# mysql certification exam study notes

exam number: 1z0-882

### chapter 11 / data types

spatial data

spatial data standard is defined in the *OpenGIS Implementation Standard for Geographic Information*

well known binary format

page 1346 ref manual 5.6

------------------------

Component Size Value

Byte order 1 byte 01

WKB type 4 bytes 01000000

X coordinate 8 bytes 000000000000F03F

Y coordinate 8 bytes 000000000000F0BF

0101000000000000000000F03F000000000000F0BF (above re-formatted)

page 1347 ref manual 5.6

------------------------

mysql> SET @g = ST\_GeomFromText('POINT(1 -1)');

mysql> SELECT LENGTH(@g);

+------------+

| LENGTH(@g) |

+------------+

| 25 |

+------------+

mysql> SELECT HEX(@g);

+----------------------------------------------------+

| HEX(@g) |

+----------------------------------------------------+

| 000000000101000000000000000000F03F000000000000F0BF |

+----------------------------------------------------+

page 1346 Ref Manual 5.6

------------------------

Component Size Value Value Bytes Data Type

--------------------------------------------------------------------------------------

Byte order 1 byte 01 1 INT UNSIGNED

WKB type 4 bytes 01000000 4 INT UNSIGNED

X coordinate 8 bytes 000000000000F03F 8 DOUBLE

Y coordinate 8 bytes 000000000000F0BF 8 DOUBLE

00000000011111111112222222222333333333344444444445

12345678901234567890123456789012345678901234567890

--------------------------------------------------

0101000000000000000000F03F000000000000F0BF [page 1346]

000000000101000000000000000000F03F000000000000F0BF [page 1347]

0000000001 11111111 12222222222333333 333344444444445

1234567890 12345678 90123456789012345 678901234567890

-----------------------------------------------------

01 01000000 000000000000F03F 000000000000F0BF

0000000001 01000000 000000000000F03F 000000000000F0BF

select p, hex(p) from geo where p is not null;

p |hex(p) [spaces added to separate components]

|byte order wkb type x coord y coord

------------------------|---------------------------------------------------

POINT (10 10) |0000000001 01000000 0000000000002440 0000000000002440

POINT (40 30) |0000000001 01000000 0000000000004440 0000000000003E40

POINT (1 -1) |0000000001 01000000 000000000000F03F 000000000000F0BF

POINT (140 807) |0000000001 01000000 0000000000806140 0000000000388940

POINT (1340 -7) |0000000001 01000000 0000000000F09440 0000000000001CC0

POINT (343 -234) |0000000001 01000000 0000000000707540 0000000000406DC0

POINT (-1031 -234) |0000000001 01000000 00000000001C90C0 0000000000406DC0

POINT (-33 -3432) |0000000001 01000000 00000000008040C0 0000000000D0AAC0

POINT (33 -12312312313) |0000000001 01000000 0000000000804040 0000C8CFF7EE06C2

Conclusions:

1. byte order
   1. byte order is always 1. (this is true for all geometry types.)
2. wkb type
   1. 1 ≤ wkb type ≤ 7
      1. point=1, linestring=2, polygon=3, multipoint=4, multilinestring=5, multipolygon=6, geometrycollection=7
   2. mysql uses 'little endian'; least significant (pair) of hex digits is on the left instead of the right.
      1. so decimal 1 in two digit little endian = 01000000 instead of 00000001
3. x coord and y coord
   1. these are (ieee floating point representations of) the x- and y- coordinates.
   2. mode complicated geometries will have > 1 pair of x- and y-coordinates since they are composed of muliple coordinate pairs.
   3. Example below:

mp |hex(mp)

---------------------------------------|--------------------------------------------------------------------------------------------------------------

MULTIPOINT ((10 10), (20 20)) |00000000010400000002000000010100000000000000000024400000000000002440010100000000000000000034400000000000003440

Examples

--------

POINT (10 10)

LINESTRING (0 0, 4 5, 5 4, 5 6)

POLYGON (

(0 0, 10 0, 10 10, 0 10, 0 0), /\* outer ring \*/

(5 5, 6 5, 6 6, 5 6, 5 5) /\* inner ring. polygons must have ≥ 1 inner ring \*/

)

MULTIPOINT ((10 10), (20 20), (30 30))

MULTILINESTRING (

(10 10, 20 20, 30 30),

(0 0, 1 1, 2 2),

(100 100, 200 200, 300 300)

)

MULTIPOLYGON (

(

(0 0, 10 0, 10 10, 0 0), (5 5, 6 5, 6 6, 5 5)

),

(

(0 0, 100 0, 100 100, 0 0), (50 50, 60 50, 60 60, 50 50)

)

)

GEOMETRYCOLLECTION (POINT (0 0), LINESTRING (10 10, 20 20, 30 30))

sample table

------------

Field |Type |Null |Key |Default |Extra

------|-------------------|-----|----|--------|------

id |int(10) unsigned |NO | |0 |

p |point |YES | | | /\* dimension = 0 \*/

ls |linestring |YES | | | /\* dimension = 1 \*/

pg |polygon |YES | | | /\* dimension = 2 \*/

mp |multipoint |YES | | | /\* dimension = 0 \*/

mls |multilinestring |YES | | | /\* dimension = 1 \*/

mpg |multipolygon |YES | | | /\* dimension = 2 \*/

gc |geometrycollection |YES | | |

spatial functions

-----------------

**set** @p = ST\_GeomFromText('POINT(10 20)'); /\* most versatile; can be used for any geometry \*/

**select** AsText(@p), X(@p), Y(@p), **hex**(@p), **length**(@p), AsBinary(@p);

AsText(@p) |X(@p) |Y(@p) |hex(@p) |length(@p) |AsBinary(@p)

-------------|------|------|---------------------------------------------------|-----------|----------------------

POINT(10 20) |10 |20 |00000000010100000000000000000024400000000000003440 |25 | $@ 4@

**select** PointFromText('POINT (10 10)');

**select** LinestringFromText('LINESTRING (0 0, 4 5, 5 4, 5 6)');

**select** PolygonFromText('POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0), (5 5, 6 5, 6 6, 5 6, 5 5))');

**select** MultipointFromText('MULTIPOINT ((10 10), (20 20), (30 30))');

**select** MultiLinestringFromText('MULTILINESTRING ((10 10, 20 20, 30 30), (0 0, 1 1, 2 2), (100 100, 200 200, 300 300))');

**select** MultiPolygonFromText('MULTIPOLYGON (((0 0, 10 0, 10 10, 0 0), (5 5, 6 5, 6 6, 5 5)), ((0 0, 100 0, 100 100, 0 0), (50 50, 60 50, 60 60, 50 50)))');

**select** GeomCollFromText('GEOMETRYCOLLECTION (POINT (0 0), LINESTRING (10 10, 20 20, 30 30))');

§ 12.15 gives a description of dozens of spatial functions. A few of the signficant ones:

**select** AsText(@ls), endpoint(@ls), GeometryType(@ls), GLength(@ls);

AsText(@ls) |endpoint(@ls) |GeometryType(@ls) |GLength(@ls)

----------------------------|--------------|------------------|------------------

LINESTRING(0 0,4 5,5 4,5 6) |POINT (5 6) |LINESTRING |9.817337799805944

**select** AsText(@ls2), GLength(@ls2);

AsText(@ls2) |GLength(@ls2)

----------------------------|--------------

LINESTRING(0 0,1 0,1 1,0 1) |3

**select** AsText(@ls), AsText(@ls2), Intersects(@ls, @ls2);

AsText(@ls) |AsText(@ls2) |Intersects(@ls, @ls2)

----------------------------|----------------------------|---------------------

LINESTRING(0 0,4 5,5 4,5 6) |LINESTRING(0 0,1 0,1 1,0 1) |1

**select** AsText(@ls2), IsClosed(@ls2);

AsText(@ls2) |IsClosed(@ls2)

----------------------------|---------------

LINESTRING(0 0,1 0,1 1,0 1) |0

**select** AsText(@pg), centroid(@pg), dimension(@pg), ExteriorRing(@pg);

AsText(@pg) |centroid(@pg) |dimension(@pg) |ExteriorRing(@pg)

---------------------------------------------------------|--------------------------------------------|---------------|-----------------------------------------

POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,6 5,6 6,5 6,5 5)) |POINT (4.994949494949495 4.994949494949495) |2 |LINESTRING (0 0, 10 0, 10 10, 0 10, 0 0)

spatial data indices

--------------------

**alter** **table** points **add** **spatial** **index**(p);

[p must be defined as NOT NULL]

### stored prodecures and functions

| **ID** | **Construct** | **Syntax** |
| --- | --- | --- |
|  | IF / ELSE IF[[1]](#footnote-1) / ELSE | IF cond THEN  ⋯  ELSEIF cond THEN  ⋯  ELSE  ⋯  END IF; |
|  | LOOP / END LOOP | my\_loop label:  LOOP  ⋯  END LOOP my\_loop\_label; |
|  | WHILE / END WHILE | while1:  **WHILE** **NOT** eof DO  ⋯  **END** **WHILE** while1; |
|  | REPEAT ⋯ UNTIL | [my\_repeat\_label:]  REPEAT  ⋯  UNTIL expr  END REPEAT [my\_repeat\_label]; |
|  | SELECT INTO | SELECT ⋯ INTO my\_var; |
|  | CURSOR | **DELIMITER //**  **DROP** **PROCEDURE** **IF** **EXISTS** demo\_sp//  **CREATE** **PROCEDURE** demo\_sp()  **BEGIN**  **DECLARE** eof **INT** **DEFAULT** 0; /\* assume eof false \*/  **DECLARE** j **INT** **DEFAULT** 0;  **DECLARE** my\_cursor **CURSOR** **FOR** **SELECT** i **FROM** random\_ints **limit** 3;  **DECLARE** **CONTINUE** HANDLER[[2]](#footnote-2) **FOR** **NOT** **FOUND** **SET** eof = 1;    **OPEN** my\_cursor;    while1:  **WHILE** **NOT** eof DO    **FETCH** my\_cursor **INTO** j;    /\* this IF statement is required;  if omitted, the last row in the table will be shown twice. \*/  **IF** eof **THEN**  **LEAVE** while1;  **END** **IF**;    **SELECT** j;    **END** **WHILE** while1;    **CLOSE** my\_cursor;  **END**//  **DELIMITER ;** |
|  | IN / OUT / INOUT | CREATE PROCEDURE|FUNCTION(IN x1 INT, OUT x2 INT, INOUT x3 INT); |
|  | CASE | CASE expr  WHEN value1 THEN  ⋯;  WHEN value2 THEN  ⋯;  ELSE  ⋯;  END CASE; |
|  | ITERATE[[3]](#footnote-3) | my\_loop label:  LOOP  ⋯  IF cond THEN  ITERATE my\_loop\_label;  END IF;  ⋯  END LOOP my\_loop\_label; |
|  | ERROR HANDLER WITH ROLLBACK | drop procedure if exists rollback\_demo;  delimiter //  create procedure rollback\_demo()  begin  declare exit handler for 1062  begin  select 'rollback!';  rollback;  end;    /\*  mysql [localhost] [stored\_procs] select \* from aircraft order by model;  +----+-------+--------------+  | id | model | name |  +----+-------+--------------+  | 10 | 300 | Airbus A-300 |  | 2 | 340 | Airbus A-340 |  | 3 | 727 | Boeing 727 |  | 9 | 737 | Boeing 737 |  | 1 | 757 | Boeing 757 |  +----+-------+--------------+    The 747 is not in the table but the 757 is.  Neither of the inserts below will work; the 747 insert will succeed at first.  But the 757 will fail since it is already in the table.  That will fire the exit handler and the transaction will be rolled back.  \*/  start transaction;  insert into aircraft(model, name) values (747, 'Boeing 747');  select 'after insert 1';  insert into aircraft(model, name) values (757, 'Boeing 757-200');  select 'after insert 2';  commit;  end//  delimiter ; |

# sample stored **procedure** **with** **cursor**

delimiter //

**CREATE** **PROCEDURE** mysql\_programming.update\_usage()

**BEGIN**

**DECLARE** d1 **date**;

**DECLARE** d2 **date**;

**DECLARE** r1 **bigint**;

**DECLARE** r2 **bigint**;

**DECLARE** c1 **CURSOR** **FOR** **SELECT** read\_date, reading **FROM** reading\_f;

**FETCH** c1 **INTO** d1, r1;

loop1:

LOOP

**FETCH** c1 **INTO** d2, r2;

**IF** **cast**(d2 **AS** **date**)-**cast**(d1 **AS** **date**) = 1 **THEN**

**INSERT** IGNORE **INTO** usage\_f(usage\_date, usage\_amount) **values**(d1, r2-r1);

**END** **IF**;

**SET** d1 = d2;

**SET** r1 = r2;

**END** LOOP loop1;

**CLOSE** c1;

**SELECT** \* **FROM** usage\_f;

**END**;//

delimiter ;

# another **cursor** example

delimiter //

**CREATE** **PROCEDURE** sp\_example4()

**BEGIN**

**DECLARE** done **INT** **DEFAULT** **FALSE**;

**DECLARE** vr\_name **VARCHAR**(100);

**DECLARE** cur1 **CURSOR** **FOR** **SELECT** username **FROM** users;

**DECLARE** **CONTINUE** HANDLER **FOR** **NOT** **FOUND** **SET** done = **TRUE**;

-- now we open the cursor

**OPEN** cur1;

read\_loop: LOOP

**FETCH** cur1 **INTO** vr\_name;

**IF** done **THEN**

LEAVE read\_loop;

**END** **IF**;

-- now do something with the return, vr\_name.

**END** LOOP;

**CLOSE** cur1;

**END**;//

delimiter ;

# sample **procedure** **with** while loop

delimiter //

**CREATE** **PROCEDURE** create\_flight\_data()

**BEGIN**

**DECLARE** i **int**;

**SET** i = 1;

WHILE i <= 1000 DO

**INSERT** IGNORE flights\_d(airline\_id, aircraft\_id, departure\_id, arrival\_id, flight\_no) **values**(

rand\_between(1, 32),

rand\_between(1, 18),

rand\_between(1, 4431),

rand\_between(1, 16),

rand\_between(1, 1000));

**SET** i = i + 1;

**END** WHILE;

**END**;//

delimiter ;

# sample **function**

**CREATE** **FUNCTION** get\_column\_value(str text, d **varchar**(1), n **int**)

**RETURNS** text

**DETERMINISTIC**

**BEGIN**

**DECLARE** start\_pos **int**;

**DECLARE** end\_pos **int**;

**SET** start\_pos = **LENGTH**(SUBSTRING\_INDEX( str, d, n ))+ 2;

**SET** end\_pos = **LENGTH**(SUBSTRING\_INDEX( str, d, n + 1 ))+ 1;

**IF** n = 0 **THEN**

**RETURN** **left**(str, end\_pos-1);

**ELSE**

**RETURN** **SUBSTRING**( str, start\_pos, end\_pos - start\_pos );

**END** **IF**;

**END**;

# **select** **into** a variable **in** a stored **procedure**

**CREATE** **PROCEDURE** sp\_example2()

**BEGIN**

**DECLARE** my\_var **VARCHAR**(511);

**SELECT** username **FROM** users **WHERE** id=4 **INTO** my\_var;

**END**;

Demo of GET DIAGNOSTICS

-----------------------

mysql [localhost] [study\_db] use study\_db;

mysql [localhost] [study\_db] insert into customers(customer) values(NULL);

ERROR 1048 (23000): Column 'customer' cannot be null

mysql [localhost] [study\_db] get diagnostics condition 1 @r=RETURNED\_SQLSTATE, @e=MYSQL\_ERRNO, @t=MESSAGE\_TEXT;

SELECT @r AS RETURNED\_SQLSTATE, @e AS MYSQL\_ERRNO, @t AS MESSAGE\_TEXT;

+-------------------+-------------+----------------------------------+

| RETURNED\_SQLSTATE | MYSQL\_ERRNO | MESSAGE\_TEXT |

+-------------------+-------------+----------------------------------+

| 23000 | 1048 | Column 'customer' cannot be null |

+-------------------+-------------+----------------------------------+

1 row in set (0.00 sec)

mysql [localhost] [study\_db] drop table xyz;

ERROR 1051 (42S02): Unknown table 'study\_db.xyz'

mysql [localhost] [study\_db] get diagnostics condition 1 @r=RETURNED\_SQLSTATE, @e=MYSQL\_ERRNO, @t=MESSAGE\_TEXT;

SELECT @r AS RETURNED\_SQLSTATE, @e AS MYSQL\_ERRNO, @t AS MESSAGE\_TEXT;

+-------------------+-------------+------------------------------+

| RETURNED\_SQLSTATE | MYSQL\_ERRNO | MESSAGE\_TEXT |

+-------------------+-------------+------------------------------+

| 42S02 | 1051 | Unknown table 'study\_db.xyz' |

+-------------------+-------------+------------------------------+

1 row in set (0.00 sec)

mysql [localhost] [study\_db] set @g=ST\_GeomFromText('POINT(10, 10)');

ERROR 3037 (22023): Invalid GIS data provided to function st\_geometryfromtext.

mysql [localhost] [study\_db] get diagnostics condition 1 @r=RETURNED\_SQLSTATE, @e=MYSQL\_ERRNO, @t=MESSAGE\_TEXT;

SELECT @r AS RETURNED\_SQLSTATE, @e AS MYSQL\_ERRNO, @t AS MESSAGE\_TEXT;

+-------------------+-------------+------------------------------------------------------------+

| RETURNED\_SQLSTATE | MYSQL\_ERRNO | MESSAGE\_TEXT |

+-------------------+-------------+------------------------------------------------------------+

| 22023 | 3037 | Invalid GIS data provided to function st\_geometryfromtext. |

+-------------------+-------------+------------------------------------------------------------+

1 row in set (0.00 sec)

delimiter //

create PROCEDURE junk()

begin

declare continue handler for 1062

begin

get diagnostics condition 1 @r=RETURNED\_SQLSTATE, @e=MYSQL\_ERRNO, @t=MESSAGE\_TEXT;

SELECT @r AS RETURNED\_SQLSTATE, @e AS MYSQL\_ERRNO, @t AS MESSAGE\_TEXT;

end;

get diagnostics condition 1 @r=RETURNED\_SQLSTATE, @e=MYSQL\_ERRNO, @t=MESSAGE\_TEXT;

SELECT @r AS RETURNED\_SQLSTATE, @e AS MYSQL\_ERRNO, @t AS MESSAGE\_TEXT;

insert into customers(customer) values('Bluefin');

end;//

delimiter ;

mysql [localhost] [study\_db] insert into customers(customer) values(NULL);

ERROR 1048 (23000): Column 'customer' cannot be null

mysql [localhost] [study\_db] get diagnostics condition 1 @r=RETURNED\_SQLSTATE, @e=MYSQL\_ERRNO, @t=MESSAGE\_TEXT;

Query OK, 0 rows affected (0.00 sec)

mysql [localhost] [study\_db] SELECT @r AS RETURNED\_SQLSTATE, @e AS MYSQL\_ERRNO, @t AS MESSAGE\_TEXT;

+-------------------+-------------+----------------------------------+

| RETURNED\_SQLSTATE | MYSQL\_ERRNO | MESSAGE\_TEXT |

+-------------------+-------------+----------------------------------+

| 23000 | 1048 | Column 'customer' cannot be null |

+-------------------+-------------+----------------------------------+

1 row in set (0.00 sec)

mysql [localhost] [study\_db] call junk();

+-------------------+-------------+----------------------------------+

| RETURNED\_SQLSTATE | MYSQL\_ERRNO | MESSAGE\_TEXT |

+-------------------+-------------+----------------------------------+

| 23000 | 1048 | Column 'customer' cannot be null |

+-------------------+-------------+----------------------------------+

1 row in set (0.00 sec)

+-------------------+-------------+----------------------------------------------+

| RETURNED\_SQLSTATE | MYSQL\_ERRNO | MESSAGE\_TEXT |

+-------------------+-------------+----------------------------------------------+

| 23000 | 1062 | Duplicate entry 'Bluefin' for key 'customer' |

+-------------------+-------------+----------------------------------------------+

1 row in set (0.02 sec)

Query OK, 0 rows affected (0.03 sec)

mysql [localhost] [study\_db] get diagnostics condition 1 @r=RETURNED\_SQLSTATE, @e=MYSQL\_ERRNO, @t=MESSAGE\_TEXT;

Query OK, 0 rows affected, 1 warning (0.00 sec)

Error (Code 1758): Invalid condition number

mysql [localhost] [study\_db] SELECT @r AS RETURNED\_SQLSTATE, @e AS MYSQL\_ERRNO, @t AS MESSAGE\_TEXT;

+-------------------+-------------+----------------------------------------------+

| RETURNED\_SQLSTATE | MYSQL\_ERRNO | MESSAGE\_TEXT |

+-------------------+-------------+----------------------------------------------+

| 23000 | 1062 | Duplicate entry 'Bluefin' for key 'customer' |

+-------------------+-------------+----------------------------------------------+

### transaction management [chapter 7, Harrison and Feuerstein]

Transaction Isolation Levels

Isolation Level: Defines the degree to which transactions in one session can can see or access data in a different session. Isolations levels are tradeoffs between *concurrency* (ability to to many things are once) and *consistency* (you can see correct data even when a lot of other things are going on).

READING UNCOMMITED: AKA *dirty read*. Transactions can read uncommmited rows. Fast but risky.

READ COMMITED: Transaction can only read committed rows. Any changed in a different session are not visible.

REPEATABLE READ: No changes to the database made in a different session are visible until the transaction is commited or rolled back. This is the default setting.)

SERIALIZABLE: Tranactions are completely isolated, and essentially run serially. Slow but 100% safe.

To set the isolation level: SET ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}

Starting a Transaction

start transation;

[⋮](https://en.wiktionary.org/wiki/%E2%8B%AE)

savepoint my\_savepoint\_name;

[⋮](https://en.wiktionary.org/wiki/%E2%8B%AE)

rollback to my\_savepoint\_name;

[⋮](https://en.wiktionary.org/wiki/%E2%8B%AE)

/\* end the transaction \*/

[commit; | roolback;]

Best practice: Do you assume that autocommit is disabled. Use START TRANSACTION in any stored programs when you need transactions.

**ACID**

Atomicity. In a transaction involving two or more discrete pieces of information, either all of the pieces are committed or none are. ('all or none')

Consistency. A transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started.

Isolation. A transaction in process and not yet committed must remain isolated from any other transaction.

Durability. Committed data is saved by the system such that, even in the event of a failure and system restart, the data is available in its correct state.

**Transactions and Locking**

START TRANSATION;

SELECT … FOR UPDATE; /\* lock the rows to you want to update to minimize deadlocks \*/

UPDATE …

COMMIT or ROLLBACK;

### crosstab

session\_a@localhost [study\_db] desc xtab;

+-------+---------------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-------+---------------------+------+-----+---------+----------------+

| id | bigint(20) unsigned | NO | PRI | NULL | auto\_increment |

| c | char(1) | YES | | NULL | |

| n | tinyint(4) | YES | | NULL | |

+-------+---------------------+------+-----+---------+----------------+

session\_a@localhost [study\_db] select \* from xtab;

+----+------+------+

| id | c | n |

+----+------+------+

| 1 | A | 1 |

| 2 | A | 2 |

| 3 | A | 5 |

| 4 | B | 6 |

| 5 | B | 7 |

| 6 | C | 8 |

+----+------+------+

select

sum(case

when c = 'A' then

n

else

NULL

end) as sum\_A,

sum(case

when c = 'B' then

n

else

NULL

end) as sum\_B,

sum(case

when c = 'C' then

n

else

NULL

end) as sum\_C

from xtab;

sum\_A |sum\_B |sum\_C

------|------|------

8 |13 |8

# subqueries

ALL / ANY aka SOME

Subqueries with ALL / ANY aka SOME can cimplify joins. (Sometimes not.)

|  |  |
| --- | --- |
| products  --------  productid  productname  supplierid  categoryid  unit  price | orderdetails  ------------  orderdetailid  orderid  productid  quantity |

Show products if there are any records in orderdetails where quantity = 10:

SELECT productname

FROM products

WHERE productid = ANY (

SELECT productid

FROM orderdetails

WHERE quantity = 10

);

Comments

1. This could have been done with a JOIN.
2. But joins can be avoided here; note that products and orderdetails do not have to be joined.
3. Note there WHERE CLAUSE: productid = ANY()…
   1. Use an operator (=, <, ≥, etc.) DO NOT USE 'IN'.
4. You will need ALL or ANY / SOME if you need to look at multiple rows for which a scalar value is ultimately required.

Another Example

---------------

|  |  |
| --- | --- |
| countries  --------  id  continent  country | languages  ---------  id  country\_id  language |

Show countries on European continent for which the country is in the worldwide list of countries in which Spanish is spoken. (Contrived example):

SELECT country

FROM countries

JOIN languages on countries.id = languages.country\_id

WHERE continent = 'Europe'

AND id = ANY (

SELECT country\_id

FROM languages

WHERE language = 'Spanish'

);

# views

Syntax:

CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] VIEW view\_name AS

SELECT …

[WITH {CASCADED | LOCAL} CHECK OPTION]

| Algorithm | Description | Updatable? |
| --- | --- | --- |
| UNDEFINED | MySQL will choose the algorithm. This is the default. |  |
| MERGE | Converts the view to a corresponding SQL statement and executes that. Requires there ∃ a 1:1 relationship between the view and its underlying table.  Example of a Legal MERGE View  create algorithm=merge view v\_good\_merge as  select id, model, name from aircraft;  CREATE ALGORITHM=MERGE DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `v\_good\_merge` AS select `aircraft`.`id` AS `id`,`aircraft`.`model` AS `model`,`aircraft`.`name` AS `name` from `aircraft`  This view could be executed as:  select id, model, name from aircraft;  Example of an Illegal MERGE View  create algorithm=merge view v\_bad\_merge as  select manf, count(\*) from aircraft group by manf;  Warning (Code 1354): View merge algorithm can't be used here for now (assumed undefined algorithm)  CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `v\_bad\_merge` AS select `aircraft`.`manf` AS `manf`,count(0) AS `count(\*)` from `aircraft` group by `aircraft`.`manf`;  The SELECT statement in this view definition does not have a 1:1 relation to aircraft RE: the count(\*) function. Therefore, cannot use a MERGE algorithm in this case.  See example of this on page 248 in 5.0 Certification Guide. | Yes |
| TEMPTABLE | Creates an intermediate temporary table that is used to finish executing the staement.  Renders the view non-updateable since it would be the intermediate temporary table that would get updated instead of the underlying table. | No |

1. ELSEIF has no space. [↑](#footnote-ref-1)
2. Handlers must be declared after cursors. [↑](#footnote-ref-2)
3. Like a goto. [↑](#footnote-ref-3)