**ABAP SQL**

**1. Introduction**

💡 ABAP SQL is a subset of SQL which is the standardized language for accessing databases.

The main ABAP SQL keywords to read and change data are the following:

|  |  |
| --- | --- |
| **Keyword** | **Purpose** |
| **SELECT** | Reads data from database tables |
| **INSERT** | Adds rows to database tables |
| **UPDATE** | Changes the content of rows of database tables |
| **MODIFY** | Inserts rows into database tables or changes the content of existing rows |
| **DELETE** | Deletes rows from database tables |

💡 ABAP SQL statements use the ABAP SQL interface. This interface transforms all ABAP SQL statements that access the standard database of an AS ABAP to platform-dependent SQL and forwards the results to the database system.

💡 Generally bear in mind the performance notes when using ABAP SQL. The considerations there are not relevant for this cheat sheet since the focus is on syntax options.

**ABAP SQL - Performance Notes**

💡 Hiệu suất chương trình thường phụ thuộc vào sự hiệu quả của các lượt đọc cơ sở dữ liệu. Trong môi trường client/server, mỗi lượt đọc cơ sở dữ liệu gây tải cho cả hệ thống cơ sở dữ liệu và kết nối với AS ABAP. Để có hiệu suất tốt, cần giữ tải này ở mức thấp nhất.

* **Giữ số lần truy vấn thấp**
  + Chọn ít hàng nhất có thể bằng cách sử dụng điều kiện cụ thể, không nên chuyển các hàng không cần thiết từ cơ sở dữ liệu đến AS ABAP.
* **Giữ khối lượng dữ liệu thấp**
  + Chỉ chuyển các cột cần thiết, có thể dùng các view hoặc biểu thức tổng hợp để giảm dữ liệu chuyển.
* **Giữ số lượt đọc thấp**
  + Sử dụng các thao tác hàng loạt thay vì thao tác đơn lẻ, tránh sử dụng câu lệnh SQL trong vòng lặp. Thay vào đó, có thể sử dụng các phép joins, các view hoặc các subquery khi đọc nhiều bảng cơ sở dữ liệu DDIC.
* **Sử dụng bộ đệm cục bộ**
  + Không đọc cùng dữ liệu nhiều lần, lưu các bảng cơ sở dữ liệu vào table buffering để tiết kiệm thời gian. Các bảng DDIC được đọc thường xuyên và ít sửa đổi nên được bộ đệm. Nếu dữ liệu giống nhau cần được sắp xếp nhiều lần theo các thứ tự khác nhau, dữ liệu nên được sắp xếp trong chương trình ABAP.
* **Tìm kiếm hiệu quả bằng cách sử dụng các chỉ mục (indexes)**
  + Trong tất cả các trường hợp mà các secondary indexes của các bảng cơ sở dữ liệu DDIC cải thiện hiệu suất chọn lọc, các chỉ mục này nên được tạo và sử dụng.

**2. Excursion: Database Tables and Views**

💡 This section provides bullet points on database tables and views which contain persisted data. Note that the code snippets in this cheat sheet focus on database tables as [data source](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abendata_source_glosry.htm) for ABAP SQL statements.

**Database tables in**[**AS ABAP**](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenas_abap_glosry.htm)**...**

* are objects of the [ABAP Dictionary (DDIC)](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenabap_dictionary_glosry.htm). The term *database table* describes a physical database table in the current [standard database](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenstandard_db_glosry.htm).
* are two-dimensional matrices consisting of rows and columns.
* contain a [table key](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abentable_key_glosry.htm), i. e. a field or a combination of fields uniquely identifies every row in a table. A [primary key](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenprimary_key_glosry.htm) must exist for every database table.
  + Note the concept of [foreign keys](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenforeign_key_glosry.htm) in which one or more columns of a database table can be primary keys of another table. See more information [here](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenddic_database_tables_forkey.htm).
* have a [flat structure](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenflat_structure_glosry.htm) type. Plus, the definition of database tables consists of [technical](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenddic_database_tables_techstruc.htm) and [semantic](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenddic_database_tables_semastruc.htm) properties.
* can be referenced as a data type and can be accessed using ABAP SQL.

Find more information in the respective (sub)topics in the ABAP Keyword Documentation [here](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenddic_database_tables.htm).

💡 **Views ...**

* are further ABAP Dictionary objects for grouping columns from one or more database tables, among others.
* usually realize a [join](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenjoin_glosry.htm) with defined [join conditions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenjoin_condition_glosry.htm).
* Note:
  + Similar to database tables, the columns of such a view form a flat structure. The view's name can be used, for example, as [data types](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abendata_type_glosry.htm) to declare [data objects](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abendata_object_glosry.htm), too.
  + The views can be accessed by ABAP SQL, especially for reading purposes using SELECT.

💡 **"Classic"** [DDIC Views](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm?file=abenddic_view_glosry.htm) ...

* are the oldest form of views and are not available in [ABAP Cloud](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenabap_cloud_glosry.htm).
* can be accessed by ABAP SQL for read and write operations, however, writing is only supported if the view is created with only one database table.
* can only be created in the [ABAP Workbench](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm?file=abenabap_workbench_glosry.htm).

💡 **"Modern" Views (since release 7.40)**

* [External views](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm?file=abenexternal_view_glosry.htm) as proxies for [SAP HANA views](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm?file=abenhana_view_glosry.htm) (attribute view, analytic view, calculation view)
  + SAP HANA Views are entities of the SAP HANA database that are defined using the [SAP HANA Studio](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm?file=abenhana_studio_glosry.htm).
  + They are based on HANA-specific data types.
  + Using external views of the ABAP dictionary, you can make those SAP HANA views "known" to the ABAP program. In doing so, the external views can be used like classic DDIC views as structured data types and as a source for reading operations with ABAP SQL.
  + To be used only if the central database of the AS ABAP is an [SAP HANA database](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenhana_database_glosry.htm).
* [ABAP Core Data Services (ABAP CDS)](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_view_glosry.htm) ...
  + serve the purpose of defining semantically rich data models.
  + have a lot more options than classic views, for example, they support [annotations](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_annotation_glosry.htm). Data sources can be combined using [associations](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_association_glosry.htm), views can be defined with input parameters, and more.
  + are used like a classic database view as structured data types and used as a source for reading operations with ABAP SQL (using [SELECT](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapselect.htm)).
  + are created using [Data Definition Language (DDL)](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenddl_glosry.htm) in the [ADT](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenadt_glosry.htm) (that is, a source code editor, in contrast to a form-based editor).
  + are, in contrast to external views, supported by all database systems (that support the ABAP CDS characteristics).

**3. Retrieving Data Using SELECT**

**3.1 Basic Syntax**

💡 You use ABAP SQL SELECT statements to read data from one or more database tables (or views). This can be done to create a multirow or single row result set by assigning the result set to a suitable data object, i. e. you can store the multirow read result in an internal table or the single row result in a structure. The SELECT statement includes several clauses that serve different purposes. The following code snippet shows the basic syntax (see the note below for a different but interchangeable syntax):

SELECT FROM source   "What database table or view to read from

  FIELDS field\_list  "What columns should be read

  WHERE condition    "Specifies conditions on which a row/rows should be read

  INTO @target.      "Data object to which the result set is assigned (preceded by @)

💡 Especially in older ABAP programs, you will see other forms of the SELECT syntax that you should no longer use. Strict syntax check modes might enforce the use of specific ABAP SQL syntax. For example, the INTO clause should be placed after the other clauses. Furthermore, host variables or host expressions are required for data objects and expressions, i. e. they must be preceded by @ or @( ... ). Further information: Release-Dependent Syntax Check Modes (F1 docu for standard ABAP).

💡 Host Variables: Là các biến dữ liệu trong chương trình ABAP, sử dụng với ký tự @ trong câu lệnh SQL để truyền dữ liệu hoặc lưu kết quả.

💡 Host Expressions: Là các biểu thức phức tạp hơn, được đánh dấu bởi @( ... ), cho phép sử dụng các phép toán và hàm ABAP trực tiếp trong câu lệnh SQL.

DATA: lv\_offset TYPE i VALUE 5,

      lt\_results TYPE TABLE OF users.

SELECT \*

  FROM users

  INTO TABLE @lt\_results

  WHERE age > @( 30 + lv\_offset ).

💡 Regarding host variables as in SELECT ... INTO @target. and since they are used in most examples below: A host variable is a data object that is

* declared in the ABAP program
* prefixed with the @ character and
* specified in an [operand position](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenoperand_position_glosry.htm) of an ABAP SQL statement. See more information [here](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenhost_variable_glosry.htm).

💡 The SELECT list, i. e. the fields that are specified, can also be specified following the SELECT keyword before the FROM clause - without FIELDS. The following two SELECT statements are basically the same but differently arranged. The code snippets in the cheat sheet randomly use one syntax or the other.

SELECT FROM dbtab

  FIELDS comp1, comp2, comp3

     ...

 SELECT comp1, comp2, comp3

   FROM dbtab

     ...

💡 Regarding the target into which data is read: Instead of using a variable that is (extra) declared beforehand, you can also make use of [inline declarations](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abeninline_declaration_glosry.htm), for example ... INTO TABLE @DATA(itab)., to comfortably create an appropriate variable in place. Note that in case of internal tables as targets, the resulting table is a standard table and has an empty key which might have an impact when further processing the internal table entries. Find more information in the ABAP cheat sheet [Internal Tables](https://github.com/SAP-samples/abap-cheat-sheets/blob/main/01_Internal_Tables.md). The declaration operator [FINAL](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenfinal_inline.htm) can be used to declare immutable variables.

💡 Many syntax examples in this cheat sheet show a selection from dbtab denoting a database table as a source. However, other data sources can also be specified.

" Views

SELECT \* FROM view\_name INTO TABLE @DATA(itab).

" Joins

SELECT a~field1, b~field2

  FROM table\_a AS a

  INNER JOIN table\_b AS b ON a~key = b~key

  INTO TABLE @DATA(itab).

" Common Table Expressions (CTEs): Các biểu thức bảng tạm thời trong SQL

WITH cte AS (SELECT field1, field2 FROM table\_name)

SELECT \* FROM cte INTO TABLE @DATA(itab).

" Table Functions

SELECT \* FROM table\_function(params) INTO TABLE @DATA(itab).

**3.2 Using SELECT for Multiple Purposes**

**Reading a single row into a structure**

SELECT FROM dbtab

  FIELDS \*              "All fields

  WHERE ...

  INTO TABLE @itab.     "itab has an appropriate row type

"Alternative syntax without the FIELDS addition

SELECT comp1, comp2, comp3          "Selected set of fields

  FROM dbtab

  WHERE ...

  INTO TABLE @DATA(lv\_itab).        "Internal table declared inline

"Selected set of fields, existing variable

"See the note on CORRESPONDING FIELDS OF above

SELECT FROM dbtab

  FIELDS comp1, comp2, comp3                "Selected set of fields

  WHERE ...

  INTO CORRESPONDING FIELDS OF TABLE @itab.

💡 Về bổ sung CORRESPONDING FIELDS OF trong mệnh đề INTO: Như đã đề cập, chỉ nội dung của các cột có tên giống hệt với các thành phần trong đích (target) mới được gán. Tuy nhiên, nếu bạn muốn đọc dữ liệu vào một đối tượng dữ liệu hiện có và các trường cụ thể được chỉ định trong danh sách SELECT mà bổ sung này không được chỉ định, bạn có thể gặp kết quả không mong muốn.

* **CORRESPONDING FIELDS OF**: Đây là một bổ sung trong mệnh đề INTO, chỉ định rằng chỉ các cột có tên giống với các thành phần trong đối tượng đích mới được gán giá trị.
* Đối tượng dữ liệu đích phải có đủ thành phần, và nội dung của các cột sẽ được gán cho các thành phần của đích từ trái sang phải theo thứ tự được chỉ định sau SELECT.
* Nội dung của các thành phần thừa của đích sẽ không thay đổi.
* Chú ý đến các quy tắc gán. Quy tắc cơ bản:
  + Không có CORRESPONDING: tên cột không quan trọng, chỉ vị trí quan trọng.
  + Có CORRESPONDING: vị trí cột không quan trọng, chỉ tên quan trọng.

💡 **Reading multiple rows into an internal table**.

SELECT FROM dbtab

  FIELDS \*              "All fields

  WHERE ...

  INTO TABLE @itab.     "itab has an appropriate row type

"Alternative syntax without the FIELDS addition

SELECT comp1, comp2, comp3          "Selected set of fields

  FROM dbtab

  WHERE ...

  INTO TABLE @DATA(lv\_itab).        "Internal table declared inline

"Selected set of fields, existing variable

"See the note on CORRESPONDING FIELDS OF above

SELECT FROM dbtab

  FIELDS comp1, comp2, comp3                "Selected set of fields

  WHERE ...

  INTO CORRESPONDING FIELDS OF TABLE @itab.

💡 **SELECT loop: Sequentially reading multiple rows**.

* A SELECT loop can be opened if the assignment is made to a structure and the addition SINGLE is not used.
* If the row is found, the system field sy-subrc is set to 0.
* The loop must be closed using ENDSELECT.
* To terminate the loop completely, you can use the statement [EXIT](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapexit_loop.htm).
* Note: As covered further down, when using the addition PACKAGE SIZE and storing the result in a table, a loop is opened, too.

**sy-subrc**: Trường hệ thống trong ABAP, được sử dụng để kiểm tra trạng thái của một thao tác. Nếu sy-subrc = 0, nghĩa là thao tác đã thành công.

DATA: lt\_results TYPE TABLE OF dbtab,   " Internal table để lưu kết quả

      ls\_result  TYPE dbtab.            " Cấu trúc để lưu từng hàng kết quả

" SELECT loop với PACKAGE SIZE

SELECT \*

  FROM dbtab

  WHERE condition = value

  INTO TABLE @lt\_results

  PACKAGE SIZE 100.

  LOOP AT lt\_results INTO ls\_result.

    IF sy-subrc = 0.

      " Thực hiện các thay đổi trên dữ liệu và thêm hàng vào internal table khác nếu cần

      MODIFY lt\_results FROM ls\_result.

    ENDIF.

  ENDLOOP.

ENDSELECT.

**3.3 Clause Variations and Additions in SELECT Statements**

SELECT/FROM clauses:

**Checking the existence of a row in a database table**

"Instead of @abap\_true, you could also use 'X'.

SELECT SINGLE @abap\_true

  FROM dbtab

  WHERE ...

  INTO @DATA(exists).

IF exists = abap\_true.

  ...

ENDIF.

**Removing rows that occur more than once in a multirow result set** using the DISTINCT addition.

* Cannot be used with the addition SINGLE.
* See more information here [here](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapselect_clause.htm).

SELECT DISTINCT comp1

  FROM dbtab

  WHERE ...

  INTO TABLE @itab.

**Lưu ý Không sử dụng DISTINCT trong các trường hợp sau**:

* Nếu sử dụng từ khóa SINGLE.
* Nếu một cột trong danh sách SELECT có kiểu STRING, RAWSTRING, LCHR, LRAW, hoặc GEOM\_EWKB.
* DISTINCT không thể được xử lý bởi Engine SQL ABAP. Câu lệnh ABAP SQL sẽ bỏ qua bộ nhớ đệm của bảng và một internal table được truy cập bởi FROM @itab phải được vận chuyển đến cơ sở dữ liệu. Điều này chỉ có thể thực hiện cho một internal table mỗi câu lệnh ABAP SQL.

**SELECT list variants** (some of them are already outlined above)

The following specifications can also be combined:

* SELECT \* ...: As outlined above, the \* character defines all columns to be read from a data source (in the order specified there).
* SELECT col1, col2, col3 ...: A comma-separated list of individual column names.
* SELECT data\_source~col1, data\_source~col2, data\_source~col3 ...: A comma-separated list of individual column names. Here, the name of the data source is explicitly specified and precedes the column name, separated by a tilde.
* SELECT data\_source~\* ...: In this case, the name of the data source is followed by a tilde and the \* character to specify all columns. Note that there are [special conditions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapselect_list.htm#!ABAP_VARIANT_1@1@) when using this variant.
* SELECT col1 AS al1, col2 AS al2, col3 AS al3 ...:
  + Defining alias names for individual columns of the result set with [AS](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapselect_list.htm).
  + Make sure that you use an alias name only once here. In the statement, the alias name can only be used after an ORDER BY clause.
  + As shown further down, in some cases (e. g. when using SQL expressions) the specification of an alias name is required. Setting an alias name for the data source is also possible (SELECT FROM dbtab AS alias\_name ...). See the section on joins further down.

💡 Có nhiều tùy chọn khác nhau về việc chỉ định các cột trong danh sách SELECT, trong đó bao gồm: **Chỉ định trực tiếp tên cột**: có thể ghi trực tiếp tên của cột bạn muốn chọn. **Biểu thức SQL**: cũng có thể sử dụng các biểu thức SQL để tính toán hoặc thay đổi dữ liệu của cột ngay trong câu lệnh SELECT (cộng trừ nhân chia, nối chuỗi, …).

"All fields

SELECT \* FROM dbtab

  WHERE ...

  INTO ...

"Comma-separated list

SELECT col1, col2, col3

  FROM dbtab

  WHERE ...

  INTO ...

"Comma-separated list, data source explicitly specified

SELECT dbtab~col1, dbtab~col2, col3

  FROM dbtab

  WHERE ...

  INTO ...

"Data source explicitly specified, all fields

SELECT dbtab~\*

  FROM dbtab

  WHERE ...

  INTO ...

"Alias names

"Consider the following: You want to read data from a database table into a target data

"object but, for example, a name in the target is different. Provided that there will

"not be an issue regarding the type (conversion) when the values are assigned, you might

"specify an alias name for the database column to match a component's name in the target data object.

SELECT FROM dbtab

  FIELDS comp1 AS comp\_a, comp2 AS comp\_b, comp3 AS comp\_c

  WHERE ...

  INTO CORRESPONDING FIELDS OF TABLE @itab.

"Alias name also possible for the data source

SELECT ds~col1, ds~col2, ds~col3

  FROM dbtab AS ds

  WHERE ...

  INTO ...

💡 SELECT list của mệnh đề SELECT của một truy vấn xác định các cột nào được sử dụng để tạo tập kết quả của câu lệnh SELECT và cách lấy giá trị của chúng. Có hai biến thể tĩnh và một biến thể động.

💡 INTO CORRESPONDING: Phần này chỉ định rằng danh sách các cột thực sự được truyền vào cơ sở dữ liệu được xác định bằng phần bổ sung INTO CORRESPONDING. Nếu một hoặc nhiều tên khớp, tất cả các cột không có tên khớp sẽ được loại bỏ khỏi danh sách SELECT và từ đó cũng khỏi tập kết quả. Nếu không có tên nào khớp, không có cột nào bị loại bỏ khỏi tập kết quả.

💡 Nếu chỉ định nhiều bảng cơ sở dữ liệu từ DDIC trong câu lệnh FROM, sẽ không thể tránh được việc nhiều cột có cùng tên khi sử dụng \*. Ví dụ: SELECT \* FROM table1, table2 có thể dẫn đến việc hai bảng có cùng tên cột, gây nhầm lẫn.

💡 Vì lý do hiệu suất, chỉ nên chỉ định các cột thực sự cần thiết. Chỉ sử dụng \* nếu cần tất cả các cột.

💡 Khi sử dụng INTO CORRESPONDING, \* có thể được chuyển đổi ngầm định thành danh sách các cột có tên khớp. Ví dụ: SELECT \* INTO CORRESPONDING FIELDS OF lt\_table FROM table sẽ chuyển \* thành danh sách các cột có tên khớp trong lt\_table.

💡 Khi truy cập một bảng nội bộ với kiểu dòng sơ cấp trong mệnh đề FROM của biểu thức bảng chung sau WITH, không thể sử dụng \* như danh sách SELECT. Ví dụ: WITH cte AS (SELECT \* FROM @itab) không được phép, thay vào đó bạn cần chỉ định các cột cụ thể như SELECT column1, column2 FROM @itab.

💡 Sử dụng SELECT table~\*: Giống SELECT \* nhưng dùng để chỉ định rõ ràng trong những trường hợp cần select nhiều cột từ nhiều table, from nhiều table…

TYPES BEGIN OF structure.

TYPES carrname TYPE scarr-carrname.

INCLUDE TYPE spfli AS spfli.

TYPES url TYPE scarr-url.

TYPES END OF structure.

DATA itab TYPE STANDARD TABLE OF structure WITH EMPTY KEY.

SELECT scarr~carrname, spfli~\*, scarr~url

       FROM scarr INNER JOIN spfli ON scarr~carrid = spfli~carrid

       INTO TABLE @itab.

cl\_demo\_output=>display( itab ).

💡 Nếu có nhiều bảng có cột tên giống nhau, bạn có thể sử dụng bí danh để phân biệt các cột. Không thể sử dụng tên bí danh như là một phần của biểu thức SQL.

SELECT fldate, AVG(loccuram AS DEC(31,2)) AS avg

       FROM sbook

       WHERE sbook~carrid = 'LH' AND

             sbook~connid = '0400'

       GROUP BY fldate

       ORDER BY avg DESCENDING

       INTO TABLE @FINAL(itab).

💡 Các kỹ thuật dynamic programming cũng có thể được sử dụng để chỉnh định, tuy nhiên có thể gây ra nguy cơ bảo mật nghiêm trọng. Bất kỳ nội dung động nào được truyền vào chương trình từ bên ngoài phải được kiểm tra kỹ lưỡng hoặc được mã hóa trước khi sử dụng trong các câu lệnh động.

**Reading data from a database table in another client** ([classic ABAP](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenclassic_abap_glosry.htm) only). Note that there are several variants of the USING ... addition for switching the [implicit client handling (F1 docu for standard ABAP)](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm?file=abenabap_sql_client_handling.htm) from the current client to other clients. See more information [here (F1 docu for standard ABAP)](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm?file=abapselect_client.htm).

💡 Trong hệ thống SAP, các client khác nhau có thể chứa dữ liệu khác nhau. Để truy vấn dữ liệu từ một client khác với client hiện tại, classic ABAP cho phép sử dụng các biến thể của từ khóa USING để chuyển đổi quản lý client ngầm định từ client hiện tại sang client khác. Các ví dụ này chỉ áp dụng cho classic ABAP và không có sẵn trong ABAP dành cho phát triển trên Cloud.

"Some examples; not available in ABAP for Cloud Development

"Replaces the current client with the specified client

SELECT \*

  FROM dbtab USING CLIENT '000'

  WHERE ...

  INTO TABLE @itab.

"Selects data of any number of clients

SELECT \*

  FROM dbtab USING ALL CLIENTS

  WHERE ...

  INTO TABLE @itab.

💡 Reading data from an internal table as data source using SELECT. Note that an alias name must be specified for the internal table used as data source. Find more information here.

SELECT \*

  FROM @itab1 AS tab

  WHERE ...

  INTO TABLE @DATA(itab2).

INTO **clause**:

**Limiting the number of returned table rows** using the optional addition [UP TO n ROWS](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapselect_up_to_offset.htm).

"A maximum of five rows are to be returned

"If the INTO clause is the last clause, the UP TO clause must be positioned after it.

SELECT \* FROM dbtab

  WHERE ...

  INTO TABLE @DATA(itab\_upto)

  UP TO 5 ROWS.

Returning only the table rows after a row with a specified count from the result set using the optional addition OFFSET n. You can only use the addition, if an ORDER BY clause is specified.

"In the example, data of all flights are retrieved, except for the 2 flights

"with the shortest flight time.

SELECT \*

  FROM ztest\_abap\_flsch

  WHERE carrid = 'LH'

  ORDER BY fltime ASCENDING

  INTO TABLE @DATA(itab)

  OFFSET 2.

💡 OFFSET giúp bỏ qua một số lượng hàng nhất định và chỉ trả về các hàng sau đó, chỉ có thể được sử dụng nếu có điều khoản ORDER BY. UP TO chỉ có thể được sử dụng sau điều khoản ORDER BY. OFFSET chỉ có thể được sử dụng sau điều khoản UP TO (ORDER BY 🡪 UP TO 🡪 OFFSET).

Giá trị n phải dương và không vượt quá 2,147,483,647, nếu không sẽ gây ra lỗi cú pháp hoặc ngoại lệ không bắt được.

💡 Không thể sử dụng bổ sung UP TO với các bổ sung SINGLE, UNION, INTERSECT, hoặc EXCEPT. UP TO n ROWS nên được ưu tiên sử dụng hơn so với việc dừng vòng lặp sau khi nhập n hàng.

Không có sự khác biệt lớn về hiệu suất giữa UP TO 1 ROWS và SINGLE, nhưng UP TO 1 ROWS cho phép sử dụng điều khoản ORDER BY.

💡 Sử dụng UP TO 1 ROWS để đọc nhiều nhất một hàng từ tập kết quả và sử dụng SINGLE để đọc một hàng cụ thể hoàn toàn xác định.

**Reading into individual elementary data objects**. Apart from reading into structures and internal tables outlined above, you can also read into individual elementary data objects. Here, the individual elementary data objects as target objects are specified in a comma-separated list (e. g. as existing host variables or declared inline with @DATA(...)) and put between a pair of parentheses. Note:

* The comma-separated list must have the same number of elements as columns in the result set.
* The content of the columns in the result set is assigned to the data objects specified in the list from left to right in accordance with the order specified in the SELECT list.
* Note the [assignment rules](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenselect_into_conversion.htm) also in this context.
* More information [here](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapinto_clause.htm#!ABAP_ALTERNATIVE_1@1@).

SELECT FROM dbtab

  FIELDS comp1, comp2, comp3

  WHERE ...

  INTO (@res1,@res2,@res3).

  "INTO (@DATA(res1),@DATA(res2),@DATA(res3)). "Using inline declarations

Các đối tượng dữ liệu res1, res2, ... có thể là:

* Biến tồn tại sẵn (existing host variables).
* Được khai báo ngay trong câu lệnh bằng cú pháp @DATA hoặc @FINAL.
* Được tạo ẩn danh bằng từ khóa NEW.

💡 Danh sách các đối tượng dữ liệu phải có số lượng phần tử tương ứng với số cột trong tập kết quả và các cột sẽ được gán cho các đối tượng theo thứ tự từ trái sang phải như trong câu lệnh SELECT.

💡 Nếu tập kết quả có một hàng, các cột của hàng đó sẽ được gán cho các đối tượng dữ liệu; nếu có nhiều hàng, câu lệnh SELECT mở một vòng lặp SELECT và phải được đóng bằng ENDSELECT hoặc ENDWITH.

💡 Nếu tập kết quả được định nghĩa bằng \* hoặc data\_source~\*, không thể sử dụng danh sách đối tượng dữ liệu phân tách bằng dấu phẩy.

💡 Trong một danh sách phân tách bằng dấu phẩy (như trong câu lệnh SQL), có thể sử dụng nhiều dòng và khoảng trắng thường không quan trọng. Tuy nhiên, nếu sau dấu ngoặc mở có khoảng trắng, thì phải có khoảng trắng trước dấu ngoặc đóng tương ứng.

💡 Đọc ba cột vào ba đối tượng dữ liệu cơ bản

DATA carrid TYPE scarr-carrid.

SELECT carrid, carrname, url

       FROM scarr

       ORDER BY carrid

       INTO (@carrid,

             @FINAL(carrname),

             NEW @FINAL(dref)).

  cl\_demo\_output=>write( |{ carrid }, { carrname }, { dref->\* } | ).

ENDSELECT.

cl\_demo\_output=>display( ).

💡 Đọc bốn cột vào bốn thành phần riêng lẻ của một cấu trúc

DATA:

  BEGIN OF wa,

    cityfrom TYPE spfli-cityfrom,

    cityto   TYPE spfli-cityto,

    carrid   TYPE spfli-carrid,

    connid   TYPE spfli-connid,

  END OF wa.

SELECT SINGLE

       FROM spfli

       FIELDS carrid, connid, cityfrom, cityto

       WHERE carrid = 'LH' AND connid = '400'

       INTO (@wa-carrid,

             @wa-connid,

             @wa-cityfrom,

             @wa-cityto).

cl\_demo\_output=>display( wa ).

**Appending the result set to an existing internal table**.

💡 Khi sử dụng từ khóa INTO, các dữ liệu trong target object sẽ được khởi tạo lại. Khi sử dụng từ khóa APPENDING, có thể giữ lại các dòng dữ liệu hiện có trong internal table. APPENDING cũng có thể được sử dụng cùng với từ khóa CORRESPONDING FIELDS OF TABLE để chỉ thêm các cột tương ứng.

SELECT \* FROM dbtab

  WHERE ...

  APPENDING TABLE @itab.

SELECT \* FROM dbtab

  WHERE ...

  APPENDING CORRESPONDING FIELDS OF TABLE @diff\_itab.

**Reading into packages of a specified number of rows** when reading into internal tables. The addition [PACKAGE SIZE n](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapinto_clause.htm#!ABAP_ONE_ADD@1@) can be specified after INTO TABLE and APPENDING TABLE. A SELECT loop ist opened. After PACKAGE SIZE, the number of rows is specified (which can be a host variable, host expression or a literal of type i) denoting the number of rows to be inserted in the target object per iteration.

SELECT FROM dbtab

  FIELDS comp1, comp2, comp3

  WHERE ...

  INTO TABLE @DATA(itab\_pack) PACKAGE SIZE n.

...

ENDSELECT.

**Specifying an**[**anonymous data object**](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenanonymous_data_object_glosry.htm)**as target object** using the addition [NEW](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapselect_into_target.htm#!ABAP_ALTERNATIVE_3@3@). Only to be used after INTO and not APPENDING.

"Here, the target object is an anonymous data object declared inline.

SELECT FROM dbtab

  FIELDS comp1, comp2, comp3

  WHERE ...

  INTO TABLE NEW @DATA(dref).

**3.4 More Clauses**

💡 [GROUP BY](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapgroupby_clause.htm) clause: Combining groups of table rows in the result set. You might also use [SQL expressions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abensql_expression_glosry.htm) here. Multiple clause elements are separated by a comma. Find more information on SQL expressions further down.

💡 Note that the GROUP BY clause requires all columns that are directly specified in the SELECT list or specified there as an argument of an SQL expression to be specified. An exception to this is [aggregate functions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenaggregate_function_glosry.htm) in [aggregate expressions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenaggregate_expression_glosry.htm) (except [grouping functions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abengrouping_glosry.htm)) as shown in the following example.

💡 In the example below, the database table rows that have the same content in column comp1 are combined. The lowest and highest values in column comp2 are determined for each of these groups and placed into the combined row.

SELECT FROM dbtab

  FIELDS comp1, MIN( comp2 ) AS min, MAX( comp2 ) AS max

  WHERE ...

  GROUP BY comp1

  INTO ...

💡 **Aggregate Functions in ABAP SQL**

Các hàm tổng hợp (aggregate functions) trong ABAP SQL giúp gom giá trị của các hàng trong một tập hợp hàng thành một giá trị duy nhất. Chúng được sử dụng trong các vị trí cụ thể của một truy vấn SQL.

**Các hàm tổng hợp**

1. **AVG( [DISTINCT] col [AS dtype] )**
   * Tính giá trị trung bình của các giá trị trong một cột.
   * Ví dụ:

SELECT AVG( num1 ) FROM demo\_expressions INTO @FINAL(avg).

1. **MEDIAN( [DISTINCT] sql\_exp )**
   * Tính giá trị trung vị (median) của một biểu thức.
   * Ví dụ:

SELECT MEDIAN( num1 ) AS median\_value FROM demo\_expressions INTO @FINAL(result).

1. **MAX( [DISTINCT] sql\_exp )**
   * Tìm giá trị lớn nhất trong kết quả của một biểu thức SQL.
   * Ví dụ:

SELECT MAX( num1 ) AS max FROM demo\_expressions INTO @FINAL(max).

1. **MIN( [DISTINCT] sql\_exp )**
   * Tìm giá trị nhỏ nhất trong kết quả của một biểu thức SQL.
   * Ví dụ:

SELECT MIN( num1 ) AS min FROM demo\_expressions INTO @FINAL(min).

1. **SUM( [DISTINCT] sql\_exp )**
   * Tính tổng các giá trị của một biểu thức SQL.
   * Ví dụ:

SELECT SUM( num1 ) FROM demo\_expressions INTO @FINAL(sum).

1. **PRODUCT( col )**
   * Tính tích của các giá trị trong một cột.
2. **STDDEV( [DISTINCT] sql\_exp )**
   * Tính độ lệch chuẩn của một biểu thức SQL.
   * Ví dụ:

SELECT STDDEV( CAST( salary AS D34N ) ) AS stddev FROM demo\_employees INTO @FINAL(result).

1. **VAR( [DISTINCT] sql\_exp )**
   * Tính phương sai của một biểu thức SQL.
   * Ví dụ:

SELECT VAR( CAST( salary AS D34N ) ) AS variance FROM demo\_employees INTO @FINAL(result).

1. **CORR( sql\_exp1, sql\_exp2 )**
   * Tính hệ số tương quan Pearson giữa hai cột.
   * Ví dụ:

SELECT CORR( num1, num2 ) AS corr FROM demo\_expressions INTO @FINAL(result).

1. **CORR\_SPEARMAN( sql\_exp1, sql\_exp2 )**
   * Tính hệ số tương quan Spearman giữa hai cột.
   * Ví dụ:

SELECT CORR\_SPEARMAN( num1, num2 ) AS corr\_spearman FROM demo\_expressions INTO @FINAL(result).

1. **STRING\_AGG( sql\_exp[, sep] [ORDER BY ...] )**
   * Chuỗi hóa kết quả của một biểu thức SQL với một dấu phân cách tùy chọn.
   * Ví dụ:

SELECT STRING\_AGG( text, ' - ' ) AS texts FROM t100 INTO @FINAL(result).

1. **COUNT( [DISTINCT] sql\_exp )**
   * Đếm số giá trị khác nhau của kết quả của một biểu thức SQL.
   * Ví dụ:

SELECT COUNT( DISTINCT carrid ) FROM spfli WHERE cityto = 'NEW YORK' INTO @FINAL(count).

1. **COUNT( \* )**
2. **COUNT(\*)**
   * Đếm số hàng trong một tập hợp hàng.
   * Ví dụ:

SELECT COUNT(\*) AS cntstar FROM demo\_expressions INTO @FINAL(result).

1. **GROUPING( col )**
   * Kiểm tra xem một cột có nằm trong phép nhóm hay không.
2. **ALLOW\_PRECISION\_LOSS( ... )**
   * Tăng hiệu suất của một biểu thức tổng hợp với khả năng mất độ chính xác.

**Lưu ý chung:**

* Sử dụng DISTINCT loại bỏ các giá trị trùng lặp khỏi phép tính.
* Nếu đối số của hàm tổng hợp (ngoại trừ COUNT) có giá trị null, nó sẽ bị bỏ qua khi tính toán.
* Hàm COUNT đếm các hàng và không bao giờ trả về giá trị null.
* Các đối số kiểu STRING, RAWSTRING, LCHR, LRAW và GEOM\_EWKB không thể sử dụng trong các hàm tổng hợp.

**Ví dụ chung:**

* Để tính giá trị trung bình, giá trị trung vị, giá trị lớn nhất, giá trị nhỏ nhất, tổng, sản phẩm, độ lệch chuẩn, phương sai, hệ số tương quan, chuỗi hóa, đếm số giá trị khác nhau, và đếm số hàng:
* SELECT AVG( num1 ), MEDIAN( num1 ), MAX( num1 ), MIN( num1 ), SUM( num1 ), COUNT( DISTINCT num1 ), COUNT(\*) FROM demo\_expressions INTO @FINAL(results).

💡 **Aggregate Expression in ABAP SQL**

**Cú pháp**: ... agg\_func ...

**Tác dụng** Biểu thức tổng hợp trong ABAP SQL sử dụng các hàm tổng hợp (agg\_func) để gộp các giá trị của nhiều hàng trong kết quả truy vấn thành một giá trị duy nhất. Nó có thể được sử dụng trong các vị trí sau:

* Như một biểu thức SQL cho một cột (col\_spec) trong danh sách SELECT của truy vấn.
* Như một toán hạng ở bên trái của các biểu thức quan hệ trong điều kiện HAVING của truy vấn.

**Các hàm tổng hợp**

* MAX, MIN, SUM, STRING\_AGG, COUNT: có thể sử dụng với các biểu thức SQL (sql\_exp) ngoại trừ các biểu thức tổng hợp và biểu thức cửa sổ.
* AVG, GROUPING: chỉ có thể sử dụng với cột của nguồn dữ liệu truy vấn.

**Lưu ý**

* Tất cả các biểu thức tổng hợp (trừ COUNT(\*)) không thể được xử lý bởi bộ máy SQL của ABAP, buộc truy vấn phải bỏ qua bộ đệm bảng và chuyển bảng nội bộ đến cơ sở dữ liệu.
* Nếu giá trị của biểu thức tổng hợp quá lớn cho vùng đích, một ngoại lệ sẽ được đưa ra.

**Ví dụ** Truy vấn sau trả về ngày bay, số lượng hành khách, trọng lượng hành lý trung bình và lớn nhất của tất cả các chuyến bay Lufthansa với số hiệu chuyến bay 0400:

SELECT FROM sbook

       FIELDS fldate,

              COUNT( \* ) AS count,

              AVG( luggweight AS DEC( 16,4 ) ) AS avg,

              MAX( luggweight ) AS max

       WHERE carrid = 'LH' AND

             connid = '0400'

       GROUP BY fldate

       INTO TABLE @FINAL(result).

cl\_demo\_output=>display( result ).

💡 **Grouping Function in ABAP SQL**

**Cú pháp**: ... GROUPING( col )

**Tác dụng** Hàm GROUPING trong ABAP SQL xác định liệu một cột cụ thể có phải là một phần của nhóm (aggregation) hay không. Hàm này trả về giá trị 0 hoặc 1, trong đó:

* 0: Cột không phải là một phần của nhóm.
* 1: Cột là một phần của nhóm.

Hàm GROUPING có kiểu dữ liệu trả về là INT1.

**Quy tắc sử dụng**

* Hàm GROUPING chỉ có thể được sử dụng trong danh sách SELECT và điều kiện HAVING.
* Chỉ sử dụng hàm GROUPING khi câu lệnh GROUP BY có thêm GROUPING SETS.
* Cột được sử dụng trong hàm GROUPING phải có trong danh sách GROUP BY.

**Ví dụ** Truy vấn sau tính số lượng chuyến bay của Lufthansa tùy thuộc vào loại máy bay (planetype) và số hiệu chuyến bay (connid). Hàm GROUPING được sử dụng để xác định xem các cột này có phải là một phần của nhóm hay không.

SELECT FROM sflight

       FIELDS carrid,

              connid,

              planetype,

              COUNT( \* ) AS flights\_count,

              grouping( connid ) AS agg\_connid,

              grouping( planetype ) AS agg\_planetype

       WHERE carrid = 'LH'

       GROUP BY GROUPING SETS ( ( carrid, planetype ),

                                ( carrid, connid ),

                                (  ) )

       ORDER BY connid, planetype

       INTO TABLE @FINAL(result\_grouping).

cl\_demo\_output=>display( result\_grouping ).

Trong ví dụ này:

* agg\_connid và agg\_planetype cho biết liệu connid và planetype có phải là một phần của nhóm hay không.
* Nếu agg\_connid hoặc agg\_planetype bằng 1, cột tương ứng là một phần của nhóm.
* Nếu agg\_connid hoặc agg\_planetype bằng 0, cột tương ứng không phải là một phần của nhóm.

💡 [HAVING](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abaphaving_clause.htm) clause: Limiting the number of table rows in groups in the result by setting conditions on these rows. The rows for which a logical expression is true are inserted in the target variable. Note that HAVING can only be used together with GROUP BY.

SELECT FROM dbtab

  FIELDS comp1, MIN( comp2 ) AS min, MAX( comp3 ) AS max

  WHERE ...

  GROUP BY comp1

  HAVING comp1 LIKE '%XYZ%' AND SUM( comp4 ) > 100

  INTO ...

💡 [ORDER BY](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abaporderby_clause.htm) clause: Sorting the result set by specified columns.

The following example shows the ordering of the result set based on the content of the primary key of the [data source](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abendata_source_glosry.htm). You can also order by any columns and by explicitly specifying the sort order. There are more ordering options, for example, by using SQL expressions.

SELECT FROM dbtab

  FIELDS comp1, comp2, comp3

  WHERE ...

  ORDER BY PRIMARY KEY

           "comp2 ASCENDING

           "comp2 DESCENDING

  INTO ...

**SQL data source**: [DDIC database table](javascript:call_link('abenddic_db_table_glosry.htm')), [CDS entity](javascript:call_link('abencds_entity_glosry.htm')) or other resource such as a [common table expression](javascript:call_link('abencommon_table_expression_glosry.htm')), a [SQL path expression](javascript:call_link('abensql_path_expression_glosry.htm')), a [SQL hierarchy](javascript:call_link('abensql_hierarchy_glosry.htm')), a [table function](javascript:call_link('abentable_function_glosry.htm')), or an [internal table](javascript:call_link('abeninternal_table_glosry.htm')) from which a [query](javascript:call_link('abenasql_query_glosry.htm')) in [ABAP SQL](javascript:call_link('abenabap_sql_glosry.htm')) or [ABAP CDS](javascript:call_link('abenabap_cds_glosry.htm')) takes its data.

**streaming data source**: Repository from which a reader [data stream](javascript:call_link('abendata_stream_glosry.htm')) takes its data in [streaming](javascript:call_link('abenstreaming_glosry.htm')). A streaming data source can be an external resource such as a [LOB](javascript:call_link('abenlob_glosry.htm')) column of a [DDIC database table](javascript:call_link('abenddic_db_table_glosry.htm')) or an internal resource such as a [string](javascript:call_link('abenstring_glosry.htm')) or an [internal table](javascript:call_link('abeninternal_table_glosry.htm')). [Streaming](javascript:call_link('abenstreaming_glosry.htm')) is currently implemented for [DDIC database tables](javascript:call_link('abenddic_db_table_glosry.htm')), [strings](javascript:call_link('abenstring_glosry.htm')), and [internal tables](javascript:call_link('abeninternal_table_glosry.htm')).

💡 **Note**

* Not specifying ORDER BY means that the order of entries in the result set is undefined.
* If ORDER BY and GROUP BY clauses are used, all columns specified after ORDER BY must also be specified after GROUP BY.
* If aggregate functions are specified after SELECT, all columns that are specified after ORDER BY and that do not have an alias name for an aggregate function must also be specified after SELECT and after the GROUP BY clause which is required in this case, too.

" Tính tổng số tiền bán (sales\_amount) cho mỗi product\_id và sắp xếp kết quả theo sales\_date.

SELECT product\_id, SUM(sales\_amount) AS total\_sales

  FROM sales

  GROUP BY product\_id, sales\_date

  ORDER BY sales\_date

  INTO TABLE @DATA(result).

" Tính tổng số tiền bán cho mỗi product\_id và sắp xếp kết quả theo tổng số tiền bán (total\_sales).

SELECT product\_id, SUM(sales\_amount) AS total\_sales

  FROM sales

  GROUP BY product\_id

  ORDER BY total\_sales

  INTO TABLE @DATA(result).

💡 [WHERE](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapwhere.htm) clause: Restricts the number of rows that are included in the result set using logical expressions. See further information on them in the following sections.

SELECT FROM dbtab

  FIELDS comp1, comp2, comp3

  WHERE comp1 = 'abc'

    AND comp2 < 123

  INTO ...

**3.** [**Retrieving Data Using SELECT**](https://github.com/SAP-samples/abap-cheat-sheets/blob/main/03_ABAP_SQL.md#retrieving-data-using-select)**: Commit 716cb73**

Đoạn trên là của commit cũ, các commit mới đã thay đổi nội dung.

|  |  |
| --- | --- |
| **Subject** | **Details/Code Snippet** |
| Retrieving a single row into a structure | "SINGLE addition  "Here, all fields of a single row a read. Specifying an  "asterisk \* indicates that all fields are to be read.  "Alternatively, you can list all the fields separated by comma.  "Note that if the selection covers more than one row, e. g. in case  "of a non-unique WHERE clause, one of these rows is included in  "the result.  SELECT SINGLE FROM dbtab    FIELDS \*    WHERE ...    INTO @struc.        "Existing structure of dbtab's row type  "Retrieving a selected set of fields of a single row  SELECT SINGLE FROM dbtab    FIELDS comp1, comp2, comp3    WHERE ...    INTO @DATA(struc2).    "Structure declared inline  "Alternative syntax without the FIELDS addition  "Here, the CORRESPONDING FIELDS OF addition is used. Only the content of  "columns that have identically named components in the target data object  "is assigned.  SELECT SINGLE comp1, comp2, comp3       "Selected set of fields    FROM dbtab    WHERE ...    INTO CORRESPONDING FIELDS OF @struc.  "Existing structure |
| Retrieving multiple rows into an internal table | SELECT FROM dbtab    FIELDS \*              "All fields    WHERE ...    INTO TABLE @itab.     "itab has an appropriate row type  "Alternative syntax without the FIELDS addition  SELECT comp1, comp2, comp3          "Selected set of fields    FROM dbtab    WHERE ...    INTO TABLE @DATA(lv\_itab).        "Internal table declared inline  "Selected set of fields, existing variable  "See the note on CORRESPONDING FIELDS OF above  SELECT FROM dbtab    FIELDS comp1, comp2, comp3                "Selected set of fields    WHERE ...    INTO CORRESPONDING FIELDS OF TABLE @itab. |
| SELECT loop: Sequentially retrieving multiple rows | If the row is found, the system field sy-subrc is set to 0.  SELECT FROM dbtab    FIELDS \*    WHERE ...    INTO @struc.      IF sy-subrc = 0.        ...  "For example, making changes on data and adding the row to an internal table.      ENDIF.  ENDSELECT. |
| Checking the existence of a row in a database table | "Instead of @abap\_true, you could also use 'X'.  SELECT SINGLE @abap\_true    FROM dbtab    WHERE ...    INTO @DATA(exists).  IF exists = abap\_true.    ...  ENDIF. |
| DISTINCT addition: Removing rows that occur more than once in a multirow result set | Cannot be used with the addition SINGLE.  SELECT DISTINCT comp1    FROM dbtab    WHERE ...    INTO TABLE @itab. |

**4. SQL Conditions**

You can formulate conditions in ABAP SQL statements, i. e. [logical expressions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenlogical_expression_glosry.htm), especially in the WHERE clause to restrict the result. Note that without a WHERE clause, all rows are respected for the operation.

See below a selection of the operators that are possible when specifying conditions. For more information, see the subtopics of the [SQL Conditions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenasql_cond.htm) topic.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| =, EQ | The content of two operands is equal. |
| <>, NE | The content of two operands is not equal. |
| <, LT | The content of one operand is less than the content of the other operand. |
| >, GT | The content of one operand is greater than the content of the other operand. |
| <=, LE | The content of one operand is less than or equal to the content of the other operand. |
| >=, GE | The content of one operand is greater than or equal to the content of the other operand. |
| ... [NOT] BETWEEN ... AND ... | The value of an operand is (not) between the value of the two other operands. |
| ... [NOT] LIKE ... | The content of an operand matches (does not match) a specified pattern. The pattern can be specified by using wildcard characters. % stands for any character string, including an empty string. \_ stands for any character. |
| ... IS [NOT] INITIAL ... | The value of an operand is (not) the initial value of its built-in dictionary type. |
| ... EXISTS ... | Checks the result set of a [subquery](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abensubquery_glosry.htm). The expression is true if the result set contains at least one row. See more information [here](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenwhere_logexp_subquery.htm). |
| ... [NOT] IN ... | Checks whether the operands on the left side match a value from a set of values specified in parentheses. On the left side, a single operand or an operand list are possible. On the right side, a comma-separated lists or subqueries can be specified. It is also possible to specify a [ranges table](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenranges_table_glosry.htm) to evaluate [ranges conditions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenranges_condition_glosry.htm). |
| ... IS [NOT] NULL ... | Checks whether the value of an operand is (not) the [null value](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abennull_value_glosry.htm). Find more information in the code snippet and in the [ABAP Keyword Documentation](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenwhere_logexp_null.htm). |

**💡 Note**

* You can combine multiple logical expressions into one logical expression using AND or OR. To further detail out the desired condition, expressions within parentheses are possible. Find more information [here](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenwhere_logexp_andornot.htm).
* In particular, the expressions specified dynamically as (cond\_syntax) are also possible as logical expressions within a combination or negation.
* The operator NOT in a WHERE clause cannot be supported by an index. For this reason, it is best to use the reverse comparison operator instead of NOT, for example col <= dobj instead of NOT col > dobj.
* Two NOT operators can only be specified consecutively if they are separated by an opening parenthesis. If not, an even number of consecutive NOT operators has the same meaning as none, and an odd number of consecutive NOT operators has the same meaning as a single NOT and should be used accordingly.
* If multiple expressions are combined using AND, rules such as the following cannot be relied on: If one expression is false, then the following expressions are not evaluated any more. For example, a condition col1 <> 0 AND col2 / col1 > 1 can lead to an exception. To avoid such exceptions, CASE expressions can be used.

Examples:

"---- SQL conditions demonstrated with the WHERE clause ----

"Note:

"- For most of the self-contained examples, an internal table is used as the

"  data source of SELECT statements to work with simple data.

"- For some examples that are covered, such as subqueries, demo database tables

"  from the cheat sheet repository are used in addition.

"- Dynamic specifications are also possible. They are not covered here. See

"  the Dynamic Programming cheat sheet.

"---- Types and internal table to work with in the examples ----

"Note: You cannot use type string columns in WHERE conditions.

TYPES: BEGIN OF demo\_struc,

          id   TYPE i,

          name TYPE c LENGTH 15,

          "name TYPE string,

        END OF demo\_struc.

DATA itab TYPE SORTED TABLE OF demo\_struc WITH UNIQUE KEY id.

"Populating internal table with data to work with in the examples

itab = VALUE #( ( id = 1 name = 'bear' )

                ( id = 2 name = 'camel' )

                ( id = 3 name = 'rabbit' )

                ( id = 4 name = 'zebra' )

                ( id = 5 name = 'dog' )

                ( id = 6 name = 'deer' )

                ( id = 7 name = 'squirrel' )

                ( id = 8 name = 'cheetah' )

                ( id = 9 name = 'elephant' )

                ( id = 10 name = 'donkey' )

                ( id = 11 name = 'fish' )

                ( id = 12 name = 'sheep' ) ).

"---- =, <>, >, >= (as a selection of possible comparison operators) ----

SELECT id FROM @itab AS tab WHERE name = 'bear' INTO TABLE @DATA(it). "1

SELECT id FROM @itab AS tab WHERE name <> 'bear' INTO TABLE @it. "2-12

SELECT id FROM @itab AS tab WHERE id > 10 INTO TABLE @it. "11,12

SELECT id FROM @itab AS tab WHERE id >= 10 INTO TABLE @it. "10,11,12

"---- Combining logical expressions using AND, OR and parentheses  ----

SELECT id FROM @itab AS tab WHERE id = 1 AND name = 'bear' INTO TABLE @it. "1

SELECT id FROM @itab AS tab WHERE name = 'bear' OR name = 'sheep' INTO TABLE @it. "1,12

"In the following example, the resulting table is initial. One of the expressions

"in parentheses is false (AND is used between the expressions in parentheses).

"In contrast, the example below returns an entry because of using OR.

SELECT id FROM @itab AS tab

  WHERE ( id = 1 AND name = 'bear' )

  AND ( id = 20 AND name = 'camel' )

  INTO TABLE @it.

SELECT id FROM @itab AS tab

  WHERE ( id = 1 AND name = 'bear' )

  OR ( id = 20 AND name = 'camel' )

  INTO TABLE @it. "1

"------------------------ [NOT] BETWEEN ------------------------

SELECT id FROM @itab AS tab WHERE id BETWEEN 1 AND 4 INTO TABLE @it. "1,2,3,4

"The condition with BETWEEN above corresponds to the following condition.

"The example makes use of a condition specified in parentheses to combine multiple

"expressions.

SELECT id FROM @itab AS tab WHERE ( id >= 1 AND id <= 4 ) INTO TABLE @it. "1,2,3,4

"Negation with NOT

SELECT id FROM @itab AS tab WHERE id NOT BETWEEN 1 AND 4 INTO TABLE @it. "5-12

"------------------------ IS [NOT] INITIAL ------------------------

SELECT id FROM @itab AS tab WHERE id IS NOT INITIAL INTO TABLE @it. "1-12

"------------------------ [NOT] LIKE ------------------------

"For (not) matching a specified pattern

"Note: % (any character string), \_ (any character).

SELECT name FROM @itab AS tab

  WHERE name LIKE '%ee%'

  OR name LIKE '\_o%'

  INTO TABLE @DATA(names). "dog,deer,cheetah,donkey,sheep

"ESCAPE addition for defining a single-character escape character

"In the following example, this character is #. It is placed before

"the % character in the specification after LIKE. In this case, %

"is escaped and does then not stand for any character string in the

"evaluation.

"Adding a table entry for this syntax example.

itab = VALUE #( BASE itab ( id = 13 name = '100%' ) ).

"Any character sequence followed by the % character

SELECT name FROM @itab AS tab

  WHERE name LIKE '%#%' ESCAPE '#'

  INTO TABLE @names. "100%

"Deleting the entry because it is not relevant for the further examples.

DELETE itab INDEX 13.

"------------------------ [NOT] IN (using a value set) ------------------------

"For (not) matching a value in a set of values specified in parentheses.

"Single operands on the left side of IN

SELECT id FROM @itab AS tab

  WHERE name IN ( 'camel', 'rabbit', 'dog', 'snake' )

  INTO TABLE @it. "2,3,5

"Negation NOT IN; note to use host variables/expressions for local/global data objects

DATA(animal) = 'sheep'.

SELECT id FROM @itab AS tab

  WHERE name NOT IN ( 'fish', @animal )

  INTO TABLE @it. "1-10

"Operand list (a parenthesized comma-separated list) on the left side of IN

"For (not) matching value tuples from a set of value tuples specified in parentheses on the right side.

"In the following example, two values are specified in the operand list on the left. Consequently,

"two values with appropriate types must be specified in parentheses on the right.

SELECT id FROM @itab AS tab

  WHERE ( id, name ) IN ( ( 1, 'bear' ), ( 3, 'rabbit' ), ( 8, 'zebra' ), ( 20, 'dog' ) )

  INTO TABLE @it. "1,3

"------------------------ [NOT] IN (using a subquery) ------------------------

"[NOT] IN for matching a value contained in the result set of a subquery

"In the following example, the subquery reads data from a demo database table.

"For a representative result, the table is cleared, and then filled with 'suitable'

"data sets.

DELETE FROM zdemo\_abap\_tab1.

MODIFY zdemo\_abap\_tab1 FROM TABLE @( VALUE #( ( key\_field = 11 num1 = 11 )

                                              ( key\_field = 12 num1 = 12 )

                                              ( key\_field = 13 num1 = 13 )

                                              ( key\_field = 14 num1 = 14 ) ) ).

SELECT id FROM @itab AS tab

  WHERE id IN ( SELECT key\_field FROM zdemo\_abap\_tab1 ) INTO TABLE @it. "11,12

"------------------------ [NOT] IN (using a ranges table) ------------------------

"[NOT] IN for checking whether the operands on the left side match a ranges condition in a ranges table

"Declaring a ranges table

DATA rangestab TYPE RANGE OF i.

"Populating a ranges table using the VALUE operator

rangestab = VALUE #( ( sign   = 'I' option = 'BT' low = 1 high = 3 )

                     ( sign   = 'I' option = 'GE' low = 10  ) ).

SELECT id FROM @itab AS tab WHERE id IN @rangestab INTO TABLE @it. "1,2,3,10,11,12

"You cannot use logical operators such as CP (conforms to pattern) in the WHERE clause.

"In a ranges table, they are possible.

"Note:

"- Regarding CP: \* (any character sequence), + (any character), # (escape character)

"- An equivalent example above uses the LIKE addition.

DATA rt TYPE RANGE OF demo\_struc-name.

rt = VALUE #( ( sign   = 'I' option = 'CP' low = '\*ee\*' ) "ee in a string

              ( sign   = 'I' option = 'CP' low = '+o\*' ) ). "o in second position

SELECT name FROM @itab AS tab

  WHERE name IN @rt

  INTO TABLE @names. "dog,deer,cheetah,donkey,sheep

"------------------------ EXISTS ------------------------

"For checking the result set of a subquery.

"The following example reads all entries from the internal table if entries having

"the same key also exist in the database table.

"Note: The SELECT list in the subquery only contains a literal to determine that

"the entry exists. Specifying explicit column names is not relevant.

SELECT id FROM @itab AS tab WHERE

  EXISTS ( SELECT @abap\_true FROM zdemo\_abap\_tab1 WHERE key\_field = tab~id )

  INTO TABLE @it. "11,12

"------------------------ IS [NOT] NULL ------------------------

"The null value is a special value that is returned by a database. It indicates an

"undefined value or result. Note that, in ABAP, there are no special null values. Do

"not confuse the null value with a type-dependent initial value. When using SELECT

"statements to read data, null values can be produced by, for example, outer joins.

"When the null values are passed to a data object, they are transformed to the

"type-dependent initial values. For more information, refer to the ABAP Keyword Documentation.

"The following example uses a left outer join to intentionally create null values. For

"this purpose, two demo database tables of the cheat sheet repository are cleared and

"populated with specific values to visualize null values.

DELETE FROM zdemo\_abap\_tab1.

DELETE FROM zdemo\_abap\_tab2.

MODIFY zdemo\_abap\_tab1 FROM TABLE @( VALUE #( ( key\_field = 1 char1 = 'a' char2 = 'y' )

                                              ( key\_field = 2 char1 = 'b' char2 = 'z' ) ) ).

MODIFY zdemo\_abap\_tab2 FROM TABLE @( VALUE #( ( key\_field = 1 char1 = 'a' )

                                              ( key\_field = 2 char1 = 'a' )

                                              ( key\_field = 3 char1 = 'b' )

                                              ( key\_field = 4 ) ) ).

"Note that for the entry 'key\_field = 4' no char1 value was passed.

"char1 is a shared column of the two database tables, and which is used in

"the ON condition of the join. Since there is no entry in char1 for 'key\_field = 4',

"the joined values are null in that case. The WHERE clause uses the addition IS NULL.

"Therefore, the result only contains this entry. char2 is assigned the type-initial

"value in the result.

SELECT tab2~key\_field, tab1~char2

    FROM zdemo\_abap\_tab2 AS tab2

    LEFT OUTER JOIN zdemo\_abap\_tab1 AS tab1 ON tab1~char1 = tab2~char1

    WHERE tab1~char1 IS NULL

    INTO TABLE @DATA(joined\_tab).

\*KEY\_FIELD    CHAR2

\*4

"The following example visualizes the null values. The INDICATORS addition of the

"INTO clause is used to specify indicators such as the null indicator. In the

"example, an appropriate target table is defined to also store information about

"which columns of the result set contain the null value and which do not.

"For more information on the syntax, refer to the ABAP Keyword Documentation.

TYPES: BEGIN OF st4null,

          BEGIN OF s2,

            key\_field TYPE zdemo\_abap\_tab2-key\_field,

            char2     TYPE zdemo\_abap\_tab1-char2,

          END OF s2,

          BEGIN OF nulls,

            key\_field TYPE c LENGTH 1,

            char2     TYPE c LENGTH 1,

          END OF nulls,

        END OF st4null.

DATA joined\_tab\_w\_null\_ind TYPE TABLE OF st4null WITH EMPTY KEY.

SELECT tab2~key\_field, tab1~char2

  FROM zdemo\_abap\_tab2 AS tab2

  LEFT OUTER JOIN zdemo\_abap\_tab1 AS tab1 ON tab1~char1 = tab2~char1

  INTO TABLE @joined\_tab\_w\_null\_ind INDICATORS NULL STRUCTURE nulls.

\*S2                      NULLS

\*KEY\_FIELD    CHAR2      KEY\_FIELD    CHAR2

\*1            y

\*KEY\_FIELD    CHAR2      KEY\_FIELD    CHAR2

\*2            y

\*KEY\_FIELD    CHAR2      KEY\_FIELD    CHAR2

\*3            z

\*KEY\_FIELD    CHAR2      KEY\_FIELD    CHAR2

\*4                                    X

"Negation IS NOT NULL

SELECT tab2~key\_field, tab1~char2

  FROM zdemo\_abap\_tab2 AS tab2

  LEFT OUTER JOIN zdemo\_abap\_tab1 AS tab1 ON tab1~char1 = tab2~char1

  WHERE tab1~char1 IS NOT NULL

  INTO TABLE @joined\_tab.

\*KEY\_FIELD    CHAR2

\*1            y

\*2            y

\*3            z

**Lưu Ý Hiệu Suất trong ABAP SQL**

1. **Giữ số lượng hits thấp**:
   * Chỉ chọn những hàng thực sự cần thiết bằng cách sử dụng các điều kiện cụ thể.
   * Tránh chuyển những hàng không cần thiết từ cơ sở dữ liệu đến AS ABAP.
2. **Giữ khối lượng dữ liệu thấp**:
   * Chỉ chọn các cột cần thiết.
   * Sử dụng các biểu thức tổng hợp và nhóm dữ liệu để giảm lượng dữ liệu chuyển giao.
3. **Giữ số lượng đọc thấp**:
   * Sử dụng các thao tác hàng loạt thay vì các thao tác đơn lẻ.
   * Tránh sử dụng các câu lệnh SQL trong vòng lặp, thay vào đó dùng phép join, view hoặc subquery.
4. **Sử dụng bộ đệm cục bộ**:
   * Không đọc lại dữ liệu giống nhau nhiều lần.
   * Sử dụng table buffering để tiết kiệm thời gian đọc cơ sở dữ liệu.
5. **Tìm kiếm hiệu quả bằng indexes**:
   * Sử dụng các chỉ mục thứ cấp (secondary indexes) để cải thiện hiệu suất lựa chọn.

**5 .SQL Operands**

|  |  |  |
| --- | --- | --- |
| **Operand Position** | **Operands** | **Arguments of SQL Expressions** |
| Column specification [**col\_spec**](javascript:call_link('abapselect_clause_col_spec.htm')) in a [**SELECT** list](javascript:call_link('abapselect_list.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) |
| Left side of a relational expression [**rel\_exp**](javascript:call_link('abenabap_sql_stmt_logexp.htm')) in [**WHERE** clauses](javascript:call_link('abapwhere.htm')) and [**ON** conditions](javascript:call_link('abapselect_join.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')) and [window expressions](javascript:call_link('abapselect_over.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')) and [window expressions](javascript:call_link('abapselect_over.htm')) |
| Right side of a relational expression [**rel\_exp**](javascript:call_link('abenabap_sql_stmt_logexp.htm')) in [**WHERE** clauses](javascript:call_link('abapwhere.htm')) and [**ON** conditions](javascript:call_link('abapselect_join.htm')) | Elementary [SQL operands](javascript:call_link('abensql_operands.htm')), no [SQL expressions](javascript:call_link('abapsql_expr.htm')) | - |
| Left side of a relational expression [**rel\_exp**](javascript:call_link('abenabap_sql_stmt_logexp.htm')) in [**HAVING** clauses](javascript:call_link('abaphaving_clause.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [window expressions](javascript:call_link('abapselect_over.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [window expressions](javascript:call_link('abapselect_over.htm')) |
| Right side of a relational expression [**rel\_exp**](javascript:call_link('abenabap_sql_stmt_logexp.htm')) in [**HAVING** clauses](javascript:call_link('abaphaving_clause.htm')) | Elementary [SQL operands](javascript:call_link('abensql_operands.htm')) and [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')), no other [SQL expressions](javascript:call_link('abapsql_expr.htm')) | - |
| Left side of a relational expression [**rel\_exp**](javascript:call_link('abenabap_sql_expr_logexp.htm')) for expressions | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')) and [window expressions](javascript:call_link('abapselect_over.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')) and [window expressions](javascript:call_link('abapselect_over.htm')) |
| Right side of a relational expression [**rel\_exp**](javascript:call_link('abenabap_sql_expr_logexp.htm')) for expressions | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')) and [window expressions](javascript:call_link('abapselect_over.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')) and [window expressions](javascript:call_link('abapselect_over.htm')) |
| Grouping criterion after [**GROUP BY**](javascript:call_link('abapgroupby_clause.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')) and [window expressions](javascript:call_link('abapselect_over.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')) and [window expressions](javascript:call_link('abapselect_over.htm')) |
| Window criterion after [**PARTITION BY**](javascript:call_link('abapselect_over.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')) and [window expressions](javascript:call_link('abapselect_over.htm')) | [SQL expressions](javascript:call_link('abapsql_expr.htm')) except [aggregate expressions](javascript:call_link('abapselect_aggregate.htm')) and [window expressions](javascript:call_link('abapselect_over.htm')) |
| Sort criterion after [**ORDER BY**](javascript:call_link('abaporderby_clause.htm')) | [Columns](javascript:call_link('abenabap_sql_columns.htm')) and alias names for [SQL expressions](javascript:call_link('abapsql_expr.htm')) defined using [**AS**](javascript:call_link('abapselect_list.htm')) | - |

Elementary operand in an [ABAP SQL statement](javascript:call_link('abenabap_sql_glosry.htm')). SQL operands are columns of [DDIC database tables](javascript:call_link('abenddic_db_table_glosry.htm')) or [views](javascript:call_link('abenview_glosry.htm')) defined in the [ABAP Dictionary](javascript:call_link('abenabap_dictionary_glosry.htm')), [host variables](javascript:call_link('abenhost_variable_glosry.htm')), or [host expressions](javascript:call_link('abenhost_expression_glosry.htm')).

Can be database table or view columns, a [literal](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenliteral_glosry.htm), [host variables](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenhost_variable_glosry.htm) (i. e. global or local data objects escaped using @: @dobj) or [host expressions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abenhost_expression_glosry.htm) (@( ... ))

**Literals** là các đối tượng dữ liệu không có tên, không cần ký tự escape @, được tạo ra ngay trong mã nguồn của chương trình. Chúng được xác định hoàn toàn bởi giá trị của chúng. Không thể thay đổi giá trị của một literal sau khi nó đã được tạo ra. Literals có hai loại: số (numeric literals) và ký tự (character literals).

* **Numeric literals** là các dãy số có thể có dấu ở phía trước. Trong numeric literals, không có dấu phân cách thập phân và cũng không có ký hiệu mantissa và exponent. Ví dụ: 183, -97, +326
* **Character literals** là các chuỗi ký tự chữ và số trong mã nguồn của một chương trình ABAP, được bao bọc trong dấu ngoặc đơn. Khi bao bọc trong dấu ngoặc đơn, chúng có kiểu dữ liệu ABAP là C và được gọi là text field literals. Khi bao bọc trong dấu “back quotes”, chúng có kiểu dữ liệu ABAP là STRING và được gọi là string literals. Độ dài của trường được xác định bởi số lượng ký tự. Trong text field literals, các khoảng trắng ở cuối bị bỏ qua, nhưng trong string literals, chúng được tính vào.
* Literals có gán kiểu dữ liệu được ưu tiên vì các lý do sau:
  + Sử dụng literals không gán kiểu dữ liệu tốn kém thêm về hiệu suất vì chúng phải được chuyển đổi bởi compiler.
  + Sử dụng literals không gán kiểu có thể dẫn đến lỗi trong runtime, trong khi literals có gán kiểu đảm bảo tương thích kiểu dữ liệu ngay lập tức.

" Text field literals:

REPORT YR\_SEP\_12.

Write 'Tutorials Point'.

Write / 'ABAP Tutorial'.

" String field literals:

REPORT YR\_SEP\_12.

Write `Tutorials Point `.

Write / `ABAP Tutorial `.

**Constants** là các đối tượng dữ liệu có tên, được tạo ra một cách tĩnh bằng cách sử dụng các câu lệnh khai báo. Một constant được khai báo bằng cách gán một giá trị cho nó, giá trị này được lưu trữ trong vùng nhớ của chương trình. Giá trị của một constant không thể thay đổi trong suốt quá trình thực thi chương trình. Các giá trị cố định này cũng có thể được coi là literals.

Complex constants bao gồm nhiều trường con.

Ví dụ:

BEGIN OF EMPLOYEE,

  Name(25) TYPE C VALUE 'Management Team',

  Organization(40) TYPE C VALUE 'Tutorials Point Ltd',

  Place(10) TYPE C VALUE 'India',

END OF EMPLOYEE.

Constant reference được sử dụng trong các so sánh hoặc có thể được truyền vào các thủ tục.

Ví dụ:

CONSTANTS null\_pointer TYPE REF TO object VALUE IS INITIAL.

Example demonstrating possible operands:

DATA number TYPE i VALUE 3.

SELECT FROM zdemo\_abap\_flsch

  FIELDS

  "Specifies a column of a data source directly using its name

  cityfrom,

  "Column selector ~ can be used to prefix every specified column.

  "Here, it is optional. It is non-optional, e. g., if multiple data

  "sources in an ABAP SQL statement are edited and the column name

  "is not unique.

  zdemo\_abap\_flsch~cityto,

  'Lufthansa' AS name, "Untyped literal

  char`X` AS flag, "Typed literal

  @number AS num, "Host variable

  @( cl\_abap\_context\_info=>get\_system\_date( ) ) as date "Host expression

  WHERE carrid = 'LH'          "Untyped literal

    AND countryfr = char`DE`   "Typed literal

  "Data object created inline and escaped with @

  INTO TABLE @DATA(it)

  "The following clause shows all options having the same effect

  UP TO 3 ROWS.             "Untyped numeric literal

  "UP TO int4`3` ROWS.      "Typed numeric literal

  "UP TO @number ROWS.        "Host variable

  "UP TO @( 10 - 7 ) ROWS.  "Host expression

**6. SQL Expressions**

* SQL expressions can be specified in various positions of ABAP SQL statements.
* They that passed to the database system for evaluation.
* For example, SQL expressions can be specified as columns in the SELECT list as demonstrated in most of the following examples.
* Find information on more possible positions and general information on SQL expressions [here](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abapsql_expr.htm) and the subtopics there.

**💡 Note**  
You can [enclose SQL expressions in parentheses](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abensql_exp_parentheses.htm).

SELECT SINGLE

   carrid,

   "SQL expressions enclosed in parenthesis in the context

   "of arithmetic expressions impacting the prioriry of

   "calculations

   ( 1 + 2 ) \* ( 3 + 4 ) \* ( 5 + 6 ) AS calc\_w\_par, "231

   1 + 2 \* 3 + 4 \* 5 + 6 AS calc\_no\_par "33

FROM zdemo\_abap\_fli

WHERE carrid = 'AA'

INTO @DATA(using\_parentheses).

**6.1 Elementary SQL Expressions**

Bao gồm:

1. **Tên cột** từ bảng dữ liệu (ví dụ: column\_name).
2. **Literal** (hằng số không đổi).
3. **Host variable** (biến host, là biến từ chương trình ABAP, ví dụ: @variable).
4. **Host expression** (biểu thức host, ví dụ: @(expression)).

DATA: lv\_fixed\_value TYPE i VALUE 100,

      lv\_dynamic\_value TYPE string VALUE 'SomeValue',

      lt\_table TYPE TABLE OF spfli.

SELECT carrid,                      " Tên cột từ bảng dữ liệu

       'LiteralValue' AS literal,   " Literal (hằng số không đổi)

       @lv\_fixed\_value AS fixed,    " Host variable (biến host)

       @( lv\_dynamic\_value && 'Suffix' ) AS host\_expression " Host expression (biểu thức host)

  FROM spfli

  INTO TABLE lt\_table.

**6.2 Arithmetic Expressions**

[Arithmetic expressions](https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abensql_arith.htm) perform arithmetic calculations using the operators +, -, \*, /.

SELECT SINGLE

  carrid,

  "Arithmethic expressions

  "operators + - \*

  "Note that / is not allowed in integer expressions as the one below

  ( 1 + 2 ) \* 3 AS calc,

  "/ used in an expression using type adjustment in ABAP SQL.

  "A cast expression converts the value of the operands to the

  "specified dictionary type. The result is a representation of the

  "source value in the specified type.

  CAST( 1 AS D34N ) / CAST( 2 AS D34N ) AS ratio

FROM zdemo\_abap\_carr

WHERE carrid = 'AA'

INTO @DATA(arithmetic\_sql\_expr).

**6.3 Cast Expressions**

**6.4 String Expressions**

**6.5 Case Expressions**

**6.6 Aggregate Expressions**

**6.7 Window Expressions**

**6.8 Null Expressions**

**7. Built-In SQL Functions**

**7.1 Numeric Functions**

**7.2 String Functions**

**7.3 coalesce Function**

**7.4 More Functions**

**8. Changing Data in Database Tables**

**8.1 Using INSERT**

**8.2 Using UPDATE**

**8.3 Using MODIFY**

**8.4 Using DELETE**

**8.5 RAP-Specific ABAP SQL Variants**

**9. More Information**

**10. Executable Example**