

CE 223 – DATABASE MANAGEMENT AND USAGE OF MySQL HELP DOCUMENT

SECTION 1. Basic Queries in MySQL

There are some basic queries and keywords you will use with MySQL and this section covers these topics. You can see [this link](#) for details.

a. ‘SELECT ... FROM ... WHERE ...’ Keyword Group

In MySQL, if you want to read some data from database(DB) tables, you always need to use “**SELECT** ... **FROM** ...” keywords. If you have also a condition to check, then you also need to add “**WHERE** ...” keyword after those two keywords. A simple example can be seen from QUERY 1.

QUERY 1:

```
SELECT c.Name, c.Region, c.GovernmentForm
FROM country c
WHERE Continent = 'Africa';
```

In this query, we want to read ‘name’, ‘region’ and ‘government form’ information from COUNTRY table if the corresponding continent for these tuples is Africa. You can see a little part of the result set from below.

RESULT 1:

	Name	Region	GovernmentForm
1	Angola	Central Africa	Republic
2	Burundi	Eastern Africa	Republic
3	Benin	Western Africa	Republic
4	Burkina Faso	Western Africa	Republic
5	Botswana	Southern Africa	Republic
6	Central African Republic	Central Africa	Republic
7	Côte d'Ivoire	Western Africa	Republic
8	Cameroon	Central Africa	Republic
9	Congo, The Democratic Republic of the	Central Africa	Republic
10	Congo	Central Africa	Republic
11	Comoros	Eastern Africa	Republic
12	Cape Verde	Western Africa	Republic
13	Djibouti	Eastern Africa	Republic
14	Algeria	Northern Africa	Republic

b. 'AND', 'OR' and 'BETWEEN' Keywords

You can use logical operators in MySQL if you want to combine some conditions together. Most used ones are **AND**, **OR** and **BETWEEN** keywords. You can see an example usage from QUERY 2.

QUERY 2:

```
SELECT c.Name, c.Continent, c.Population
FROM country c
WHERE (c.Population BETWEEN 10000 AND 1000000)
AND
(c.Continent = 'Asia' OR
c.Continent = 'Europe');
```

RESULT:

	Name	Continent	Population
1	Andorra	Europe	78000
2	Bahrain	Asia	617000
3	Brunei	Asia	328000
4	Cyprus	Asia	754700
5	Faroe Islands	Europe	43000
6	Gibraltar	Europe	25000
7	Iceland	Europe	279000
8	Liechtenstein	Europe	32300
9	Luxembourg	Europe	435700
10	Macao	Asia	473000
11	Monaco	Europe	34000
12	Maldives	Asia	286000
13	Malta	Europe	380200
14	Qatar	Asia	599000

c. 'DISTINCT', 'COUNT', 'MIN/MAX', 'SUM' and 'AVG' Keywords

In MySQL, **DISTINCT** keyword is used to fetch only distinct (not same) data. **COUNT** keyword is used as a function to count the rows. **MIN** function returns the minimum value and **MAX** function returns the maximum value. **SUM** is the summation of all values and **AVG** is the average value of the selected rows. The examples we gave here are very simple; but these keywords can also be used with conditions also.

QUERY GROUP 3:

```
SELECT DISTINCT Language
FROM countrylanguage;
```

```

SELECT COUNT(Language)
FROM countrylanguage;

SELECT MAX(cl.Percentage)
FROM countrylanguage cl;

SELECT SUM(cl.Percentage)
FROM countrylanguage cl;

SELECT AVG(cl.Percentage)
FROM countrylanguage cl;

```

There can be duplicate names for languages in `COUNTRYLANGUAGE` table, but when you use `DISTINCT`, it only selects and shows the distinct ones. `COUNT` function gives the value 984 as the result, but if we have used the keywords like `COUNT(DISTINCT ...)`, then it gives us the value 457. `SUM` function sums up all of the percentage values in the table and shows the result.

RESULTS:

457 rows	
Language	
1 Dutch	
2 English	
3 Papiamentu	
4 Spanish	
5 Balochi	
6 Dari	
7 Pashto	
8 Turkmenian	
9 Uzbek	
10 Ambo	
11 Chokwe	
12 Kongo	
13 Luchazi	
14 Luimbe-nganguela	

1 row	
1	984

1 row	
1	100

1 row	
1	20048.4

1 row	
1	20.37439

d. 'ORDER BY ... ASC/DESC' Keyword

If you want to order the data you read from DB, you need to use `ORDER BY ...` keyword. From QUERY 4 example, it can be seen that the data which is read from DB is id, name, country code and population of the cities whose populations are less than 15000. And the data is ordered by the country code descending. `ASC` keyword after `ORDER BY` is used for ordering the data ascending; and `DESC` keyword after `ORDER BY` is used for ordering the data descending. If you have not written `ASC` or `DESC`, the default ordering scheme for `ORDER BY` is ascending.

QUERY 4:

```
SELECT ci.ID, ci.Name, ci.CountryCode, ci.Population
FROM city ci
WHERE ci.Population < 15000
ORDER BY ci.CountryCode DESC;
```

RESULT:

51 rows					Tx: Manual				
	ID	Name	CountryCode	Population					
1	3536	Mata-Utu	WLF	1137					
2	4067	Charlotte Amalie	VIR	13000					
3	537	Road Town	VGB	8000					
4	3538	Citt� del Vaticano	VAT	455					
5	3424	Funafuti	TUV	4600					
6	3333	Fakaofo	TKL	300					
7	3423	Cockburn Town	TCA	4800					
8	3067	Saint-Pierre	SPM	5808					
9	3171	San Marino	SMR	2294					
10	3170	Serravalle	SMR	4802					
11	938	Longyearbyen	SJM	1438					
12	3063	Jamestown	SHN	1500					
13	2881	Koror	PLW	12000					
14	2912	Adamstown	PCN	42					

e. 'GROUP BY ... HAVING ...' Keyword

If you want to group some data which you read from DB according to some criteria, you need to use **GROUP BY** keyword. The attribute you used after this keyword is your ordering criteria ('region' for QUERY 5). You use **HAVING** keyword if you have some condition for the group you have selected. For QUERY 5, we have grouped the countries according to their regions in Africa continent and we have selected these groups if they have more than 5 countries.

QUERY 5:

```
SELECT COUNT(c.Code) AS number_of_countries, c.Region
FROM country c
WHERE Continent = 'Africa'
GROUP BY c.Region
HAVING COUNT(c.Code) > 5;
```

RESULT:

	number_of_countries	Region
1	9	Central Africa
2	20	Eastern Africa
3	7	Northern Africa
4	17	Western Africa

Usage of **DISTINCT** does not affect **GROUP BY**, because it already groups the data distinctly by using grouping criteria. You usually use **COUNT** function with **GROUP BY ...** keyword.

QUERY 6:

```
SELECT DISTINCT ci.CountryCode, COUNT(ci.CountryCode) AS CITY_COUNT
FROM city ci
GROUP BY ci.CountryCode;
```

RESULT:

232 rows		
	CountryCode	CITY_COUNT
1	ABW	1
2	AFG	4
3	AGO	5
4	AIA	2
5	ALB	1
6	AND	1
7	ANT	1
8	ARE	5
9	ARG	57
10	ARM	3
11	ASM	2
12	ATG	1
13	AUS	14
14	AUT	6

f. 'IN' and 'LIKE' Keywords

IN keyword returns some values if the corresponding value list in parentheses includes these values. It can also be used for subqueries to detect if the certain values which return from the subquery is in that certain range of values. It can be used in queries instead of multiple **OR** conditions. **LIKE** keyword determines if a character string matches a pattern. You can use percent sign '%' or underscore '_' as wildcard characters. '%' matches any string with zero or more characters, '_' matches only a single character. So if you use '___s%' after **LIKE** keyword; it matches the string that has three characters before character 's' and has zero or more characters afterwards; such as Alaste, Merst, Dors, Kulskado, etc. In QUERY 7, we selected the names and the country codes from **CITY** table where the city name has 'ka' string somewhere in it but a single character comes after this 'ka' string and also the country code is 'IND' or 'DEU'.

QUERY 7:

```
SELECT ci.Name, ci.CountryCode
FROM city ci
WHERE ci.Name LIKE '%ka_'
AND ci.CountryCode IN ('IND', 'DEU');
```

RESULT:

	Name	CountryCode
1	Sikar	IND
2	Kulti-Barakar	IND
3	Guntakal	IND
4	Zwickau	DEU
5	Esslingen am Neckar	DEU

SECTION 2. Usage of Subqueries and Joins in MySQL

-TBD-

SECTION 3. Table Creation and Alteration, Keys and Constraints in MySQL

-TBD-

```
CREATE TABLE country
(
  Code          CHAR(3) DEFAULT '' NOT NULL PRIMARY KEY,
  Name          CHAR(52) DEFAULT '' NOT NULL,
  Continent     ENUM ('Asia', 'Europe', 'North America', 'Africa', 'Oceania',
'Antarctica', 'South America') DEFAULT 'Asia' NOT NULL,
  Region        CHAR(26) DEFAULT '' NOT NULL,
  SurfaceArea   FLOAT(10, 2) DEFAULT '0.00' NOT NULL,
  IndepYear     SMALLINT(6) NULL,
  Population    INT DEFAULT '0' NOT NULL,
  LifeExpectancy FLOAT(3, 1) NULL,
  GNP           FLOAT(10, 2) NULL,
  GNPOld        FLOAT(10, 2) NULL,
  LocalName     CHAR(45) DEFAULT '' NOT NULL,
  GovernmentForm CHAR(45) DEFAULT '' NOT NULL,
  HeadOfState   CHAR(60) NULL,
  Capital       INT NULL,
  Code2         CHAR(2) DEFAULT '' NOT NULL
);
```

SECTION 4. Trigger Usage in MySQL

```
USE world;

/* This is a trigger for checking if before updating a city, the city population is not
greater than the country population where this city belongs. If it is greater, it does
not change this city population and also deletes the countries whose codes matches with
country codes of the old city table.*/
DELIMITER //
CREATE TRIGGER countryAgeTrigger
    BEFORE UPDATE ON world.country
    FOR EACH ROW
BEGIN
    IF NEW.IndepYear >
        (SELECT country.IndepYear FROM country WHERE NEW.IndepYear > country.Code) THEN
        SET NEW.Population = OLD.Population;
        DELETE FROM country WHERE country.Code = OLD.CountryCode;
    END IF ;
END; //
DELIMITER ;

-- Try to change populations
UPDATE world.country
SET world.Population = 158001
WHERE Name = 'Turkey';
```

If the trigger has more than one statements between **BEGIN ... END** keywords; then you need to use **DELIMITER** keyword as in the example.

Triggers can only be used to alter or update the attributes of a table which is different than the table used in trigger itself. You cannot insert or delete any value according to the result of your control condition. But you can do it for other tables, as in the example. (i.e. Example trigger uses **CITY** table, but it deletes from and hence changes **COUNTRY** table.)

You can use only **NEW** keyword with **INSERT** operation, because you have not any old value for insertion. You can use only **OLD** keyword with **DELETE** operation, because you have not any new value for deletion. And you can use both **NEW** and **OLD** keywords with **UPDATE** operation, because you have both changed and unchanged values. (See also: [\[1\]](#))

With using **AFTER** keyword, you CAN NOT update any **NEW** values. However, when using **BEFORE** keyword; you can update and change the **NEW** values that you have for the table you are using in your trigger. Both in usage of **AFTER** and **BEFORE** keywords, you CAN NOT update any **OLD** values; because it is read-only. (See also: [\[2\]](#))

In our example, we used **BEFORE UPDATE ON city**. It works before an update operation on **CITY** table. With IF condition, it controls if city's population is greater than the population of the country that city belongs. If it is, trigger **SETS** updated value to the **OLD** value and also **DELETES** matching rows from the **COUNTRY** table. If it is not, then it does not do anything rather than this checking operation.

SECTION 5. Stored Procedures in MySQL

```

/* This is a stored procedure which takes a population size as an input argument, and
returns the number of cities whose population sizes are smaller than this input. */
DROP PROCEDURE IF EXISTS COUNT_SMALL_CITIES;
DELIMITER //
CREATE PROCEDURE COUNT_SMALL_CITIES(IN popSize INT(11), OUT cityCount INT)
BEGIN
    SELECT COUNT(Name)
    INTO cityCount
    FROM city
    WHERE Population < popSize;
END; //
DELIMITER ;

CALL COUNT_SMALL_CITIES(1000, @cityCount); -- cityCount is an output parameter

SELECT @cityCount AS small_cities;

-- For checking if our procedure is right
SELECT * FROM city WHERE Population < 1000;

```

With stored procedures, you can write and store some complicated queries before the time when you need to use them.

You have three different parameter options for procedures, `IN`, `OUT` and `INOUT`. You declare your parameter like a code fragment in C or Java in parentheses after the procedure name. When you use `IN`, it is only an input parameter. When you use `OUT`, it can give output to your query as in the example. If you use `INOUT`, then you can use this parameter as an input to your procedure and change its value in your procedure to take it as an output parameter.

You can declare some local variables by using `DECLARE` keyword. (e.g. `DECLARE name VARCHAR(45)`) You need to declare them between `BEGIN ... END` keywords.

You can also use a `CURSOR`, if you have more than one data tuple you need when you use a `SELECT` query. (e.g. `DECLARE nameCursor CURSOR FOR SELECT c.Name FROM country c WHERE c.Name LIKE 'T%';`) When you need to use a `CURSOR`, you first need to use “`OPEN nameCursor;`” before any loop and you need to use “`CLOSE nameCursor;`” after the loop finishes. (See also: [\[3\]](#))

You need to use a special variable when you are using a `CURSOR` which is the `CONTINUE HANDLER`. You need this for when the cursor does not have any other rows anymore to fetch, it needs to stop. If you do not use this, you can see an error like “No data - zero rows fetched, selected, or processed”. You must declare this variable first (i.e. `DECLARE done INT DEFAULT FALSE;`) and use it as a continue handler after declaration (i.e. `DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;`). (See also: [\[3\]](#) [\[4\]](#))

You can use a `LOOP` structure in your procedures if you need to manipulate `CURSOR` data tuple by tuple. You have some loop types such as `LOOP ... END LOOP`, `WHILE ... DO`, `REPEAT ... UNTIL`. In a loop structure, you can use `CURSOR` values one by one by using `FETCH` keyword. (e.g. `FETCH nameCursor INTO tempName;`) (See also: [\[5\]](#))