MySQL 개발자 가이드

목차

[1 데이터 타입 변환 5](#_Toc14388400)

[2 SQL 관련 8](#_Toc14388401)

[2-1 SELECT 8](#_Toc14388402)

[2-2 UPDATE 8](#_Toc14388403)

[2-3 SUBQUERY 8](#_Toc14388404)

[2-3-1 IN대신 EXISTS를 활용 8](#_Toc14388405)

[2-3-2 NOT IN 대신 NOT EXISTS 활용 8](#_Toc14388406)

[2-3-3 DISTINCT 대신 EXISTS를 활용 8](#_Toc14388407)

[2-3-4 SUBQUERY 내에서는 가급적 ORDER BY 구문을 사용하지 않는다 8](#_Toc14388408)

[2-3-5 SUBQUERY 또는 INLINE VIEW 사용시, 가능하면 1레벨 까지만 사용한다. 8](#_Toc14388409)

[2-4 OUTER JOIN 8](#_Toc14388410)

[2-5 GROUP 함수 9](#_Toc14388411)

[2-6 DECODE 대신 CASE 구문 사용 9](#_Toc14388412)

[2-7 OR 사용제한 9](#_Toc14388413)

[2-8 MERGE 구문 대체 9](#_Toc14388414)

[2-9 UNION vs. UNION ALL 9](#_Toc14388415)

[2-10 index 9](#_Toc14388416)

[2-10-1 index 설계절차 9](#_Toc14388417)

[2-10-2 index 컬럼 적용시 고려 사항 9](#_Toc14388418)

[2-10-3 index가 적용되지 않는 경우 9](#_Toc14388419)

[3 옵티마이저 관련 10](#_Toc14388420)

[3-1 실행 계획 분석 10](#_Toc14388421)

[3-1-1 실행계획 기본정보 10](#_Toc14388422)

[3-1-2 실행계획 내용(explain 정보) 10](#_Toc14388423)

[3-1-3 분석시 주의 사항 10](#_Toc14388424)

[5 함수인덱스 변환 (generated column 이용) 10](#_Toc14388425)

[6 오라클 Job Scheduler 변환 (MariaDB Event Scheduler) 11](#_Toc14388426)

[6-1 Event Scheduler 생성 방법 12](#_Toc14388427)

[6-2 Event Scheduler 변경 방법 13](#_Toc14388428)

[6-3 Event Scheduler 삭제 방법 14](#_Toc14388429)

[6-4 Event Scheduler 모니터링 14](#_Toc14388430)

[7 profile 관련 변환 16](#_Toc14388431)

[8 Sequence 변환 17](#_Toc14388432)

[8-1 Oracle에서 MariaDB로의 sequence 이관 18](#_Toc14388433)

[8-2 Galera Cluster(또는 multi-mater 환경)에서의 sequence 설정 18](#_Toc14388434)

[8-3 alter sequence 관련 SR 진행 상황 19](#_Toc14388435)

[9 함수변환 20](#_Toc14388436)

[10 그외 Oracle to MariaDB 관련 주의할 점 28](#_Toc14388437)

[10.1 오라클과 MariaDB의 NULL 값 28](#_Toc14388438)

[10.2 후행공백(trailing space) 처리 28](#_Toc14388439)

[10.3 Galera cluster에서의 event scheduler 28](#_Toc14388440)

[10.4 Galera cluster에서의 sequence 28](#_Toc14388441)

[10.5 Galera cluster에서의 통계정보 29](#_Toc14388442)

# 1 데이터 타입 변환

|  |  |  |
| --- | --- | --- |
|  | Oracle | MySQL |
| CHARACTER | CHAR(n)  : 1 <= n <=255  n은 byte 와 character 선택 가능 | CHAR(n)  : 1 <= n <=255  n은 character 만 선택가능  (ex, char(2) character 2개를 의미) |
| CHAR(n)  : n 이 255 보다 큰 경우 varchar로 변환  256 <= n <= 2000 | VARCHAR(n)  : character set이 utf8 인 경우, 1 <= n <= 21844  character set이 latin1인 경우, 1 <= n <= 65532 |
| NCHAR(n)  : Fixed-length UTF-8 string, 1 <= n <= 255 | NCHAR(n) |
| NCHAR(n)  : Fixed-length UTF-8 string, 256 <= n <= 2000 | NVARCHAR(n) |
| NCHAR VARYING(n)  : Varying-length UTF-8 string, 1 <= n <= 4000 | NCHAR VARYING(n) |
| VARCHAR(n)  : Variable-length string, 1 <= n <= 4000 | VARCHAR(n)  : character set이 utf8 인 경우, 1 <= n <= 21844  character set이 latin1인 경우, 1 <= n <= 65532 |
| VARCHAR2(n)  : Variable-length string, 1 <= n <= 4000 | VARCHAR(n) |
| NVARCHAR2(n)  : Variable-length UTF-8 string, 1 <= n <= 4000 | NVARCHAR(n) |
| NUMBER | NUMBER(18,0) | BIGINT  : -9223372036854775808 ~ 9223372036854775807 |
| NUMBER(9,0) | INT, INTEGER  : -2147483648 ~ 2147483647 |
| NUMBER(6,0) | MEDIUMINT  : -8388608 ~ 8388607 |
| NUMBER(4,0) | SMALLINT  : -32768 ~ 32767 |
| NUMBER(2,0) | TINYINT  : -128 ~ 127 |
| NUMBER  : Default (38,0) | DECIMAL(38,0) |
| 그 외 NUMBER(p,s) | DECIMAL(p,s)  : p(s를 포함한 전체 자리수), s(소수점이하 자리수)  P의 max는 65, s의 max (10.2.1 이후) 38 |
| FLOAT(p)  : Floating-point number.  p는 이진수 자릿수.(1 <= p <= 126, default는 126)  ex) Float(126) = NUMBER(38) | DOUBLE |
| BINARY\_FLOAT  : 32-bit floating-point number | FLOAT |
| BINARY\_DOUBLE  : 64-bit floating-point number | DOUBLE |
| DECIMAL(p,s), DEC(p,s)  : Fixed-point number | DECIMAL(p,s), DEC(p,s) |
| DOUBLE PRECISION  : Floating-point number | DOUBLE PRECISION |
| INTEGER, INT  : 38 digits integer(NUMBER(38)과 동일) | DECIMAL(38) |
| LONG and RAW | LONG  : Character data, <= 2G | LONGTEXT |
| LONG RAW  : Binary data, <= 2G | LONGBLOB |
| RAW(n)  : Variable-length binary string, 1 <= n <= 255 | BINARY(n) |
| RAW(n)  : Variable-length binary string, 256 <= n <= 2000 | VARBINARY(n) |
| DATETIME | DATE  : Date and time | DATETIME |
| TIMESTAMP(p)  : Date and time with fraction | DATETIME(p) |
| TIMESTAMP(p) WITH TIME ZONE  : Date and time with fraction and time zone | DATETIME(p)  : 오라클 SYS\_EXTRACT\_UTC 함수를 이용하여 오라클에 있는 TZ정보를 가져오고 같은 테이블 혹은 다른 테이블에 저장 후 사용해야 함.  (아래 URL 참조  <http://www.sqlines.com/oracle-to-mysql/timestamp_with_time_zone> ) |
| INTERVAL YEAR(p) TO MONTH  : Date interval | VARCHAR(30) |
| INTERVAL DAY(p) TO SECOND(s)  : Day and time interval | VARCHAR(30) |
| LARGE OBJECT | BLOB  : Binary Large OBject, <= 4G | LONGBLOB |
| CLOB  : Character Large OBject, <= 4G | LONGTEXT |
| NCLOB  : Variable-length Unicode string, <= 4G | NVARCHAR(max)  : max는 National character set에 따라 달라짐.  또는  LONGTEXT  : utf8, utf8mb4의 경우 |
| BFILE  : Pointer to binary file, <= 4G | VARCHAR(255) |
| ROWID | ROWID  : Physical row address | CHAR(10) |
| UROWID(n)  : Logical row addresses, 1 <= n <= 4000 | VARCHAR(n) |
| XML | XMLTYPE  : XML data | LONGTEXT |
|  |  |  |

아래 URL 참조

<https://dev.mysql.com/doc/refman/8.0/en/other-vendor-data-types.html>

# 2 SQL 관련

## 2-1 SELECT

2-1-1 SELECT LIST 절에서 다음과 같은 것은 배제하도록 합니다.

1) SELECT “\*” 는 가능하면 사용하지 않는다. 불필요한 컬럼 들을 Sort, Hash 작업에서 Size가 큰 경우 불필요한 디스크/메모리 I/O가 발생하여 성능 저하의 원인이 되므로 반드시 필요한 Column들을 하나씩 기술합니다.

2) Distinct 는 항상 Sort 연산을 수행하므로, 처리 결과값이 항상 unique한 경우 DISTINCT 키워드를 사용하지 않는다.

3) UNION은 SORT를 수행하므로 가능하면 UNION 대신 SORT를 수행하지 않는 UNION ALL를 사용한다.

4) 꼭 필요한 부분이 아니면 IFNULL 함수를 사용하지 하지 않는다.

|  |
| --- |
| - NOT NULL 컬럼에 IFNULL 함수 사용은 불필요하다.  - 한 개 컬럼을 SUM하는 경우 NULL 값을 제외하고 SUM이 수행되므로 IFNULL 함수가 불필요하다.  SELECT SUM(IFNULL(COL1, 0)) ; => SELECT SUM(COL1) ;  - 여러 컬럼을 연산하는 경우 적절하게 IFNULL 함수를 사용한다.  SELECT SUM(IFNULL(COL1,0) + IFNULL(COL2, 0)) => SELECT IFNULL(SUM(COL1) + SUM(COL2), 0)  (왼쪽 Query의 경우 ROW수 만큼 IFNULL 함수 수행, 오른쪽의 경우 최종 결과에 대해 한번만 수행) |

2-1-2 SELECT WHERE 절에서 주의할 내용입니다.

1) LIKE 사용시 첫 문자는 '%'의 사용을 지양한다. (index 이용 불가)

2) 조건절에 있는 컬럼 중 인덱스를 구성하는 컬럼은 가공을 하지 않는다. (가공되는 경우 인덱스 이용불가)

|  |
| --- |
| - COL1 = CONCAT(COL2,’123’), CONCAT(COL1,COL2) = abc123, substr(COL1,1,4) = ‘AAAA’  첫번째 COL1을 이용한 조건은 인덱스를 이용하지만, 나머지 두 경우처럼 COL1이 함수를 이용하는 순간 인덱스를 이용하지 못한다. |

3) OR를 사용하여 Full Table Scan이 되는 경우, UNION ALL을 사용하여 SQL 구문을 분리하고 이를 통하여 인덱스 사용을 유도한다.

2-1-3 SUBQUERY 관련 주의할 내용입니다.

## 2-2 UPDATE

## 2-3 SUBQUERY

### 2-3-1 IN대신 EXISTS를 활용

### 2-3-2 NOT IN 대신 NOT EXISTS 활용

### 2-3-3 DISTINCT 대신 EXISTS를 활용

### 2-3-4 SUBQUERY 내에서는 가급적 ORDER BY 구문을 사용하지 않는다

### 2-3-5 SUBQUERY 또는 INLINE VIEW 사용시, 가능하면 1레벨 까지만 사용한다.

## 2-4 OUTER JOIN

## 2-5 GROUP 함수

## 2-6 DECODE 대신 CASE 구문 사용

## 2-7 OR 사용제한

## 2-8 MERGE 구문 대체

## 2-9 UNION vs. UNION ALL

## 2-10 index

### 2-10-1 index 설계절차

### 2-10-2 index 컬럼 적용시 고려 사항

### 2-10-3 index가 적용되지 않는 경우

# 3 옵티마이저 관련

데이터 이관이 끝나면 오라클에 함수 인덱스가 있는지 확인하고, 있으면 MariaDB에서 이에 대응하는 인덱스는 방법을 사용합니다.(현재, generated column에서 사용할 수 있는 함수에 제약사항이 많아서 100% 전환이 되지는 않습니다.)

## 3-1 실행 계획 분석

### 3-1-1 실행계획 기본정보

### 3-1-2 실행계획 내용(explain 정보)

### 3-1-3 분석시 주의 사항

DB USER 관련 부분

user@host

trigger , procedure, event sample

# 5 함수인덱스 변환 (generated column 이용)

데이터 이관이 끝나면 오라클에 함수 인덱스가 있는지 확인하고, 있으면 MariaDB에서 이에 대응하는 인덱스를 추가적으로 만들어줘야 합니다. 오라클의 함수 인덱스에 대응되는 MariaDB 쪽 대응 방법은 해당 함수 인덱스에서 사용하는 함수 및 계산 결과를 담을 generated column을 만든 후 해당 column에 index를 생성하는 방법을 사용합니다.(현재, generated column에서 사용할 수 있는 함수에 제약사항이 많아서 100% 전환이 되지는 않습니다.)

자세한 내용은 다음의 URL에 설명되어 있습니다. (<https://mariadb.com/kb/en/library/generated-columns/>)

테이블 CSMI\_ITMS0401 과 CSMI\_ITMS0402의 경우를 예로 들면 다음과 같습니다.

|  |
| --- |
| CSMI\_ITMS0401  - ALTER TABLE CSMI\_ITMS0401 ADD ASST\_NO\_VIRTUAL VARCHAR(15) AS (TRIM(ASST\_NO)) PERSISTENT AFTER BUDG\_MGR\_EENO;  - CREATE INDEX CSMI\_ITMS0401\_IDX01 ON CSMI\_ITMS0401(ASST\_NO\_VIRTUAL); |
| CSMI\_ITMS0402  - ALTER TABLE CSMI\_ITMS0402 ADD ASST\_NO\_VIRTUAL VARCHAR(15) AS (TRIM(ASST\_NO)) PERSISTENT AFTER MKR\_CD;  - ALTER TABLE CSMI\_ITMS0402 ADD ASST\_MGMT\_NO\_VIRTUAL VARCHAR(15) AS (TRIM(ASST\_MGMT\_NO)) PERSISTENT AFTER ASST\_NO\_VIRTUAL;  - CREATE INDEX CSMI\_ITMS0402\_IDX01 ON CSMI\_ITMS0402(ASST\_NO\_VIRTUAL);  - CREATE INDEX CSMI\_ITMS0402\_IDX02 ON CSMI\_ITMS0402(ASST\_MGMT\_NO\_VIRTUAL); |

CSMI\_ITMS0401의 경우 ASST\_NO를 TRIM 함수로 처리한 결과값을 넣을 ASST\_NO\_VIRTUAL 이라는 generated column을 만든 다음 해당 ASST\_NO\_VIRTUAL 컬럼에 인덱스를 만듭니다.

CSMI\_ITMS0402의 경우도 같은 방법으로 처리합니다.

# 6 오라클 Job Scheduler 변환 (MariaDB Event Scheduler)

오라클 Job Scheduler는 MariaDB의 Event Scheduer로 변환을 합니다. 어떤 USER가 Event를 create/alter/drop 하기 위해서는 DATABASE Level의 권한 중에 EVENT 권한이 있어야 합니다.

Ex) mysql> GRANT EVENT on ITIMS.\* to ‘user\_A’@’%’ ;

그리고, event scheduler가 실행되기 위해서는 event\_scheduler 변수가 enable 되어 있어야 하고, enable 시키는 방법은 my.cnf를 이용한 방법과 set 명령을 이용한 방법이 있고, 사용방법은 다음과 같습니다.

|  |
| --- |
| 1)번 2)번 두가지 방법 중 한가지를 사용합니다.  1) my.cnf를 이용한 방법(권장 방법)  Engine을 기동하기 전에 /etc/my.cnf 에 다음과 같이 등록시킵니다.(등록 후 재기동 해야 함)  event\_scheduler = ON  2) mysql 에서 set 명령으로 등록.(DB가 restart 되면 다시 설정 해줘야 합니다.)  SET GLOBAL event\_scheduler = ON;  3) 등록 되었는지 확인  MariaDB [(none)]> show global variables like '%event\_sche%';  +-----------------+-------+  | Variable\_name | Value |  +-----------------+-------+  **| event\_scheduler | ON |**  +-----------------+-------+  1 row in set (0.010 sec) |

## 6-1 Event Scheduler 생성 방법

6-1-1 Create Event Syntax 구문

|  |
| --- |
| CREATE [OR REPLACE]  [DEFINER = { user | CURRENT\_USER | role | CURRENT\_ROLE }]  EVENT  [IF NOT EXISTS]  event\_name  ON SCHEDULE schedule  [ON COMPLETION [NOT] PRESERVE]  [ENABLE | DISABLE | DISABLE ON SLAVE]  [COMMENT 'comment']  DO sql\_statement;  schedule:  AT timestamp [+ INTERVAL interval] ...  | EVERY interval  [STARTS timestamp [+ INTERVAL interval] ...]  [ENDS timestamp [+ INTERVAL interval] ...]  interval:  quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |  WEEK | SECOND | YEAR\_MONTH | DAY\_HOUR | DAY\_MINUTE |  DAY\_SECOND | HOUR\_MINUTE | HOUR\_SECOND | MINUTE\_SECOND} |

6-1-2 최소한의 Event Scheduler 생성 조건

- EVENT NAME : Event를 구분 짓는 이름은 반드시 존재해야 합니다.

- ON SCHEDULE 절 : Event가 실행되는 시기와 빈도를 결정하는 ON SCHEDULE 절은 반드시 있어야 합니다.

- DO 절 : Event에 의해 실행될 SQL 문은 반드시 포함되어야 합니다.

6-1-3 이벤트 생성 예제

|  |
| --- |
| CREATE EVENT myevent  ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL 1 HOUR  DO  UPDATE myschema.mytable SET mycol = mycol + 1;  Event name : myevent  On schedule : 현재 시간에서 1시간 후에 실행을 하고 이벤트가 자동으로 삭제됩니다.  Do : myschema DB의 mytable 이라는 테이블의 mycol 값을 1 증가 시킵니다. |

|  |
| --- |
| CREATE EVENT example  ON SCHEDULE EVERY 1 HOUR  STARTS CURRENT\_TIMESTAMP + INTERVAL 1 MONTH  ENDS CURRENT\_TIMESTAMP + INTERVAL 1 MONTH + INTERVAL 1 WEEK  DO call sp\_something();  Event name : example  On schedule : 현재 시간에서 1개월 후 부터 일주일 동안 매번 1시간 단위로 실행을 합니다.  Do : sp\_something이라는 프로시저를 실행 합니다. |

|  |
| --- |
| Event의 Do문에 block 지정시 반드시 DELIMITER 를 사용하여 event를 생성해야 합니다.  DELIMITER //  CREATE EVENT BLOCK\_EVENT  ON SCHEDULE EVERY 1 DAY STARTS '2019-03-15 03:00:00' ENDS '2050-12-31 04:00:00'  DO  BEGIN  call procedure\_A();  call procedure\_B();  END  //  DELIMITER ;  Event name : BLOCK\_EVENT  On schedule : 2019년 3월 15일 03시부터 2050년 12월31일 04시 까지 하루에 한번 03시에 이벤트가 실행됩니다.  Do : procedure\_A와 procedure\_B 라는 프로시저 두 개가 순차적으로 실행이 됩니다. |

## 6-2 Event Scheduler 변경 방법

6-2-1 Alter Event Syntax 구문

|  |
| --- |
| ALTER  [DEFINER = { user | CURRENT\_USER }]  EVENT event\_name  [ON SCHEDULE schedule]  [ON COMPLETION [NOT] PRESERVE]  [RENAME TO new\_event\_name]  [ENABLE | DISABLE | DISABLE ON SLAVE]  [COMMENT 'comment']  [DO sql\_statement] |

6-2-2 이벤트 변경 예제

|  |
| --- |
| ALTER EVENT myevent  ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL 2 HOUR  DO  UPDATE myschema.mytable SET mycol = mycol + 1;  기존 myevent 의 on schedule 에 있는 interval 1 hour 를 2 hour 로 변경합니다. |

## 6-3 Event Scheduler 삭제 방법

6-3-1 Drop Event Syntax 구문

|  |
| --- |
| DROP EVENT [IF EXISTS] event\_name |

6-3-2 이벤트 삭제 예제

|  |
| --- |
| DROP EVENT myevent ; |

## 6-4 Event Scheduler 모니터링

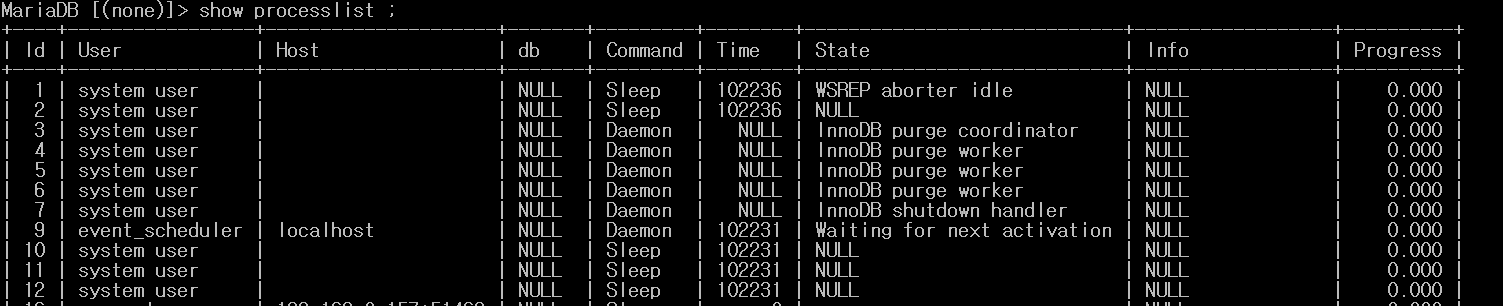
6-4-1 현재 DB에 생성된 event 확인 (show events)

|  |
| --- |
| MariaDB [sample]> show events \G  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Db: sample  Name: myevent  Definer: maria@%  Time zone: SYSTEM  Type: ONE TIME  Execute at: 2019-03-31 00:27:41  Interval value: NULL  Interval field: NULL  Starts: NULL  Ends: NULL  Status: ENABLED  Originator: 1  character\_set\_client: utf8  collation\_connection: utf8\_general\_ci  Database Collation: utf8\_general\_ci  1 row in set (0.017 sec) |

6-4-2 해당 event의 내용 확인 (show create event 이벤트이름)

|  |
| --- |
| MariaDB [sample]> show create event myevent \G  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Event: myevent  sql\_mode: STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION  time\_zone: SYSTEM  Create Event: CREATE DEFINER=`maria`@`%` EVENT `myevent` ON SCHEDULE AT '2019-03-31 00:27:41' ON COMPLETION NOT PRESERVE ENABLE DO UPDATE myschema.mytable SET mycol = mycol + 1  character\_set\_client: utf8  collation\_connection: utf8\_general\_ci  Database Collation: utf8\_general\_ci  1 row in set (0.003 sec)  또는 information\_schema 의 EVENTS 를 확인 합니다.  MariaDB [(none)]> select \* from information\_schema.EVENTS \G  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  EVENT\_CATALOG: def  EVENT\_SCHEMA: sample  EVENT\_NAME: myevent  DEFINER: maria@%  TIME\_ZONE: SYSTEM  EVENT\_BODY: SQL  EVENT\_DEFINITION: UPDATE myschema.mytable SET mycol = mycol + 1  EVENT\_TYPE: ONE TIME  EXECUTE\_AT: 2019-03-31 00:27:41  INTERVAL\_VALUE: NULL  INTERVAL\_FIELD: NULL  SQL\_MODE: STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION  STARTS: NULL  ENDS: NULL  STATUS: ENABLED  ON\_COMPLETION: NOT PRESERVE  CREATED: 2019-03-30 14:27:41  LAST\_ALTERED: 2019-03-30 14:27:41  LAST\_EXECUTED: NULL  EVENT\_COMMENT:  ORIGINATOR: 1  CHARACTER\_SET\_CLIENT: utf8  COLLATION\_CONNECTION: utf8\_general\_ci  DATABASE\_COLLATION: utf8\_general\_ci  1 row in set (0.010 sec) |

6-4-3) 현재 event scheduler가 작동 중인지 확인 (show processlist )



User 부분의 “event\_scheduler”가 State 부분에 “Waiting for next activation” 이라는 상태에 있으면 현재 작동중인 상태입니다.

# 7 profile 관련 변환

이관 당시 오라클에 설정된 프로파일을 기준으로 작성합니다.

|  |  |
| --- | --- |
| 오라클 | MariaDB |
| SESSIONS\_PER\_USER UNLIMITED | MAX\_USER\_CONNECTIONS 0  (User 생성시 설정합니다.) |
| CPU\_PER\_SESSION UNLIMITED | 기본적으로 UNLIMITED |
| CPU\_PER\_CALL UNLIMITED | 기본적으로 UNLIMITED |
| CONNECT\_TIME UNLIMITED | 기본적으로 UNLIMITED |
| IDLE\_TIME UNLIMITED | Active mode인 경우,  wait\_timeout (1~3153600 초)을 이용하여 설정 |
| Interactive mode인 경우,  interactive\_timeout (1~3153600 초)을 이용하여 설정 |
| LOGICAL\_READS\_PER\_SESSION UNLIMITED | 기본적으로 UNLIMITED |
| LOGICAL\_READS\_PER\_CALL UNLIMITED | 기본적으로 UNLIMITED |
| COMPOSITE\_LIMIT UNLIMITED | N/A |
| PRIVATE\_SGA UNLIMITED | 기본적으로 공용 메모리는 UNLIMITED  (PRIVATE\_SGA에 매칭되는 내용이 MariaDB에는 없음) |
| FAILED\_LOGIN\_ATTEMPTS 10 | N/A |
| PASSWORD\_LIFE\_TIME 90 | N/A  (10.4.3 버전에 expire 기능 추가) |
| PASSWORD\_REUSE\_TIME UNLIMITED | 기본적으로 UNLIMITED |
| PASSWORD\_REUSE\_MAX UNLIMITED | 기본적으로 UNLIMITED |
| PASSWORD\_LOCK\_TIME 0.0833 | N/A |
| PASSWORD\_GRACE\_TIME 7 | N/A |
| PASSWORD\_VERIFY\_FUNCTION VERIFY\_FUNCTION | 완벽히 호환되지는 않지만 Simple Password Check Plugin 를 이용하여 패스워드 설정에 제한을 가할 수 있습니다. 아래 URL을 참조하시기 바랍니다.  https://mariadb.com/kb/en/library/simple-password-check-plugin/ |

# 8 Sequence 변환

MariaDB의 sequence는 10.3 부터 제공되는 object 입니다. 자세한 기능 및 사용법은 다음의 URL을 참조 하시기 바랍니다.(<https://mariadb.com/kb/en/library/sequence-overview/>)

Sequence 사용법

|  |  |
| --- | --- |
| Oracle | MariaDB |
| sequence\_name.nextval | NEXT VALUE FOR sequence\_name  또는  NEXTVAL(sequence\_name) |
| sequence\_name.currval | PREVIOUS VALUE FOR sequence\_name  또는  LASTVAL(sequence\_name) |
|  |  |

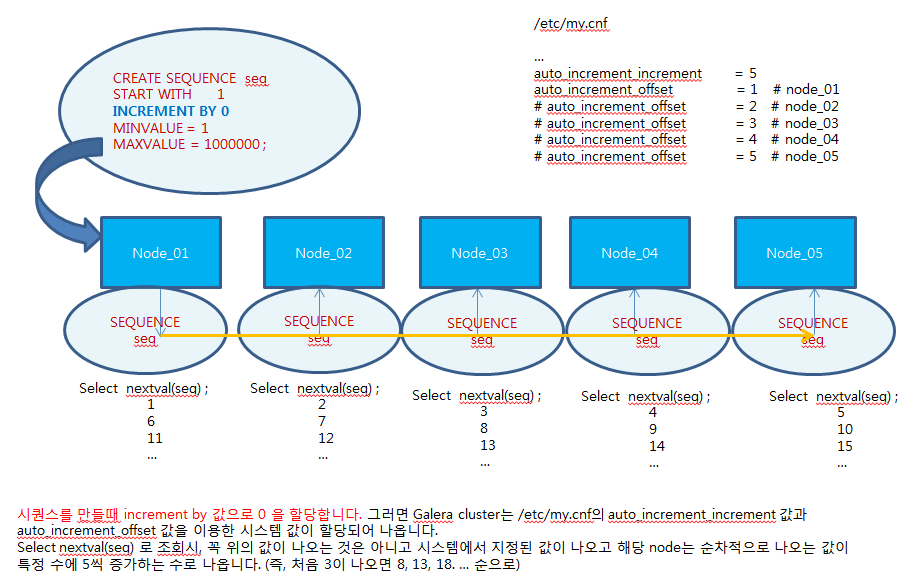
## 8-1 Oracle에서 MariaDB로의 sequence 이관

다음의 절차에 의해서 Oracle sequence를 MariaDB에 생성합니다.

|  |
| --- |
| 1. Oracle에서 각 sequence에 대한 create sequence 구문을 스크립트 파일로 백업 합니다.  2. Oracle에서 각 sequence에 대한 currval 값을 구합니다.  3. 1.번에서 구한 Oracle sequence 생성 스크립트 파일을 MariaDB의 문법에 맞게 수정하고,  start with 부분에는 2번에서 구한 currval 값을 지정합니다.  4. 3에서 만든 스크립트를 MariaDB에서 생성합니다. |

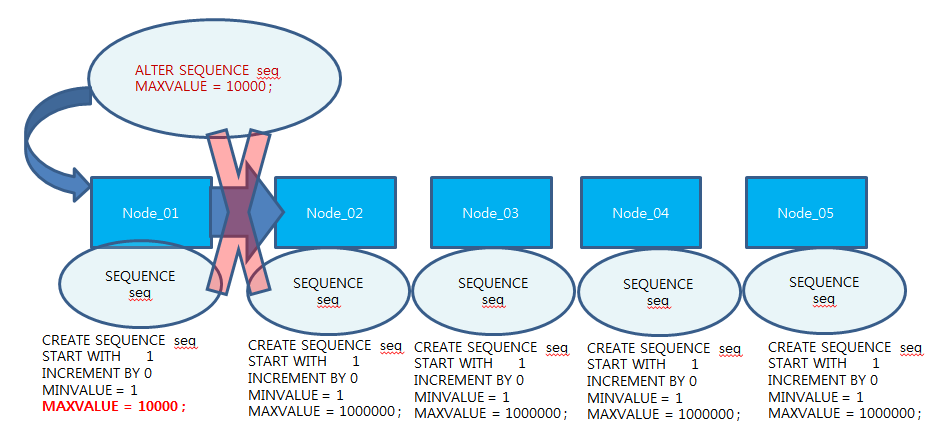
## 8-2 Galera Cluster(또는 multi-mater 환경)에서의 sequence 설정

Sequence 생성시 increment 값을 0으로 설정 하고, my.cnf에는 auto\_increment\_increment 와 auto\_increment\_offset 값을 node 환경에 맞게 설정해야 합니다. 설정이 정상적으로 되면, Galera Cluster의 각 node는 시스템 환경에 맞게 중복되지 않는 값을 return 합니다. 단 값의 증가분이 auto\_increment\_offset 값 만큼씩 증가 합니다.



## 8-3 alter sequence 관련 SR 진행 상황

Galera Cluster에서 어느 하나의 node에 Sequence를 create 하면, 8-1의 그림에서 보는 바와 같이 나머지 4개의 node에 자동으로 복제가 됩니다. 물론, alter sequence 의 경우도 하나의 node에서 변경이 발생하면 나머지 4개의 node에도 그 변경이 반영되어야 합니다. 그런데, 현재 버전 MariaDB 10.3.10(galera 25.3.24)에서는 alter sequence를 하나의 node에서 실행하면 해당 node 하나에만 변경이 일어나고 나머지 4개의 node에는 변경이 발생하지 않습니다. 해당 내용은 SR Issue ID: 27959 로 오픈 되어서 진행중 입니다.(Bug 번호 : [MDEV-19353](https://jira.mariadb.org/browse/MDEV-19353))



# 9 함수변환

|  |  |  |
| --- | --- | --- |
|  | Oracle | mariaDB |
| 1 | ABS(num) | ABS(num) |
| 2 | ACOS(num) | ACOS(num) |
| 3 | ADD\_MONTHS(date, num) | TIMESTAMPADD(MONTH, num, date)  : SELECT TIMESTAMPADD(MONTH, 1, '2019-03-28'); |
| 4 | ASCII(str) | ASCII(str)  : SELECT ASCII('TEST'); |
| 5 | ASCIISTR(string) | N/A |
| 6 | ASIN(num) | ASIN(num) |
| 7 | ATAN(num) | ATAN(num) |
| 8 | ATAN2(x, y) | ATAN2(x, y) |
| 9 | BIN\_TO\_NUM(bit1, bit2, …) | N/A |
| 10 | BITAND(exp1, exp2) | (exp1 & exp2)  : SELECT 29 & 15; |
| 11 | CEIL(num) | CEIL(num)  : SELECT CEIL(1.23); |
| 12 | CHR(num) | CHR(num) : 10.3.1부터 지원  : SELECT CHR(67); |
| 13 | COALESCE(exp1, exp2, …) | COALESCE(exp1, exp2, …)  : SELECT COALESCE(NULL,1); |
| 14 | CONCAT(char1, char2) | CONCAT(char1, char2)  : SELECT CONCAT(c1, c2) FROM tab\_A; |
| 15 | CONVERT(string, charset) | CONVERT(string USING charset)  : SELECT CONVERT('abc' USING utf8); |
| 16 | COS(num) | COS(num) |
| 17 | COSH(num) | (EXP(num) + EXP(-num)) / 2 |
| 18 | CURRENT\_DATE | CURDATE()  : SELECT CURDATE();  NOW()  : SELECT DATE\_FORMAT(NOW(),'%Y-%m-%d'); |
| 19 | CURRENT\_TIMESTAMP | CURRENT\_TIMESTAMP()  : SELECT CURRENT\_TIMESTAMP(6) ;  NOW()  : SELECT NOW(6); |
| 20 | DECODE(exp, when, then, …)  : SELECT DECODE(foods, '한식',1, '중식',2, '양식',3 , 4) | CASE expression  : SELECT CASE foods  WHEN '한식' THEN 1  WHEN '중식' THEN 2  WHEN '양식' THEN 3  ELSE 4  END; |
| 21 | EXP(n) | EXP(n) |
| 22 | EXTRACT(YEAR FROM date) | YEAR(date)  : SELECT YEAR('2019-03-28') ; |
| 23 | EXTRACT(MONTH FROM date) | MONTH(date)  : SELECT MONTH('2019-03-28') ; |
| 24 | EXTRACT(DAY FROM date) | DAY(date)  : SELECT DAY('2019-03-28') ; |
| 25 | EXTRACT(HOUR FROM time) | HOUR(time)  : SELECT HOUR('10:05:03'); |
| 26 | EXTRACT(MINUTE FROM time) | MINUTE(time)  : SELECT MINUTE('2019-03-28 11:04:03'); |
| 27 | EXTRACT(SECOND FROM time) | SECOND(time)  : SELECT SECOND('10:05:03'); |
| 28 | FLOOR(num) | FLOOR(num)  : SELECT FLOOR(18.567); |
| 29 | GREATEST(exp, exp2, …) | GREATEST(exp, exp2, …)  : SELECT GREATEST(34.5, 3.6, 5.2, 767.1); |
| 30 | INITCAP(string) | N/A (사용자 함수로 만들어서 사용)  < 생성 >  DELIMITER $$  CREATE FUNCTION `initcap`(x varchar(255)) RETURNS varchar(255) CHARSET utf8  RETURN concat( upper(substring(x,1,1)),lower(substring(x,2)) );  END$$  DELIMITER ;  < 사용 >  SELECT initcap('this is chanchal wankhade.') ; |
| 31 | INSTR(str, substr) | INSTR(str, substr)  : SELECT INSTR('foobarbar', 'bar'); |
| 32 | INSTR(str, substr, pos) | LOCATE(substr, str, pos)  : SELECT LOCATE('bar', 'foobarbar', 5); |
| 33 | INSTR(str, substr, pos, num) | 파라메터 4개의 경우 사용자 함수로 만들어서 사용  < 생성 >  DROP FUNCTION IF EXISTS `INSTR\_4` ;  DELIMITER //  CREATE FUNCTION `INSTR\_4`( str VARCHAR(8000),  sub\_str VARCHAR(255),  str\_start INT,  occurrence INT )  RETURNS INT  BEGIN  DECLARE cnt INT ;  DECLARE pos INT ;  SET pos = str\_start;  SET cnt = occurrence;  `EXIT\_WHILE`:  WHILE TRUE DO  -- Find the next occurrence  SET pos = LOCATE(sub\_str, str, pos);  -- Nothing found  IF pos = 0 THEN  RETURN pos;  END IF;  -- The required occurrence found  IF cnt = 1 THEN  LEAVE `EXIT\_WHILE`;  END IF;  -- Prepare to find another one occurrence  SET cnt = cnt - 1;  SET pos = pos + 1;  END WHILE;  RETURN pos;  END  //  DELIMITER ;  < 사용 >  SELECT `INSTR\_4`("foobarbarbarbarbar", "bar", 5, 2); |
| 34 | LAST\_DAY(date) | LAST\_DAY(date)  : SELECT LAST\_DAY('2019-02-01');  SELECT LAST\_DAY('2019-02-01 12:30:30'); |
| 35 | LEAST(exp, exp2, …) | LEAST(exp, exp2, …)  : SELECT LEAST(34.2, 3.5, 3.2, 767.0);  SELECT LEAST('B', 'A', 'C'); |
| 36 | LENGTH(string)  : 문자의 개수 | CHAR\_LENGTH(string)  : SELECT CHAR\_LENGTH('마리아'); |
| 37 | LENGTHB(string)  : 문자의 byte 수 | LENGTH(string) 또는 LENGTHB(string)  : SELECT LENGTH('마리아'); |
| 38 | LN(num) | LN(num) |
| 39 | LOCALTIMESTAMP  : default 소수점이하 6자리 | LOCALTIMESTAMP (내부적으로 now()와 같다)  : SELECT LOCALTIMESTAMP(6); |
| 40 | LOCALTIMESTAMP([prec])  : SELECT LOCALTIMESTAMP(0) FROM DUAL ; | LOCALTIMESTAMP()  : SELECT LOCALTIMESTAMP(0); |
| 41 | LOG(num1, num2) | LOG(num1, num2) |
| 42 | LOWER(string) | LOWER(string)  : SELECT LOWER('QUADRATICALLY'); |
| 43 | LPAD(string, len) | LPAD(string, len, ' ') 또는 10.3.1버전부터 LPAD(string, len)  : SELECT LPAD('hello',10); |
| 44 | LPAD(string, len, pad) | LPAD(string, len, pad)  : SELECT LPAD('hello',10, '\*'); |
| 45 | LTRIM(string) | LTRIM(string)  : SELECT LTRIM(' MariaDB ')); |
| 46 | LTRIM(string, set)  : select ltrim('aabacdbaeaa' , 'a') from dual; | TRIM(LEADING set FROM string)  : SELECT TRIM(LEADING 'a' FROM 'aabacdbaeaa') ; |
| 47 | MONTHS\_BETWEEN(date1, date2) | N/A 사용자 함수로 만들어서 사용  MONTHS\_BETWEEN 함수는 두 날짜간의 MONTH 값의 차이를 정수부분과 소수부분으로 리턴합니다.  다음의 경우에는 정수 값만 리턴합니다.  - 두 날짜의 DAY 부분이 모두 같습니다 (2월 13일과 3월 13일).  - 두 날짜가 모두 해당 월의 마지막 날 입니다. (1월 31일과 4월 30일)  소수 부분이 참조될 때, 다음과 같이 계산 합니다.  If day\_of\_date1 > day\_of\_date2 , then (day\_of\_date1 - day\_of\_date2) / 31  If day\_of\_date1 < day\_of\_date2 , then (31 - day\_of\_date2 + day\_of\_date1) / 31  그리고, MONTHS\_BETWEEN 은 소수 부분을 계산할 때, 모든 달을 31일로 계산합니다.  < 생성 >  DROP FUNCTION IF EXISTS month\_between ;  DELIMITER //  CREATE FUNCTION month\_between( date1 DATETIME, date2 DATETIME )  RETURNS FLOAT  BEGIN  DECLARE months FLOAT;  BEGIN  SET months = MONTH(date1) - MONTH(date2);  IF (DAY(date1) <> DAY(date2)) AND  (MONTH(date1) = MONTH(date1 + INTERVAL 1 DAY) OR MONTH(date2) = MONTH(date2 + INTERVAL 1 DAY))  THEN  IF DAY(date1) < DAY(date2)  THEN  SET months = months + (((31 - DAY(date2) + DAY(date1)) / 31) - 1);  ELSE  SET months = months + ((DAY(date1) - DAY(date2)) / 31);  END IF;  END IF;  END;  RETURN months;  END  //  DELIMITER ;  < 사용 >  SELECT month\_between('2019-04-3', '2019-03-31'); |
| 48 | MOD(dividend, divisor) | MOD(dividend, divisor) |
| 49 | NEXT\_DAY  1) SELECT NEXT\_DAY(SYSDATE, 'Sun') FROM dual;  NEXT\_DAY(SYSDATE,'SUN')  -----------------------  2019-05-19  1 row selected.  2) SELECT NEXT\_DAY(SYSDATE, 'Sun') - 7 FROM dual;  NEXT\_DAY(SYSDATE,'SUN')-7  -------------------------  2019-05-12  1 row selected. | N/A 사용자 함수로 만들어서 사용  < 생성 >  DELIMITER //  CREATE FUNCTION next\_day(start\_date DATETIME, weekday CHAR(20))  -- RETURNS DATETIME  RETURNS DATE  BEGIN  DECLARE start DATETIME;  DECLARE i INT;    -- Select the next date  SET start = ADDDATE(start\_date, 1);  SET i = 1;    days:  LOOP  -- Compare the day names  IF SUBSTR(DAYNAME(start), 1, 3) = SUBSTR(weekday, 1, 3) THEN  LEAVE days;  END IF;  -- Select the next date  SET start = ADDDATE(start, 1);  SET i = i + 1;  -- Not valid weekday specified  IF i > 7 THEN  SET start = NULL;  LEAVE days;  END IF;    END LOOP days;    RETURN start;  END;  //    DELIMITER ;  < 사용 >  1) MariaDB [sample]> SELECT NEXT\_DAY(SYSDATE(), 'Sun') FROM dual;  +----------------------------+  | NEXT\_DAY(SYSDATE(), 'Sun') |  +----------------------------+  | 2019-05-19 |  +----------------------------+  1 row in set, 1 warning (0.003 sec)  2) MariaDB [sample]> SELECT SUBDATE(NEXT\_DAY(NOW(), 'Sun'), 7);  +------------------------------------+  | SUBDATE(NEXT\_DAY(NOW(), 'Sun'), 7) |  +------------------------------------+  | 2019-05-12 |  +------------------------------------+  1 row in set, 1 warning (0.003 sec)  MariaDB [sample]> |
| 50 | NULLIF(exp1, exp2) | NULLIF(exp1, exp2)  : SELECT NULLIF(1,1); |
| 51 | NVL(exp, replacement) | IFNULL(exp, replacement)  : SELECT IFNULL(NULL,10); |
| 52 | NVL2(exp1, exp2, exp3)  : 1) SELECT NVL2(NULL, 'Maria', 'NULL') FROM DUAL;  2) SELECT NVL2(1+2, 'Maria', 'NULL') FROM DUAL; | CASE expression  CASE WHEN exp1 IS NOT NULL THEN exp2 ELSE exp3 END  1)  SELECT CASE WHEN NULL IS NOT NULL  THEN 'Maria'  ELSE 'NULL'  END AS NVL2 ;  2)  SELECT CASE WHEN 1+2 IS NOT NULL  THEN 'Maria'  ELSE 'NULL'  END AS NVL2 ; |
| 53 | POWER(value, n) | POWER(value, n) |
| 54 | REMAINDER(n1, n2) | (n1 - n2\*ROUND(n1/n2)) |
| 55 | REPLACE(str, search)  : SELECT REPLACE('oracle to mariadb', 'ora') FROM DUAL; | REPLACE(str, search, '')  : SELECT REPLACE('oracle to mariadb', 'ora', ‘’); |
| 56 | REPLACE(str, search, replace)  : SELECT REPLACE('oracle to mariadb', 'oracle', 'MySQL') FROM DUAL; | REPLACE(str, search, replace)  : SELECT REPLACE('oracle to mariadb', 'oracle', 'MySQL'); |
| 57 | ROUND(num, integer) | ROUND(num, integer) |
| 58 | RPAD(string, len)  : SELECT RPAD('31', 5) FROM DUAL; | RPAD(string, len, ' ')  : SELECT RPAD('31', 5); |
| 59 | RPAD(string, len, pad)  : SELECT RPAD('31', 5, '0') FROM DUAL; | RPAD(string, len, pad)  : SELECT RPAD('31', 5, '0'); |
| 60 | RTRIM(string)  : SELECT RTRIM (' Sample ') FROM DUAL; | RTRIM(string)  : SELECT RTRIM (' Sample '); |
| 61 | RTRIM(string, set)  : SELECT RTRIM (' Sample000', '000') FROM DUAL; | TRIM(TRAILING set FROM string)  : SELECT TRIM(TRAILING '000' FROM ' Sample000'); |
| 62 | SIGN(exp) | SIGN(exp) |
| 63 | SIN(num) | SIN(num) |
| 64 | SINH(num) | (EXP(num) - EXP(-num)) / 2 |
| 65 | SOUNDEX(string) | SOUNDEX(string) |
| 66 | SQRT(num) | SQRT(num) |
| 67 | SUBSTR(string, pos)  : SELECT SUBSTR ('ABCDEFG', -2) FROM DUAL; | SUBSTR(string, pos)  : SELECT SUBSTR ('ABCDEFG', -2); |
| 68 | SUBSTR(string, pos, len)  : SELECT SUBSTR('oracle to mariadb', 8, 8) FROM DUAL; | SUBSTR(string, pos, len)  : SELECT SUBSTR('oracle to mariadb', 8, 8); |
| 69 | SYS\_GUID()  : SELECT SYS\_GUID() FROM DUAL; | REPLACE(UUID(), '-', '')  : SELECT REPLACE (UUID(), '-', '') ; |
| 70 | SYSDATE  : SELECT SYSDATE FROM DUAL;  : SELECT TO\_CHAR(SYSDATE, 'YYYY-MM-DD HH:MI:SS') FROM DUAL; | SYSDATE()  : SELECT DATE\_FORMAT(SYSDATE(), '%Y-%m-%d');  : SELECT SYSDATE(); |
| 71 | SYSTIMESTAMP  : SELECT SYSTIMESTAMP FROM DUAL; | SYSDATE(6)  : SELECT SYSDATE(6); |
| 72 | TAN(num) | TAN(num) |
| 73 | TANH(num) | (EXP(2\*num) - 1)/(EXP(2\*num) + 1) |
| 74 | TO\_CHAR(datetime, format)  : SELECT TO\_CHAR(SYSDATE, 'YYYYMMDD') FROM DUAL;  : SELECT TO\_CHAR(SYSDATE - 1, 'yyyymmdd') FROM DUAL; | DATE\_FORMAT(datetime, format)  : SELECT DATE\_FORMAT(SYSDATE(), '%Y%m%d') ;  : SELECT DATE\_FORMAT(DATE\_ADD(SYSDATE(), INTERVAL -1 DAY), '%Y%m%d') |
| 75 | TO\_CHAR(number, format)  : SELECT TO\_CHAR('1234.5', '999,999.99') FROM DUAL;  : SELECT TO\_CHAR('12345', 'fm$9,999,999') FROM DUAL;  : SELECT TO\_CHAR('12345', '000,000,000') FROM DUAL;  : SELECT TO\_CHAR(1111) FROM DUAL; | FORMAT(number, decimal\_digits)  : SELECT FORMAT(1234.5, 2);  : SELECT CONCAT('$', FORMAT(12345, 0);  : SELECT LPAD(FORMAT(12345,0), 9, 0);  : SELECT CAST(1111 AS CHAR); |
| 76 | TO\_DATE(string, format)  : SELECT TO\_DATE('2019-04-04', 'YYYY-MM-DD') FROM dual;  : SELECT TO\_DATE('30/04/19', 'DD/MM/RR') FROM DUAL;  : SELECT TO\_DATE('20190415', 'YYYYMMDD') FROM DUAL; | STR\_TO\_DATE(string, format)  : SELECT STR\_TO\_DATE('2019-04-04', '%Y-%m-%d');  : SELECT STR\_TO\_DATE('30/04/19','%d/%m/%y');  : SELECT STR\_TO\_DATE('20190415', '%Y%m%d'); |
| 77 | TO\_LOB(exp) | N/A |
| 78 | TO\_NCHAR(exp) | N/A |
| 79 | TO\_NCLOB(exp) | N/A |
| 80 | TO\_NUMBER(exp)  : SELECT TO\_NUMBER('12345') from dual; | CAST(exp as datatype)  : SELECT CAST( '12345' AS UNSIGNED INTEGER); |
| 81 | TO\_SINGLE\_BYTE(exp) | N/A |
| 82 | TO\_TIMESTAMP(str, format)  : SELECT TO\_TIMESTAMP ('2019-04-07 14:10:10.123000', 'YY-MM-DD HH24:MI:SS.FF') FROM DUAL; | STR\_TO\_DATE(str, format)  : SELECT STR\_TO\_DATE('2019-04-07 14:10:10 123000','%Y-%m-%d %H:%i:%s %f'); |
| 83 | TRANSLATE(string, from, to)  : 1) SELECT TRANSLATE('ABCDEFG', 'C', '!@#') FROM dual;  TRANSLATE('ABCDEFG','C','!@#')  ------------------------------  AB!DEFG  1 row selected.  2) SELECT TRANSLATE('ABCDEFG', 'CDE', '!@#') FROM dual;  TRANSLATE('ABCDEFG','CDE','!@#')  --------------------------------  AB!@#FG  1 row selected.  3) SELECT TRANSLATE('ABCDEFG', 'CDEAB', '!@#') FROM dual;  TRANSLATE('ABCDEFG','CDEAB','!@#')  ----------------------------------  !@#FG  1 row selected. | N/A 사용자 함수로 만들어서 사용  < 생성 >  DELIMITER //  CREATE FUNCTION `translate`(subject varchar(255), what varchar(255), replace\_to varchar(255))  RETURNS varchar(255)  begin  declare c int unsigned default 0;  declare result varchar(255);  set result = subject;  while c <= length(subject) do  set result = replace(result, mid(what, c, 1), mid(replace\_to, c, 1) );  set c=c+1;  end while;  return result;  END  //  DELIMITER ;  < 사용 >  1) SELECT TRANSLATE('ABCDEFG', 'C', '!@#') FROM dual;  +----------------------------------+  | TRANSLATE('ABCDEFG', 'C', '!@#') |  +----------------------------------+  | AB!DEFG |  +----------------------------------+  1 row in set (0.007 sec)  2) SELECT TRANSLATE('ABCDEFG', 'CDE', '!@#') FROM dual;  +------------------------------------+  | TRANSLATE('ABCDEFG', 'CDE', '!@#') |  +------------------------------------+  | AB!@#FG |  +------------------------------------+  1 row in set (0.004 sec)  3) SELECT TRANSLATE('ABCDEFG', 'CDEAB', '!@#') FROM dual;  MariaDB [sample]> SELECT TRANSLATE('ABCDEFG', 'CDEAB', '!@#') FROM dual;  +--------------------------------------+  | TRANSLATE('ABCDEFG', 'CDEAB', '!@#') |  +--------------------------------------+  | !@#FG |  +--------------------------------------+  1 row in set (0.004 sec) |
| 84 | TRIM([type trim FROM] string) | TRIM([type trim FROM] string) |
| 85 | TRUNC(num) | TRUNCATE(num, 0) |
| 86 | TRUNC(num, num2) | TRUNCATE(num, num2) |
| 87 | TRUNC(datetime)  : 1) SELECT TRUNC(SYSDATE) FROM dual;  2) SELECT TRUNC(SYSDATE, 'DD') FROM dual;  3) SELECT TRUNC(SYSDATE, 'MM') FROM dual; | DATE(datetime), DATE\_FORMAT  : 1) SELECT DATE(SYSDATE());  2) SELECT DATE(SYSDATE());  3) SELECT DATE\_FORMAT(SYSDATE(), '%Y-%m-01'); |
| 88 | UNISTR(string) | CHAR(string USING UCS2) |
| 89 | UPPER(string)  : SELECT UPPER('String Data Type') FROM DUAL; | UPPER(string)  : SELECT UPPER('String Data Type'); |
| 90 | USER  : SELECT User FROM DUAL;  USER  ------------------------------  SCOTT  1 row selected. | USER()  : SELECT USER() ;  +-----------------+  | USER() |  +-----------------+  | maria@localhost |  +-----------------+  1 row in set (0.001 sec) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 10 그외 Oracle to MariaDB 관련 주의할 점

## 10.1 오라클과 MariaDB의 NULL 값

오라클의 경우는 NULL 과 “”(zero space) 둘 다 NULL로 인식하는데 MariaDB의 경우 NULL 만 인식합니다. 이와 관련하여 사용자 프로그램에서 NULL 값 체크하는 루틴에서 잘못된 결과가 발생할 수 있습니다.

## 10.2 후행공백(trailing space) 처리

오라클의 경우는 후행공백을 의미 있는 문자로 인식하고 CHAR, VARCHAR 타입의 컬럼에 해당 데이터가 insert 되는 데, MariaDB의 경우는 디폴트로 후행 공백을 의미 있는 문자로 처리 하지 않습니다. VARCHAR의 경우는 후행공백이 컬럼에 insert 는 되지만 해당 VARCHAR 컬럼에 대한 index를 만들거나 하는 처리를 할 때는 제거 되어서 사용되고, CHAR 의 경우는 후행공백이 처음부터 잘려서 insert 됩니다. 후행공백을 의미 있는 문자로 처리하려면 4장에서 설명이 되었던 No Pad Collation을 사용하셔야 합니다. (<https://mariadb.com/kb/en/library/supported-character-sets-and-collations/#no-pad-collations>)

## 10.3 Galera cluster에서의 event scheduler

Galera cluster 환경에서 event scheduler를 만들고 생성된 상태를 show events \G 명령으로 확인을 하면, create event 구문을 실행시킨 node만 Status 부분이 ENABLED 로 되고, 나머지 생성 구문이 복제 된 node들은 SLAVESIDE\_DISABLED 로 됩니다. 그리고, Event 실행은 Status가 ENABLED로 된 node만 실행이 됩니다.

## 10.4 Galera cluster에서의 sequence

Galera cluster 환경에서의 sequence 사용은 8장에서 설명을 드린 것처럼 약간의 설정이 필요하며 기존 오라클에서 사용하던 increment by 1, 또는 10 과 같은 증가분에 대한 제어를 할 수 없습니다. increment by 0 을 설정하고 시스템에서 제공하는 수를 사용합니다.

## 10.5 Galera cluster에서의 통계정보

Galera cluster 환경에서는 각 node별로 통계정보 데이터를 관리합니다. 따라서, 통계정보 갱신 작업을 하는 경우 해당 작업을 모든 node에서 똑같이 실행 해야 합니다. 그렇지 않으면 같은 쿼리가 node에 따라서 성능이 일정하게 나오지 않을 수 있습니다.