6주차

▼ 1. 전문 검색

전문 검색(Full-text Search): 용량이 큰 문서를 단어 수준으로 잘게 쪼개어 문서 검색을 하게 해주는 기능

기존에는 일부 스토리지 엔진을 사용하는 테이블만 전문 검색 기능을 활용할 수 있었지만 8.0 부터 InnoDB 스토리지 엔진에서도 사용할 수 있도록 개선

한국어나 중국어, 일본어 보다는 서구권 언어에 적합한 형태소 분석과 이를 보완하기 위해 8.0 부터 도입된 특정 길이 조각(Token)으로 인덱싱하는 n-gram 파서가 도입

> InnoDB 스토리지 엔진의 전문 검색 엔진에서만 내장 불용어 처리를 하지 않게 설정하려면 `inodb_ft_stopword` 시스템 변수 값을 `OFF`로 변경하면 된다.

1.1 전문 검색 인덱스의 생성과 검색

인덱싱 토큰 분리 알고리즘

- 형태소 분석(서구권 언어의 경우 어근 분석)
  - 문장의 공백과 같은 띄어쓰기 단위로 단어를 분리 후 각 단어의 조사를 제거해 명사 또는 어근을 찾아 인덱싱(MySQL에서는 단순히 공백과 같은 띄어쓰기 기준으로 토큰을 분리해 인덱싱)
- n-gram 파서
  - 문장 자체에 대한 이해 없이 공백과 같은 띄어쓰기 단위로 단어를 분리하고, 그 단어를 단순히 주어진 길이(n-gram의 n은 1~10 사이의 수)로 쪼개서 인덱싱

n-gram

n: `ngram_token_size` 시스템 변수로 설정하며 숫자 값(기본값은 2, 최소 1, 최대 10)

n이 1면 uni-gram, 2이면 bi-gram, 3이면 tri-gram

일반적으로 bi-gram 또는 tri-gram이 많이 사용

`ngram_token_size` 시스템 변수는 MySQL 서버 설정 파일을 이용해 서버 시작 시에만 변경할 수 있다. 또한 테이블의 전문 검색 인덱스를 생성할 때 `WITH PARSER ngram` 옵션을 추가해야 n-gram 파서를 사용해 토큰을 생성할 수 있다.(그렇지 않으면 기본 파서를 사용)

규칙

- 검색어의 길이가 `ngram_token_size` 값보다 작은 경우 검색 불가능
- 검색어의 길이가 `ngram_token_size` 값보다 크거나 같은 경우 검색 가능

n-gram 전문 검색 인덱스는 검색어가 단어의 시작 부분이 아니고, 단어의 중간이나 마지막 부분이어도 n-gram이 검색할 수 있다.

```
단편적인 -> 단편 | 편적 | 적인
이 -> 없음
```

전문 검색 쿼리가 오면 인덱싱할 때와 동일하게 검색어를 ngram_token_size 시스템 변수값에 맞게 토큰을 자른다. 그리고 그 토큰들을 검색된 결과들의 도큐먼트 ID로 그루핑하고, 그루핑된 결과에서 각 단어의 위치를 이용해 최종 검색어를 포함하는지 식별한다.

> **도큐먼트 ID**
>
> 전문 검색 인덱스의 경우 PK와 별개로 레코드별 id를 가지는데, 이를 도큐먼트 ID(Document ID)라고 한다. MySQL 서버의 전문 검색 인덱스에서 도큐먼트는 레코드 또는 로우(Row)와 동의어로 사용된다.

> 테이블 생성, 데이터 `INSERT`/`UPDATE`, `SELECT` 쿼리로 전문 검색 실행 시 모두 `ngram_token_size` 시스템 변수는 영향을 미친다. 그래서 한 번 전문 검색 인덱스가 생성되면 `ngram_token_size` 시스템 변수는 변경하면 안 된다.
>
> InnoDB의 n-gram 파서를 이용하는 경우 다음 4개의 시스템 변수는 아무런 영향을 미치지 않는다. 이 시스템 변수들은 MySQL 서버가 형태소 분석 알고리즘으로 토큰을 만들어 낼 때만 영향을 미친다.
>
> `innodb_ft_min_token_size`, `Innodb_ft_max_token_size`, `ft_min_word_len`, `ft_max_word_len`

1.2 전문 검색 쿼리 모드

- 자연어(`NATURAL LANGUAGE`) 검색 모드
- 불리언(`BOOLEAN`) 검색 모드

전문 검색 쿼리에서 특정 모드를 지정하지 않으면 자연어 검색 모드

자연어 검색 모드와 함께 사용 가능한 검색어 확장(Query Expansion) 기능도 지원

1. 자연어 검색(NATURAL LANGUAGE MODE)

검색어에 제시된 단어들을 많이 가지고 있는 순서대로 정렬해 결과를 반환한다.

단일 단어뿐 아니라 문장을 검색어로 사용 가능하다. 이 경우 공백이나 뉴 라인과 같은 띄어쓰기 문자 등을 기준으로 단어를 분리하고 n-gram 파서로 토큰을 생성한 후 각 토큰에 대해 일치하는 단어의 개수를 확인해 일치율을 계산

> 검색어가 단일 단어 또는 문장인 경우 “.” 또는 “,”등과 같은 문장 기호는 모두 무시

```
SELECT id, title, body, MATCH(title, body) AGAINST('MySQL' IN NATURAL LANGUAG
FROM tb_bi_gram
WHERE MATCH(title, body) AGAINST('MySQL' IN NATURAL LANGUAGE MODE);
```

2. 불리언 검색(BOOLEAN MODE)

쿼리에 사용되는 검색어의 존재 여부에 대해 논리적 연산이 가능하다.

```
SELECT id, title, body, MATCH(title, body) AGAINST('+MySQL -manual' IN BOOLEA
FROM tb_bi_gram
WHERE MATCH(title, body) AGAINST('+MySQL -manual' IN BOOLEAN MODE);
```

“MySQL”은 포함하지만 “manual”은 포함하지 않는 레코드를 검색하는 쿼리

불리언 검색에서 쌍따옴표(`"`)로 묶인 구는 하나의 단어인 것처럼 취급

```
SELECT id, title, body, MATCH(title, body) AGAINST('+"MySQL man"' IN BOOLEAN
FROM tb_bi_gram
WHERE MATCH(title, body) AGAINST('+"MySQL man"' IN BOOLEAN MODE);
```

띄어쓰기까지 정확히 일치하는 것을 찾는 게 아닌 “MySQL” 단어 뒤에 “man”이라는 단어가 나오면 일치하는 것으로 판단

```
SELECT id, title, body, MATCH(title, body) AGAINST('MySQL man' IN BOOLEAN MOD
FROM tb_bi_gram
WHERE MATCH(title, body) AGAINST('MySQL man' IN BOOLEAN MODE);
```

검색어에 포함된 단어 중 아무거나 하나라도 있으면 일치하는 것으로 판단

```
SELECT id, title, body, MATCH(title, body) AGAINST('sour*' IN BOOLEAN MODE) A
FROM tb_bi_gram
WHERE MATCH(title, body) AGAINST('sour*' IN BOOLEAN MODE);
```

와일드카드 문자를 이용해 패턴 검색 가능이 가능하다. 하지만 n-gram 인덱스에 대한 불리언 검색에서는 와일드카드가 아무런 효과를 가지지 않는다.

3. 검색어 확장(QUERY EXPANSION)

사용자가 쿼리에 사용한 검색어로 검색된 결과에서 공통으로 발견되는 단어들을 모아 다시 한 번 더 검색을 수행하는 방식

Blind query expansion 알고리즘을 사용하는데, 이는 검색어 결과에서 자주 사용된 단어들을 모아 다시 전문 검색 쿼리를 실행할 뿐이다.

1.3 전문 검색 인덱스 디버깅

```
SET GLOBAL innodb_ft_aux_table = 'test/tb_bi_gram';
```

`innodb_ft_aux_table` 시스템 변수에 전문 검색 인덱스를 가진 테이블이 설정되면 `information_schema` DB의 다음 테이블을 통해 전문 검색 인덱스가 어떻게 저장 및 관리되는지 확인할 수 있도록 한다.

- `information_schema.innodb_ft_config`: 전문 검색 인덱스의 설정 내용을 보여준다.
- `information_schema.innodb_ft_index_table`: 전문 검색 인덱스가 가지고 있는 인덱스 엔트리의 목록을 보여준다.
- `information_schema.innodb_ft_index_cache`: 테이블에 레코드가 새롭게 INSERT 되면 토큰을 분리해 즉시 디스크에 저장하지 않고 메모리에 임시로 저장하는데, 이 공간이 `innodb_ft_index_cache` 테이블이다. `innodb_ft_cache_size` 크기를 넘어서면 한꺼번에 모아 디스크의 파일로 저장한다.
- `information_schema.innodb_ft_deleted`, `innodb_ft_being_deleted`: 테이블의 레코드가 삭제되면 어떤 레코드가 삭제되었는지, 그리고 어떤 레코드가 현재 전문 검색 인덱스에서 삭제되고 있는지를 보여준다.

▼ 2. 공간 검색

2.1 용어 설명

- OGC(Open Geospatial Consortium): 위치 기반 데이터에 대한 표준을 수립하는 단체

  정부와 기업체, 학교 등 모든 기관이 자유롭게 가입 가능하다.
- OpenGIS: OGC에서 제정한 지리 정보 시스템(GIS, Geographic Information System) 표준

  WKT나 WKB 같은 지리 정보 데이터를 표기하는 방법과 저장하는 방법, 그리고 SRID와 같은 표준을 포함
- SRS(Spatial Reference System): 공간 참조 시스템으로, 우리가 흔히 이야기하는 좌표계(Coordinate System)라고 생각하면 된다.
  - GCS(Geographic Coordinate System): 지구 구체상의 특정 위치나 공간을 표현하는 좌표계(위도와 경도)
  - PCS(Projected Coordinate System): 구체 형태의 지구를 종이 지도와 같은 평면으로 투영(Projection)시킨 좌표계(meter와 같은 선형적 단위)
  - GCS는 위치(where), PCS는 어떻게(how) 표현할지가 관심사 (PCS는 GCS도 포함)
- SRID(Spatial Reference ID): 특정 SRS를 지칭하는 고유 번호

  SRS-ID라고도 표현하며, MySQL 서버에서는 SRS 고유 번호를 저장하는 컬럼의 이름으로는 SRS_ID라고 사용하고, 함수의 인자나 식별자로 사용될 경우 SRID를 사용한다.
- WKT(Well-Known Text format) • WKB(Well-Known Binary format): OGC에서 제정한 OpenGIS에 명시된 위치 좌표의 표현 방법이다. WKT는 사람의 눈으로 쉽게 확인 가능한 텍스트 포맷, WKB는 컴퓨터에 저장할 수 있는 형태의 이진 포맷의 저장 표준이다.
  WKT는

  `POINT(15 20)` 또는 `LINESTRING(0 0, 10 10, 20 25, 50 60)` 등과 같이 점이나 도형의 위치 정보를 정의하는 표준
  MySQL 서버가 내부적으로 사용하는 이진 데이터 포맷은 WKB 포맷과 흡사하지만 완전히 동일하지는 않음
- MBR(Minimum Bounding Rectangle): 어떤 도형을 감싸는 최소의 사각 상자

MtySQL 서버의 공간 인덱스(Spatial Index)가 도형의 포함 관계를 이용하기 때문이다. 이 도형들의 포함 관계를 이용해 만들어진 인덱스를 R-Tree라고 한다.

2.2 SRS(Spatial Reference System)

MySQL 서버에서 지원하는 SRS는 5000여 개가 넘는데, `information_schema` 데이터베이스의 `ST_SPATIAL_REFERENCE_SYSTEM` 테이블을 통해 확인 가능하다.

`ST_SPATIAL_REFERENCE_SYSTEM` 테이블에서 중요한 것은 `SRS_ID` 컬럼과 `DEFINITION` 컬럼이다.

`SRS_ID` 컬럼의 고유 번호는 컬럼에 데이터를 저장하거나 검색할 때 필요하기 때문에 사용자는 저장하는 공간 데이터가 어느 좌표계를 사용하는지 알고 있어야 한다.

`DEFINITION` 컬럼의 값은 항상 `GEOGCS` 또는 `PROJCS`로 시작한다.

- `GEOGCS` : 지리 좌표계(GCS) ⇒ MySQL 서버는 483개 지원
- `PROJCS` : 투영 좌표계(PCS) ⇒ MySQL 서버는 4668개 지원

`DEFINITION` 컬럼에서 하나 더 중요한 내용은 `AXIS` 필드이다.

```
mysql> SELECT *
    -> FROM information_schema.ST_SPATIAL_REFERENCE_SYSTEMS
    -> WHERE SRS_ID = 4326 \G
*************************** 1. row ***************************
                SRS_NAME: WGS 84
                  SRS_ID: 4326
            ORGANIZATION: EPSG
ORGANIZATION_COORDSYS_ID: 4326
              DEFINITION: GEOGCS["WGS 84",DATUM["World Geodetic System 1984",
                                  SPHEROID["WGS 84",6378137,298.257223563
                                  AUTHORITY["EPSG","7030"]],AUTHORITY["EP
                                  PRIMEM["Greenwich",0,AUTHORITY["EPSG","
                                  UNIT["degree",0.017453292519943278,AUTH
                                  AXIS["Lat",NORTH],AXIS["Lon",EAST],AUTH
             DESCRIPTION: NULL
1 row in set (0.00 sec)
```

위 예제에서 두 번의 `AXIS`가 표시되는데 위도와 경도 순서로 나열되어 있다. 따라서 WGS 84 좌표계를 사용하는 위치 정보에서 특정 위치를 표시할 때 `POINT(Latitude Longitude)` 형식으로 표현해야 한다. 나열된 순서대로 첫 번째 `AXIS`가 X 축, 두 번째 `AXIS`가 Y 축에 해당한다. 그래서 WGS 84 좌표를 사용하는 좌표에 대해 `ST_X()` 함수는 위도 값을 반환하며 `ST_Y()` 함수는 경도 값을 반환한다.

> 위도/경도를 사용하는 공간 좌표 시스템(SRS)에서는 `ST_X()`, `ST_Y()` 대신 `ST_Latitude()`, `ST_Longitude()` 함수를 사용할 수 있다. 하지만 `SRID`가 0인 `POINT` 데이터의 경우에는 `ST_X()`, `ST_Y()` 함수만 사용 가능하다.

```
mysql> SELECT *
    -> FROM information_schema.ST_SPATIAL_REFERENCE_SYSTEMS
    -> WHERE SRS_ID = 3857 \G
*************************** 1. row ***************************
                SRS_NAME: WGS 84 / Pseudo-Mercator
                  SRS_ID: 3857
            ORGANIZATION: EPSG
ORGANIZATION_COORDSYS_ID: 3857
              DEFINITION: PROJCS["WGS 84 / Pseudo-Mercator",
                                 GEOGCS["WGS 84",
                                 DATUM["World Geodetic System 1984",
                                 SPHEROID["WGS 84",6378137,298.257223563
                                 AUTHORITY["EPSG","6326"]],
                                 PRIMEM["Greenwich",0,AUTHORITY["EPSG","
                                 UNIT["degree",0.0174532925199433278,AUTH
                                 AXIS["Lat",NORTH],AXIS["Lon",EAST],AUTH
                                 PROJECTION["Popular Visualisation Pseud
                                 PARAMETER["Latitude of natural origin",
                                 PARAMETER["Longitude of natural origin"
                                 PARAMETER["False easting",0,AUTHORITY["
                                 PARAMETER["False northing",0,AUTHORITY[
                                 UNIT["metre",1,AUTHORITY["EPSG","9001"]
                                 AXIS["X",EAST],AXIS["Y",NORTH],AUTHORIT
             DESCRIPTION: NULL
1 row in set (0.00 sec)
```

SRID가 3857인 투영 좌표계이다. 이 좌표계의 이름도 WGS 84이다. 하지만 이 좌표계는 입력 값의 단위(UNIT)가 meter이다. AXIS 내용을 보면 SRID 4326과 반대로 `POINT(Longitude Latitude)` 순서로 명시해야 한다.

8.0 부터 공간 데이터를 저장할 때 SRID를 지정하거나 사용할 수 있도록 추가되었다. 별도로 명시하지 않으면 SRID 0으로 자동 인식된다.

SRID를 명시함으로써 공간 데이터 자체가 SRS에 대한 정보를 가지게 되면 MySQL 서버에서 제공되는 공간 함수들을 이용해 필요한 값을 즉시 계산할 수 있다. (거리를 계산하는 `ST_Distance()` 함수를 이용할 때 SRID 0인 공간 데이터를 이용하면 단순히 피타고라스 정리를 이용한 계산이지만, SRID 4326 공간 데이터를 이용하면 지구 구체 기반으로 거리를 계산)

> SRID 0 공간 데이터를 이용한다고 해서 실제 km나 meter 단위 계산을 절대 못 하는 것은 아니다. 그저 자동으로 필요한 값을 계산하지 못할 뿐이다. 8.0에서 지원하는 공간 함수들이 모두 SRID를 지원하는 것도 아니다. 아직 많은 공간 함수들이 SRID 0 공간 데이터에만 적용되므로 확인하고 사용하자.

2.3 투영 좌표계와 평면 좌표계

평면 좌표계는 단위를 가지지 않으며 X축과 Y축의 값 제한이 없다.(투영 좌표계와의 가장 큰 차이점) 무한 평면 좌표계라고도 한다.

> MySQL 서버에서 공간 데이터를 처리하는 작업은 대부분 함수를 통해 처리된다. 함수의 이름에 `ST_` 접두사가 붙은 함수들은 모두 MySQL 최근 버전부터 지원되기 시작한 함수이다. `ST_` 접두사를 가지는 함수들은 OpenGIS 표준을 준수해 만들어진 함수들이며 8.0 부터는 기존의 모든 함수를 사용하지 않고 `ST_` 접두사를 가진 함수만으로 필요한 작업을 모두 수행할 수 있다. `ST_` 접두사를 가지는 함수들만 사용하자.

`ST_PointFromText()` 함수를 이용해 `POINT(X Y)` 문자열(WKT)을 Geometry 타입 값으로 변환할 때 SRID를 명시하지 않으면 `SRID = 0`으로 설정된다.(평면 좌표계)

테이블을 생성할 때 SRID를 명시적으로 정의하지 않으면 해당 컬럼은 모든 SRID를 저장할 수 있다. 하지만 하나의 컬럼에 저장된 데이터의 SRID가 제각각이라면 MySQL 서버는 인덱스를 이용해 빠른 검색을 수행할 수 없게 된다.

공간 데이터를 그대로 출력하면 MySQL 서버가 내부적으로 사용하는 이진 포맷 데이터로 출력되고, `ST_AsWKB()` 함수를 사용해 출력하면 WKB 포맷의 공간 데이터로 출력된다. MySQL 서버 내부 이진 데이터 포맷은 WKB 앞쪽에 SRID를 위한 4바이트 공간이 추가된다.

`ST_AsText()` 함수를 이용해 이진 데이터를 WKT 포맷으로 변환하여 출력할 수 있다. `ST_SRID()` 함수를 통해 공간 데이터가 어떤 SRID를 사용하는지 확인할 수 있다.

지구(구면체)상의 두 점 간의 거리를 계산하기 위해서는 `ST_Distance_Sphere()` 함수를 사용하면 된다. 이 함수는 SRID 4326과 같은 지리 좌표에 대해서만 사용할 수 있다.

> 실제 애플리케이션에서 평면 좌표계는 잘 사용되지 않지만 투영 좌표계는 지도나 화면에 지도를 표현하는 경우 자주 사용된다.

2.4 지리 좌표계

1. 지리 좌표계 데이터 관리

지리 좌표계에서 두 점(POINT)의 거리는 `ST_DistanceSphere()` 함수를 이용하여 구할 수 있다. 미터(meter) 단위로 반환한다.

MySQL 서버는 아직 인덱스를 이용한 반경 검색 기능을 제공하지 않는다. 따라서 테이블 풀 스캔을 하게 되며, `ST_DistanceSphere()` 함수의 결과를 상수와 비교하는 형태는 인덱스를 사용할 수 없는 형태이다.

> **MySQL 서버가 지원하는 여러 형태의 점, 선, 다각형 데이터 타입**
>
> - `POINT` & `MULTIPOINT`
> -
> `LINESTRING` & `MULTILINESTRING`
> -

> `POLYGON` & `MULTIPOLYGON`
> -
> `GEOMETRY` & `GEOMETRYCOLLECTION`

차선책으로 MBR(Minimum Bounding Rectangle)을 이용하는 것이다. 첫 번째 파라미터의 공간 데이터가 두 번째 파라미터의 공간 데이터에 포함되는지 체크하는 함수이다.

MBR을 이용해 위치 검색 쿼리를 작성하려면 주어진 위치 기준으로 반경 `n`km의 원을 감싸는 사각형(MBR)을 만들어야 한다. POLYGON 객체로 중심 지점으로부터 특정 km 반경의 원을 둘러싸는 사각형을 생성하고 `ST_Contains()` 또는 `ST_Within()` 함수를 사용해 반경 검색을 실행할 수 있다.(두 함수의 파라미터 순서가 반대)

```
SELECT id, name
FROM sphere_coord
WHERE ST_CONTAINS(getDistanceMBR(ST_POINTFROMTEXT('POINT(37.547027 127.047337

SELECT id, name
FROM sphere_coord
WHERE ST_WITHIN(location, getDistanceMBR(ST_POINTFROMTEXT('POINT(37.547027 12
```

두 쿼리의 실행 계획은 모두 공간 인덱스에서 range 접근 방법으로 데이터를 읽는 것을 확인할 수 있다.

이 쿼리들은 1km 반경의 MBR 사각형 내의 점들을 검색하기 때문에 사각형의 모서리 부분은 1km를 넘어서는 부분도 결과에 포함될 수 있다. 이 부분을 보완하기 위해 8각형, 16각형으로 MBR을 생성하면 더 효율적으로 작동할 것이다. 인덱스를 통해 검색된 결과에서 한 번 더 거리 계산 조건을 적용하는 방법도 있다.

▼ `ST_Buffer()`

`ST_Buffer()` 함수는 주어진 위치에서 일정 거리만큼 떨어진 모든 점을 찾아 반환한다.

```
SET @origin = ST_GeomFromText('POINT(0 0)');
SET @pt_strategy = ST_Buffer_Strategy('point_circle', 8);

SELECT ST_AsText(ST_Buffer(@origin, 2, @pt_strategy)) AS bounding_circle;
```

8.0.24 이하 버전에서는 ST_Buffer() 함수가 평면 좌표계(SRID 0)만 지원한다.

2. 지리 좌표계 주의 사항

정확성 주의 사항

지리 좌표계에서 `ST_Contains()` 함수가 정확하지 않은 결과를 반환하는 오류가 있다. 간단히 거리 비교 조건을 추가해 문제를 회피할 수 있다.

```
SET @center:= ST_GeomFromText('POINT(37.398899 126.966064)', 4326);

SELECT (ST_Contains(@polygon, @point) AND ST_Distance_Sphere(@point, @center)
```

또한 풀 테이블 스캔으로 데이터를 조회할 때와 공간 인덱스를 사용해 조회할 때 `ST_Contains()` 또는 `ST_Within()`의 결과가 상이할 수 있다. 풀 테이블 스캔을 사용하는 경우 잘못된 결과가 검색될 수 있다.

성능 주의 사항

지리 좌표계는 투영 좌표계에 비해 `ST_Contains()` 함수 등으로 포함 관계를 비교하는 경우 느린 성능을 보인다. 거의 3배??

좌표계 변환

MySQL 서버에서 `ST_Transform()` 함수를 사용해 SRID를 변환할 수 있지만 SRID 3857에 대한 변환은 아직 지원되지 않는다. 투영 좌표계는 거리 계산 시 실제 구면체에서의 거리와 오차가 있기 때문에 `ST_Distance()` 함수의 결과가 실제 원하는 값이 아닐 수 있다. 이때 SRID 4326으로 변환하여 거리를 계산해야 할 수 있는데 다음과 같은 함수를 정의하여 사용할 수 있다.

```
/* SRID 4326 -> SRID 3857 */
CREATE
    DEFINER = 'root'@'%' FUNCTION convert4326To3857(p_4326 POINT) RETURNS POI
    DETERMINISTIC
    SQL SECURITY INVOKER
BEGIN
    DECLARE lon DOUBLE;
    DECLARE lat DOUBLE;
    DECLARE x DOUBLE;
    DECLARE y DOUBLE;

    /* Check SRID for the safety */
    IF ST_SRID(p_4326) = 3857 THEN
        RETURN p_4326;
    ELSEIF ST_SRID(p_4326) <> 4326 THEN
        SIGNAL SQLSTATE 'HY000' SET MYSQL_ERRNO = 1108, MESSAGE_TEXT = 'Incor
    END IF;

    SET lon = ST_Longitude(p_4326);
    SET lat = ST_Latitude(p_4326);

    -- // 20037508.34 미터 = 적도의 지구 둘레의 절반
    -- // = (π * EarthRadius) = (π * 6378137 meters)
    SET x = lon * 20037508.34 / 180;
    SET y = LOG(TAN((90 + lat) * PI() / 360)) / (PI() / 180);
    SET y = y * 20037508.34 / 180;

    RETURN ST_POINTFROMTEXT(CONCAT('POINT(', x, ' ', y, ')'), 3857);
END;;

//------------------------------------------------------------------------
```

```
/* SRID 3857 -> SRID 4326 */
CREATE
    DEFINER = 'root'@'%' FUNCTION convert3857To4326(p_3857 POINT) RETURNS POI
    DETERMINISTIC
    SQL SECURITY INVOKER
BEGIN
    DECLARE lon DOUBLE;
    DECLARE lat DOUBLE; DECLARE x DOUBLE; DECLARE y DOUBLE; /* Check SRID for
    IF ST_SRID(p_3857) = 4326 THEN
        RETURN p_3857;
    ELSEIF ST_SRID(p_3857) <> 3857 THEN
        SIGNAL SQLSTATE 'HY000' SET MYSQL_ERRNO = 1108, MESSAGE_TEXT = 'Incor
    END IF;

    SET x = ST_X(p_3857);
    SET y = ST_Y(p_3857);

    /* 적도의 지구 둘레의 절반 = (2 * π * R)/2 = (π * EarthRadius) = (π * 6378137
    SET lon = x * 180 / 20037508.34;
    SET lat = ATAN(EXP(y * PI() / 20037508.34)) * 360 / PI() - 90;

    RETURN ST_POINTFROMTEXT(CONCAT('POINT(', lat, ' ', lon, ')'), 4326);
END;;

//------------------------------------------------------------------------

/* SRID 3857 좌표의 두 위치 간 거리 계산(지구 구체면 상) */
CREATE
    DEFINER = 'root'@'%' FUNCTION distanceInSphere(p1_3857 POINT, p2_3857 POI
    DETERMINISTIC
    SQL SECURITY INVOKER
BEGIN
    DECLARE p1_4326 POINT;
    DECLARE p2_4326 POINT;
    SET p1_4326 = convert3857To4326(p1_3857);
    SET p2_4326 = convert3857To4326(p2_3857);

    -- // 6370986 = Default Radius meters in MySQL server
    RETURN ST_DISTANCE_SPHERE(p1_4326, p2_4326, 6370986);
END;;
```