Command	Syntax	Description	Example
SELECT	`SELECT column1, column2, ... FROM table_name;`	`SELECT` statement is used to fetch data from a database.	`SELECT city FROM placeofinterest;`
WHERE	`SELECT column1, column2, ...FROM table_name WHERE condition;`	`WHERE` clause is used to extract only those records that fulfill a specified condition.	`SELECT * FROM placeofinterest WHERE city == 'Rome' ;`
COUNT	`SELECT COUNT * FROM table_name ;`	`COUNT` is a function that takes the name of a column as argument and counts the number of rows when the column is not NULL.	`SELECT COUNT(country) FROM placeofinterest WHERE country='Canada';`
DISTINCT	`SELECT DISTINCT columnname FROM table_name;`	`DISTINCT` function is used to specify that the statement is a query which returns unique values in specified columns.	`SELECT DISTINCT country FROM placeofinterest WHERE type='historical';`
LIMIT	`SELECT * FROM table_name LIMIT number;`	`LIMIT` is a clause to specify the maximum number of rows the result set must have.	`SELECT * FROM placeofinterest WHERE airport="pearson" LIMIT 5;`
INSERT	`INSERT INTO table_name (column1,column2,column3...) VALUES(value1,value2,value3...);`	`INSERT` is used to insert new rows in the table.	`INSERT INTO placeofinterest (name,type,city,country,airport) VALUES('Niagara Waterfalls','Nature','Toronto','Canada','Pearson');`
UPDATE	`UPDATE table_name SET[[column1]=[VALUES]] WHERE [condition];`	`UPDATE` used to update the rows in the table.	`UPDATE placeofinterest SET name = 'Niagara Falls' WHERE name = "Niagara Waterfalls";`
DELETE	`DELETE FROM table_name WHERE [condition];`	`DELETE` statement is used to remove rows from the table which are specified in the WHERE condition.	`DELETE FROM placeofinterest WHERE city IN ('Rome','Vienna');`

Command	Syntax	Description	Example
CREATE TABLE	`CREATE TABLE table_name (col1 datatype optional keyword, col2 datatype optional keyword,col3 datatype optional keyword,..., coln datatype optional keyword)`	`CREATE TABLE` statement is to create the table. Each column in the table is specified with its name, data type and an optional keyword which could be **PRIMARY KEY**, **NOT NULL**, etc.,	`CREATE TABLE employee ( employee_id char(2) PRIMARY KEY, first_name varchar(30) NOT NULL, mobile int);`
ALTER TABLE - ADD COLUMN	`ALTER TABLE table_name ADD COLUMN column_name_1 datatype....ADD COLUMN column_name_n datatype;`	`ALTER TABLE` statement is used to add the columns to a table.	`ALTER TABLE employee ADD COLUMN income bigint;`
ALTER TABLE - ALTER COLUMN	`ALTER TABLE table_name ALTER COLUMN column_name_1 SET DATA TYPE datatype;`	`ALTER TABLE ALTER COLUMN` statement is used to modify the data type of columns.	`ALTER TABLE employee ALTER COLUMN mobile SET DATA TYPE CHAR(20);`
ALTER TABLE - DROP COLUMN	`ALTER TABLE table_name DROP COLUMN column_name_1 ;`	`ALTER TABLE DROP COLUMN` statement is used to remove columns from a table.	`ALTER TABLE employee DROP COLUMN mobile ;`
ALTER TABLE - RENAME COLUMN	`ALTER TABLE table_name RENAME COLUMN current_column_name TO new_column_name;`	`ALTER TABLE RENAME COLUMN` statement is used to rename the columns in a table.	`ALTER TABLE employee RENAME COLUMN first_name TO name ;`
TRUNCATE TABLE	`TRUNCATE TABLE table_name IMMEDIATE;`	`TRUNCATE TABLE` statement is used to delete all of the rows in a table. The IMMEDIATE specifies to process the statement immediately and that it cannot be undone.	`TRUNCATE TABLE employee IMMEDIATE ;`
DROP TABLE	`DROP TABLE table_name ;`	Use the `DROP TABLE` statement to delete a table from a database. If you delete a table that contains data, by default the data will be deleted alongside the table.	`DROP TABLE employee ;`

Topic	Syntax	Description	Example
Cross Join	`SELECT column_name(s) FROM table1 CROSS JOIN table2;`	The `CROSS JOIN` is used to generate a paired combination of each row of the first table with each row of the second table.	`SELECT DEPT_ID_DEP, LOCT_ID FROM DEPARTMENTS CROSS JOIN LOCATIONS;`
Inner Join	`SELECT column_name(s) FROM table1 INNER JOIN table2 ON table1.column_name = table2.column_name; WHERE condition;`	You can use an `inner join` in a SELECT statement to retrieve only the rows that satisfy the join conditions on every specified table.	`select E.F_NAME,E.L_NAME, JH.START_DATE from EMPLOYEES as E INNER JOIN JOB_HISTORY as JH on E.EMP_ID=JH.EMPL_ID where E.DEP_ID ='5';`
Left Outer Join	`SELECT column_name(s) FROM table1 LEFT OUTER JOIN table2 ON table1.column_name = table2.column_name WHERE condition;`	The `LEFT OUTER JOIN` will return all records from the left side table and the matching records from the right table.	`select E.EMP_ID,E.L_NAME,E.DEP_ID,D.DEP_NAME from EMPLOYEES AS E LEFT OUTER JOIN DEPARTMENTS AS D ON E.DEP_ID=D.DEPT_ID_DEP;`
Right Outer Join	`SELECT column_name(s) FROM table1 RIGHT OUTER JOIN table2 ON table1.column_name = table2.column_name WHERE condition;`	The `RIGHT OUTER JOIN` returns all records from the right table, and the matching records from the left table.	`select E.EMP_ID,E.L_NAME,E.DEP_ID,D.DEP_NAME from EMPLOYEES AS E RIGHT OUTER JOIN DEPARTMENTS AS D ON E.DEP_ID=D.DEPT_ID_DEP;`
Full Outer Join	`SELECT column_name(s) FROM table1 FULL OUTER JOIN table2 ON table1.column_name = table2.column_name WHERE condition;`	The `FULL OUTER JOIN` clause results in the inclusion of rows from two tables. If a value is missing when rows are joined, that value is null in the result table.	`select E.F_NAME,E.L_NAME,D.DEP_NAME from EMPLOYEES AS E FULL OUTER JOIN DEPARTMENTS AS D ON E.DEP_ID=D.DEPT_ID_DEP;`
Self Join	`SELECT column_name(s) FROM table1 T1, table1 T2 WHERE condition;`	A `self join` is regular join but it can be used to joined with itself.	`SELECT B.* FROM EMPLOYEES A JOIN EMPLOYEES B ON A.MANAGER_ID = B.MANAGER_ID WHERE A.EMP_ID = 'E1001';`

Joins in MySQL using phpMyAdmin

Full Outer Join	`SELECT column_name(s) FROM table1 LEFT OUTER JOIN table2 ON table1.column_name = table2.column_name WHERE condition`
	`UNION`
	
	`SELECT column_name(s) FROM table1 RIGHT OUTER JOIN table2 ON table1.column_name = table2.column_name WHERE condition`
		The `UNION operator` is used to combine the result-set of two or more SELECT statements.	`select E.F_NAME,E.L_NAME,D.DEP_NAME from EMPLOYEES AS E LEFT OUTER JOIN DEPARTMENTS AS D ON E.DEP_ID=D.DEPT_ID_DEP`
			`UNION`
			
			`select E.F_NAME,E.L_NAME,D.DEP_NAME from EMPLOYEES AS E RIGHT OUTER JOIN DEPARTMENTS AS D ON E.DEP_ID=D.DEPT_ID_DEP`

Command	Syntax	Description	Example
LIKE	`SELECT column1, column2, ... FROM table_name WHERE columnN LIKE pattern;`	`LIKE` operator is used in a WHERE clause to search for a specified pattern in a column. There are two wildcards often used in conjunction with the LIKE operator which are percent sign(%) and underscore sign (_).	`SELECT f_name , l_name FROM employees WHERE address LIKE '%Elgin,IL%';`
BETWEEN	`SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;`	The `BETWEEN` operator selects values within a given range. The values can be numbers, text, or dates. The `BETWEEN` operator is inclusive: begin and end values are included.	`SELECT * FROM employees WHERE salary BETWEEN 40000 AND 80000;`
ORDER BY	`SELECT column1, column2, ... FROM table_name ORDER BY column1, column2, ... ASC|DESC;`	`ORDER BY` keyword is used to sort the result-set in ascending or descending order. The default is ascending.	`SELECT f_name, l_name, dep_id FROM employees ORDER BY dep_id DESC, l_name;`
GROUP BY	`SELECT column_name(s) FROM table_name WHERE condition GROUP BY column_name(s) ORDER BY column_name(s);`	`GROUP BY` clause is used in collaboration with the SELECT statement to arrange identical data into groups.	`SELECT dep_id, COUNT(*) FROM employees GROUP BY dep_id;`

Command	Syntax	Description	Example
COUNT	`SELECT COUNT(column_name) FROM table_name WHERE condition;`	`COUNT` function returns the number of rows that matches a specified criterion.	`SELECT COUNT(dep_id) FROM employees;`
AVG	`SELECT AVG(column_name) FROM table_name WHERE condition;`	`AVG` function returns the average value of a numeric column.	`SELECT AVG(salary) FROM employees;`
SUM	`SELECT SUM(column_name) FROM table_name WHERE condition;`	`SUM` function returns the total sum of a numeric column.	`SELECT SUM(salary) FROM employees;`
MIN	`SELECT MIN(column_name) FROM table_name WHERE condition;`	`MIN` function returns the smallest value of the SELECTed column.	`SELECT MIN(salary) FROM employees;`
MAX	`SELECT MAX(column_name) FROM table_name WHERE condition;`	`MAX` function returns the largest value of the SELECTed column.	`SELECT MAX(salary) FROM employees;`
ROUND	`SELECT ROUND(2number, decimals, operation) AS RoundValue;`	`ROUND` function rounds a number to a specified number of decimal places.	`SELECT ROUND(salary) FROM employees;`
LENGTH	`SELECT LENGTH(column_name) FROM table;`	`LENGTH` function returns the length of a string (in bytes).	`SELECT LENGTH(f_name) FROM employees;`
UCASE	`SELECT UCASE(column_name) FROM table;`	`UCASE` function that displays the column name in each table in uppercase.	`SELECT UCASE(f_name) FROM employees;`
DISTINCT	`SELECT DISTINCT(column_name) FROM table;`	`DISTINCT` function is used to display data without duplicates.	`SELECT DISTINCT(UCASE(f_name)) FROM employees;`
DAY	`SELECT DAY(column_name) FROM table`	`DAY` function returns the day of the month for a given date	`SELECT DAY(b_date) FROM employees where emp_id = 'E1002';`
CURRENT DATE	`SELECT (CURRENT DATE - COLUMN) FROM table;`	`CURRENT DATE` is used to display the current date.This can be subtracted from the previous date to get the difference.	`SELECT YEAR(CURRENT DATE - b_date) As AGE, CURRENT_DATE, b_date FROM employees;`

Subquery	`SELECT column_name [, column_name ] FROM table1 [, table2 ] WHERE column_name OPERATOR (SELECT column_name [, column_name ] FROM table1 [, table2 ] [WHERE])`	`Subquery` is a query within another SQL query and embedded within the WHERE clause.
		A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
			`SELECT emp_id, fmame, lname, salary FROM employees where salary < (SELECT AVG(salary) FROM employees);`
			`SELECT * FROM ( SELECT emp_id, f_name, l_name, dep_id FROM employees) AS emp4all;`
			
			`SELECT * FROM employees WHERE job_id IN (SELECT job_ident FROM jobs);`

Implicit Inner Join	`SELECT column_name(s) FROM table1, table2 WHERE table1.column_name = table2.column_name;`	`Implicit Inner Join` combines the two or more records but displays only matching values in both tables. Inner join applies only the specified columns.	`SELECT * FROM employees, jobs where employees.job_id = jobs.job_ident;`
Implicit Cross Join	`SELECT column_name(s) FROM table1, table2;`	`Implicit Cross Join` defines as a Cartesian product where the number of rows in the first table multiplied by the number of rows in the second table..	`SELECT * FROM employees, jobs;`

Views

Topic	Syntax	Description	Example
Create View	`CREATE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;`	A `CREATE VIEW` is an alternative way of representing data that exists in one or more tables.	`CREATE VIEW EMPSALARY AS SELECT EMP_ID, F_NAME, L_NAME, B_DATE, SEX, SALARY FROM EMPLOYEES;`
Update a View	`CREATE OR REPLACE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;`	The `CREATE OR REPLACE` VIEW command updates a view.	`CREATE OR REPLACE VIEW EMPSALARY AS SELECT EMP_ID, F_NAME, L_NAME, B_DATE, SEX, JOB_TITLE, MIN_SALARY, MAX_SALARY FROM EMPLOYEES, JOBS WHERE EMPLOYEES.JOB_ID = JOBS.JOB_IDENT;`
Drop a View	`DROP VIEW view_name;`	Use the `DROP VIEW` statement to remove a view from the database.	`DROP VIEW EMPSALARY;`

Stored Procedures on IBM Db2 using SQL

Stored Procedures	`--#SET TERMINATOR @ CREATE PROCEDURE PROCEDURE_NAME LANGUAGE`
	
	`BEGIN`
	
	`END @`
		A `stored procedure` is a prepared SQL code that you can save, so the code can be reused over and over again.
		The default terminator for a stored procedure is semicolon(;). To set a different terminator we use `SET TERMINATOR` clause followed by the terminator such as '@'.
			`--#SET TERMINATOR @ CREATE PROCEDURE RETRIEVE_ALL`
			`LANGUAGE SQL`
			`READS SQL DATA`
			
			`DYNAMIC RESULT SETS 1`
			`BEGIN`
			
			`DECLARE C1 CURSOR`
			`WITH RETURN FOR`
			
			`SELECT * FROM PETSALE;`
			
			`OPEN C1;`
			
			`END @`

Stored Procedures in MySQL using phpMyAdmin

Stored Procedures

```
DELIMITER //

CREATE PROCEDURE PROCEDURE_NAME

BEGIN

END //

DELIMITER ;
```

A `stored procedure` is a prepared SQL code that you can save, so the code can be reused over and over again.
The default terminator for a stored procedure is semicolon (;).
To set a different terminator we use `DELIMITER` clause followed by the terminator such as $$ or //.

```
DELIMITER //
CREATE PROCEDURE RETRIEVE_ALL()

BEGIN

SELECT * FROM PETSALE;

END //

DELIMITER ;
```

Transactions in MySQL using phpMyAdmin

Commit command

```
DELIMITER //

CREATE PROCEDURE PROCEDURE_NAME

BEGIN

COMMIT;

END //

DELIMITER ;
```

A `COMMIT command` is used to persist the changes in the database.
The default terminator for a COMMIT command is semicolon (;).

```
DELIMITER //
CREATE PROCEDURE TRANSACTION_ROSE()

BEGIN

DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN ROLLBACK;
RESIGNAL; END;

START TRANSACTION; UPDATE BankAccounts SET Balance =
Balance-200 WHERE AccountName = 'Rose';

UPDATE BankAccounts SET Balance = Balance-300 WHERE
AccountName = 'Rose';

COMMIT;

END //

DELIMITER ;\
```

Rollback command	DELIMITER // CREATE PROCEDURE PROCEDURE_NAME BEGIN ROLLBACK; COMMIT; END // DELIMITER ;	A `ROLLBACK command` is used to rollback the transactions which are not saved in the database. The default terminator for a ROLLBACK command is semicolon (;).	DELIMITER // CREATE PROCEDURE TRANSACTION_ROSE() BEGIN DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN ROLLBACK; RESIGNAL; END; START TRANSACTION; UPDATE BankAccounts SET Balance = Balance-200 WHERE AccountName = 'Rose'; UPDATE BankAccounts SET Balance = Balance-300 WHERE AccountName = 'Rose'; COMMIT; END // DELIMITER ;\

Transactions in MySQL using db2

Commit command	--#SET TERMINATOR @
	
	CREATE PROCEDURE PROCEDURE_NAME
	
	BEGIN
	
	COMMIT;
	
	END @
		A `COMMIT command` is used to persist the changes in the database.
		The default terminator for a COMMIT command is semicolon (;).
			--#SET TERMINATOR @ CREATE PROCEDURE TRANSACTION_ROSE
			LANGUAGE SQL MODIFIES SQL DATA
			BEGIN
			
			DECLARE SQLCODE INTEGER DEFAULT 0;
			DECLARE retcode INTEGER DEFAULT 0;
			DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
			SET retcode = SQLCODE;
			
			UPDATE BankAccounts SET Balance = Balance-200 WHERE AccountName = 'Rose';
			
			UPDATE BankAccounts SET Balance = Balance-300 WHERE AccountName = 'Rose';
			
			IF retcode < 0 THEN
			ROLLBACK WORK;
			
			ELSE COMMIT WORK;
			
			END IF;
			
			END @\

Rollback command

```
--#SET TERMINATOR @
CREATE PROCEDURE PROCEDURE_NAME

BEGIN

ROLLBACK;

COMMIT;

END @
```

A `ROLLBACK command` is used to rollback the transactions which are not saved in the database.
The default terminator for a ROLLBACK command is semicolon (;).

```
--#SET TERMINATOR @ CREATE PROCEDURE TRANSACTION_ROSE
LANGUAGE SQL MODIFIES SQL DATA
BEGIN

DECLARE SQLCODE INTEGER DEFAULT 0;
DECLARE retcode INTEGER DEFAULT 0;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
SET retcode = SQLCODE;

UPDATE BankAccounts SET Balance = Balance-200 WHERE
AccountName = 'Rose';

UPDATE BankAccounts SET Balance = Balance-300 WHERE
AccountName = 'Rose';

IF retcode < 0 THEN
ROLLBACK WORK;

ELSE COMMIT WORK;

END IF;

END @\
```