Speed up your ABAP development using shortcuts

Edit

Ctrl+Shift+A	Open development object
Ctrl+F2	Check development object
Ctrl+F3	Activate development object
Ctrl+Shift+F3	Activate all inactive objects
Ctrl+Space	Code completion
Ctrl+1	Quick fix proposal
Ctrl+<	Add comment
Ctrl+Shift+<	Remove comment
Shift+F1	Format source aka pretty printer

Help

F1	ABAP keyword documentation
F2	Show code element information
Ctrl+3	Search for commands & views
Ctrl+Shift+L	List all keyboard shortcuts

Navigate

F3	Open definition
Alt+Left	Backward history
Alt+Right	Forward history
Ctrl+T	Quick hierarchy
F4	Open Type Hierarchy
Ctrl+O	Quick outline
Ctrl+Shift+G	Where-used list

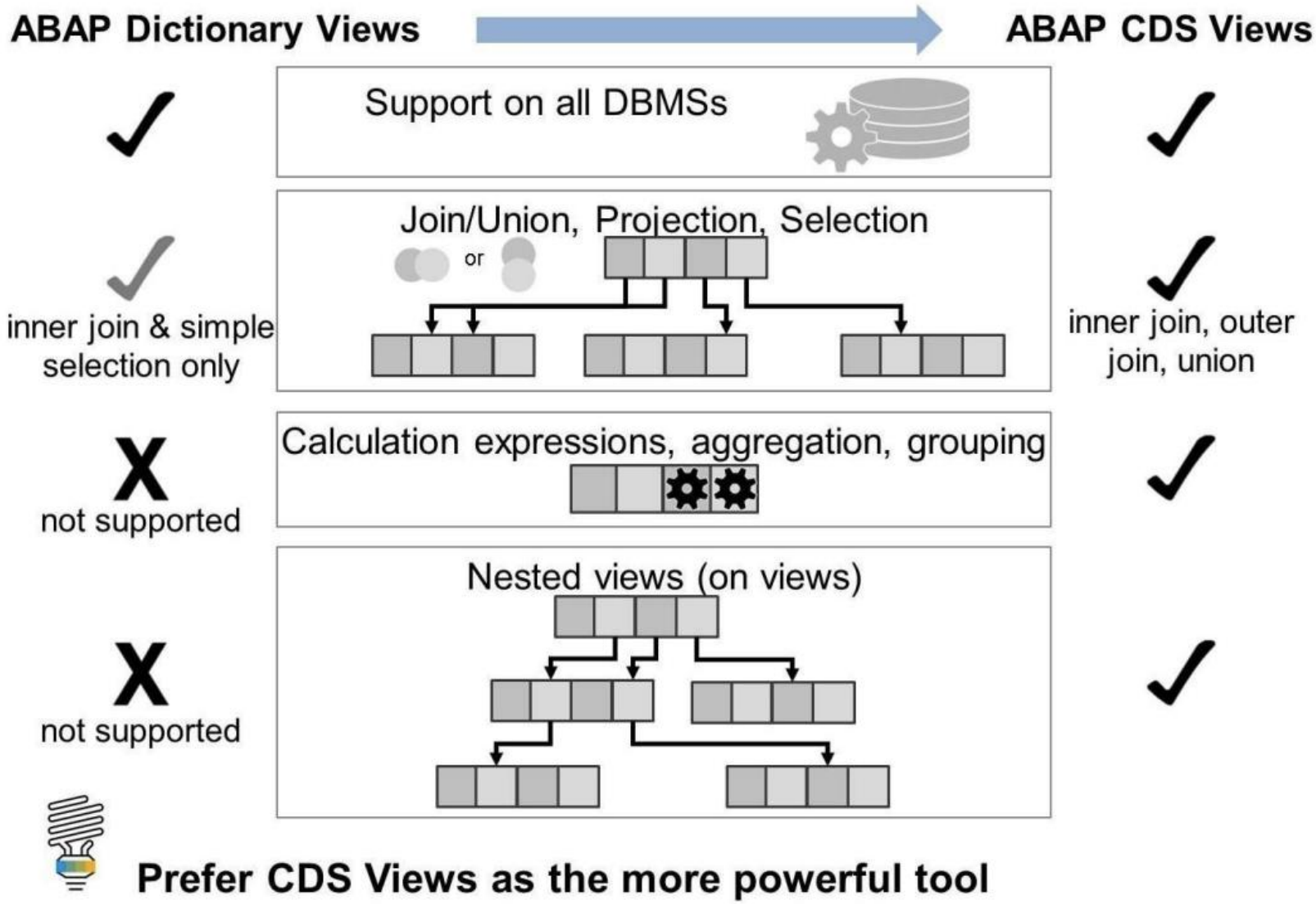
Run, Debug

F8	Run current ABAP object
Alt+F8	Select & run ABAP application
Ctrl+Shift+B	Toggle breakpoint
F5, F6, F7, F8	Step into, over, return, resume
Ctrl+Shift+F10	Execute ABAP unit tests
Alt+F9	Profile development object

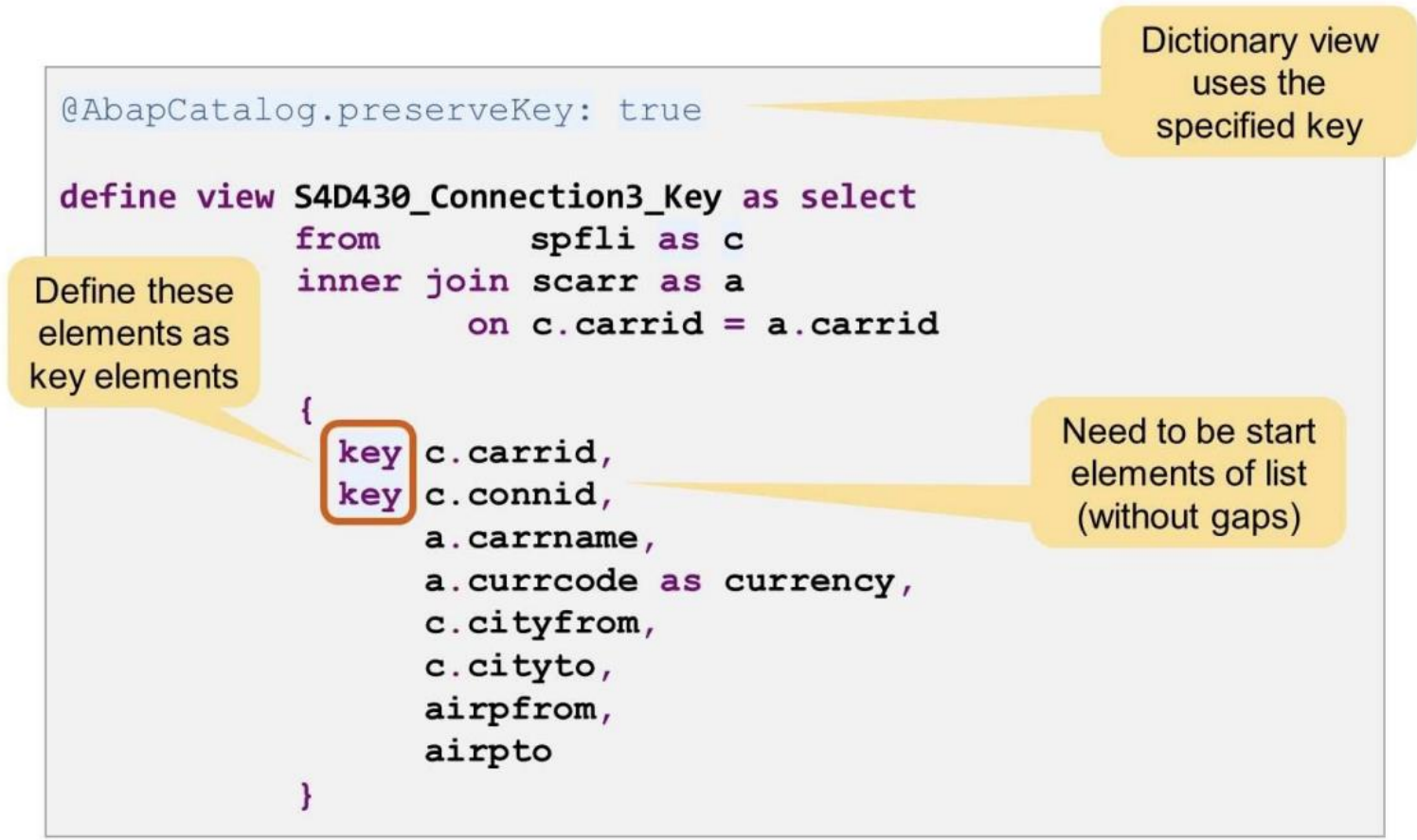
General Syntax Rules of CDS DDL

- **Allowed Characters**
  Only ASCII-characters.
- **Keywords:**
  All uppercase, all lowercase, or lowercase with uppercase initial letter.
  No mixed uppercase and lowercase.
  Allowed: SELECT, select, Select. Not allowed: SeLect, seleCT.
- **Literals:**
  Numeric literals always in full, with decimal point if necessary.
  Allowed: 1, 2.0, or 0.5. Not allowed: .5, 1,3

  Character literals enclosed in single quotations marks (').
  'LH', '00001'
- **Comments:**
  Explicit end: enclosed by /* and */
  Rest of line: two forward slashes (//)
- **Separators:**
  Statements can be closed using a semicolon (;). This is optional.
- **Protected Words:**
  Certain keywords cannot be used as self-defined names.


ABAP Dictionary Views
ABAP CDS Views
Support on all DBMSs
Join/Union, Projection, Selection
or
inner join & simple selection only
inner join, outer join, union
Calculation expressions, aggregation, grouping
not supported
Nested views (on views)
not supported
Prefer CDS Views as the more powerful tool

Key Definition



```
define view S4D430_Connection4_Selection as select
            from       spfli as c
            inner join scarr as a
                    on c.carrid = a.carrid

            {
              key c.carrid,
              key c.connid,
                  a.carrname,
                  a.currcode as currency,
                  c.cityfrom,
                  c.cityto,
                  airpfrom,
                  airpto
            }
            where c.fltype <> 'X' //exclude charter flights
```

Add WHERE clause to define a pre-selection

Aliases for Tables And Fields

```
define view S4D430_Connection2_Alias as select
            from       spfli as c
            inner join scarr as a
                    on c.carrid = a.carrid

            {
              c.carrid,
              c.connid,
              a.carrname,
              a.currcode as currency,
              c.cityfrom,
              c.cityto,
              airpfrom,
              airpto
            }
```

Alias for table name

Alias for element

(mandatory for certain expressions)

Grouping of Annotations

Individual Annotations

```
...
@AbapCatalog.compiler.compareFilter: true

@AbapCatalog.buffering.type: #GENERIC
@AbapCatalog.buffering.numberOfKeyFields: 2
@AbapCatalog.buffering.status: #ACTIVE
...
```

Grouped Sub-Annotations

```
@AbapCatalog: { compiler.compareFilter: true,

                buffering: { type: #GENERIC,
                             numberOfKeyFields: 2,
                             status: #ACTIVE
                           }
              }
```

Group of *ABAPCatalog*-annotations

Sub-Group of three *buffering*-Annotations

Literals

Character Literals

```
...
'Hello' as col_char,  //Type CHAR
'32768' as col_numc,  //Type NUMC (only digits)
...
```

Numeric Literals

```
...
32768   as col_int4, //Type INT4
4711    as col_int2, //Type INT2 in Range [-32768,32767]
255     as col_int1, //Type INT1 in Range [0,255]
1.5     as col_fltp, //Type FLTP
...
```

Built-in Functions for Calculations

- **div(*arg1,arg2*)**
  - Input: Only integer values
    (INT1, INT2, INT4, INT8 or DEC, CURR, QUAN with decimals = 0)
  - Result type: type of *arg1*
  - Result always rounded off to next integer value
- **mod(*arg1,arg2*)**
  - Input: Only integer types (INT1, INT2, INT4, INT8)
  - Result type: type of *arg1*
  - Result can be negative
- **division(*arg1, arg2, dec*)**
  - Input: Integer values, values with fixed decimal
    (Types INT1, INT2, INT4, INT8 and DEC, CURR, QUAN with any number of decimals)
  - Result type: DEC with *dec* decimal places, length depends on type of *arg1*
  - Result is always rounded to *dec* decimals ((commercial rounding)

Built-in Rounding Functions

- **abs(*arg*)**
  - returns the absolut value of *arg*

```
abs(  1.5) =  1.5
abs( -1.5) =  1.5
```

- **floor(arg)**
  - rounds to the next lower integer

    i.e. towards zero if arg > 0, away from zero if arg < 0

```
floor(  1.5) =   1
floor( -1.5) =  -2
```

- **ceil(*arg*)**
  - rounds to the next higher integer

    i.e. away from zero if arg > 0, towards zero if arg < 0

```
ceil(  1.5) =   2
ceil( -1.5) =  -1
```

- **round(*arg*,*pos*)**
  - pos > 0: Round *arg* to *pos* decimal places
  - *pos* < 0: Round *arg* to position

```
round(3.1514,  2) = 3.15
round(273.15,-1) =  270
```

Some Built-in Functions for String Processing (NW 7.40)

- **concat(*arg1*,*arg2*)**
  - Returns result of type CHAR or SSTRING (depending on types of *arg1* and *arg2*)
  - All trailing blanks are removed
  - Corresponds to ABAP statement **`CONCATENATE`** without addition **`SEPARATED BY`**
- **replace(*arg1*, *arg2*, *arg3*)**
  - Result type depends on type of *arg1*
  - corresponds to ABAP statement

    ```
    REPLACE ALL OCCURENCES OF arg2 IN arg1 WITH arg3.
    ```
- **substring(*arg*,*pos*,*len*)**
  - Result type depends on type of *arg*
  - Similar to ABAP function substring( ) or direct substring access:

    ```
    dobj[+off][(len)]
    ```

    Important difference: *pos* denotes the position, not the offset!

Some Built-in Functions for String Processing (NW 7.50)

- **concat_with_space(*arg1*,*arg2*,*count*)**
  - Like concat( ) but with *count* spaces between *arg1* and *arg2*
  - Similar to ABAP statement `CONCATENATE` with addition `SEPARATED BY`
- **length(*arg*)**
  - Returns result of type INT4
  - Trailing blanks do not count
  - Corresponds to ABAP built-in function `numofchar( )`
- **left(*arg*,*n*) and right(*arg*,*n*)**
  - Similar to substring( ), but returns first or last *n* characters of *arg*
  - left( ) corresponds to ABAP expression `arg(n)`

Built-in Functions for String Processing (NW 7.51)

- **lower(*arg*)**
  - Result type identical to type of *arg*
  - NUMC, DATS and TIMS not allowed as input type
  - Corresponds to ABAP statement **`TRANSLATE arg TO LOWER CASE`**
- **upper(*arg*)**
  - Result type identical to type of *arg*
  - NUMC, DATS and TIMS not allowed as input type
  - Corresponds to ABAP statement **`TRANSLATE arg TO UPPER CASE`**

Built-in functions

- `Unit_Conversion(p1 => a1, p2 => a2, ... )`
  - Returns result of type *abap.quan*
  - Converts a quantity in source unit into a value in target unit
  - Rules maintained in transaction CUNI and stored in database table T006
- `Currency_Conversion(p1 => a1, p2 => a2, ... )`
  - Returns result of type *abap.curr*
  - Converts an amount in source currency into a value in target currency
  - Based on the exchange rate valid on a target date
  - Rules maintained in transaction OB08 and stored in database tables TCUR... .

General Remarks

- Parameter assignment with operator “=>”
- Comma-separated parameters
- Optional parameter *error_handling* with default value "FAIL_ON_ERROR“
  Other options: "SET_TO_NULL“ and "KEEP_UNCONVERTED“
- Result may depend on the database (different rounding rules)

```
@AbapCatalog.sqlViewName: 'S4D430_FUNCCONV1'
define view S4d430_Function_Convertion1 as select
  from spfli
     { carrid,
       connid,
       @Semantics.quantity.unitOfMeasure: 'DISTID'
       Unit_Conversion( quantity    => distance,
                        source_unit => distid,
                        target_unit => cast('MI' as abap.unit)
                      ) as distance,
       @Semantics.unitOfMeasure: true
       cast('MI' as abap.unit) as distid
     }
```

Casting of literal 'MI' into expected type

Parameter assignment with operator „=>“

```
@AbapCatalog.sqlViewName: 'S4D430_FUNCCONV2'
define view s4d430_Function_Conversion2 as select
  from sflight
     { carrid,
       connid,
       fldate,
       @Semantics.amount.currencyCode:'CURRENCY'
       currency_conversion(
             amount              => price,
             source_currency     => currency,
             round               => 'X',
             target_currency     => cast( 'USD' as abap.cuky),
             exchange_rate_type  => 'M',
             exchange_rate_date  => fldate,
             error_handling      => 'SET_TO_NULL'
             ) as price,
       @Semantics.currencyCode: true
       cast('USD' as abap.cuky) as currency
      }
```

Casting of literal 'USD' into expected type

Behaviour in Case of error

Built-in functions, new in NW 7.50

- **dats_is_valid(*date*)**
  - Returns result of type INT4
  - Returns 1 if *date* contains a valid date, 0 otherwise
- **dats_days_between(*date1*,*date2*)**
  - Returns result of type INT4
  - Calculates number of days between two dates (corresponds to *date2* – *date1* in ABAP
- **dats_add_days(*date*,*count*,*on_error*)**
  - Returns result of type DATS
  - Adds count days to the given date (corresponds to *date* + *count* in ABAP)
- **dats_add_months(*date*,*count*,*on_error*)**
  - Returns result of type DATS
  - Adds *count* months to the given date (no simple equivalent in ABAP)

General Remarks:

- All dates in format YYYYMMDD (technical format on database)
- Possible values for *on_error*: 'FAIL', 'NULL', 'INITIAL', 'UNCHANGED'

```
 @AbapCatalog.sqlViewName: ,S4D430_FUNCDAYS'
define view S4D430_Function_Days as select
       from sbook
          {
            carrid,
            connid,
            fldate,
            bookid,

            dats_days_between(order_date, fldate) as days_ahead,

            dats_add_days( order_date, 14, 'FAIL' ) as due_date

          }
```

„Subtract“ two fields of type DATE

What happens in case of an error? (Here: Raise exception)

Aggregate Functions

- **MIN(*operand*) and MAX(*operand*)**
  - Returns the smallest/greatest value in *operand*
- **SUM(*operand*)**
  - Calculates the sum of the values of *operand*
- **AVG(*operand*)**
  - Calculates the average value of the values of *operand*
- **COUNT(*)**
  - Returns the number of entries in the result set
- **COUNT(DISTINCT *operand*)**
  - Returns the number of distinct values of *operand*

Operand can be

- a field of the data source
- a literal
- a case distinction

```
@AbapCatalog.sqlViewName: 'S4D430AGGREGATE'
define view S4d430_Aggregates as select
      from sflight
  {
   min( seatsocc )                          as col_min,
   max( seatsocc )                          as col_max,
   sum( seatsocc )                          as col_sum,

   avg( seatsocc )                          as col_avg,
   avg( seatsocc as abap.dec(16,2) )        as col_avg_conv,

   count(*)                                 as col_count,
   count(distinct planetype)                as col_cnt_dist,

   cast( sum( 1 ) as abap.int4 )            as col_literal,
   sum(
     case
       when seatsocc > seatsmax
       then cast( 1 as abap.int4 )
       else 0
     end )                                  as col_overbooked
}
```

Alias name mandatory for aggregate expressions

Result type of avg( ) is FLTP – change it with addition as

Sum of literal 1 (Same result as count(*))

Cast required to avoid overflow

Sum of a case distinction (Count overbooked flights)

```
@AbapCatalog.sqlViewName: 'S4D430_AGGRGRP1'
define view S4D430_Aggregates_Group_By_1 as select
  from sflight
  {
    carrid,
    connid,
    count(*)                    as col_count,
    avg( seatsocc )             as col_avg,
}
group by carrid, connid
```

Field list consists not only of aggregate expressions

GROUP BY needed for all fields in the field list

```
@AbapCatalog.sqlViewName: 'S4D430_AGGRGRP2'
define view S4D430_Aggregates_Group_By_2 as select
  from sflight
  {
    concat_with_space(carrid, connid, 1) as ID,
    count(*)                             as col_count,
    avg( seatsocc )                      as col_avg,
}
group by carrid, connid
```

Field list contains expression or function

Arguments of the expression or function listed individually

Important Restrictions for Aggregate Functions in CDS

- **Argument of aggregate function**
  - only fields, literals, case distinctions
  - no other expressions or functions (e.g. avg (seatsmax – seatsocc))
  - use „View on View“ to aggregate calculated results
- **Argument of function SUM**
  - has to be numeric
  - INT8, DFL16, DFL34 are not supported
- **NULL values**
  - Are not considered in the aggregation
- **CDS View Extensions (see later)**
  - CDS Views with aggregates cannot be extended before Rel. 7.51

Example: Arithmetic Expressions in CDS Views

```
@AbapCatalog.sqlViewName: 'S4D430_EXPRARITH'
define view s4d430_Expression_arithmetic as select
  from sflight
  {
    seatsmax,
    seatsocc,

    seatsmax - seatsocc                as seatsfree,
    seatsocc + seatsocc_b + seatsocc_f as seatsmax_tot,
    2 * price                          as double_price

}
```

Session Variables in CDS ABAP

- **Are global variables of the Database**
- **Correspond to system fields of the ABAP runtime**

Session Variable	Related ABAP system field
$session.client	*sy-mandt*
$session.system_date	*sy-datum*
$session.system_language	*sy-langu*
$session.user	*sy-uname*
$session.user_date	*sy-datlo*
$session.user_timezone	*sy-zonlo*

- **Are set to their values when the view is used in ABAP SQL**
- **Have undefined content if the CDS View is not used in ABAP SQL**

> To access ABAP system fields, it is generally preferable to use annotated input parameters

```
@AbapCatalog.sqlViewName: 'S4D430_SESVAR'

define view S4D430_Session_Variables
   as select from scarr
  left outer join tcurt
               on scarr.currcode = tcurt.waers
              and tcurt.spras    = $session.system_language

                  {

                    key scarr.carrid,
                        scarr.carrname,
                        scarr.currcode,
                        tcurt.ltext

                  }
```

Type Conversion with CAST Expression

- **What it looks like:**

```
CAST ( operand AS target_type [PRESERVING TYPE] )
```

- **What it does:**
  - Converts the value of *operand* into *target_type*
- **Many Options for *operand*:**
  - Literal (without a domain prefix)
  - Field of a data source
  - Arithmetic expression
  - Case distinction with CASE
  - Predefined function
  - …
- **Two Options for *target_type*:**
  - A predefined dictionary type, e.g. *abap.int4, abap.char(10), abap.dec(8,2)*
  - Any Dictionary data element, e.g. *S_CARRID* , *BUKRS*
  - Addition PRESERVING TYPE to change semantic attributes, only

Example: Type Conversions With CAST

```
@AbapCatalog.sqlViewName: 'S4D430_EXPRCAST'
define view S4d430_Expression_Cast as select
  from sflight
  {
    '19891109'                                          as col_char,

    cast('19891109' as abap.int4)                       as col_int4,
    cast('19891109' as abap.dec(16,2))                  as col_dec,
    cast('19891109' as abap.fltp)                       as col_fltp,

    cast('19891109' as abap.dats)                       as col_dats,
    cast('19891109' as s_date)                          as col_ddic,
    cast('19891109' as s_customer preserving type)      as col_cust,

    cast(seatsocc as abap.fltp) / cast(seatsmax as abap.fltp)
                                                        as ratio
 }
```

col_char	col_int4	col_dec	col_fltp	col_dats	col_ddic	ratio
19891109	19.891.109	19891109.00	1.9891109000000000E+07	1989-11-09	1989-11-09	9.7402597402597402E-01
19891109	19.891.109	19891109.00	1.9891109000000000E+07	1989-11-09	1989-11-09	9.6623376623376622E-01

Case Distinctions

Simple Case Distinction

```
CASE operand
        WHEN operand1 THEN result1
       [WHEN operand2 THEN result2]
        ...
       [ELSE resultn]
 END
```

- Comparable to ABAP Statement CASE … WHEN … ENDCASE.
- Result depends on a series of “EQUALS”-comparisons

Complex Case Distinction (= Searched Case)

```
CASE WHEN sql_condition1 THEN result1
    [WHEN sql_condition2 THEN result2]
         ...
    [ELSE resultn]
 END
```

- Comparable to ABAP Statement IF … ELSEIF …. ENDIF.
- Result depends on a sequence of SQL conditions (logical expressions)

Example 1: A Simple Case Expression

```
@AbapCatalog.sqlViewName: 'S4D430_EXPRCASE'
define view S4d430_Expression_Case as select
  from sbook
  {

 ...

// Simple Case

      case class
         when 'Y' then 'Economy'
         when 'C' then 'Business'
         when 'F' then 'First'
       end                              as class_txt
...

  }
```

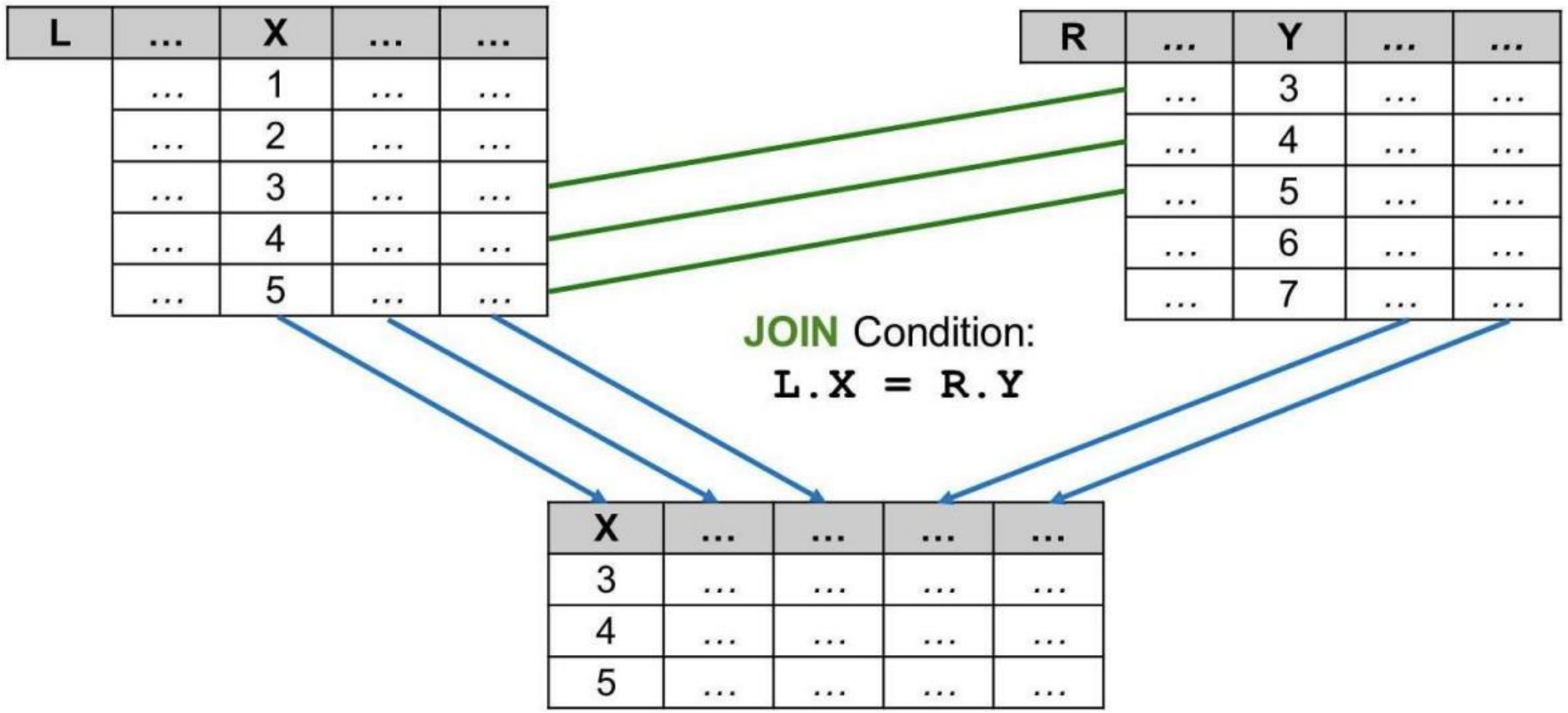
Example 3: Two Nested Case Expressions

```
@AbapCatalog.sqlViewName: 'S4D430_EXPRCASE'
define view S4d430_Expression_Case as select
  from sbook
  {

...

// Nested Case
       case class
         when 'F' then ' '
         else case
                when ( wunit = 'KG' and luggweight > 20 )
                  or ( wunit = 'LB' and luggweight > 44 )
                then 'X'
                else ' '
              end
        end                                          as excess_luggage2
  }
```

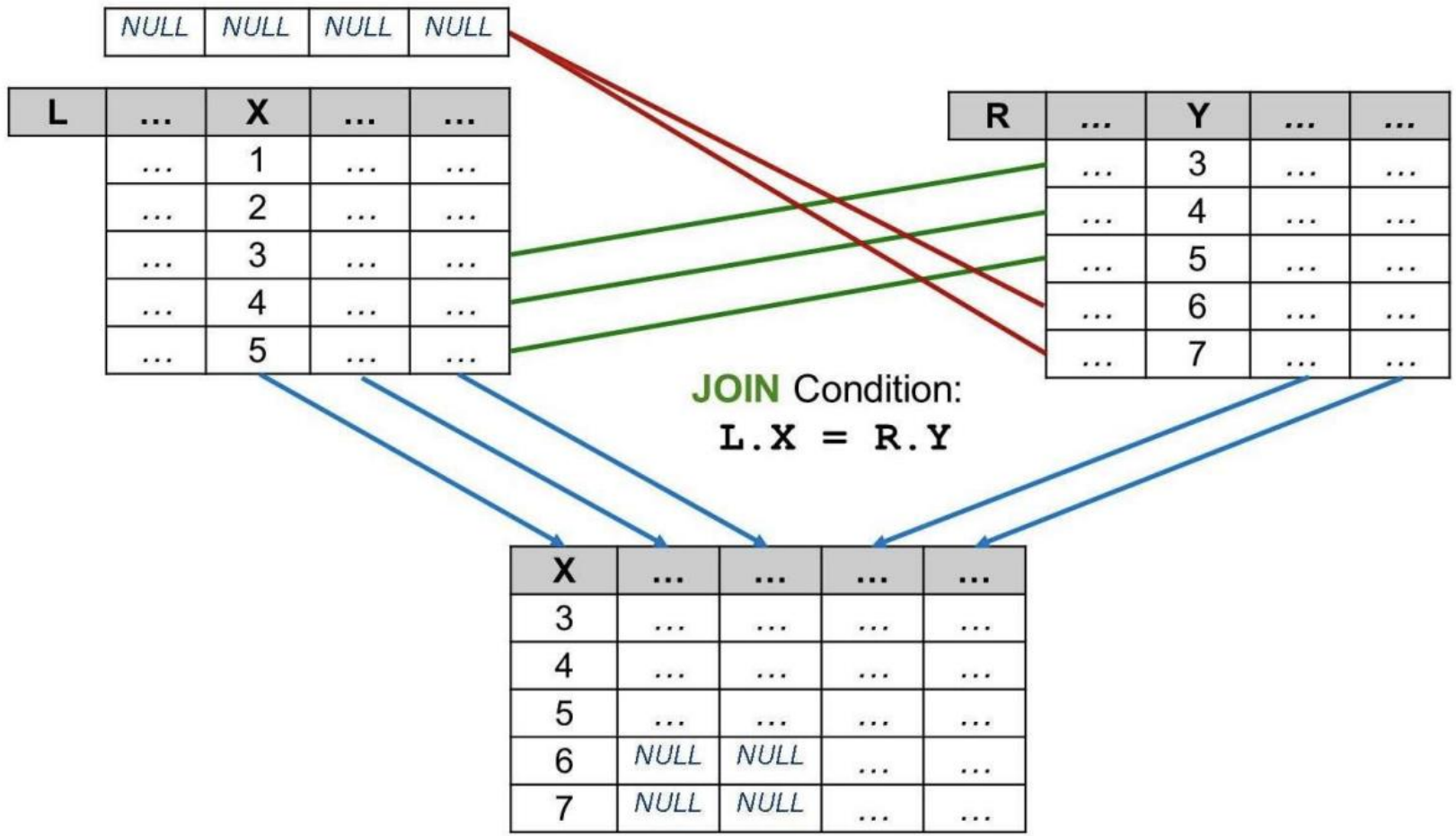
Example 2: A Complex Case Expression

```
@AbapCatalog.sqlViewName: 'S4D430_EXPRCASE'
define view S4d430_Expression_Case as select
  from sbook
  {

...

// Complex Case
       case
        when class = 'F' then ' '
        when wunit = 'KG' and luggweight > 20 then 'X'
        when wunit = 'LB' and luggweight > 44 then 'X'
        else ' '
       end                                          as excess_luggage1,

...

  }
```

Reminder: Inner Join



Right Outer Join



Example: Right Outer Join

```
@AbapCatalog.sqlViewName: 'S4D430_JOINRIGHT'
define view S4d430_Join_Right_Outer as select
       from  spfli as c right outer join scarr as a
         on c.carrid = a.carrid
      {
        key a.carrid,
            a.carrname,
            c.connid,
            c.cityfrom,
            c.cityto
}
```

Raw Data

Filter pattern

36 rows retrieved - 62 ms

carrid	carrname	connid	cityfrom	cityto
AA	American Airlines	0017	NEW YORK	SAN FRANCISCO
AA	American Airlines	0064	SAN FRANCISCO	NEW YORK
AB	Air Berlin	0000		
AC	Air Canada	0000		
AF	Air France	0000		
AZ	Alitalia	0555	ROME	FRANKFURT

No flight connections in table SPFLI for Carriers “AB“, “AC“ and “AF“

Figure 86: Left Outer Join

Figure 96: UNION

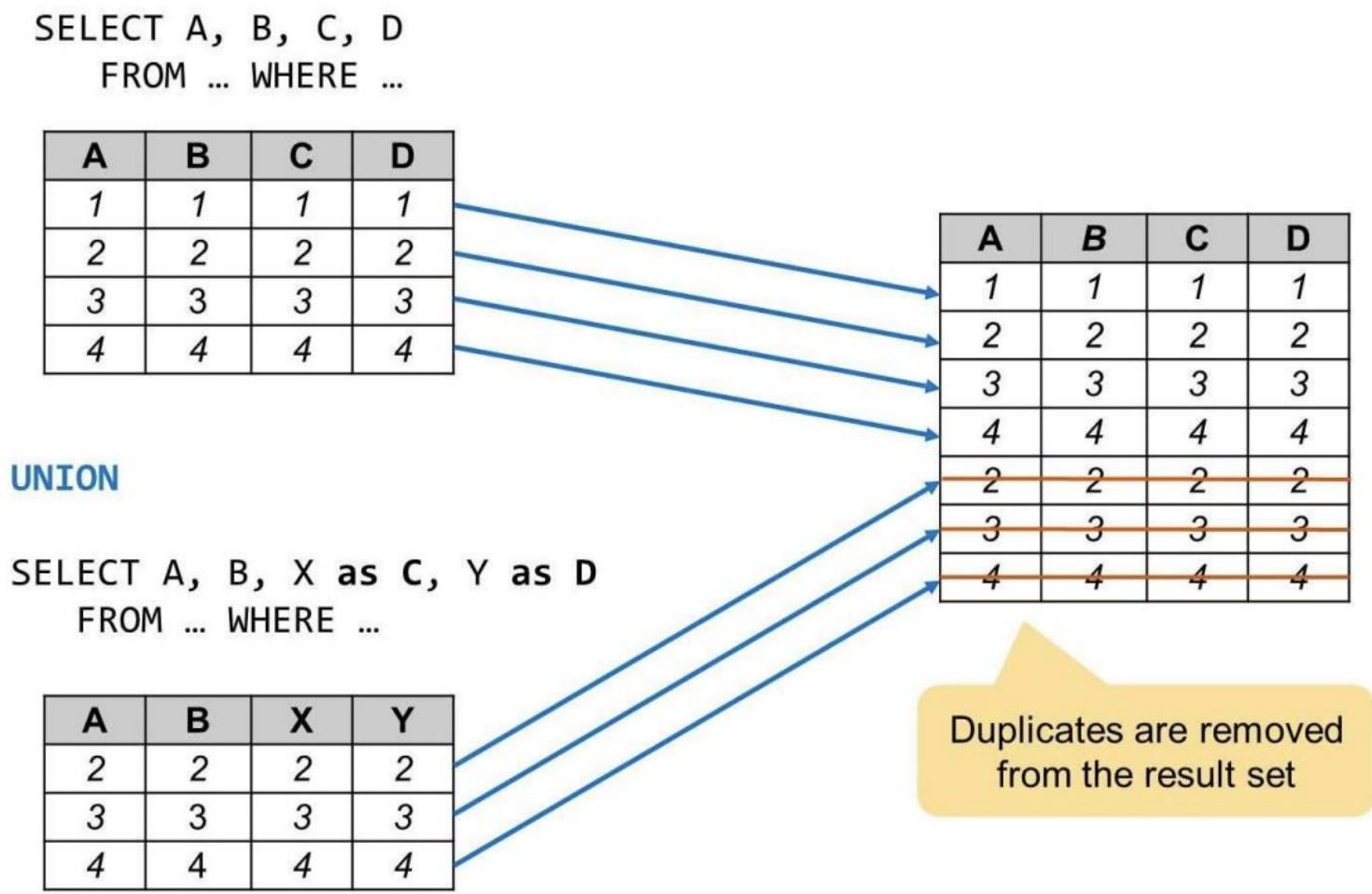


Figure 98: Example: UNION

```
@AbapCatalog.sqlViewName: 'S4D430_UNIO'
define view S4d430_Union as
select from scustom
          {
            key id,
            key 'Customer'       as type,
                name,
                city,
                country
           }
union
select from stravelag
          {
            key agencynum        as id,
                'Agency  '       as type,
                name,
                city,
                country
     }
```

Key definition in second select is ignored

Alias *id* is needed to have identical element names

Additional blanks needed to have compatible types

Figure 87: Unions

Merge the results of queries using keyword `UNION`

- Column lists of the queries must
  - have equal number of columns
  - Be of compatible types in matching order
- `UNION` implies a distinct semantic
- `UNION ALL` does not include distinct

```
define view HA400D_07_Union as
select from scustom
 {  id,
    form, name,
    street,
    postcode, city,
    country,
    postbox, custtype
} where postbox <> ' '
union
select from scustom
 {  id,
    form, name,
    street,
    postcode, city,
    country,
    postbox, custtype
} where custtype = 'B'
```

First query

Second query

Figure 99: UNION ALL

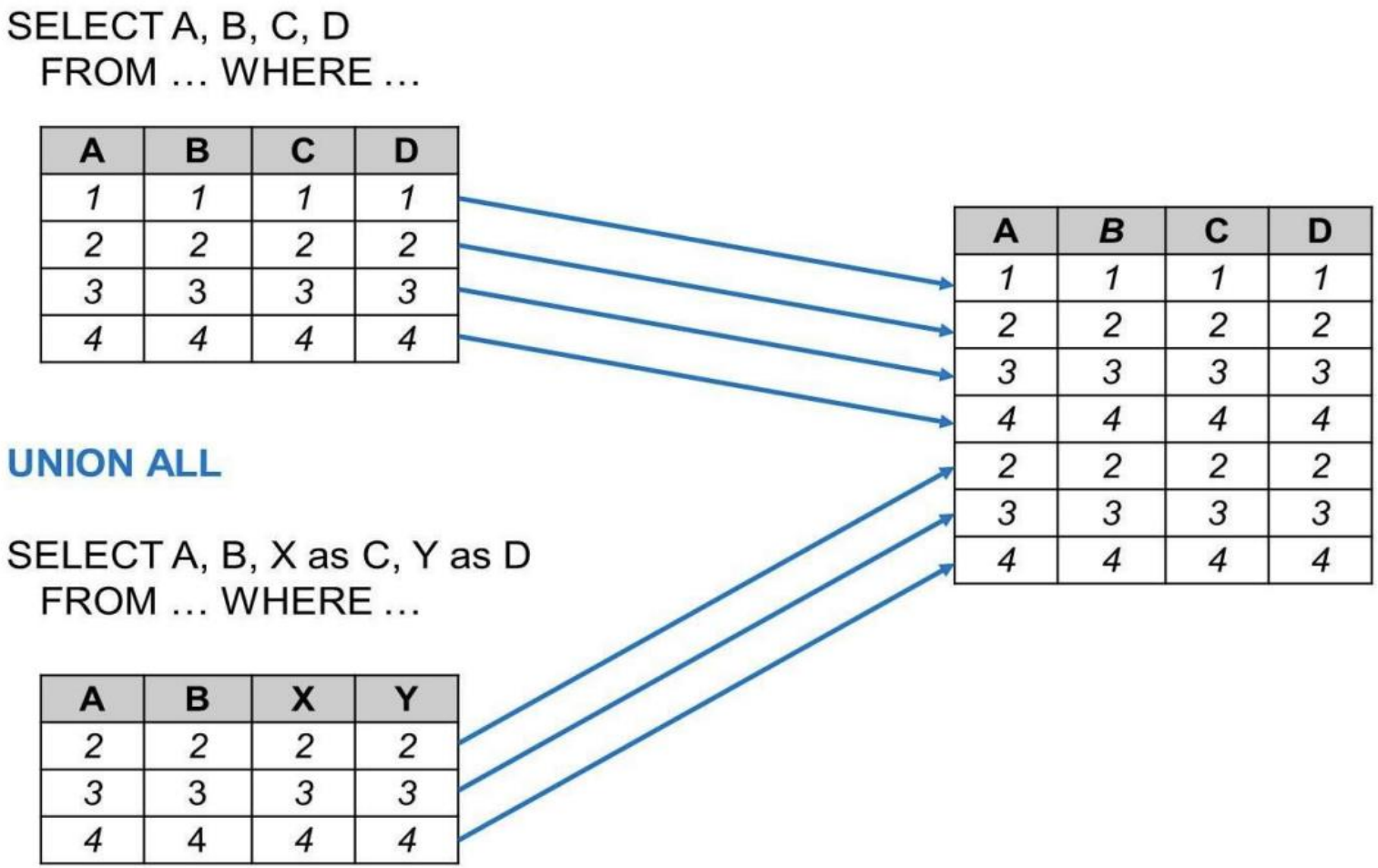
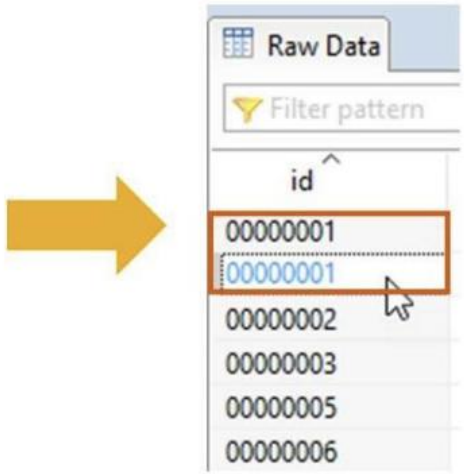


Figure 100: Example: UNION vs UNION ALL

```
@AbapCatalog.sqlViewName: 'S4D430_UNIOALL1'
define view S4d430_Union_All_1 as
select from scustom
          { key id }
      where city = 'Walldorf'
union all
select from sbook
          { key customid as id }
      where agencynum = '00000100'
```




```
@AbapCatalog.sqlViewName: 'S4D430_UNIOALL2'
define view S4d430_Union_All_2 as
select from scustom
          { key id }
      where city = 'Walldorf'
union
select from sbook
          { key customid as id }
      where agencynum = '00000100'
```

Same definition without ALL

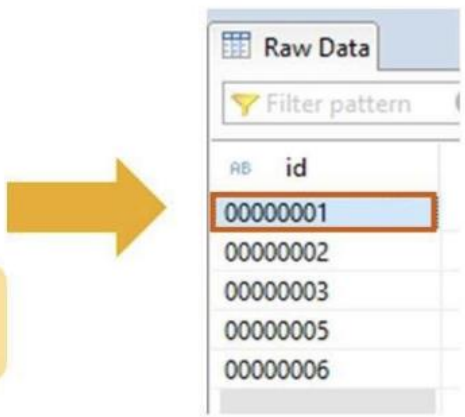


Figure 90: Cross Join (As off Release 7.51)

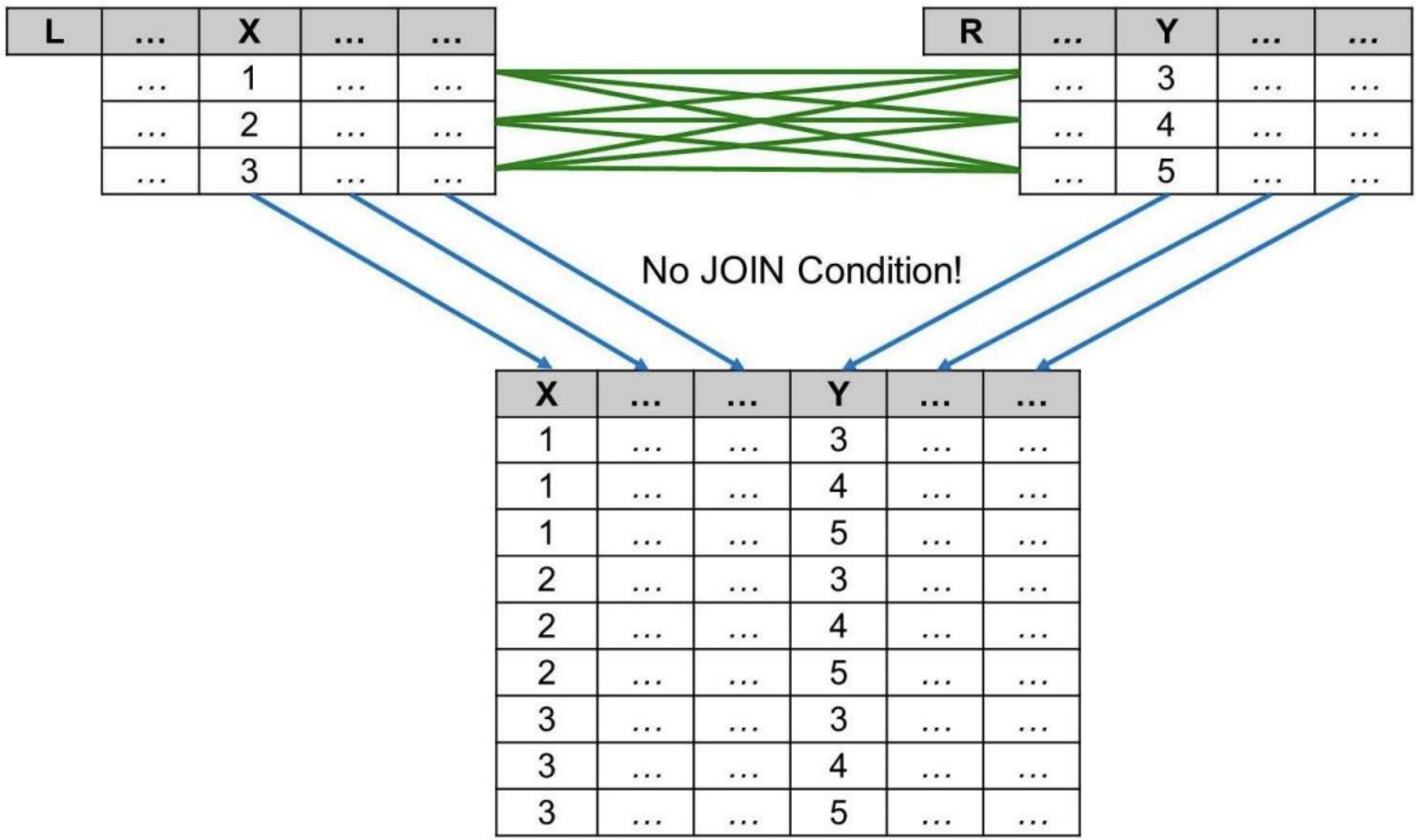


Figure 91: Example: Cross Join

```
@AbapCatalog.sqlViewName: 'S4D430_JOINCROSS'
define view S4D430_JOIN_CROSS as select
       from  spfli as c cross join scarr as a
      {
        key a.carrid as carrid_scarr,
            a.carrname,
            c.carrid as carrid_spfli,
            c.connid,
            c.cityfrom,
            c.cityto
}
```

S4D430_JOIN_CROSS

Raw Data

Filter pattern | 100 rows retrieved - 3 ms (partial result)

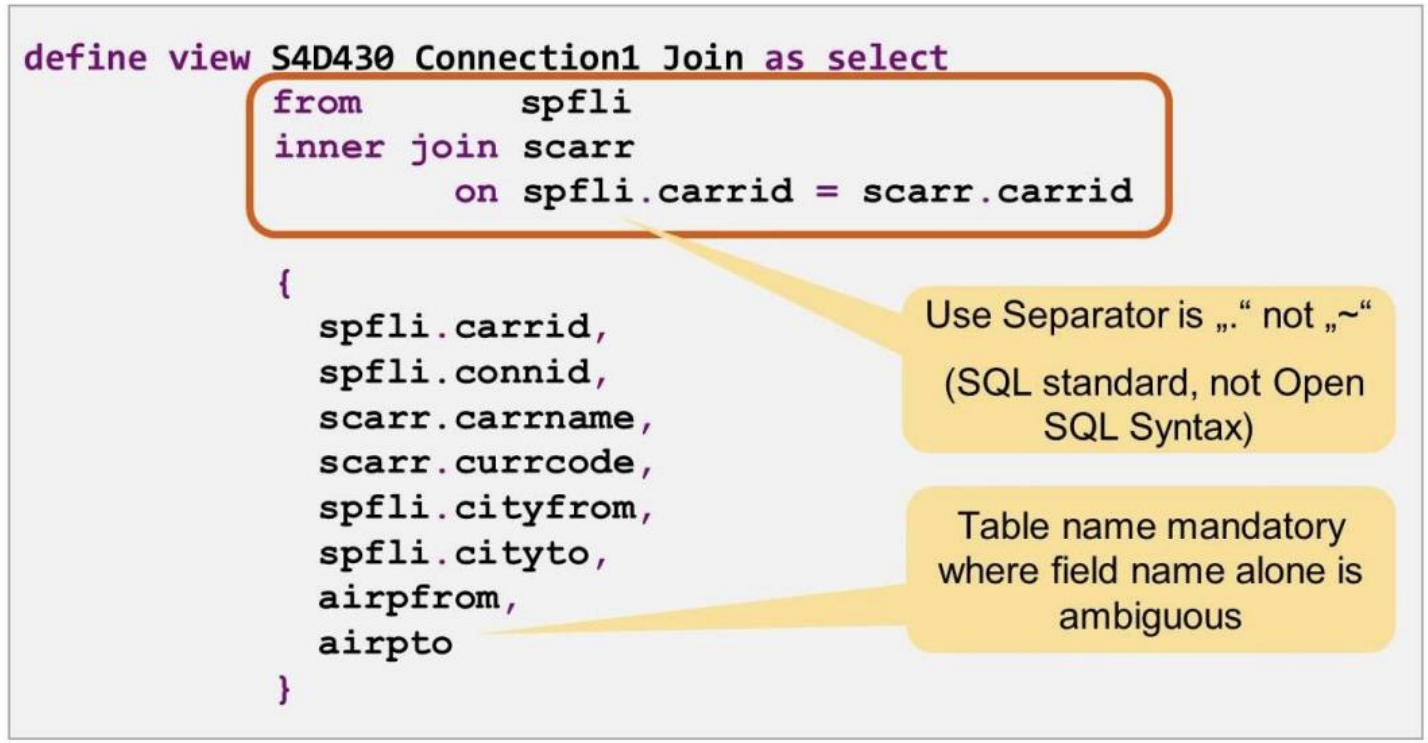
carrid_scarr	carrname	carrid_spfli	connid	cityfrom	cityto
AA	American Airlines	AZ	0555	ROME	FRANKFURT
AA	American Airlines	LH	2402	FRANKFURT	BERLIN
AA	American Airlines	UA	0941	FRANKFURT	SAN FRAN...
AA	American Airlines	AZ	0789	TOKYO	ROME
AA	American Airlines	LH	0402	FRANKFURT	NEW YORK
AA	American Airlines	QF	0005	SINGAPORE	FRANKFURT

Figure 94: Example: Nested Join Expression

```
@AbapCatalog.sqlViewName: 'S4D430_JOINNEST'
define view S4d430_Join_Nested as select
       from scarr as a
       left outer join (
                         sairport as p
                         left outer join scounter as c
                                      on p.id = c.airport
                       )
                    on a.carrid = c.carrid
     {
       a.carrid   as carrier_id,
       p.id       as airport_id,
       c.countnum as counter_number
     }
```

Join in parantheses is evaluated first

Figure 44: Inner Join in DDL View Definition



View Definition with Association

```
@AbapCatalog.sqlViewName: 'S4D430_ASSO1'
define view S4d430_Association_1 as select
  from spfli association to scarr
    on spfli.carrid = scarr.carrid
   { key carrid,
     key connid,
         scarr.carrname
   }
```

Association target

Name of data source mandatory for all fields from association target

View Definition with Join

```
@AbapCatalog.sqlViewName: 'S4D430_JOININN'
define view S4d430_JOIN_INNER as select
  from spfli inner join scarr
    on spfli.carrid = scarr.carrid
     { key spfli.carrid,
       key connid,
           carrname
     }
```

Name of data source only mandatory for non-unique field names

Technical Realization of Associations

View Definition with Association

```
@AbapCatalog.sqlViewName: 'S4D430_ASSO1'
define view S4d430_Association_1 as select
  from spfli association to scarr
    on spfli.carrid = scarr.carrid
   { key carrid,
     key connid,
         scarr.carrname
   }
```

Corresponding SQL Create Statement

```
CREATE VIEW "S4D430_ASSO1" AS SELECT
  "SPFLI"."MANDT" AS "MANDT",
  "SPFLI"."CARRID",
  "SPFLI"."CONNID",
  "SPFLI"."CITYFROM",
  "SPFLI"."CITYTO",
  "=A0"."CARRNAME"
FROM "SPFLI" "SPFLI" LEFT OUTER JOIN "SCARR" "=A0" ON (
  "SPFLI"."MANDT" = "=A0"."MANDT" AND
  "SPFLI"."CARRID" = "=A0"."CARRID"
)
```

Example: Additional Semantics

```
@AbapCatalog.sqlViewName: 'S4D430_ASSO3'
define view S4d430_Association_3 as select

from spfli as c
     association[1..1] to scarr as _Carrier
       on c.carrid = _Carrier.carrid
   {
      key c.carrid as CarrierID,
      key c.connid,
          c.cityfrom,
          c.cityto,
          _Carrier.carrname
   }
```

Cardinality (exactly one carrier related to a connection)

Name of the association

Replaced with Alias in SQL Create Statement

Extract From the SQL Create Statement

```
  "=A0"."CARRNAME"
FROM "SPFLI" "C" LEFT OUTER JOIN "SCARR" "=A0" ON (
  "C"."MANDT" = "=A0"."MANDT" AND
  "C"."CARRID" = "=A0"."CARRID"
)
```

On-Condition with $Projection

```
@AbapCatalog.sqlViewName: 'S4D430_ASSO3'

define view S4D430_Association_3 as select
  from spfli as c
     association[1..1] to scarr as _Carrier

            on $projection.CarrierID = _Carrier.carrid
   {
      key c.carrid as CarrierID,
      key c.connid,
          c.cityfrom,
          c.cityto,
          _Carrier.carrname
   }
```

Use $projection on the left hand side of the ON-clause

If the field has an alias, the alias has to be used after $projection

Some Rules Regarding Cardinality

- **Cardinality is optional**
  - Default cardinality is [0..1]
- **Minimum value is optional**
  - Default Value for mimimum is 0

    [1] means [0..1]

    [4] means [0..4]

    [*] means [0..*]
- **Forbidden Values**
  - Minimum can not be *
  - Maximum can not be 0
- **Syntax Check for Maximum > 1**
  - Association cannot be used in a WHERE condition (syntax error)
  - Association should not be used outside aggregate expressions (syntax warning)

Example: Syntax Warning for Cardinality [*]

```
@AbapCatalog.sqlViewName: 'S4D430_ASSOWARN'
define view S4d430_Association_WARNING as select
  from spfli as c
    association[*] to sflight as _Flights     // short for [0..*]
                   on $projection.carrid = _Flights.carrid
                  and $projection.connid = _Flights.connid
   {
     key carrid,
     key connid,
         _Flights.fldate
}
```

Access to association with cardinality [*] increases result set

Result without field fldate

Raw Data

Filter pattern 26 rc

carrid	connid
AA	0017

… and with field fldate

Raw Data

Filter pattern 100 rows retrieved

carrid	connid	fldate
AA	0017	2015-11-18
AA	0017	2015-12-16
AA	0017	2016-01-13
AA	0017	2016-02-10
AA	0017	2016-03-09
AA	0017	2016-04-06

Example: Exposed Association

```
@AbapCatalog.sqlViewName: 'S4D430_ASSOEXP'
define view S4d430_Association_Exposed as select
 from spfli as c
     association[1] to scarr as _Carrier
                    on $projection.carrid = _Carrier.carrid
     association[*] to sflight as _Flights
                    on $projection.carrid = _Flights.carrid
                   and $projection.connid = _Flights.connid

{
      key c.carrid,
      key c.connid,
          c.cityfrom,
          c.cityto,

          _Carrier,
          _Flights
   }
```

Prerequisite: all fields used in the ON condition also in element list

Association target visible to consumer of this CDS View

Difference Between Exposed and Ad-hoc Associations

```
@AbapCatalog.sqlViewName: 'S4D430_ASSOEXP'
define view S4d430_Association_Exposed as select
 from spfli as c
     association[1] to scarr as _Carrier
                    on $projection.carrid = _Carrier.carrid
     association[*] to sflight as _Flights
                    on $projection.carrid = _Flights.carrid
                   and $projection.connid = _Flights.connid
   {
      key c.carrid,
      key c.connid,
          c.cityfrom,
          c.cityto,
          _Carrier,
          _Flights
   }
```

No warning if association with cardinality [*] is exposed

Exposed Associations are not (yet) joined on database level

```
CREATE VIEW "S4D430_A    XP" AS SELECT
  "C"."MANDT" AS "MA    ,
  "C"."CARRID",
  "C"."CONNID",
  "C"."CITYFROM",
  "C"."CITYTO"
FROM "SPFLI" "C"
```

Figure 86: Support for Joins, WHERE, Aggregations, Grouping, and Filtering in CDS Views

Like in Open SQL:

JOINs supported

- **INNER**, **LEFT OUTER**, **RIGHT OUTER** join
- Before or after the field list
- No SELECT * together with joins

Aggregations supported

WHERE clause supported

Grouping, Filtering supported

- All columns not aggregated must be listed in **GROUP BY**
- **HAVING** clause filters resulting groups, aggregate functions supported

```
define view HA400D_06_Complex as
  select from sbook as book
    inner join scustom as customer
            on book.bookid = customer.id
{
    book.carrid,
    book.loccurkey as currcode,

    sum(book.loccuram) as total_amount,
    max(book.loccuram) as max_amount,
    min(book.loccuram) as min_amount,

    customer.region,
    customer.discount
}
  where     customer.country = 'US'

  group by book.carrid, book.loccurkey,
           customer.region,
           customer.discount
  having sum(book.loccuram) > 100
```

Figure 88: CDS Views in ABAP Programming

CDS views in ABAP Programming

- CDS view name as ABAP data type
- CDS view name in the FROM clause of Open SQL SELECT statements
- New Open SQL Syntax mandatory
- CDS views not allowed in INSERT, UPDATE, MODIFY, DELETE

```
DATA lt_book TYPE STANDARD TABLE
                OF HA400D_02_Field_List.

SELECT *
  FROM HA400D_02_Field_List
  INTO TABLE @lt_book
 WHERE customer = '931'.
```

Figure 85: Select List – Support for Expressions in CDS Views

String & arithmetic expressions

- Literals
- Operators: +, - , *, /, unary –
- Built-in functions, e.g. CONCAT

CAST expression

- Length determined at activation time
- No nesting of CAST expressions

CASE expression

Special built-in functions

- Currency and unit conversion
- COALESCE

Alias required for result columns

```
define view HA400D_05_Expressions as
select from sbook {
CONCAT( carrid, connid) as flightno,

fldate, bookid,

CONCAT( CONCAT(
  SUBSTRING(order_date,5,2), '-') ),
  SUBSTRING(order_date,1,4)) as ordermonth,

case smoker
  when 'X' then
        cast(loccuram as abap.fltp)*1.03
  else  cast(loccuram as abap.fltp)*0.98
end as adjusted_amount,

currency_conversion(
   amount           => loccuram,
   source_currency => loccurkey,
   target_currency =>
                cast('EUR' as abap.cuky(5)),
   exchange_rate_date => order_date
  ) as euro_amount   }
```

```
@EndUserText.label: 'Billing Header'
@Metadata.ignorePropagatedAnnotations: true
@ObjectModel.usageType:{
    serviceQuality: #X,
    sizeCategory: #S,
    dataClass: #MIXED
}
define view entity ZI_BILLING_HEADER as select from vbrk
{
    @Consumption.valueHelpDefinition: [{ entity: {name: 'ZI_BILLING_HEADER', element:
    key cast(vbeln as vbeln_vf preserving type ) as BillingDocument,
        fkdat as BillingDate,
        kunrg as Payer,
        @Semantics.amount.currencyCode: 'CurrencyKey'
        netwr as NetValue,
        clrst as Status,
        cast('USD' as abap.cuky( 5 )) as CurrencyKey,
        changed_on as LastChanged,
        cast(0 as timestampl) as LocalLastChanged


}

```