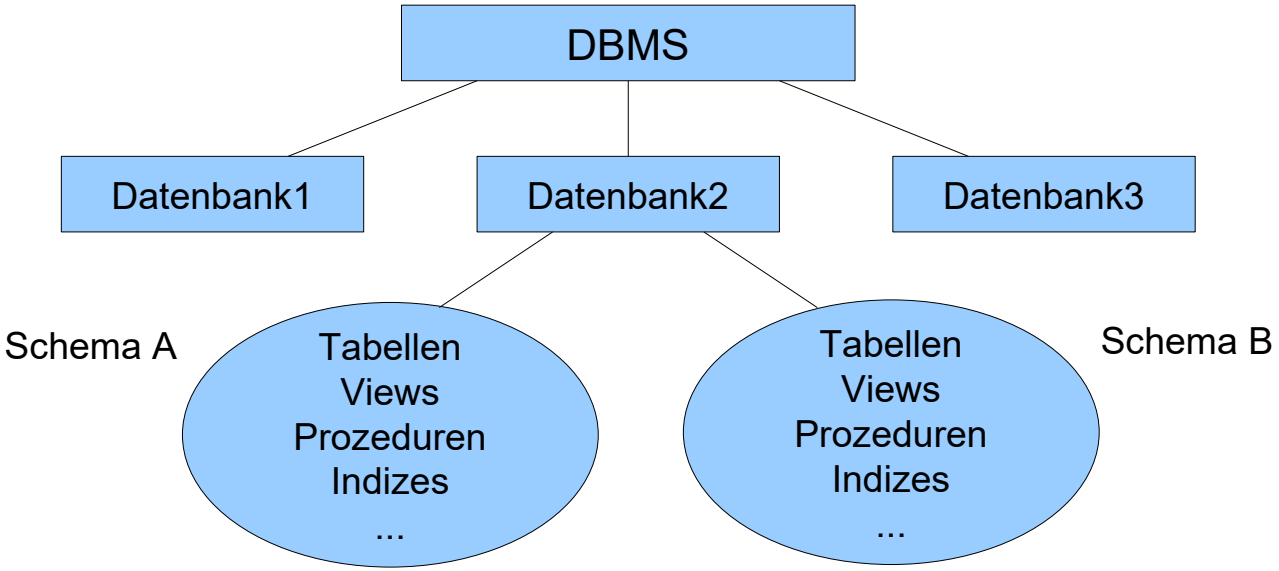
Snowflake – Einfache Abfragen

Stephan Karrer

Schemata fassen Datenbankobjekte zu logischen Gruppen zusammen



- Entspricht einem Verzeichnis im Dateisystem, allerdings in der Regel ohne Schachtelung (so auch bei Snowflake).
- Die Hersteller setzen allerdings Schemata durchaus unterschiedlich um.
- Die jeweilige Umsetzung hat nur grob etwas mit dem Schema-Begriff des Datenbankentwurfs zu tun.

Datenbanken und Namensauflösung bei Snowflake

- Adressierung eines Schema-gebundenen Objekts, z.B. einer Tabelle erfolgt via Datenbank-Name.Schema-Name.Tabellen-Name, z.B:

```
SELECT   *   FROM  TEST.HR.EMPLOYEES
```

- Ein Benutzer kann mehrere Datenbanken und Schemata besitzen und anderen Nutzern den Zugriff auf die darin enthaltenen Datenbankobjekte erteilen.

- Wird kein Datenbank-Name zur Qualifizierung verwendet, wählt Snowflake die jeweils aktuelle Datenbank (abhängig von den Sitzungs- und Konto-Einstellungen). Diese kann auch abgefragt bzw. gesetzt werden:

```
SELECT CURRENT_DATABASE();
USE DATABASE test;
```

- Das Erzeugen einer neuen Datenbank (CREATE DATABASE) setzt diese als aktuelle Datenbank.

Schemata und Namensauflösung bei Snowflake

- Es existiert ein PUBLIC Schema (in der Regel default bei DML und DDL-Anweisungen, wenn kein anderes Schema adressiert wird).

- Wird kein Schema-Name zur Qualifizierung verwendet, wählt Snowflake das jeweils aktuelle Schema (abhängig von den Sitzungs- und Konto-Einstellungen). Dieses kann auch abgefragt bzw. gesetzt werden:

```
SHOW SCHEMAS;
SELECT CURRENT_SCHEMA();
USE SCHEMA hr;
```

- Wird kein Schema-Name zur Qualifizierung verwendet, sucht Snowflake die Tabelle bzw. das Datenbankobjekt anhand eines Suchpfads, der standardmäßig nur das aktuelle und das PUBLIC Schema berücksichtigt.

```
SELECT current_schemas();
```

- Wir können diesen aber anpassen, z.B:

```
ALTER SESSION SET search_path='$current, testdb.public';
SHOW PARAMETERS LIKE 'search_path';
```

Abfragen mit SQL: SELECT-Anweisung

```
SELECT [ALL|DISTINCT] Auswahlliste
       FROM Quelle
       [WHERE Where-Klausel]
       [ORDER BY (Sortierungsattribut)[ASC|DESC]]
```

```
SELECT * FROM employees;
SELECT last_name, job_id, salary, department_id
       FROM employees;
SELECT first_name AS "Vorname", last_name "Nach""name"
       FROM employees;
```

- Generell gilt: SQL ist nicht Case-Sensitiv. Schlüsselworte (wie SELECT, FROM, …) und nicht-maskierte (unquoted) Namen können beliebig groß/klein geschrieben werden.
- Wollen wir Namen case-sensitiv bzw. in den Namen nicht-konforme Zeichen verwenden, können wir diese maskieren.
- Die max. Namenslänge ist bei Snowflake standardmäßig 255 Zeichen.
- Generell sind Spalten-Aliase für die angelieferten Spalten möglich.

SELECT-Anweisung mit Sortierung

```
SELECT [ALL|DISTINCT] Auswahlliste
       FROM Quelle
       [WHERE Where-Klausel]
       [ORDER BY (Sortierungsattribut)[ASC|DESC]]
```

```
SELECT last_name, job_id, department_id  FROM employees
       ORDER BY department_id NULLS FIRST, last_name DESC;

SELECT last_name, job_id  FROM employees
       ORDER BY department_id, last_name DESC;

SELECT DISTINCT job_id  FROM employees;
```

- Es sind durchaus mehrere Sortier-Kriterien mit expliziter Angabe absteigend/aufsteigend und Sortierung der Null-Werte möglich.
- Die Reihenfolge der Ausgabespalten muss nicht der Reihenfolge der Sortierkriterien entsprechen bzw. diese überhaupt enthalten.
- Vorsicht! Sortierung kostet Performance, also nicht unnötig sortieren
- DISTINCT bedingt in der Regel intern Sortierung!

Abfragen mit SQL: SELECT-Anweisung mit Ausdrücken

```
SELECT last_name, salary, 12*(salary+100) AS        new_annual_sal
        FROM employees;

SELECT first_name || last_name || ' is a ' || job_id AS "Is A"
        FROM employees;

SELECT last_name "employees in department 50"
        FROM employees WHERE department_id = 50;

SELECT DISTINCT job_id FROM employees WHERE salary <= 5000;

-- reine Berechnung
SELECT (2+8)/5 "Ergebnis";
```

- Überall dort, wo ein Wert erwartet wird, darf auch ein Ausdruck stehen.
- Natürlich hängt die Funktionalität vom jeweiligen Datentyp ab.
- Runde Klammern regeln wie üblich Ausführungsreihenfolge.
- FROM-Klausel ist bei reinen Berechnungen obsolet!

Snowflake Spezialitäten: SELECT-Anweisung

```
SELECT * ILIKE '%id%' FROM employees;

SELECT * EXCLUDE (department_id, employee_id) FROM employees;

SELECT * RENAME (department_id AS department, employee_id AS id)
    FROM employees;

SELECT * ILIKE '%id%' RENAME department_id AS department
    FROM employees;
```

- Mit ILIKE kann man ein Muster für die gewünschten Spalten angeben (Muster wie beim LIKE-Operator ohne Berücksichtigung Groß/Klein-Schreibung). Wird das Muster nicht getroffen gibt es Kompilierungsfehler.
- EXCLUDE schließt nicht gewünschte Spalten aus.
- Mit RENAME können eine oder mehrere Spalten Aliase bekommen.
- Die Klauseln können auch kombiniert eingesetzt werden.

ANSI SQL Basis-Datentypen

CHARACTER CHARACTER VARYING (or VARCHAR) CHARACTER LARGE OBJECT NCHAR NCHAR VARYING	NUMERIC DECIMAL SMALLINT INTEGER BIGINT FLOAT REAL DOUBLE PRECISION
BINARY BINARY VARYING BINARY LARGE OBJECT	DATE TIME TIMESTAMP INTERVAL
BOOLEAN	

Kaum ein Hersteller setzt das 1:1 um !
(Bei den komplexeren Typen: Object, XML, ... ist die Umsetzung noch unsicherer)
Siehe auch: https://en.wikibooks.org/wiki/SQL_Dialects_Reference

Zeichenketten als Datentyp

Name	Beschreibung
VARCHAR(n), STRING(n), TEXT(n)	Variable Länge mit Max. n (default und max 16.777.216 Bytes)
CHARACTER(n), CHAR(n)	Wie varchar mit Länge n, aber default 1

```
SELECT 'Max Muster';  -- z.B. für VARCHAR(15) oder TEXT
SELECT 'otto';            -- z.B. für CHAR(4)
SELECT 'Mother''s Day';   -- Escape
SELECT 'Hello\tWorld', 'C:\\user', '-\u26c4-'   -- Escape erweitert
```

- VARCHAR ist der native Snowflake-Typ, die anderen existieren zwecks ANSI-und Hersteller-Konformität (z.Bsp. auch VARCHAR2, NVARCHAR, …).
- Snowflake weicht von der üblichen CHAR-Semantik dadurch ab, dass Zeichenfolgen, die kürzer als die maximale Länge sind, am Ende nicht mit Leerzeichen aufgefüllt werden !
- CHAR bringt also bei Snowflake keinen Performance-Gewinn !
- Länge wird in Bytes gemessen, da aber intern Unicode (je Zeichen 2-4 Byte) verwendet wird, können es weniger Zeichen sein.

Operationen auf Zeichenketten: Konkatenation

Operator	Beschreibung	Beispiel
`||`	Konkatenation von Zeichenketten und CLOB-Daten	`SELECT 'Name is ' || last_name FROM employees;`

```
CREATE TABLE tab1 (col1 VARCHAR2(6), col2 CHAR(6),
                   col3 VARCHAR2(6), col4 CHAR(6) );

INSERT INTO tab1 (col1, col2, col3, col4)
        VALUES ('abc', 'def ', 'ghi ', 'jkl');

SELECT col1 || col2 || col3 || col4 "Concatenation"
        FROM tab1;
```

Concatenation
----
abcdef ghi jkl

- Bei Snowflake werden bei Zeichenkettenoperationen Leer- und Sonder-Zeichen berücksichtigt !

Fixpunktzahlen als Datentyp

Name	Bereich
NUMBER(p,s)	p: Gesamtzahl Stellen (38 max/default) s: Anzahl Nachkommastellen (0 default)
DECIMAL , DEC , NUMERIC	Alias für NUMBER

```
SELECT 3.5 + .001 AS erg;       -- liefert 3.501
SELECT 5e2 - 1.1e-3 AS erg;     -- liefert 499.9989
SELECT 2 * 3.5 AS erg;          -- liefert 7.0
SELECT 7.0 / 2.0 AS erg;        -- liefert 3.500000
SELECT 7.0 % 3 AS erg;             -- Modulo-Operator: liefert 1
```

- Wissenschaftliche und numerische Schreibweise können beliebig gemischt werden.
- Die Gesamtzahl der Stellen wird Precision genannt, die Anzahl Nachkommastellen Scale.
- Die Gesamtzahl hat bei Snowflake nur begrenzten Einfluß auf die Speicheranforderung (da für jede Mikropartition Snowflake die Mindest- und Höchstwerte für eine bestimmte Spalte festlegt).

Ganzzahlen als Datentyp

Name	Bereich
INT , INTEGER , BIGINT , SMALLINT , TINYINT , BYTEINT	entspricht stets NUMBER(38, 0) ! -99999999999999999999999999999999999999 bis +99999999999999999999999999999999999999

```
SELECT -1 AS erg;      -- liefert -1
SELECT 2*2 AS erg;     -- liefert 4
SELECT 2+3 AS erg;     -- liefert 5
SELECT 7-3 AS erg;     -- liefert 4
SELECT 7/3 AS erg;     -- Keine Ganzzahl-Division: liefert 2.333333
SELECT 7%3 AS erg;     -- Modulo-Operator: liefert 1
```

- Ganzzahlen werden als Spezialfall von NUMBER betrachtet.
- Es stehen die üblichen arithmetischen Operatoren zur Verfügung.

IEEE Gleitpunktzahlen als Datentyp

Name	Speichergröße	Bereich
FLOAT	8 Bytes	ca. 1E-307 bis 1E+308 mit einer Präzision von 15 Dezimalstellen (IEEE-Format)
FLOAT4, FLOAT8, DOUBLE , REAL, DOUBLE PRECISION	Alias für FLOAT	

- Das IEEE-Format ist intern ein Binärformat und nicht für kaufmännische Berechnungen geeignet !!
- Schreibweisen und Operatoren wie bei NUMBER.

ANSI Datentypen: Datum und verschiedene Zeitstempel

ANSI	Beschreibung
DATE	nur Datum
TIME	nur Zeit
TIMESTAMP	Datum und Zeit ohne Zeitzone
TIME WITH TIME ZONE	Zeit mit Zeitzone
TIMESTAMP WITH TIME ZONE	Datum und Zeit mit Zeitzone
INTERVAL DAY TO SECOND(n)	Zeitintervall in Stunden, Minuten und Sekunden(-bruchteilen)
INTERVAL YEAR TO MONTH	Zeitintervall in Jahren und Monaten

Datum und Zeit in Snowflake

Name	Bereich	Auflösung
DATE	von 1582 bis 9999 (empfohlen)	Tag
TIME [(p)]	von 00:00:00 bis 23:59:59.999999999	Nanosekunde (p=9 default)
TIMESTAMP_LTZ [(p)]	Kombination von DATE und TIME mit lokaler Zeitzone	Nanosekunde
TIMESTAMP_NTZ [(p)] (DATETIME [(p)])	Kombination von DATE und TIME ohne Zeitzone	Nanosekunde
TIMESTAMP_TZ [(p)]	Kombination von DATE und TIME mit explizitem Zeitzonen-Offset	Nanosekunde

- Alle Zeiten beziehen sich auf UTC Zeit bzw. den Gregorianischen Kalender.
- TIMESTAMP ist ein Alias für eine der TIMESTAMP-Varianten.
  Gesteuert via TIMESTAMP_TYPE_MAPPING Session-Parameter.
  Default ist TIMESTAMP_NTZ.
- Vorsicht es findet keine automatische Aktualisierung der Zeitzonen-Info statt, da nur der Offset einmalig bei Erstellung des Werts gespeichert wird.
- Sommer/Winter-Zeit wird nicht berücksichtigt.

Datum und Zeit: Eingabe-Beispiele

```
SELECT  TIMESTAMP '2003-06-17';
SELECT  TIMESTAMP '2003-06-17T13:45:30';
SELECT  TIMESTAMP '2003-06-17T13:45:30.67';
SELECT  TIMESTAMP
        '1999-01-08 13:05:06 -8:00';
```

```
SELECT  DATE '2003-06-17';
SELECT  DATE '1/8/2023';
SELECT  DATE '08-Jan-1999';

SELECT  TIME '13:05:06';
SELECT  TIME '13:05';
SELECT  TIME '13:05:06.45';
SELECT  TIME
      '07:57:01.123456789 AM';
```

- Snowflake unterstützt einige Datums- bzw. Zeit-Formate direkt (AutoDetection).
- Die ISO 8601 – Formate sind am portabelsten (fett im Listing).

Datum und Zeit: Explizites Format vorgeben

Eingabe-Formate	Ausgabe-Formate
DATE_INPUT_FORMAT TIME_INPUT_FORMAT TIMESTAMP_INPUT_FORMAT	DATE_OUTPUT_FORMAT TIME_OUTPUT_FORMAT TIMESTAMP_OUTPUT_FORMAT TIMESTAMP_LTZ_OUTPUT_FORMAT TIMESTAMP_NTZ_OUTPUT_FORMAT TIMESTAMP_TZ_OUTPUT_FORMAT

```
SELECT TO_DATE('01:1994:JAN', 'dd:yyyy:mon');
SELECT TO_TIME('22/12/01', 'mi/hh24/ss');
SELECT TO_TIMESTAMP('2020.02.24 04:00:00', 'YYYY.MM.DD HH:MI:SS');

ALTER SESSION SET DATE_INPUT_FORMAT = 'dd:yyyy:mon';  -- für die Session
```

- Bei expliziter Angabe der Format-Spezifikation sind wir auf der sicheren Seite !

Datum und Zeit: Direkte Arithmetik

- Wie bei den meisten DBMS können auch bei Snowflake direkt Zeiteinheiten, aber nur mit Hilfe des Schlüsselworts INTERVAL, addiert bzw. subtrahiert werden.

```
SELECT TO_DATE ('2019-02-28') + INTERVAL '1 day, 1 year';  -- 2020-03-01

SELECT TO_DATE ('2024-02-29') + INTERVAL '1 year';     -- 2025-02-28

SELECT TO_TIME('04:15:29') + INTERVAL '3 hours, 18 minutes';  -- 07:33:29

SELECT TO_DATE('2025-01-17')
    + INTERVAL '1 y, 3 q, 4 mm, 5 w, 6 d, 7 h, 9 m, 8 s,1000 ms,
                445343232 us, 898498273498 ns'
    as complex_interval;                          -- 2027-03-30 07:31:32.841

SELECT last_name, hire_date from employees where
        hire_date > current_date - INTERVAL '17 y, 3 month';
```

Vergleichsoperatoren für alle Datentypen mit Ordnung (Zeichenketten, Zahlen, Datum und Zeit)

=	gleich
<>, !=	ungleich
>	größer
>=	größer oder gleich
<	kleiner
<=	kleiner oder gleich

```
SELECT * FROM employees
       WHERE salary = 2500;

SELECT * FROM employees
       WHERE salary != 2500;

SELECT * FROM employees
       WHERE salary > 2500;
```

Wahrheitswerte

Name	Bereich
BOOLEAN	TRUE, FALSE, NULL

```
SELECT  TRUE OR FALSE;
SELECT  TRUE AND FALSE;
SELECT  NOT TRUE;
SELECT  *  FROM employees WHERE TRUE;
SELECT 1=1;
SELECT 1=1 = TRUE;
SELECT 1=1  = (NOT FALSE);
SELECT  'TRUE' OR 'FALSE';   -- impliziter CAST
SELECT  10 OR 0;   -- impliziter CAST
SELECT 'WAHR ist ' || TRUE AS "String";
```

- Es existiert impliziter Cast von String- bzw. Zahlenwerten auf BOOLEAN:
  - Bei Zeichenketten: 'true' oder 'false' (Nicht Case-sensitiv)
  - Bei Zahlen: 0 wird FALSE, jeder andere Wert wird TRUE
- und umgekehrt bei Zeichenketten.

Logik-Operatoren

- Logische Verküpfungsoperatoren können auf logische Ausdrücke angewendet werden.

Operator	Kommentar
AND, OR, NOT	Basisoperatoren

```
SELECT * FROM employees
    WHERE NOT (job_id IS NULL)
    ORDER BY employee_id;

SELECT * FROM employees
    WHERE job_id = 'PU_CLERK' AND department_id = 30;
```

Spezielle Vergleichsoperatoren

Operator	Kommentar
BETWEEN	Prüft, ob der Operand im Intervall liegt
IN	Prüft, ob der Operand in der Aufzählung enthalten ist
LIKE	Prüft, ob der Operand einem Muster gleicht

```
SELECT * FROM employees
        WHERE salary BETWEEN 5000 AND 10000;

SELECT * FROM employees
        WHERE job_id IN ('SA_MAN', 'SA_REP');

SELECT * FROM employees
    WHERE (first_name, last_name, email) IN
        (('Guy', 'Himuro', 'GHIMURO'),
         ('Karen', 'Colmenares', 'KCOLMENA'));
```

- Tupelvergleiche sind verfügbar.

LIKE-Operator für Vergleiche

```
<subject> [NOT] LIKE <pattern> [ESCAPE <escape>]
```

Zur Bildung von Mustern können verwendet werden:

- % beliebig viele Zeichen (auch keines)
- _ genau ein Zeichen

```
SELECT salary
   FROM employees
   WHERE last_name LIKE 'R%';

SELECT last_name
   FROM employees
   WHERE last_name LIKE '%A\\_B%' ESCAPE '\\';
```

- LIKE ist rudimentär, deshalb bieten viele DBMS wie auch Snowflake Erweiterungen und Unterstützung von regulären Ausdrücken an.
- '\\' ist nötig, da der Backslash Standard-Escape-Zeichen bei Snowflake ist.

Erweiterungen des LIKE-Operator

```
SELECT *
  FROM employees
  WHERE first_name LIKE ALL ('%Jo%','J%n')
  ORDER BY first_name;

SELECT *
  FROM employees
  WHERE first_name LIKE ANY ('%Jo%','J%n')
  ORDER BY first_name;
```

- Bei ALL muüssen alle Muster erfüllt sein, bei ANY irgendeines.

Explizite Typkonvertierung (Cast)

Für den expliziten Cast stehen 3 Optionen zur Verfügung:

- Die CAST-Funktion
- Der CAST-Operator ::
- Die entsprechenden SQL-Funktionen zur Typkonvertierung, z.Bsp. TO_NUMBER, TO_DATE, ...

```
SELECT CAST('2022-04-01' AS DATE);

SELECT '2022-04-01'::DATE;

SELECT TO_DATE('2022-04-01');
```

Nullwerte (Null Values)

- Nullwerte stehen für nicht verfügbare bzw. unbekannte Werte und können in Tabellen als Werte von Zeilen vorkommen
- Werte können explizit auf NULL gesetzt bzw. daraufhin überprüft werden
- Ist ein Operand in arithmetischen Ausdrücken ein Nullwert, so ergibt die Auswertung stets NULL.
- Vergleiche mit Nullwerten liefern stets NULL (außer die speziellen Tests auf Nullwerte)
- Bei der Auswertung logischer Ausdrücke wird durch Nullwerte die Prädikatenlogik erweitert.

Prüfung auf NULL-Wert

- Die blau eingefärbte Variante ist Snowflake-spezifisch.

Operator	Kommentar
IS NULL	Prüft, ob der Operand ein NULL-Wert ist
IS NOT NULL	Prüft, ob der Operand kein NULL-Wert ist
EQUAL_NULL(<expr1> , <expr2>)	Prüft auf Gleichheit, auch bei NULL-Werten

```
SELECT last_name
    FROM employees
    WHERE commission_pct IS NULL
    ORDER BY last_name;

SELECT EQUAL_NULL(null, 1 = 2);
```

SQL-Funktionen: Numerik (Auszug)

Funktion	Beschreibung
ABS(<num_expr>)	Absolutbetrag
CEIL(<input_expr> [, <scale_expr>])	Nächstgrößere Zahl (Scale für Nachkommastellen)
FLOOR(<input_expr> [, <scale_expr>])	Nächstkleinere Zahl (Scale für Nachkommastellen)
ROUND(<input_expr> [, <scale_expr> [, <rounding_mode>]])	Runden auf n Stellen
TRUNC(<input_expr> [, <scale_expr>])	Abschneiden auf n Stellen
MOD(<expr1> , <expr2>)	Rest-Operation (Modulo)
SQRT(<expr>)	Quadratwurzel
SQUARE(<expr>)	Quadrat
POW(<expr>, <n>)	n-te Potenz von zahl
...	und viele weitere (siehe Doku)

SQL-Funktionen: Zeichenketten (Auszug)

Funktion	Beschreibung
LOWER(<expr>)	Konvertierung zu Kleinbuchstaben
UPPER(<expr>)	Konvertierung zu Großbuchstaben
LEN[GTH](<expr>)	Länge der Zeichenkette
SUBSTR[ING] (<base_expr>, <start_expr> [, <length_expr>])	Teilzeichenkette von Position start mit Länge length
LEFT(<string_expr> , <length_expr>)	Linke Teilzeichenkette der Länge n
RIGHT(<string_expr> , <length_expr>)	Rechte Teilzeichenkette der Länge n
INITCAP(<expr> [, <delimiters>])	1. Buchstabe eines jeden Worts groß, Rest klein
LPAD / RPAD(<base>, <length_expr> [, <pad>])	Links bzw. rechts auffüllen auf Länge n mit Zeichenkette pad (default Leerzeichen)
LTRIM / RTRIM(<expr> [, <characters>])	Entfernen führender/folgender Zeichen
...	und viele weitere (siehe Doku)

SQL-Funktionen: Zeitstempel (Auszug)

Funktion	Beschreibung
CURRENT_DATE CURRENT_TIME [(precision)] CURRENT_TIMESTAMP[(precision)]	Aktuelles Datum bzw. Zeit zur lokalen Zeitzone, optionale precision reduziert die Sekundenbruchteile
LOCALTIME [(precision)] LOCALTIMESTAMP [(precision)]	Alias für CURRENT_TIME bzw. CURRENT_TIMESTAMP
EXTRACT(<field> FROM <source>) DATE_PART (<field>, <source>)	Extrahiert Datums bzw. Zeitanteil, Vielzahl von field-Parametern möglich
DATE_TRUNC (<field>, <source>)	Reduziert auf die durch field angebene Genauigkeit
DATEADD (<part>, <value>, <source>)	Addiert Zeit/Datumswerte zum vorhanden Wert
DATEDIFF (<part>, <value>, <source>)	Subtrahiert Zeit/Datumswerte zum vorhanden Wert
...	und viele weitere (siehe Doku)

SQL-Funktionen: Konvertierung von Datentypen

Von	Zu	Funktion
TIMESTAMP, TIME, DATE	VARCHAR	TO_CHAR (<source>, <format>)
NUMERIC	VARCHAR	TO_CHAR (<source>, <format>)
VARCHAR	DATE	TO_DATE (<source>, <format>)
VARCHAR	TIMESTAMP	TO_TIMESTAMP (<source>, <format>)
VARCHAR	TIME	TO_TIME (<source>, <format>)
VARCHAR	NUMERIC	TO_NUMBER (<source>, <format>)
...	und weitere (siehe Dokumentation)	

```
SELECT TO_CHAR(hire_date, 'DD-MM-YYYY') FROM employees;
SELECT TO_CHAR(hire_date, 'DD-Mon-YYYY hh24-mi-ss') FROM employees;
SELECT TO_DATE('2023-17-08', 'YYYY-DD-MM');
SELECT TO_CHAR(salary, '000G000D00L') FROM employees;
SELECT TO_NUMBER('-12,454.8', '99G999D9', 8, 2);
```

Funktionen für NULL-Behandlung

Funktion	Beschreibung
IFNULL(<expr1> , <expr2>) NVL(<expr1> , <expr2>)	Liefert den 2.Wert, falls der 1.Wert NULL ist, sonst den 1.Wert
NVL2(<expr1> , <expr2> , <expr3>)	Liefert den 2.Wert, falls der erste nicht NULL ist, sonst den 3. Wert
COALESCE(<expr1> , <expr2> [, ...])	Liefert den ersten Wert, der nicht NULL ist, sofern alle Werte NULL sind ist das Ergenis NULL
NULLIF(<expr1> , <expr2>)	Liefert NULL, falls expr1 = expr2, ansonsten den Wert der expr1
NULLIFZERO(<expr>)	Liefert NULL, falls der Ausdruck 0 liefert, ansonsten den numerischen Wert (NUMBER)
ZEROIFNULL(<expr>)	Liefert 0, falls der Ausdruck NULL liefert, ansonsten den numerischen Wert (NUMBER)

- Die Werte müssen einen gemeinsamen Typ haben !

Größter und Kleinster Wert

Funktion	Beschreibung
GREATEST(<expr1> [, <expr2> ...])	Liefert den größten Wert aus einer Menge
GREATEST_IGNORE_NULLS(<expr1> , [...])	Liefert den größten Wert aus einer Menge, ignoriert NULL-Werte
LEAST(<expr1> [, <expr2> ...])	Liefert den kleinsten Wert aus einer Menge
LEAST_IGNORE_NULLS (<expr1> , [...])	Liefert den kleinsten Wert aus einerMenge, ignoriert NULL-Werte

```
SELECT GREATEST(2*5, 17-8, 3*4);
SELECT LEAST(2*5, 17-8, 3*4);
SELECT GREATEST_IGNORE_NULLS(2*5, 17-8, NULL, 3*4);
SELECT LEAST_IGNORE_NULLS(2*5, 17-8, NULL, 3*4);
```

- Die Werte müssen einen gemeinsamen Typ haben !

CASE – Ausdruck

```
SELECT  last_name,
        CASE salary
            WHEN 2000 THEN 'Low'
             WHEN 5000 THEN 'High'
             ELSE 'Medium'
        END AS sal
FROM employees;
```

- Der ELSE-Zweig ist optional, nicht getroffene Werte werden nicht ersetzt.
- Auch hier gilt: Die einzelnen Zweige müssen einen gemeinsamen Typ liefern !
- Wurde in der Vergangenheit oft für das Aufbereiten der Ausgabe im Sinne von Reporting benutzt.

DECODE-Funktion statt CASE

```
SELECT last_name,
       DECODE( salary,
               2000, 'Low',   -- <search>, <result>
               5000, 'High',  -- <search>, <result>
               NULL, 'Null-Value',
               'Medium' )    -- default
        AS sal
FROM employees;
```

- Der Default ist optional, nicht getroffene Werte werden nicht ersetzt.
- Anders als bei CASE produzieren gemeinsame NULL-Werte einen Treffer.
- Auch hier gilt: Die einzelnen Zweige müssen einen gemeinsamen Typ liefern !
- Wurde in der Vergangenheit oft für das Aufbereiten der Ausgabe im Sinne von Reporting benutzt.

Allgemeiner CASE - Ausdruck (Searched CASE)

```
SELECT  last_name,
        CASE  WHEN salary < 2000 THEN 'Low'
              WHEN salary > 5000 THEN 'High'
              ELSE 'Medium' END AS sal
FROM employees;
```

- Der ELSE-Zweig ist optional, nicht getroffene Werte werden nicht ersetzt.
- Auch hier gilt: Die einzelnen Zweige müssen einen gemeinsamen Typ liefern !
- Ist natürlich viele flexibler, da beliebige Bedingungen formulierbar sind.

Spezialfall: Simples Ersetzen von Werten in der Ausgabe

```
SELECT * REPLACE ('x' AS first_name) FROM employees;

SELECT * REPLACE ('DEPT-' || department_id AS department_id)
     FROM departments;


-- ist quasi Spezialfall der replace-Funktion:

SELECT last_name, replace(first_name, first_name, 'x')
FROM employees;
```

- Da Ersetzen ohne Bedingung, nur eingeschränkt verwendbar.

TOP-N mit Snowflake

```
SELECT TOP 10 * FROM employees ORDER BY job_id;

SELECT * FROM employees ORDER BY job_id LIMIT 7;

SELECT * FROM employees ORDER BY job_id LIMIT 2 OFFSET 10;

SELECT * FROM employees ORDER BY job_id FETCH FIRST 10 ROWS;

SELECT * FROM employees ORDER BY job_id
                        OFFSET 6 FETCH FIRST 2 ROWS;
```

- Alle Varianten schneiden einfach ab. Das ist nicht so ganz ok, falls es mehrere gleiche Ergebnisse auf der letzten Position gibt. (WITH TIES wird nicht unterstützt)
- Nur via Zeilenzahl begrenzbar, nicht via Prozent-Angabe.