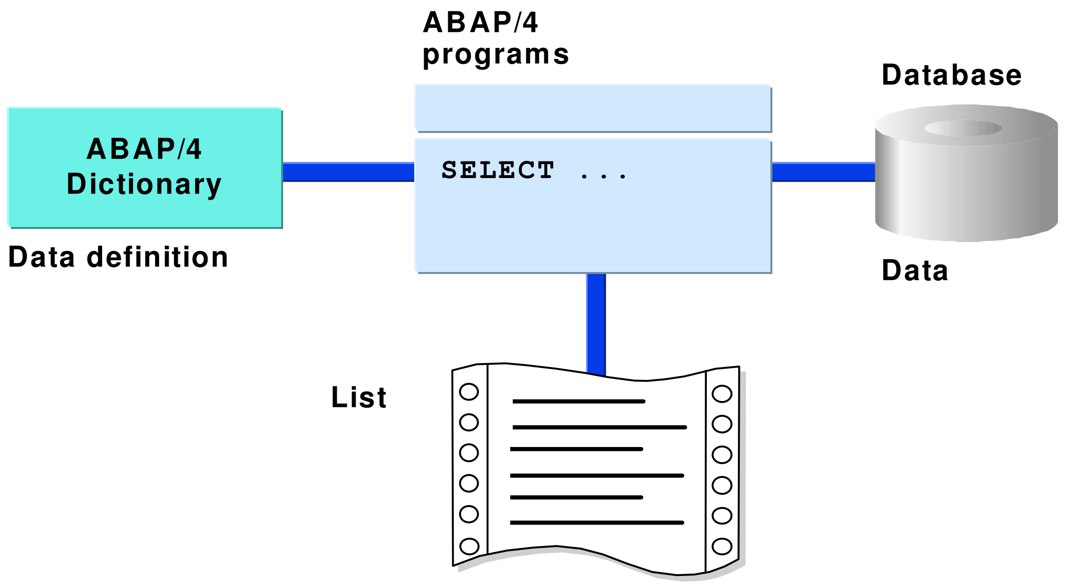
Chapter 08 ABAP/4 Open SQL

- **ABAP/4 Open SQL**

Chapter 08 **Objectives**

- **How to read, change, delete and insert data in database tables with the ABAP/4 Open SQL key words SELECT, MODIFY, UPDATE, DELETE and INSERT**

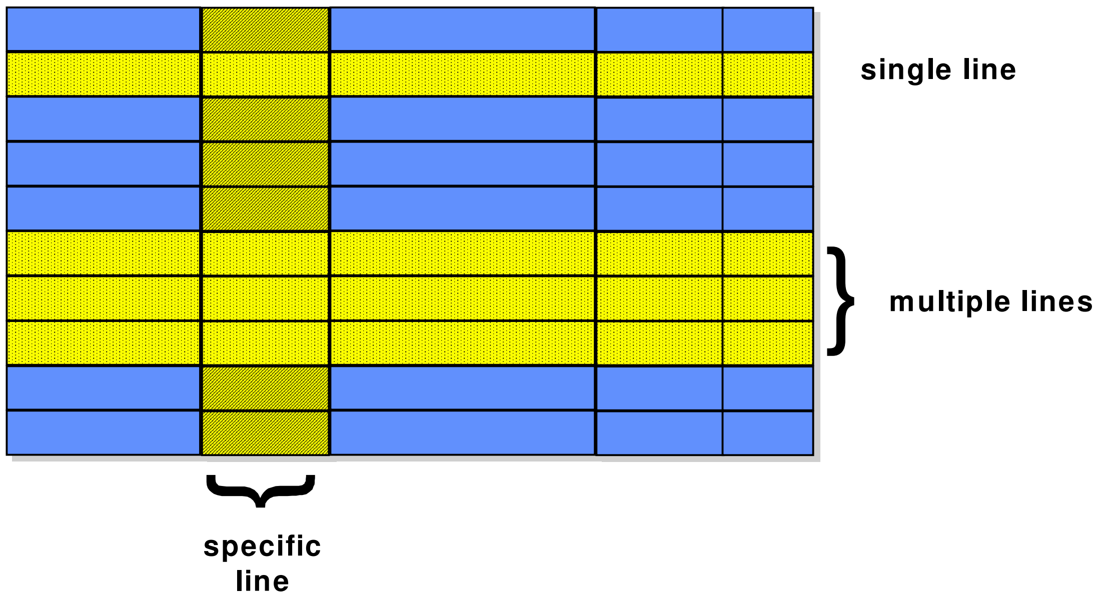
Overview I



- To read data from database tables, you use the SELECT statement.
- ABAP/4 contains statements similar to standard SQL which you can use to access your database. ABAP/4 Open SQL has the following features:

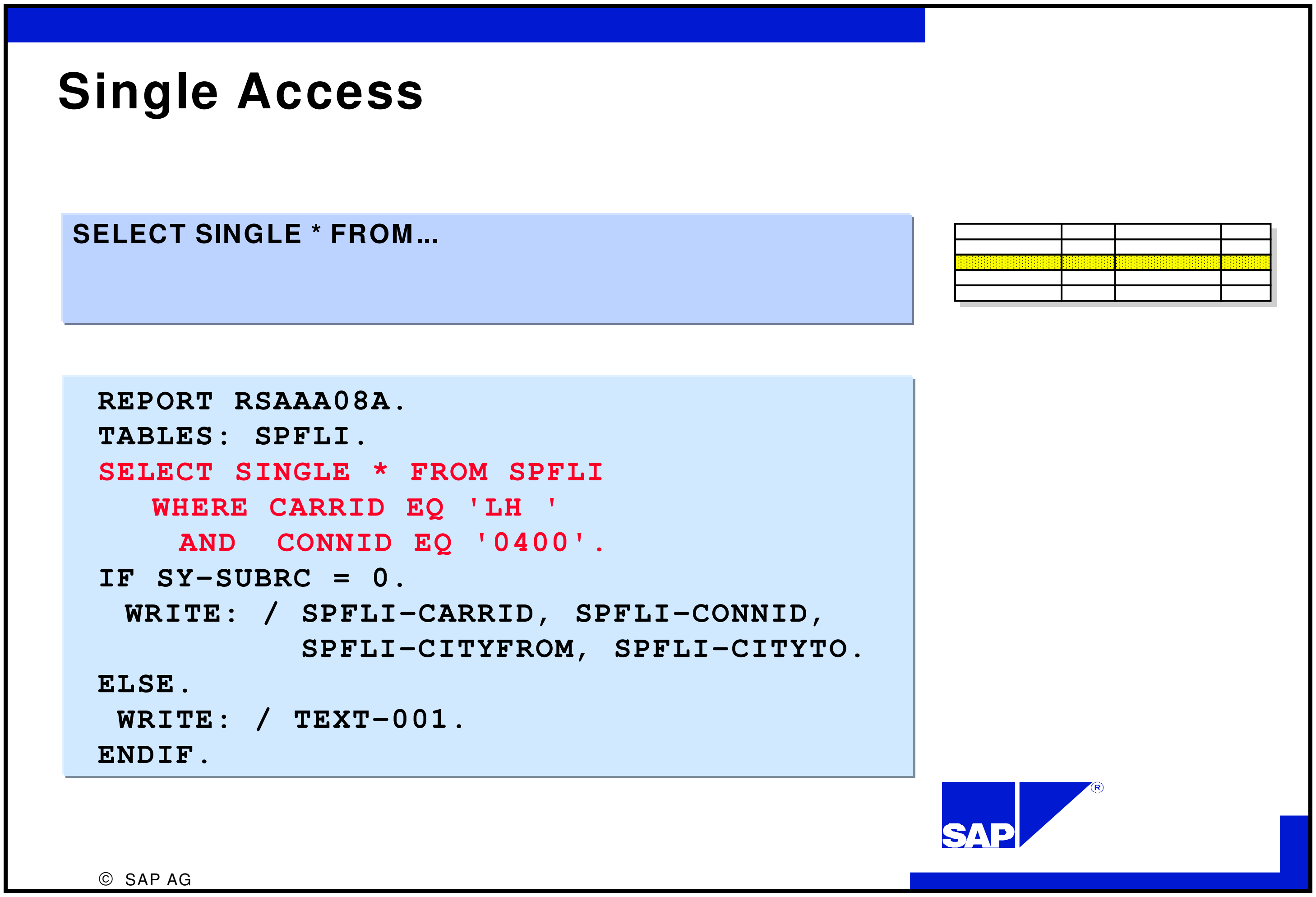
  - ABAP/4 Open SQL syntax corresponds to standard SQL syntax.
  - ABAP/4 Open SQL is a subset of standard SQL.
  - ABAP/4 Open SQL contains SAP short forms.
- You can use the SELECT statement to evaluate database tables via the ABAP/4 Open SQL interface. The tables must be defined in the ABAP/4 Repository and have a primary key.
- **SELECT does not perform any authorization checks.**

Overview II



- The SELECT clause has three variants which are all described in this chapter. They include:

  - reading all the data of a single entry
  - reading all the data of several entries
  - reading the data of specific columns



Single Access

SELECT SINGLE * FROM...

```
REPORT RSAAA08A.
TABLES: SPFLI.
SELECT SINGLE * FROM SPFLI
   WHERE CARRID EQ 'LH '
     AND  CONNID EQ '0400'.
IF SY-SUBRC = 0.
  WRITE: / SPFLI-CARRID, SPFLI-CONNID,
           SPFLI-CITYFROM, SPFLI-CITYTO.
ELSE.
 WRITE: / TEXT-001.
ENDIF.
```

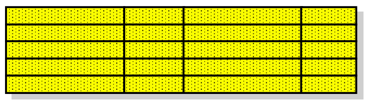
- By using the SELECT statement with the addition SINGLE you can access a single table entry. To do this, you specify the full key of this table entry in the WHERE clause. If the key is not fully qualified, you get a warning message when the syntax check is performed.

  In this case, the first line of the solution set is selected.
- After the statement has been executed, the record appears in the table work area defined with TABLES.
- The return code values have the following meaning:
  - 0: The read was successful
  - 4: The table entry does not exist

Loop Processing without Restriction

```
SELECT * FROM...

ENDSELECT.
```

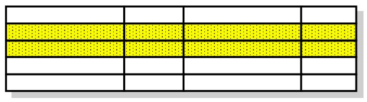


```
REPORT RSAAA08B.
TABLES: SPFLI.
SELECT * FROM SPFLI.
    WRITE: / SPFLI-CARRID, SPFLI-CONNID,
             SPFLI-CITYFROM, SPFLI-CITYTO.
ENDSELECT.
IF SY-SUBRC NE 0.
 WRITE: / TEXT-001.
ENDIF.
```

- If specified as a loop without **any** restriction, the SELECT statement performs a purely sequential read access for every record in the specified table. With larger tables, this obviously affects the runtime.
- Each time the loop is processed, a data record is placed in the table work area defined with TABLES.
- The return codes have the following meaning:
  - 0: The read was successful
  - 4: No entries exists in the table

Loop Processing with Restriction

```
SELECT * FROM...
         WHERE.....
ENDSELECT.
```



```
REPORT RSAAA08C.
TABLES: SPFLI.
SELECT * FROM SPFLI
         WHERE CITYFROM EQ 'FRANKFURT'.
     WRITE: / SPFLI-CARRID, SPFLI-CONNID,
              SPFLI-CITYFROM, SPFLI-CITYTO.
ENDSELECT.
IF SY-SUBRC NE 0.
 WRITE: / TEXT-001.
ENDIF.
```

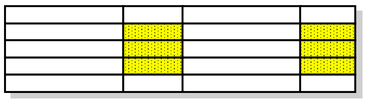
< comparison operators >:

EQ	=	
GE	>=	=>
LE	<=	=<
NE	<>	><
GT	>	
LT	<	

- In the above example, ABAP/4 reads all records in the table SPFLI where the field CITYFROM contains the value FRANKFURT. Each time the loop is processed, the found records are placed in the table work area defined with TABLES.
- You output the fields using a WRITE statement inside the loop.
- The table fields inside the WHERE clause must be addressed without a table prefix (i.e. you should specify ...WHERE CITYFROM = ... and not ...WHERE SPFLI-CITYFROM = ...).
- The conditions you formulate after WHERE in the SELECT statement can consist of several comparisons which you link together with OR or AND operators.
- The return codes have the following meaning:
  - 0: At least one entry satisfies the condition
  - 4: No entry satisfies the condition.

Reading Single Columns I

```
SELECT <a1> <a2>...INTO (<f1>, <f2>, ...) FROM...
         WHERE.....
ENDSELECT.
```

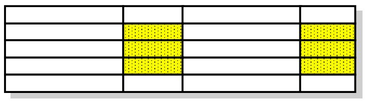


```
REPORT RSAAA08D.
TABLES: SPFLI.
DATA: ZIEL  LIKE SPFLI-CITYTO,
      START LIKE SPFLI-CITYFROM.
SELECT CITYTO CITYFROM INTO (ZIEL, START)
       FROM SPFLI WHERE CARRID = 'LH'.
    WRITE: / START, ZIEL.
ENDSELECT.
IF SY-SUBRC NE 0.
 WRITE: / TEXT-001.
ENDIF.
```

- Until now, we have seen how SELECT * ... always reads all the fields of one or several data records. The above example shows how you can handle single columns.
- Here, just the fields CITYTO and CITYFROM are read from the table SPFLI for the airline carrier 'LH'.
- After the addition INTO of the SELECT statement, you have to specify a work area (in this case the fields ZIEL and START) which is filled each time the loop is processed.
- The argument list of the SELECT clause <a1> <a2>... must contain the same number of elements as the INTO clause (<f1>, <f2>...).

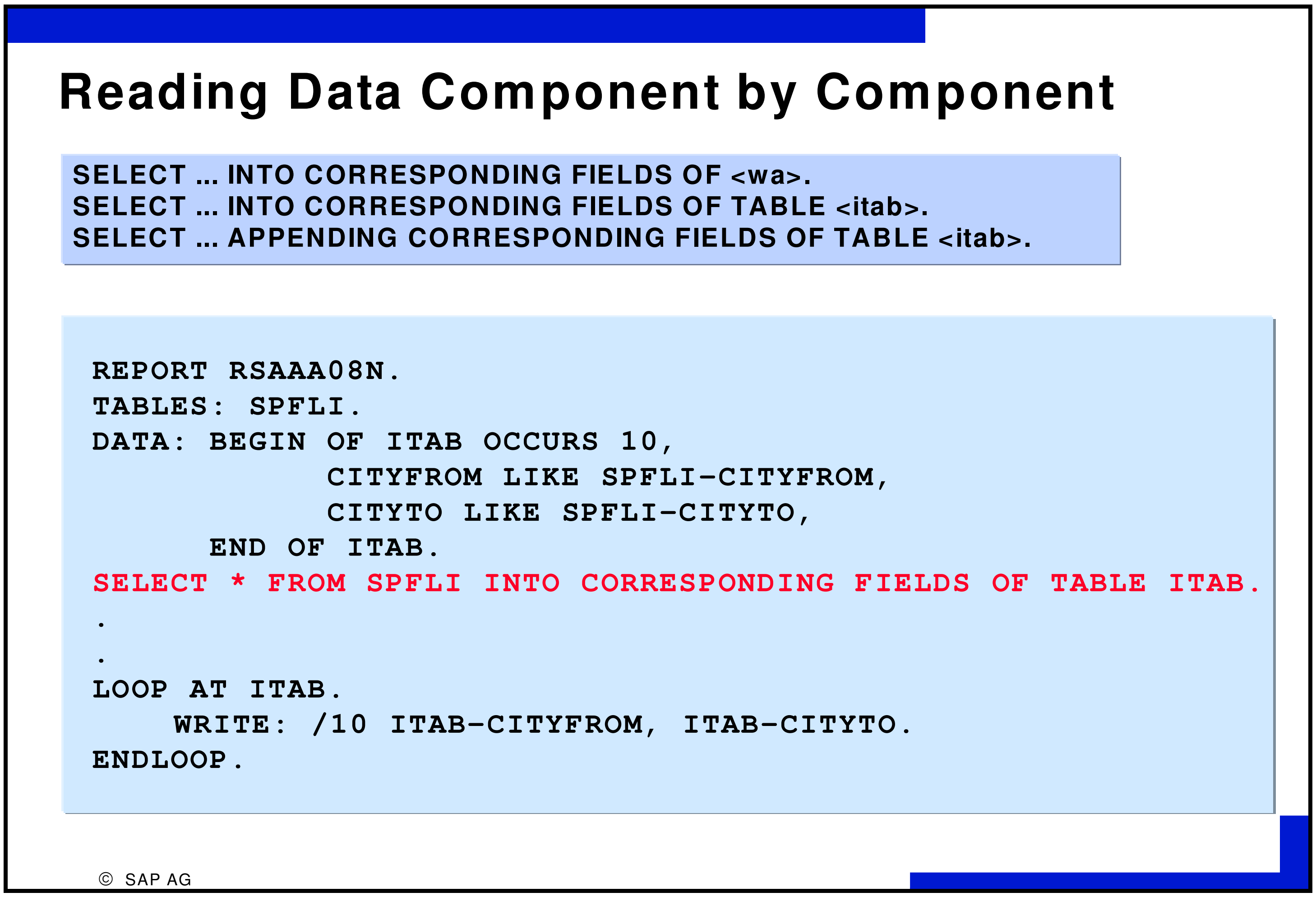
Reading Single Columns II (Aggregate Functions)

```
SELECT MAX( DISTANCE )
       MIN( DISTANCE )
       COUNT(*) FROM <table> INTO (..., ..., ...)
       WHERE.....
```



```
REPORT RSAAA08E.
TABLES: SPFLI.
DATA: MAXFIELD LIKE SFLIGHT-DISTANCE,
      MINFIELD LIKE SFLIGHT-DISTANCE, COUNTER TYPE I.
SELECT MAX( DISTANCE )
       MIN( DISTANCE )
       COUNT(*)
       FROM SPFLI INTO
       (MAXFIELD, MINFIELD, COUNTER).
    WRITE: / TEXT-001, MAXFIELD,
           / TEXT-002, MINFIELD,
           / TEXT-003, COUNTER.
```

- The above example determines the total number of data records in the table SPFLI, as well as the greatest and shortest distance between the departure airport and the destination airport.
- Besides the above-mentioned aggregate functions MIN, MAX and COUNT, you can also use the functions AVG and SUM (but only for numeric fields).
- The opening parenthesis must immediately follow the aggregat ID with no gaps in between and each component in the expression must be separated from the others by ar least one blank.
- If you specify the optional addition DISTINCT, the aggregate function only operates on the different values in a column.
- No ENDSELECT.
- See also the online documentation for the SELECT statement.

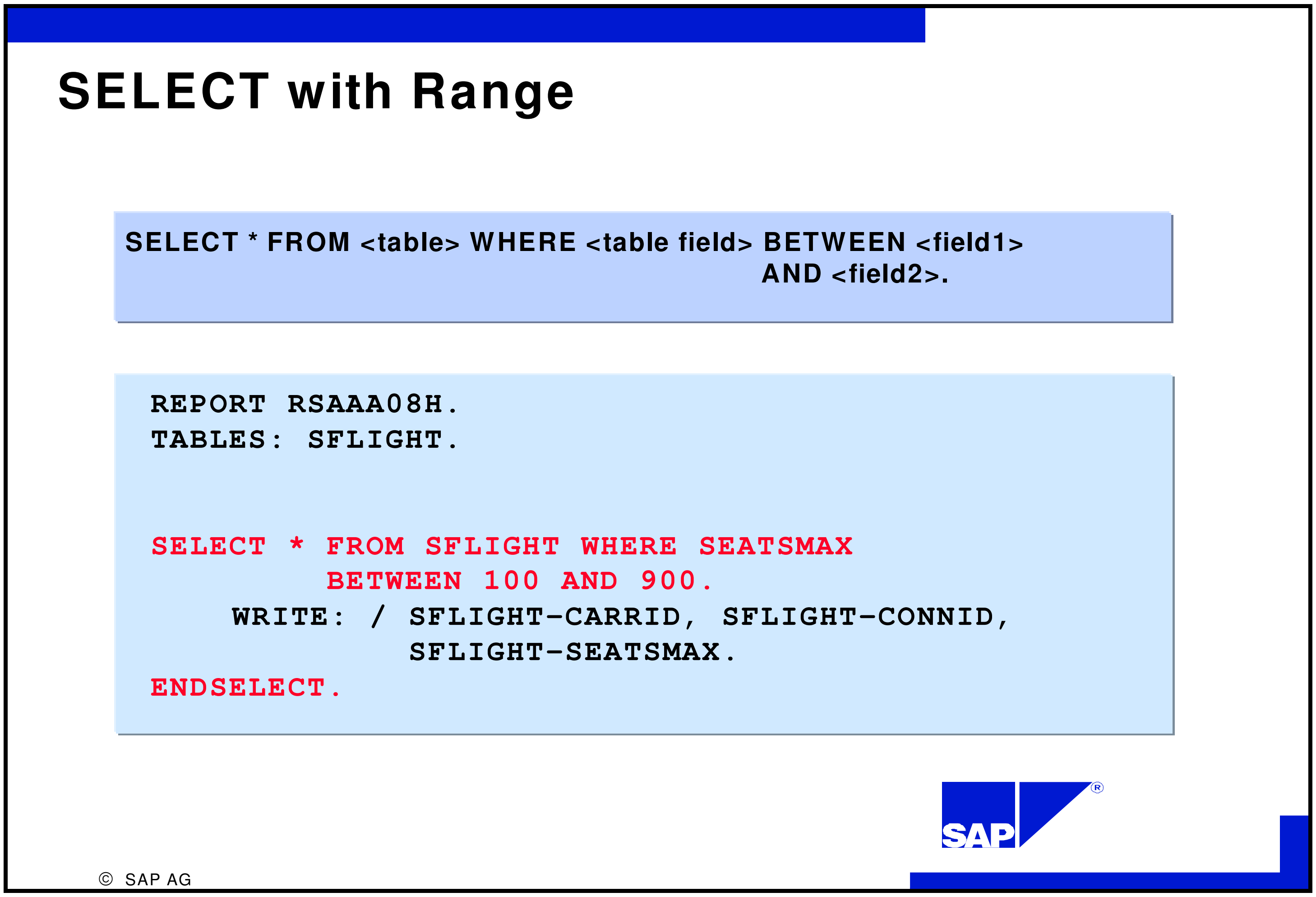


Reading Data Component by Component

```
SELECT ... INTO CORRESPONDING FIELDS OF <wa>.
SELECT ... INTO CORRESPONDING FIELDS OF TABLE <itab>.
SELECT ... APPENDING CORRESPONDING FIELDS OF TABLE <itab>.
```

```
REPORT RSAAA08N.
TABLES: SPFLI.
DATA: BEGIN OF ITAB OCCURS 10,
            CITYFROM LIKE SPFLI-CITYFROM,
            CITYTO LIKE SPFLI-CITYTO,
      END OF ITAB.
SELECT * FROM SPFLI INTO CORRESPONDING FIELDS OF TABLE ITAB.
.
.
LOOP AT ITAB.
    WRITE: /10 ITAB-CITYFROM, ITAB-CITYTO.
ENDLOOP.
```

- To read data component by component into a target area, use the INTO CORRESPONDING FIELDS OF... option.
- Here, not all the fields of the selected lines are read into the work area, but just those columns for which there are identically named components in the target area.
- In the example, only the columns CITYFROM and CITYTO from the table SPFLI are read and transported to the identically named fields in the work area defined for the purpose.
- The addition ...CORRESPONDING FIELDS allows you to fill simple field strings as well as internal tables.
- With internal tables, you can use ...APPENDING CORRESPONDING FIELDS... which does not overwrite any existing table entries, but appends the new ones to those already there.
- When processing internal tables with ...CORRESPONDING... or ...APPENDING CORRESPONDING..., no ENDSELECT is needed in the program code.



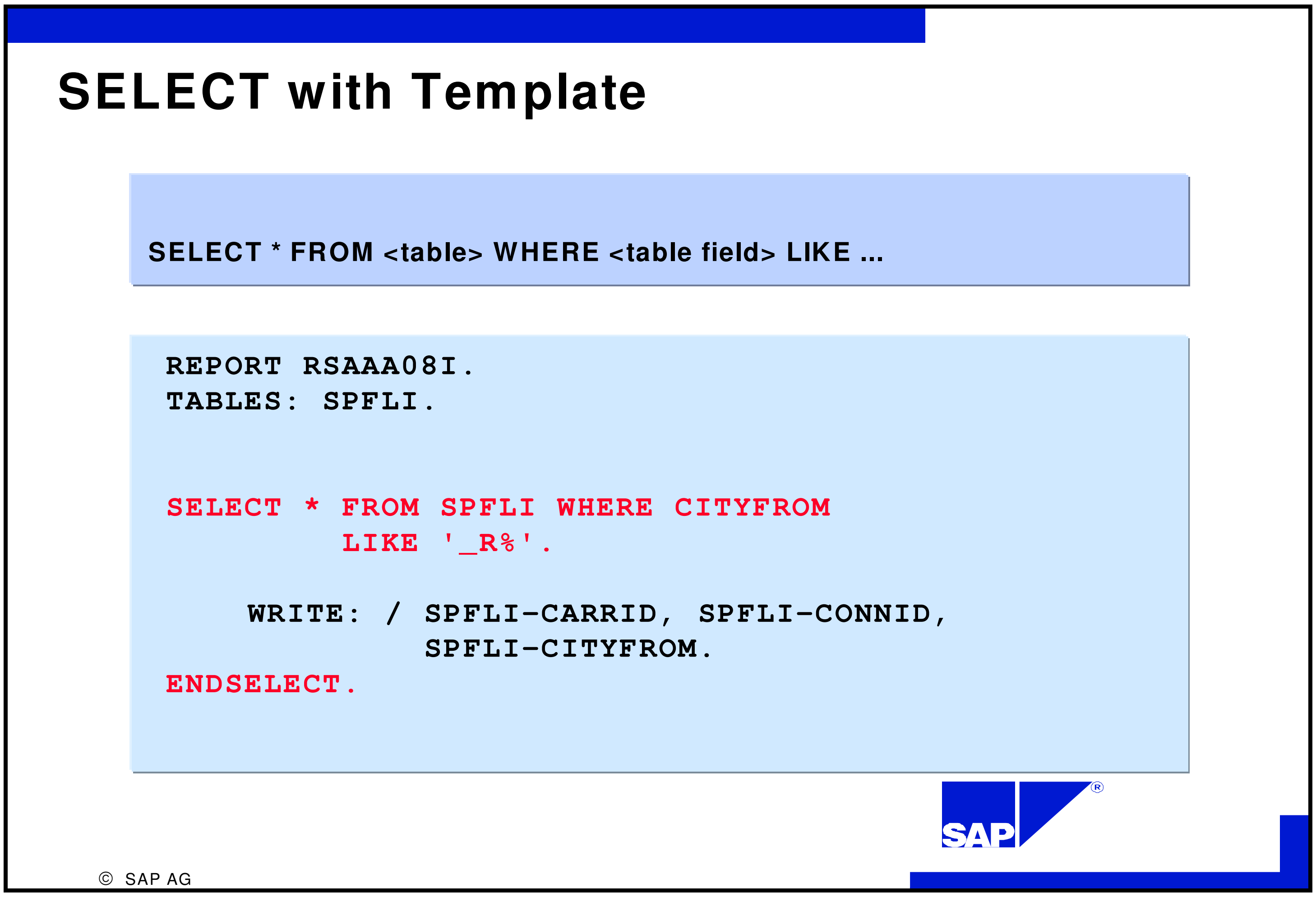
SELECT with Range

```
SELECT * FROM <table> WHERE <table field> BETWEEN <field1>
                                              AND <field2>.
```

```
REPORT RSAAA08H.
TABLES: SFLIGHT.


SELECT * FROM SFLIGHT WHERE SEATSMAX
         BETWEEN 100 AND 900.
    WRITE: / SFLIGHT-CARRID, SFLIGHT-CONNID,
             SFLIGHT-SEATSMAX.
ENDSELECT.
```

- The addition ...BETWEEN... allows you to process a range within the WHERE clause.
- ABAP/4 interprets the lower and upper limits of ranges as part of the specified range.
- The example above reads all flights from the table SFLIGHT where the maximum number of available seats lies between 100 and 900.



SELECT with Template

SELECT * FROM <table> WHERE <table field> LIKE ...

```
REPORT RSAAA08I.
TABLES: SPFLI.


SELECT * FROM SPFLI WHERE CITYFROM
         LIKE '_R%'.

    WRITE: / SPFLI-CARRID, SPFLI-CONNID,
             SPFLI-CITYFROM.
ENDSELECT.
```

- The above example selects all records where an “R” occurs in the 2nd position in the field CITYFROM and is followed by any sequence of characters.
- The characters "_" and "%" have a particular meaning:

" _ "	stands for a single character
"%"	stands for any sequence of characters

  - You can only use wildcard characters with text fields.
  - This usage of wildcard characters in the LIKE condition corresponds to SQL standards.

SELECT with List

SELECT * FROM <table> WHERE <table field> IN   (...., ...).

```
REPORT RSAAA08J.
TABLES: SFLIGHT.


SELECT * FROM SFLIGHT WHERE PRICE
         IN (123, 1000).

    WRITE: / SFLIGHT-CARRID, SFLIGHT-CONNID,
             SFLIGHT-PRICE.
ENDSELECT.
```

- You use the IN operator of the WHERE clause to formulate a comparison with a list of single values.
- Listing single values with IN is the same as linking with OR.
- In the list, you can use either fields or literals.

SELECT with IN Operator

SELECT * FROM <table> WHERE <table field> IN <itab>.

```
REPORT RSAAA08K.
TABLES: SFLIGHT.
DATA: BEGIN OF ITAB OCCURS 10,
        SIGN(1),OPTION(2),LOW LIKE SFLIGHT-PRICE,
                          HIGH LIKE SFLIGHT-PRICE,
      END OF ITAB.
*RANGES: ITAB FOR SFLIGHT-PRICE.
MOVE: 'I'    TO ITAB-SIGN, 'BT' TO ITAB-OPTION,
      '500'  TO ITAB-LOW,  '1000' TO ITAB-HIGH.
      APPEND ITAB.
MOVE: 'I'    TO ITAB-SIGN, 'EQ' TO ITAB-OPTION.
      '440'  TO ITAB-LOW.
      APPEND ITAB.
SELECT * FROM SFLIGHT WHERE PRICE IN ITAB.
      WRITE: / SFLIGHT-CARRID, SFLIGHT-CONNID,
               SFLIGHT-PRICE.
ENDSELECT.
```

- The IN operator of the WHERE clause also allows you make comparisons listed in an internal table.
- You can create a table suitable for this purpose with ...BEGIN OF ... OCCURS... END OF...., with the key word RANGES, or with the SELECT-OPTIONS... statement.
- This internal table must always include the fields SIGN, OPTION, LOW and HIGH.
- After filling the table (with APPEND ITAB.), its contents are evaluated by the WHERE clause of the SELECT statement.
- The example above selects only those records where the price is between 500 and 1000, or is equal to 440.

Dynamic Table Name

```
SELECT * FROM (<table>) INTO <work area>.
```

```
REPORT RSAAA08F.
DATA: BEGIN OF WA,
        LINE(100),
      END OF WA.
PARAMETERS: TABNAME(10) DEFAULT 'SPFLI'.

SELECT * FROM (TABNAME) INTO WA.
     WRITE: / WA-LINE.
ENDSELECT.
```

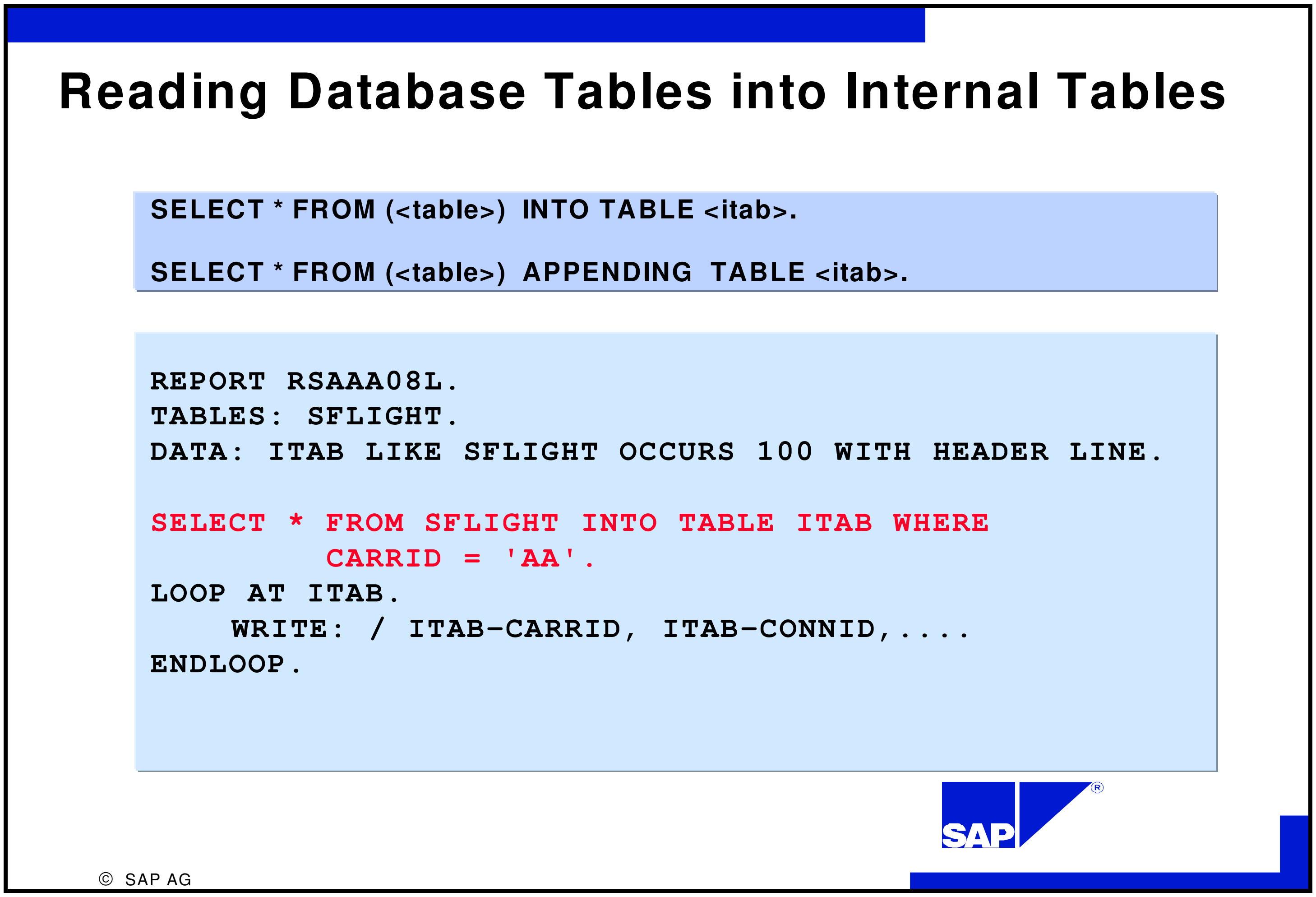
- The name of the database table is not known until specified in a PARAMETERS field at runtime. The contents of this fields thus determine the table name.
- The TABLES statement is not required for this type of name assignment.
- This variant of the SELECT statement can only be used with the addition INTO... .
- Specifying the table name dynamically in a SELECT statement always takes up more CPU time than specifying the name statically in the program.
- The table name is case-sensitive (i.e. upper/lower case should be taken into account).
- Numeric characters in text fields cannot be correctly displayed.

Dynamic WHERE Clause

```
SELECT * FROM (<table>)  WHERE (<itab>).
```

```
REPORT RSAAA08G.
DATA: LINE(72) OCCURS 10 WITH HEADER-LINE, AND(3).
PARAMETRS: PCARRID LIKE SFLIGHT-CARRID,
           PCONNID LIKE SFLIGHT-CONNID.
CONCATENATE 'CARRID = ''' PCARRID '''' INTO LINE.
      APPEND LINE. AND = 'AND'.
CONCATENATE AND ' CONNID = ''' PCONNID '''' INTO LINE.
      APPEND LINE.
SELECT * FROM SPFLI WHERE (LINE).
    WRITE: / SPFLI-CARRID, SPFLI-CONNID,....
ENDSELECT.
```

- With a dynamic WHERE clause, you can leave specification of the selection condition until runtime. The WHERE condition is then taken from an internal table with lines consisting of type C fields and a maximum length of 72 characters.
- You specify the name of the internal table in parentheses without leaving any blanks between each parenthesis and the table name.
- The condition formulated in the internal table must have the same form as a corresponding condition in the source code. Syntax checking cannot, of course, take place until runtime.
- Only literals are allowed as values.



Reading Database Tables into Internal Tables

```
SELECT * FROM (<table>)  INTO TABLE <itab>.

SELECT * FROM (<table>)  APPENDING  TABLE <itab>.
```

```
REPORT RSAAA08L.
TABLES: SFLIGHT.
DATA: ITAB LIKE SFLIGHT OCCURS 100 WITH HEADER LINE.

SELECT * FROM SFLIGHT INTO TABLE ITAB WHERE
         CARRID = 'AA'.
LOOP AT ITAB.
    WRITE: / ITAB-CARRID, ITAB-CONNID,....
ENDLOOP.
```

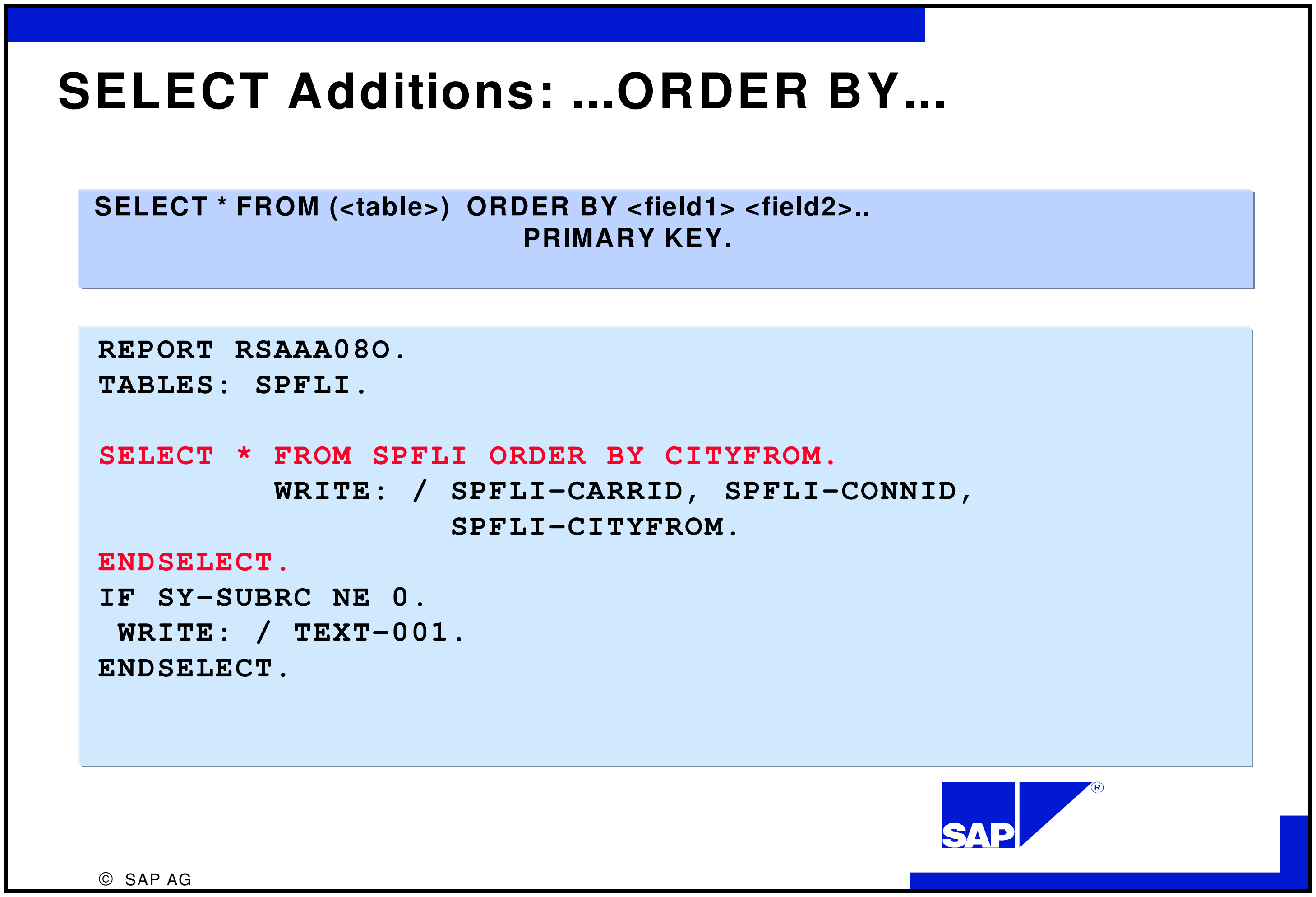
- You can use a single SELECT statement to read records from a database table into an internal table.
- The database system reads the records as a set, not individually, into the internal table. This process is faster than reading the database table in a loop and placing each record in the internal table one-by-one.
- Since there is no loop processing, no ENDSELECT is needed.
- The basic form (...INTO TABLE...) of the statement fills the internal table with the found data records and overwrites any existing entries.
- The variant ...APPENDING TABLE... appends records to existing entries.

SELECT Additions: ...FOR ALL ENTRIES...

```
SELECT * FROM (<table>)  FOR ALL ENTRIES IN  <itab>
                                    WHERE <condition>.
```

```
REPORT RSAAA08M.
TABLES: SPFLI.
DATA: BEGIN OF ITAB OCCURS 10,
            CITYFROM LIKE SPFLI-CITYFROM,
            CITYTO LIKE SPFLI-CITYTO,
      END OF ITAB.
ITAB-CITYFROM = 'FRANKFURT'. ITAB-CITYTO = 'BERLIN'.
APPEND ITAB.
ITAB-CITYFROM = 'NEW YORK'. ITAB-CITYTO = 'SAN FRANCISCO'.
APPEND ITAB.
SELECT * FROM SPFLI FOR ALL ENTRIES IN ITAB WHERE
              CITYFROM = ITAB-CITYFROM AND
              CITYTO   = ITAB-CITYTO.
              WRITE: / ....
ENDSELECT.
```

- The SELECT statement with the addition ...FOR ALL ENTRIES... retrieves all entries that result when the fields of the internal table addressed by the WHERE condition are replaced by the corresponding values of a table entry.
- Duplicate lines are not included in the result set.
- If the internal table contains no entries, the processing acts as if there is no WHERE condition.
- The database field and the field with which it is being compared must have the same type and length.
- ...FOR ALL ENTRIES... excludes the additions ...ORDER BY PRIMARY KEY and ...ORDER BY f1...fn.



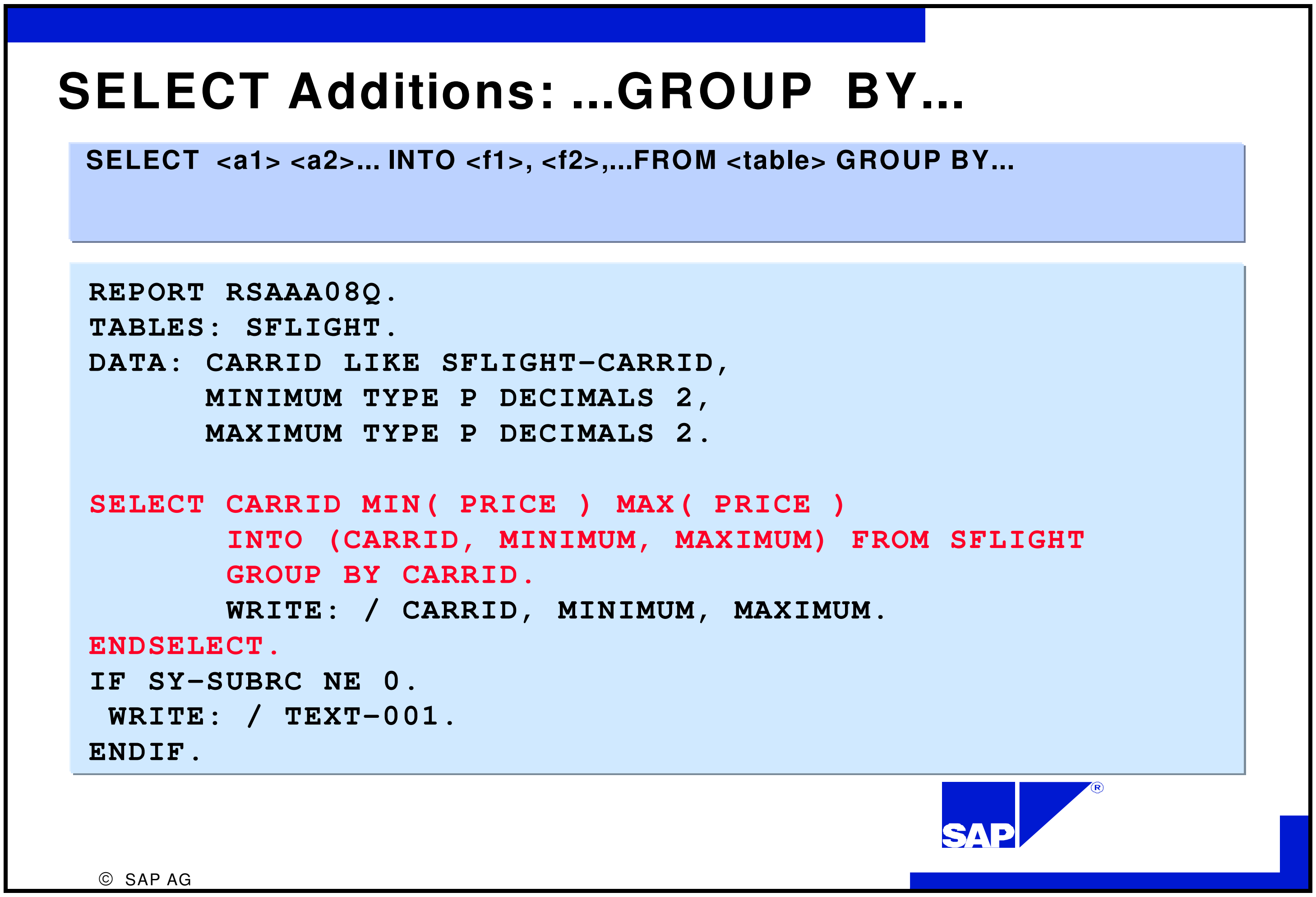
SELECT Additions: ...ORDER BY...

```
SELECT * FROM (<table>)  ORDER BY <field1> <field2>..
                                  PRIMARY KEY.
```

```
REPORT RSAAA080.
TABLES: SPFLI.

SELECT * FROM SPFLI ORDER BY CITYFROM.
         WRITE: / SPFLI-CARRID, SPFLI-CONNID,
                  SPFLI-CITYFROM.
ENDSELECT.
IF SY-SUBRC NE 0.
 WRITE: / TEXT-001.
ENDSELECT.
```

- The addition ...ORDER BY... allows you to retrieve table entries in a particular order.
- ...ORDER BY PRIMARY KEY sorts the entries read by the primary key in ascending order.
- ...ORDER BY f1...fn sorts the entries by the specified table fields.
- You can define the sort sequence explicitly with the additions ASCENDING and DESCENDING.
- The standard sort sequence is ascending order.
- You can also specify the sort fields at runtime, e.g. ...ORDER BY <itab>. In this case, the internal table <itab> must contain the list <f1>...<fn>. The entries of <itab> must be type C and have a maximum length of 72.



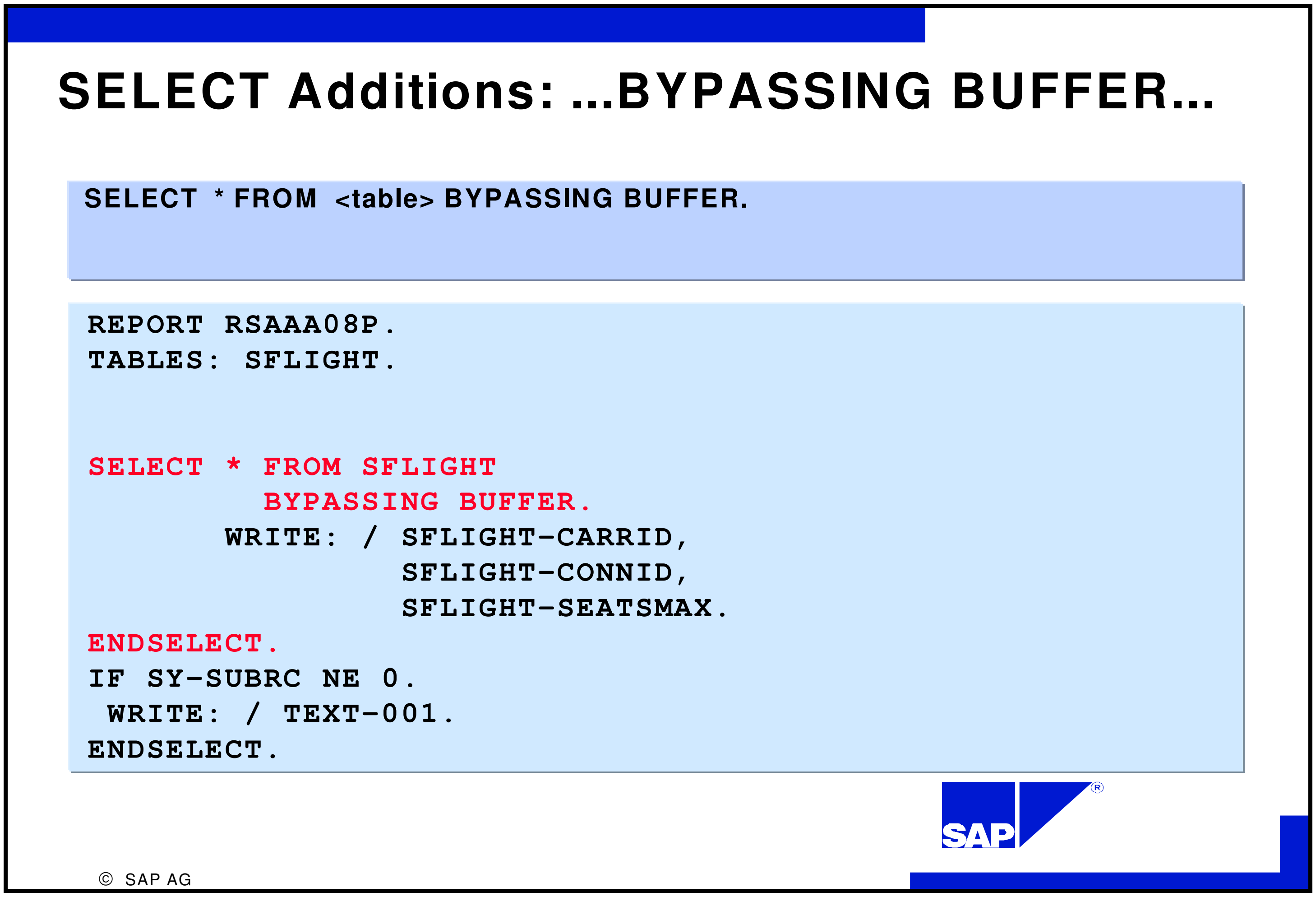
SELECT Additions: ...GROUP BY...

SELECT <a1> <a2>... INTO <f1>, <f2>,...FROM <table> GROUP BY...

```
REPORT RSAAA08Q.
TABLES: SFLIGHT.
DATA: CARRID LIKE SFLIGHT-CARRID,
      MINIMUM TYPE P DECIMALS 2,
      MAXIMUM TYPE P DECIMALS 2.

SELECT CARRID MIN( PRICE ) MAX( PRICE )
       INTO (CARRID, MINIMUM, MAXIMUM) FROM SFLIGHT
       GROUP BY CARRID.
       WRITE: / CARRID, MINIMUM, MAXIMUM.
ENDSELECT.
IF SY-SUBRC NE 0.
 WRITE: / TEXT-001.
ENDIF.
```

- The above example groups the entries by the contents of the field CARRID. It uses aggregate functions to determine the smallest and largest values of the PRICE fields and outputs them.
- A group consists of the entries which are listed in the columns specified after GROUP BY and have the same values.
- With the exception of aggregate expressions, all database fields specified in the argument list of the SELECT clause (in this case CARRID) must also appear in the argument list of the GROUP BY clause.
- If the name of the database table is not known until runtime, you cannot use ...GROUP-BY... .



SELECT Additions: ...BYPASSING BUFFER...

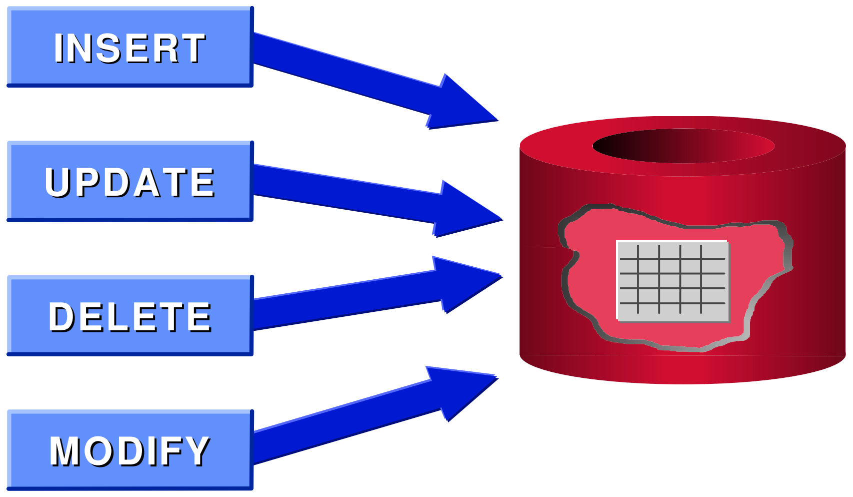
```
SELECT * FROM <table> BYPASSING BUFFER.
```

```
REPORT RSAAA08P.
TABLES: SFLIGHT.


SELECT * FROM SFLIGHT
         BYPASSING BUFFER.
       WRITE: / SFLIGHT-CARRID,
                SFLIGHT-CONNID,
                SFLIGHT-SEATSMAX.
ENDSELECT.
IF SY-SUBRC NE 0.
 WRITE: / TEXT-001.
ENDSELECT.
```

- To keep access times as short as possible, much of the data in the R/3 System is stored in buffers on the application servers, as well as in the database.
- Since these table buffers have to be updated at regular intervals, the data there sometimes may not match the data on the database server.
- If you want to be sure of reading the data in the database, and not the data in the table buffer of the application server, you should use the addition ...BYPASSING BUFFER.

Overview



- The ABAP/4 Open SQL subset includes the update functions UPDATE, INSERT and DELETE.
- These three update accesses are supported by return code values. This means that if the access is successful, the system field SY-SUBRC is set to 0. Otherwise, it contains a value other than 0.
- You can use the MODIFY <table> statement either to change an existing record or to add a new one.
- For further information, please see the appendix or refer to the relevant online documentation.

Chapter 08 Summary

- **You perform read-only and update database accesses with ABAP/4 Open SQL commands.**
- **You can use the SELECT statement to read data from particular columns, as well as single or multiple entries.**

Exercises Chapter 8: ABAP/4 Open SQL

1.

Name of your report:	ZBCA##H1
##:	Group number
Development class:	$TMP (local)

Task:

Create a list containing data from table SPFLI.

The list should include the fields airline carrier (SPFLI-CARRID), flight connection code (SPFLI-CONNID), departure city (SPFLI-CITYFROM) and destination (SPFLI-CITYTO).

The entries should be sorted by airline carrier, departure city and destination.

Allow the user to enter particular airline carriers on the selection screen.

a) Example list (extract)

CARRID	CONNID	CITYFROM	CITYTO
A A	0026	FRANKFURT	NEW YORK
A A	0017	NEW YORK	SAN FRANC.
A A	0064	SAN FRANCISCO	NEW YORK
DL	1699	NEW YORK	SAN FRANC.
DL	1984	SAN FRANCISCO	NEW YORK
LH	2415	BERLIN	FRANKFURT
LH	2407	BERLIN	FRANKFURT
LH	2463	BERLIN	FRANKFURT
LH	2462	FRANKFURT	BERLIN
LH	2402	FRANKFURT	BERLIN
LH	2436	FRANKFURT	BERLIN
LH	0400	FRANKFURT	NEW YORK
LH	0402	FRANKFURT	NEW YORK
LH	0454	FRANKFURT	SAN FRANC.
LH	3577	ROM	FRANKFURT
LH	0455	SAN FRANCISCO	FRANKFURT
.	.	.	.
.	.	.	.

2. Name of your report: ZBCA##H2

Name of your report:	ZBCA##H2
##:	Group number
Development class:	$TMP (local)

Task:

Create a report containing the number of occupied seats for individual airline carriers.

To do this, determine the maximum, the minimum and the average for the field SFLIGHT-SEATSOCC (occupied seats) from the table SFLIGHT.

Output these values in the list.

Allow the user to choose an airline carrier on the selection screen.

b) Example list

```
----------------------------------------------------------------------------
Occupied seats for airline carrier: LH
----------------------------------------------------------------------------
MAXIMUM    :          38
MINIMUM    :          10
Average    :          26.65
```

3. Name of your report:	ZBCA##H3
##:	Group number
Development class:	$TMP (local)

Task:

Use Native SQL to determine the

flight duration (FLTIME)

departure city (CITYFROM)

destination (CITYTO)

of the flight "LH" (CARRID) "0400" (CONNID) from the table SPFLI.

Solutions Chapter 8: ABAP/4 Open SQL

1. REPORT RSAAA081.

```
TABLES: SPFLI.

SELECT-OPTIONS: SGES FOR SPFLI-CARRID DEFAULT ´AA´ TO ´LH´

SELECT * FROM SPFLI WHERE CARRID IN SGES ORDER BY CARRID CITYFROM

                 CITYTO.

WRITE: /10 SPFLI-CARRID,

        20 SPFLI-CONNID,

        30 SPFLI-CITYFROM,

        50 SPFLI-CITYTO.

ENDSELECT.


IF SY-SUBRC NE0.

  WRITE: / TEXT-001.

ENDIF.
```

2. REPORT RSAAA082.

```
TABLES: SPFLIGHT

PARAMETERS: SGES LIKE SFLIGHT-CARRID DEFAULT ´LH´.


DATA:   MAXFIELD LIKE SFLIGHT-SEATSOCC,

        MINFIELD LIKE SFLIGHT-SEATSOCC,

        AVGFIELD(6) TYPE P DECIMALS 2.


SELECT MAX( SEATSECC )

         MIN( SEATSOCC )

         AVG( SEATSOCC )

         FROM SFLIGHT INTO (MAXFIELD, MINFIELD, AVGFIELD)

              WHERE CARRID = SGES.

SKIP.


WRITE: / TEXT-001, SGES.

SKIP.

ULINE.

WRITE: / TEXT-002, MAXFIELD.

        SKIP.

WRITE: / TEXT-003, MINFIELD.

        SKIP.

WRITE: / TEXT-004, AVGFIELD.
```

1. REPORT RSAAA083.

```
DATA:   DCARRI  LIKE SPFLI-CARRID VALUE ´LH´,
        DCONN   LIKE SPFLI-CONNID VALUE ´0400´,
        DCITYF  LIKE SPFLI-CITYFROM,
        DCITYT  LIKE SPFLI-CITYTO,
        DFLTIME LIKE SPFLI-FLTIME.


EXEC SQL.
SELECT CARRID, CONNID, CITYFROM, CITYTO, FLTIME
         INFO :DCARRI, :DCONN, :DCITYF, :DCITYT, :DFLTIME
         FROM SPFLI  WHERE CARRID = :DCARRI
                         AND CONNID = :DCONN
ENDEXEC.


WRITE: / DCARRI, DCONN, DCITYF, DCITYT, DFLTIME.
SKIP.
ULINE.
```