Using binary data

The usefulness and flexibility of the BINARY data type is best demonstrated by example. This topic provides practical examples of tasks that involve the BINARY data type and its three encoding schemes.

Converting between hex and base64

The BINARY data type can be used as an intermediate step when converting between hex and base64 strings.

Convert from hex to base64 using TO_CHAR:

```
SELECT c1, TO_CHAR(TO_BINARY(c1, 'hex'), 'base64') FROM hex_strings;
```

```
+----------------------+-----------------------------------------+
| C1                   | TO_CHAR(TO_BINARY(C1, 'HEX'), 'BASE64') |
|----------------------+-----------------------------------------|
| df32ede209ed5a4e3c25 | 3zLt4gntWk48JQ==                        |
| AB4F3C421B           | q088Qhs=                                |
| 9324df2ecc54         | kyTfLsxU                                |
+----------------------+-----------------------------------------+
```

```
SELECT c1, TO_CHAR(TO_BINARY(c1, 'base64'), 'hex') FROM base64_strings;
```

```
+------------------+-----------------------------------------+
| C1               | TO_CHAR(TO_BINARY(C1, 'BASE64'), 'HEX') |
|------------------+-----------------------------------------|
| 3zLt4gntWk48JQ== | DF32EDE209ED5A4E3C25                    |
| q088Qhs=         | AB4F3C421B                              |
| kyTfLsxU         | 9324DF2ECC54                            |
+------------------+-----------------------------------------+
```

Converting between text and UTF-8 bytes

Strings in Snowflake are composed of Unicode characters, while binary values are composed of bytes. By converting a string to a binary value with the UTF-8 format, you can directly manipulate the bytes that make up the Unicode characters.

Convert single-character strings to their UTF-8 representation in bytes using TO_BINARY:

```
SELECT c1, TO_BINARY(c1, 'utf-8') FROM characters;
```

```
+----+-----------------------+
| C1 | TO_BINARY(C1, 'UTF-8') |
```

```
| π  | CF80                    |
+----+-------------------------+
```

Convert a UTF-8 byte sequence to a string using TO_CHAR , TO_VARCHAR:

```
SELECT TO_CHAR(X'41424320E29D84', 'utf-8');
```

```
+-------------------------------------+
| TO_CHAR(X'41424320E29D84', 'UTF-8') |
|-------------------------------------|
| ABC ❄                               |
+-------------------------------------+
```

Getting the MD5 digest in base64

Convert the binary MD5 digest to a base64 string using TO_CHAR:

```
SELECT TO_CHAR(MD5_BINARY(c1), 'base64') FROM variants;
```

```
+----------+-----------------------------------+
| C1       | TO_CHAR(MD5_BINARY(C1), 'BASE64') |
```

```
| "abcdef" | 6AtQFwmJUPxYqtg8jBSXjg==           |
| "côté"   | H6G3w1nEJsUY4Do1BFp2tw==           |
+----------+------------------------------------+
```

Convert to binary with variable format

Convert strings to binary values using a binary format extracted from the string. The statement includes the TRY_TO_BINARY and SPLIT_PART functions:

```
SELECT c1,
       TRY_TO_BINARY(SPLIT_PART(c1, ':', 2), SPLIT_PART(c1, ':', 1)) AS binary_value
  FROM strings;
```

```
+-------------------------+----------------------+
| C1                      | BINARY_VALUE         |
|-------------------------+----------------------|
| hex:AB4F3C421B          | AB4F3C421B           |
| base64:c25vd2ZsYWtlCg== | 736E6F77666C616B650A |
| utf8:côté               | 63C3B474C3A9         |
| ???:abc                 | NULL                 |
+-------------------------+----------------------+
```

Try multiple formats for the conversion:

```
          x'00' || TRY_TO_BINARY(c1, 'hex'),
          x'01' || TRY_TO_BINARY(c1, 'base64'),
          x'02' || TRY_TO_BINARY(c1, 'utf-8')) AS binary_value
  FROM strings;
```

```
+------------------+------------------------+
| C1               | BINARY_VALUE           |
|------------------+------------------------|
| ab4f3c421b       | 00AB4F3C421B           |
| c25vd2ZsYWtlCg== | 01736E6F77666C616B650A |
| côté             | 0263C3B474C3A9         |
| 1100             | 001100                 |
+------------------+------------------------+
```

Note

Since the above queries use TRY_TO_BINARY, the result is NULL if the format isn't recognized or if the string can't be parsed with the given format.

Convert the results from the previous example back to strings using SUBSTR and DECODE:

```
SELECT c1,
       TO_CHAR(
       SUBSTR(c1, 2),
       DECODE(SUBSTR(c1, 1, 1), x'00', 'hex', x'01', 'base64', x'02', 'utf-8')) AS string_value
  FROM bin;
```

```
|------------------------+------------------|
| 00AB4F3C421B           | AB4F3C421B       |
| 01736E6F77666C616B650A | c25vd2ZsYWtlCg== |
| 0263C3B474C3A9         | côté             |
| 001100                 | 1100             |
+------------------------+------------------+
```

Custom decoding with JavaScript UDF

The BINARY data type allows the storage of arbitrary data. Since JavaScript UDFs support the data type via `Uint8Array` (see Introduction to JavaScript UDFs), it is possible to implement custom decoding logic in JavaScript. This isn't the most efficient way to work, but it is very powerful.

Create a function that decodes based on the first byte:

```
CREATE OR REPLACE FUNCTION my_decoder (b BINARY)
  RETURNS VARIANT
  LANGUAGE JAVASCRIPT
AS '
  IF (B[0] == 0) {
      var number = 0;
      FOR (var i = 1; i < B.length; i++) {
          number = number * 256 + B[i];
      }
      RETURN number;
  }
  IF (B[0] == 1) {
```

```
        }
        RETURN str;
    }
    RETURN NULL;';


SELECT c1, my_decoder(c1) FROM bin;
```

```
+----------------+----------------+
| C1             | MY_DECODER(C1) |
|----------------+----------------|
| 002A           | 42             |
| 0148656C6C6F21 | "Hello!"       |
| 00FFFF         | 65535          |
| 020B1701       | null           |
+----------------+----------------+
```