Chapter 6: Conversion Functions and Conditional Expressions

Conversion Functions Overview

Conversion functions are used to **convert data from one type to another** without changing the value, only the type. They are required when operations (e.g., math functions) expect specific data types.

Common Categories

- NUMBER
- VARCHAR2
- DATE
- TIMESTAMP

Use Cases & Examples

- '1230' stored in a VARCHAR2 column can be converted to NUMBER for calculations
- Conversion can help detect input issues (e.g., the letter "O" instead of zero in '123O')

Formatting Capability

Conversion functions can **format output**, not just change type:

- Format a DATE into "Thursday, July the Fourth, Seventeen Seventy-Six" using TO_CHAR
- Used for **international number formats**, currency, punctuation, etc.

Explicit vs Implicit Conversion

Explicit Conversion

Done with functions like:

- `TO_NUMBER(text)`
- `TO_CHAR(date_or_number)`
- `TO_DATE(text)`

Benefits:

- Preferred for **clarity, control, and performance**
- Prevents errors from future schema/data changes
- Recommended by Oracle

Implicit Conversion

Happens **automatically** when needed. SQL detects type mismatches and converts on its own:

```sql
SELECT 'Chapter ' || 1 || ' ... I am born.' FROM DUAL;
-- Output: Chapter 1 ... I am born.
```

Examples:

```sql
SELECT SYSDATE,
       ADD_MONTHS(SYSDATE, SUBSTR('plus 3 months', 6, 1)) PLUS_THREE
FROM DUAL;
-- SUBSTR returns '3' as text, SQL converts it to number implicitly

SELECT 'TRUE' FROM DUAL WHERE '3' > '20';  -- TRUE (text comparison)
SELECT 'TRUE' FROM DUAL WHERE '3' > 20;    -- FALSE (numeric comparison)
```

Implicit Conversion Gotchas

- Can cause **unexpected behavior** and **performance issues**, especially with indexes
- Example: Comparing strings '3' > '20' returns TRUE (alphabetic), but '3' > 20 is FALSE (numeric)

Automatic Conversion Rules

Will NOT convert:

- Numeric → Date
- Numeric/Text → LOB, CLOB, RAW, etc.
- Very large types → each other (LOB ↔ RAW, etc.)

Best Practice

Always **use explicit conversion** in production code to ensure clarity, stability, and performance.

Core Conversion Functions

TO_NUMBER

Converts character data to numeric format.

Syntax:

```sql
TO_NUMBER(e1, format_model, nls_parms)
```

- `e1`: the expression to convert (usually character to number)
- `format_model`: guides parsing format ($999,999.99, 9G999D99, etc.)
- `nls_parms`: for locale-specific settings (NLS_CURRENCY, NLS_NUMERIC_CHARACTERS, etc.)

Example:

```sql
TO_NUMBER('$17,000.23', '$999,999.99')
-- Returns: 17000.23

-- NLS Example:
TO_NUMBER('17.000,23', '999G999D99', 'nls_numeric_characters = '',.'' ')
```

Format Elements:

- `9` = digit
- `0` = leading/trailing zero
- `G` = group separator (e.g., comma)
- `D` = decimal
- `L` = local currency
- `MI`, `PR`, `S` = signs
- `EEEE` = scientific notation

TO_CHAR

Converts data to character format. Three overloaded versions:

1. TO_CHAR(Character)

```sql
TO_CHAR(c)
```

Converts NCHAR, NVARCHAR2, CLOB, etc. → VARCHAR2

2. TO_CHAR(Number)

```sql
TO_CHAR(n, format_model, nls_parms)
```

Converts number → formatted string

Example:

```sql
TO_CHAR(198, '$999.99')  -- Returns: $198.00
```

3. TO_CHAR(Date)

```sql
TO_CHAR(d, format_model, nls_parms)
```

Converts date to string using date format elements

Example:

```sql
TO_CHAR(SYSDATE, 'DAY, "the" DDth "of" Month, YYYY')
-- Returns: Wednesday, the 17th of February, 2016
```

Common Date Format Elements:

- `YYYY`, `RRRR` = year
- `MM` = month
- `DD` = day
- `HH24`, `HH12`, `MI`, `SS` = hour, minute, second
- `DY`, `DAY` = name of day
- `TH`, `th` = 1st, 2nd, etc.
- `FM` = remove leading/trailing blanks
- `FX` = exact match with input

⚠️ **Important:** In date formats, `MM` means **month**, not **minute** -- use `MI` for minutes.

```sql
TO_CHAR(SYSDATE, 'DD-MON-RRRR HH:MM:SS')  -- WRONG: MM is month
TO_CHAR(SYSDATE, 'DD-MON-RRRR HH:MI:SS')  -- CORRECT: MI is minute
```

TO_DATE

Converts character data to DATE format.

Syntax:

```sql
TO_DATE(c, format_model, nls_parms)
```

Example:

```sql
TO_DATE('2016-01-31', 'RRRR-MM-DD')
```

TO_TIMESTAMP

Converts string to TIMESTAMP (includes fractional seconds).

Syntax:

```sql
TO_TIMESTAMP(c, format_model, nls_parms)
```

Example:

```sql
TO_TIMESTAMP('2020-JAN-01 13:34:00.093423', 'RRRR-MON-DD HH24:MI:SS:FF')
```

TO_TIMESTAMP_TZ

Converts string to TIMESTAMP WITH TIME ZONE.

Syntax:

```sql
TO_TIMESTAMP_TZ(c, format_model, nls_parms)
```

Example:

```sql
TO_TIMESTAMP_TZ('17-04-2016 16:45:30', 'DD-MM-RRRR HH24:MI:SS')
```

TO_YMINTERVAL

Converts to INTERVAL YEAR TO MONTH.

Syntax:

```sql
TO_YMINTERVAL('y-m')
```

Example:

```sql
TO_YMINTERVAL('04-06')  -- Returns: 4 years 6 months
```

TO_DSINTERVAL

Converts string to INTERVAL DAY TO SECOND.

Syntax:

```sql
TO_DSINTERVAL('sql_format', nls_parms)
```

Example:

```sql
TO_DSINTERVAL('40 08:30:00.03225')
```

NUMTOYMINTERVAL

Converts number to INTERVAL YEAR TO MONTH.

Syntax:

```sql
NUMTOYMINTERVAL(n, 'YEAR'|'MONTH')
```

Example:

```sql
NUMTOYMINTERVAL(27, 'MONTH')  -- Returns: 2 years 3 months
```

NUMTODSINTERVAL

Converts number to INTERVAL DAY TO SECOND.

Syntax:

```sql
NUMTODSINTERVAL(n, 'DAY'|'HOUR'|'MINUTE'|'SECOND')
```

Example:

```sql
NUMTODSINTERVAL(36, 'HOUR')  -- Returns: 1 day 12 hours
```

CAST Function

Converts data type using ANSI SQL syntax.

Syntax:

```sql
CAST(expression AS data_type)
```

Example:

```sql
CAST('19-JAN-16 11:35:30' AS TIMESTAMP WITH LOCAL TIME ZONE)

-- With nested format:
CAST(TO_TIMESTAMP('19-JAN-16 14:35:30', 'DD-MON-RR HH24:MI:SS')
     AS TIMESTAMP WITH LOCAL TIME ZONE)
```

Pro Tip: Nesting Conversion Functions

You can nest conversion functions for complex transformations:

```sql
TO_CHAR(TO_DATE('04-JUL-1776'), 'fmDay, Month "the" Ddthsp, Year')
```

Conditional Expressions in SELECT Statements

Oracle SQL doesn't have an IF statement, but uses conditional expressions:

1. CASE Expression

Syntax:

```sql
CASE expression
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE result_final
END
```

- Compares expression to each condition. Returns the matching result
- ELSE is optional. If no match and no ELSE, returns NULL
- Can use constants, columns, or expressions

Example:

```sql
SELECT CASE 'option1'
           WHEN 'option1' THEN 'found it'
           WHEN 'option2' THEN 'did not find it'
       END AS Answer
FROM DUAL;
```

2. DECODE Function

Syntax:

```sql
DECODE(e, search1, result1, search2, result2, ..., default)
```

- Compares `e` to each search value (search1, search2, etc.)
- Returns the corresponding result if matched
- If no match: returns default (optional)
- If no default: returns NULL

NULL Behavior:

- `DECODE(NULL, NULL, result)` returns result
- `= NULL` is always false, but DECODE treats two NULLs as equal

Example:

```sql
SELECT STATE,
       DECODE(STATE, 'CA', 'California', 'IL', 'Illinois', 'Other') AS DECODED_STATE
FROM ADDRESSES;
```

3. NVL Function

Syntax:

```sql
NVL(e1, e2)
```

- If `e1` is NULL, returns `e2`. Otherwise, returns `e1`

- `e1` and `e2` must be same or convertible data types
- Useful in calculations where NULL would break logic

Example:

```sql
SELECT NVL(NULL, 0) FROM DUAL;  -- returns 0

SELECT SQ_FT + NVL(BALCONY_SQ_FT, 0)
FROM SHIP_CABINS;
```

4. NULLIF Function

Syntax:

```sql
NULLIF(e1, e2)
```

- If `e1 = e2`, returns NULL
- Otherwise, returns `e1`

Use Case: Filter out values that haven't changed or are duplicates

Example:

```sql
SELECT TEST_SCORE,
       UPDATED_TEST_SCORE,
       NULLIF(UPDATED_TEST_SCORE, TEST_SCORE) AS REVISION_ONLY
FROM SCORES;
```