIN**) y qué tipo de unión queremos hacer entre estas tablas (elegir entre **INNER JOIN** u **OUTER JOIN**).



INNER JOIN

Mediante un **INNER JOIN** seleccionamos los registros que tienen correspondencia entre las 2 tablas. Es decir, para cada línea de la tabla de la izquierda del Join existe una o más líneas en la tabla de la derecha que cumplan las condiciones de unión. Por defecto cuando hablamos de JOIN nos referimos a **INNER JOIN**.

Espacio reservado para los elementos de
identificación visual de la Entidad

INNER JOIN



```
REPORT sapbc405_gdad_inner_join_2tab.  
...  
SELECT spfli~carrid spfli~connid  
       spfli~cityfrom spfli~cityto  
       sflight~fldate sflight~seatsmax  
       sflight~seatsocc  
  INTO TABLE itab flights  
 FROM spfli INNER JOIN sflight  
       ON spfli~carrid = sflight~carrid  
       AND spfli~connid = sflight~connid  
       WHERE spfli~carrid IN so_carr  
       AND spfli~connid IN so_conn.  
...
```

INNER JOIN

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

A	B	D
a1	b1	d1
a3	b2	d2
a3	b3	d3

A	B	C	D
a1	b1	c1	d1
a3	b3	c3	d3

Espacio reservado para los elementos de identificación visual de la Entidad

OUTER JOIN

Mediante un OUTER JOIN seleccionamos registros de la tabla de la izquierda del join aunque no tengan correspondencia en la tabla de la derecha. Es decir, tomamos como principal la tabla de la izquierda y buscamos las coincidencias de unión en la tabla de la derecha si existen.

OUTER JOIN ✓

```
REPORT sapbc405_gdad_outer_join.
...
SELECT scarr-carrid scarr-carrname
       spfli-connid spfli-cityfrom
       spfli-cityto
INTO TABLE itab flights
FROM   scarr LEFT OUTER JOIN spfli
ON     scarr-carrid = spfli-carrid
ORDER BY scarr-carrid spfli-connid.
...
```

LEFT OUTER JOIN

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

A	D	E
a1	d1	e1
a3	d2	e2
a3	d3	e3

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b2	c2		
a3	b3	c3	d2	e2
a3	b3	c3	d3	e3

Otras sentencias SQL:

Además de la sentencia Select de obtención de datos podemos usar otras sentencias para modificarlos.

```
DATA: gv_carrid      TYPE sflight-carrid,
      gv_connid      TYPE sflight-connid,
      gv_fldate      TYPE sflight-fldate,
      gv_seatsmax     TYPE sflight-seatsmax.

START-OF-SELECTION.

SELECT carrid connid fldate seatsmax
FROM sflight
INTO (gv_carrid, gv_connid, gv_fldate, gv_seatsmax)
WHERE...
```

- Añadir una línea a la base de datos:

INSERT <TAB> FROM <WA>.
- Añadir varias líneas de una tabla interna.

INSERT <TAB> FROM TABLE <ITAB>.
- Modificar una línea

**Espacio reservado para los elementos de
identificación visual de la Entidad**

UPDATE <TAB> FROM <WA>.

- Modificar varias líneas

UPDATE <TAB>
SET <f1> = <c1> <f2> = <c2>
WHERE <CONDICIÓN>.

- Modificar varias líneas usando una tabla interna.

UPDATE <TAB> FROM TABLE <ITAB>.

- Añadir o modificar una línea

MODIFY <TAB> FROM <WA>.

- Añadir o modificar varias líneas

MODIFY <TAB> FROM TABLE <ITAB>.

- Borrar una línea

DELETE <TAB> FROM <WA>.

- Borrar varias líneas

DELETE FROM <TAB> WHERE <CONDICIÓN>.

- Borrar varias líneas usando una tabla interna.

DELETE <TAB> FROM TABLE <ITAB>.

**Espacio reservado para los elementos de
identificación visual de la Entidad**

Otros operadores SQL:

Dentro de la cláusula WHERE podemos:

- Usar una máscara dentro del campo especificado mediante **LIKE**.

Estas son las opciones de máscara:

- **'_'** Especifica un carácter:
- **'%'** Especifica una cadena de caracteres
- Todos los símbolos (1...2, A...Z,&,\$,.....)

**Sintaxis : SELECT * FROM <nombre_tabla>
WHERE <campo_tabla> LIKE <máscara>.**

```
REPORT ZCURS0ABAP02 .
*-----
* Declaración de tablas
*-----
tables: spfli.
*-----
* Start-of-selection
*-----
select * from spfli
  where cityfrom like '_R%'.
      write: /10 spfli-carrid,
            20 spfli-connid,
            50 spfli-distance.
endselect.
```

CARRID	CONNID	CITYFROM
AB	0350	SEVILLA
AB	0400	MADRID
AC	0350	ROMA
AC	0400	BARCELONA
DD	0300	BILBAO
LH	0400	ROTTERDAM
LH	0450	LAS PALMAS

- Utilizar **IN** para realizar búsquedas en una tabla. De esta forma se seleccionan todos los registros que en el campo del where contengan los valores especificados. También permite comparar dentro de una tabla interna de selección generada automáticamente mediante **SELECT-OPTIONS** o **RANGES**.

**Sintaxis : SELECT * FROM <nombre_tabla>
WHERE <campo_tabla> IN (<valor1>, <valor2>, ...).**

```
REPORT ZCURS0ABAP02 .
*-----
* Declaración de tablas
*-----
tables: spfli.
*-----
* Start-of-selection
*-----
select * from spfli
  where cityfrom in ('Roma', 'Madrid', 'Bilbao').
      write: /10 spfli-carrid,
            20 spfli-connid,
            50 spfli-distance.
endselect.
```

CARRID	CONNID	CITYFROM
AB	0350	SEVILLA
AB	0400	MADRID
AC	0350	ROMA
AC	0400	BARCELONA
DD	0300	BILBAO
LH	0400	ROTTERDAM
LH	0450	LAS PALMAS

**Espacio reservado para los elementos de
identificación visual de la Entidad**

Confirmar o anular cambios:

A veces puede ser necesario confirmar o retroceder modificaciones en el contenido de una tabla antes de que sean definitivamente almacenadas, actuando sobre una LUW (Logical Unit Work)

Para confirmar cambios en la Base de Datos para toda la LUW.

COMMIT WORK. => marca el final de la LUW e inicia la tarea de actualización.



Para confirmar cambios y esperar a que acabe la tarea de actualización.

COMMIT WORK. AND WAIT.

Para revertir cambios en la Base de Datos.

ROLLBACK WORK.

Una vez se ha ejecutado el COMMIT WORK todos los cambios en la Base de Datos son irreversibles.

Ambas instrucciones cierran todos los cursores de la Base de Datos, por tanto, no deben utilizarse dentro de SELECT.

