Database : MySQL

OPERATORS

Arithmetic

{v} + {v}
{v} - {v}
{v} * {v}
{v} / {v}
{v} % {v}

Bitwise

{v} & {v}
{v} | {v}
{v} ^ {v}

Compound

{v} += {v}
{v} -= {v}
{v} *= {v}
{v} /= {v}
{v} %= {v}
{v} &= {v}
{v} |*= {v}
{v} ^-= {v}

Comparison

{v} < {v}
{v} <= {v}
{v} = {v}
{v} >= {v}
{v} > {v}
{v} <> {v}

Logical

NOT {cd}
Displays a record if the condition(s) is NOT TRUE

EXISTS({s})
TRUE if the subquery returns one or more records

{cd} AND {cd} | {cd} & {cd}
TRUE if all the conditions separated by AND is TRUE

{cd} OR {cd} | {cd} | {cd}
TRUE if any of the conditions separated by OR is TRUE

{v} {cp} ALL({s})
TRUE if all of the subquery values meet the condition

{v} {cp} ANY({s} | {v} {cp} SOME({s}
TRUE if any of the subquery values meet the condition

{v} IN({s})
TRUE if the operand is equal to one of a list of expressions

{v} BETWEEN {v} AND {v}
TRUE if the operand is within the range of comparisons

{v} LIKE '%_'
TRUE if the operand matches a pattern

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

DATATYPE

String

BINARY(size)
parameter size : [1-255] default = 1

VARBINARY(size)
parameter size : [1;65 535]

CHAR(size)
parameter size : [1;255] default = 1

VARCHAR(size)
parameter size : [1;65 535]

TINYTEXT
values : [0;255]

TEXT(size)
parameter size : [1;65 535]

MEDIUMTEXT
values : [0;16 777 215]

LONGTEXT
values : [0;4 294 967 295]

ENUM('val1', '...', 'val65535')
SET('val1', '...', 'val64')

TINYBLOB
values : [0;255]

BLOB(size)
parameter size : [0;65 535]
values : [0;65 535]

MEDIUMBLOB
values : [0;16 777 215]

LONGBLOB
values : [0;4 294 967 295]

Numeric

BIT(size)
parameter size : [1,64] default = 1

BOOL | BOOLEAN
values : [0;1]

TINYINT(size)
parameter size : [1,255]
values : [-128;127]

SMALLINT(size)
parameter size : [1,255]
values : [-32 768;32 767]

MEDIUMINT(size)
parameter size : [1,255]
values : [-8 388 608;8 388 607]

INT(size) | INTEGER(size)
parameter size : [1,255]
values : [-2 147 483 648; 2 147 483 647]

BIGINT(size)
parameter size : [1,255]
values : [-9 223 372 036 854 775 808; 9 223 372 036 854 775 807]

UNSIGNED TINYINT(size)
values : [0;255]

UNSIGNED SMALLINT(size)
values : [0;65 535]

UNSIGNED MEDIUMINT(size)
values : [0;16 777 215]

UNSIGNED INT(size) | UNSIGNED INTEGER(size)
values : [0;4 294 967 295]

UNSIGNED BIGINT(size)
values : [0;18 446 744 073 709 551 615]

FLOAT(size, d)
parameter size : [1;65] default 10,
parameter d : [0,30] default 0

FLOAT(p)
parameter p : [0;53]

DEC(size, d) | DECIMAL(size, d)
parameter size : [1;65] default 10
parameter d : [0,30] default 0

DOUBLE(size, d)
parameter size : [1;65] default 10
parameter d : [0,30] default 0

DOUBLE PRECISION(size, d)
parameter size : [1;65] default 10
parameter d : [0,30] default 0

Date & Time

DATE
format : YYYY-MM-DD
values : [1000-01-01; 9999-12-31]

DATETIME(fsp
format : YYYY-MM-DD hh:mm:ss
values : [1000-01-01 00:00:00; 9999-12-31 23:59:59]

TIMESTAMP(fsp)
format : YYYY-MM-DD hh:mm:ss
values : [1970-01-01 00:00:01; 2038-01-09 03:14:07]

TIME(fsp)
format : hh:mm:ss
values : [-838:59:59; 838:59:59]

YEAR
format : YYYY
values : [0000] & [1901;2155]

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

FUNCTIONS

String

SPACE(number)
Returns a string of the specified number of space characters

FORMAT(number, decimal_place)
Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places

LOWER(string) | LCASE(string)
Converts a string to lower-case

UCASE(string) | UPPER(string)
Converts a string to upper-case

LTRIM(string
Removes leading spaces from a string

RTRIM(string)
Removes trailing spaces from a string

TRIM(string)
Removes leading and trailing spaces from a string

LPAD(str, length, lpad_str)
Left-pads a string with another string, to a certain length

RPAD(string, length, rpad_string)
Right-pads a string with another string, to a certain length

REPEAT(string, number)
Repeats a string as many times as specified

LEFT(string, number_of_chars)
Extracts a number of characters from a string (starting from left)

MID(string, start, length) | SUBSTR(string, start, length) | SUBSTRING(string, start, length)
Extracts a substring from a string (starting at any position)

RIGHT(string, number_of_chars
Extracts a number of characters from a string (starting from right)

SUBSTRING_INDEX(string, delimiter, number)
Returns a substring of a string before a specified number of delimiter occurs

CONCAT(str1, ..., strX)
Adds two or more expressions together

Numeric

PI()
Returns the value of PI

RAND(seed)
Returns a random number

ABS(number)
Returns the absolute value of a number

SIGN(number)
Returns the sign of a number

{v} DIV {v}
Used for integer division

EXP(number)
Returns e raised to the power of a specified number

SQRT(number)
Returns the square root of a number

POW({v},{v}) | POWER({v},{v})
Returns the value of a number raised to the power of another number

AVG(col1)
Returns the average value of an expression

COUNT(col1)
Returns the number of records returned by a select query

MAX({s})
Returns the maximum value in a set of values

MIN({s})
Returns the minimum value in a set of values

SUM({s})
Calculates the sum of a set of values

CEIL(number) | CEILING
Returns the smallest integer value that is >= to a number

GREATEST({s})
Returns the greatest value of the list of arguments

LEAST({s})
Returns the smallest value of the list of arguments

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

CONCAT_WS(separator, str1, ... strX)
Adds two or more expressions together with a separator

INSERT(string, position, number, str_insert)
Inserts a string within a string at the specified position and for a certain number of characters

REPLACE(string, substring, new_string)
Replaces all occurrences of a substring within a string, with a new substring

REVERSE(string)
Reverses a string and returns the result = palindrome

ASCII(character)
Returns the ASCII value for the specific character

CHAR_LENGTH(string) | CHARACTER_LENGTH(string)
Returns the length of a string (in characters)

LENGTH(string)
Returns the length of a string (in bytes)

STRCMP(str1, str2)
Compares two strings

FIELD(search, val1, ..., valX)
Returns the index position of a value in a list of values

FIND_IN_SET(string, "str1, ..., strX")
Returns the position of a string within a list of strings

POSITION(substr IN string) | LOCATE(substr, str, start)
Returns the position of the first occurrence of a substring in a string

FLOOR(number)
Returns the largest integer value that is <= to a number

ROUND(number, decimals)
Rounds a number to a specified number of decimal places

TRUNCATE(number, decimals)
Truncates a number to the specified number of decimal places

DEGREES(number)
Converts a value in radians to degrees

RADIANS(number)
Converts a degree value into radians

LN(number)
Returns the natural logarithm of a number

LOG(number)
Returns the natural logarithm of a number, or the logarithm of a number to a specified base

LOG10(number)
Returns the natural logarithm of a number to base 10

LOG2(number)
Returns the natural logarithm of a number to base 2

ACOS(number)
Returns the arc cosine of a number

ASIN(number)
Returns the arc sine of a number

ATAN(number)
Returns the arc tangent of one or two numbers

ATAN2(a, b)
Returns the arc tangent of two numbers

COS(number)
Returns the cosine of a number

COT(number)
Returns the cotangent of a number

SIN(number)
Returns the sine of a number

TAN(number)
Returns the tangent of a number

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

Date & Time

UNITS

- MICROSECOND
- SECOND
- MINUTE
- HOUR
- DAY
- MONTH
- YEAR
- WEEK
- QUARTER
- SECOND_MICROSECOND
- MINUTE_MICROSECOND
- HOUR_MICROSECOND
- DAY_MICROSECOND,
- MINUTE_SECOND
- HOUR_SECOND
- DAY_SECOND,
- HOUR_MINUTE
- DAY_MINUTE
- DAY_HOUR
- YEAR_MONTH

FORMATS

%a	Abbreviated weekday name (Sun to Sat)
%b	Abbreviated month name (Jan to Dec)
%c	Numeric month name (0 to 12)
%D	Day of the month as a numeric value, followed by suffix (1st, 2nd, 3rd, ...)
%d	Day of the month as a numeric value (01 to 31)
%e	Day of the month as a numeric value (0 to 31)
%f	Microseconds (000000 to 999999)
%H	Hour (00 to 23)
%h	Hour (00 to 12)
%I	Hour (00 to 12)
%i	Minutes (00 to 59)
%j	Day of the year (001 to 366)

DAY(date) | DAYOFMONTH(date)
Returns the day of the month for a given date

DAYNAME(date)
Returns the weekday name for a given date

DAYOFWEEK(date)
Returns the weekday index for a given date

DAYOFYEAR(date)
Returns the day of the year for a given date

EXTRACT({u} FROM date)
Extracts a part from a given date

HOUR(datetime)
Returns the hour part for a given date

MICROSECOND(datetime)
Returns the microsecond part of a time/datetime

MINUTE(datetime)
Returns the minute part of a time/datetime

MONTH(date)
Returns the month part for a given date

MONTHNAME(date)
Returns the name of the month for a given date

QUARTER(date)
Returns the quarter of the year for a given date value

SECOND(datetime)
Returns the seconds part of a time/datetime

WEEK(date, firstdayofweek) | WEEKOFYEAR(date) | YEARWEEK(date, firstdayofweek)
Returns the week number for a given date

WEEKDAY(date)
Returns the weekday number for a given date

YEAR(date)
Returns the year part for a given date

CURDATE() | CURRENT_DATE()
Returns the current date

CURTIME() | CURRENT_TIME()
Returns the current time

CURRENT_TIMESTAMP() | LOCALTIMESTAMP()
Returns the current date and time

LOCALTIME() | NOW() | SYSDATE()
Returns the current date and time

DATE("YYYY-MM-DD hh:mm:ss")
Extracts the date part from a datetime expression

MAKEDATE(year, day)
Creates and returns a date based on a year and a number of days value

MAKETIME(hour, minute, second)
Creates and returns a time based on an hour, minute, and second value

SEC_TO_TIME(seconds)
Returns a time value based on the specified seconds

STR_TO_DATE(string, "{f}")
Returns a date based on a string and a format

TIME(string)
Extracts the time part from a given time/datetime

TIME_FORMAT(string, "{f}")
Formats a time by a specified format

TO_DAYS(date)
Returns the number of days between a date and date "0000-00-00"

LAST_DAY(date)
Extracts the last day of the month for a given date

Advanced

LAST_INSERT_ID()
Returns the AUTO_INCREMENT id of the last row that has been inserted or updated in a table

BIN(number)
Returns a binary representation of a number

BINARY {v}
Converts a value to a binary string

CONV(number, from_base, to_base)
Converts a number from one numeric base system to another

CONVERT({v}, {dt})
Converts a value into the specified datatype or character set

CAST({v} AS {dt})
Converts a value (of any type) into a specified datatype

COALESCE({s})
Returns the first non-null value in a list

ISNULL(expression)
Returns 1 or 0 depending on whether an expression is NULL

NULLIF(expr1, expr2)
Compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned

IFNULL(expression, {v})
Return a specified value if the expression is NULL, otherwise return the expression

IF(condition, {vTrue}, {vFalse})
Returns a value if a condition is TRUE, or another value if a condition is FALSE

CASE
Goes through conditions and return a value when the first condition is met

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

%k	Hour (0 to 23)
%l	Hour (1 to 12)
%M	Month name in full (January to December)
%m	Month name as a numeric value (00 to 12)
%p	AM or PM
%r	Time in 12 hour AM or PM format (hh:mm:ss AM/PM)
%S	Seconds (00 to 59)
%s	Seconds (00 to 59)
%T	Time in 24 hour format (hh:mm:ss)
%U	Week where Sunday is the first day of the week (00 to 53)
%u	Week where Monday is the first day of the week (00 to 53)
%V	Week where Sunday is the first day of the week (01 to 53). Used with %X
%v	Week where Monday is the first day of the week (01 to 53). Used with %x
%W	Weekday name in full (Sunday to Saturday)
%w	Day of the week where Sunday=0 and Saturday=6
%X	Year for the week where Sunday is the first day of the week. Used with %V
%x	Year for the week where Monday is the first day of the week. Used with %v
%Y	Year as a numeric, 4-digit value
%y	Year as a numeric, 2-digit value

ADDDATE(date, INTERVAL {v} {u}) | DATE_ADD(date, INTERVAL {v} {u})
Adds a time/date interval to a date and then returns the date

ADDTIME(datetime, addtime)
Adds a time interval to a time/datetime and then returns the time/datetime

DATEDIFF(date1, date2)
Returns the number of days between two date values

DATE_SUB(date, INTERVAL {v} {u}) | SUBDATE(date, INTERVAL {v} {u})
Subtracts a time/date interval from a date and then returns the date

FROM_DAYS(number)
Returns a date from a numeric datevalue

PERIOD_ADD(period, number)
Adds a specified number of months to a period

PERIOD_DIFF(period1, period2)
Returns the difference between two periods

SUBTIME(datetime, time_interval)
Subtracts a time interval from a datetime and then returns the time/datetime

TIME_TO_SEC(time)
Converts a time value into seconds

TIMEDIFF(time1, time2)
Returns the difference between two time/datetime expressions

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

VERSION()
Returns the current version of the MySQL database

DATABASE()
Returns the name of the current database

CURRENT_USER()
Returns the user name and host name for the MySQL account that the server used to authenticate the current client

SESSION_USER() | SYSTEM_USER() | USER()
Returns the current MySQL user name and host name

CONNECTION_ID()
Returns the unique connection ID for the current connection

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

REQUESTS DDL

Data Definition Language (DDL) refers to the CREATE, ALTER and DROP statements.
DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. DDL allows to add / modify / delete the logical structures which contain the data or which allow users to access / maintain the data (databases, tables, keys, views...). DDL is about "metadata".

`CREATE DATABASE databasename;` `USE databasename;`	`SHOW databases;`	`DROP databasename;`

`CREATE TABLE table_name(`
`  column_name DATATYPE,`
`  ...`
`);`
	`ALTER TABLE table_name`
	`ADD columnX_name {dt};`
	`ALTER TABLE table_name`
	`MODIFY COLUMN columnX_name {dt};`
		`DROP TABLE table_name;`
		
		`ALTER TABLE table_name`
		`DROP COLUMN columnX_name;`

`SHOW tables;`	`ALTER TABLE table_name MODIFY COLUMN column_name {dt} AUTO_INCREMENT;` `ALTER TABLE table_name AUTO_INCREMENT={v};`	

	`ALTER TABLE table_name`
	`MODIFY COLUMN column_string_name DEFAULT '{v}';`
	`ALTER TABLE table_name`
	`MODIFY COLUMN column_bool_name DEFAULT {v};`
	`ALTER TABLE table_name`
	`MODIFY COLUMN column_numeric_name DEFAULT {v};`
	`ALTER TABLE table_name`
	`MODIFY COLUMN column_date_name DEFAULT CURRENT_DATE();`	

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

	ALTER TABLE table_name ADD CONSTRAINT NN_column_name NOT NULL(column_name); ALTER TABLE table_name ADD CONSTRAINT U_column_name UNIQUE(column_name); ALTER TABLE table_name ADD CONSTRAINT PK_column1_name_column2_name PRIMARY KEY (column1_name, column2_name); ALTER TABLE table_name ADD CONSTRAINT FK_column_name FOREIGN KEY (column_name) REFERENCES table2_name(column_name); ALTER TABLE table_name ADD CONSTRAINT CHK_column_name_min CHECK (column_name {cp} {v});	ALTER TABLE table_name DROP INDEX U_column_name; ALTER TABLE table_name DROP PRIMARY KEY; ALTER TABLE table_name DROP FOREIGN KEY FK_column_name; ALTER TABLE table_name DROP CHECK CHK_column_name_min;
CREATE [UNIQUE] INDEX index_name ON table_name (column1_name, column2_name);	SHOW index FROM table_name; ALTER TABLE table_name;	DROP INDEX index_name;
CREATE VIEW view_name AS {s};	CREATE OR REPLACE VIEW view_name AS {s};	DROP VIEW view_name;

REQUESTS DML

Data Manipulation Language (DML) refers to the INSERT, UPDATE and DELETE statements
DML allows to add / modify / delete data itself.

INSERT INTO table_name [(col1, ..., colX)] VALUES (val1, ..., valX);	UPDATE table_name SET col1 = val1, ..., colX = valX [WHERE condition];	DELETE FROM table_name [WHERE condition];

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

REQUESTS DQL

Data Query Language (DQL) refers to the SELECT, SHOW and HELP statements (queries)
SELECT is the main DQL instruction. It retrieves data you need. SHOW retrieves infos about the metadata. HELP... is for people who need help.

```
SELECT col1 [AS alias], ..., colX [AS
alias] | * | fct() [AS alias]
FROM table_name | view_name [AS alias]
[[LEFT | INNER | RIGHT | CROSS] JOIN
table_name | view_name]
[WHERE condition]
[GROUP BY col1, ..., colX
[HAVING condition]]
[ORDER BY col1, ..., colX [ASC | DESC]]
[LIMIT number [OFFSET number]]
[UNION
{sWithSameNameColumn}];
```

```
SHOW BINARY LOG STATUS
SHOW BINARY LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM
pos] [LIMIT [offset,] row_count]
SHOW {CHARACTER SET | CHARSET}
[like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM
db_name] [like_or_where]
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION func_name
SHOW CREATE PROCEDURE proc_name
SHOW CREATE TABLE tbl_name
SHOW CREATE TRIGGER trigger_name
SHOW CREATE VIEW view_name
SHOW DATABASES [like_or_where]
SHOW ENGINE engine_name {STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW EVENTS
SHOW FUNCTION CODE func_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW OPEN TABLES [FROM db_name]
[like_or_where]
SHOW PLUGINS
SHOW PROCEDURE CODE proc_name
SHOW PROCEDURE STATUS [like_or_where]
```

```
HELP 'search_string'
HELP 'contents'
HELP 'data types'
HELP 'ascii'
HELP 'create table'

...
```

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

```
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW PROFILE [types] [FOR QUERY n]
[OFFSET n] [LIMIT n]
SHOW PROFILES
SHOW RELAYLOG EVENTS [IN 'log_name']
[FROM pos] [LIMIT [offset,] row_count]
SHOW REPLICA STATUS [FOR CHANNEL channel]
SHOW REPLICAS
SHOW [GLOBAL | SESSION] STATUS
[like_or_where]
SHOW TABLE STATUS [FROM db_name]
[like_or_where]
SHOW [FULL] TABLES [FROM db_name]
[like_or_where]
SHOW TRIGGERS [FROM db_name]
[like_or_where]
SHOW [GLOBAL | SESSION] VARIABLES
[like_or_where]
SHOW WARNINGS [LIMIT [offset,] row_count]

like_or_where: {
    LIKE 'pattern'
  | WHERE expr
}
```

REQUEST DCL

Data Control Language (DCL) refers to the GRANT and REVOKE statements
DCL is used to grant / revoke permissions on databases and their contents. DCL is simple, but MySQL's permissions are rather complex. DCL is about security.

CREATE USER 'nom_login'[@'adresse_client'] [IDENTIFIED BY [PASSWORD] 'mot_de_passe'];	RENAME USER 'ancien_login'@'ancienne_adresse_client' TO 'nouveau_login'@'nouvelle_adresse_client' ; SET PASSWORD FOR 'nom_login'@'adresse_client' = PASSWORD('nouveau_mot_de_passe');	DROP USER 'nom_login'@'adresse_client';
CREATE ROLE [IF NOT EXISTS] role1 [, role2];		DROP ROLE [IF EXISTS] role1 [, role2];
CREATE USER [IF NOT EXISTS] nom_login DEFAULT ROLE role1 [, role1] ...	ALTER USER [IF EXISTS] nom_login DEFAULT ROLE {NONE | ALL | role1 [, role2] ...};	
GRANT privilège [(col1, col2,...)] ON [type_élément] type_domaine TO nom_login [IDENTIFIED BY mot_de_passe]; GRANT ALL [PRIVILEGES] ON [type_élément] type_domaine TO nom_login [IDENTIFIED BY mot_de_passe]; GRANT USAGE ON *.* -- toujours sur tous les éléments TO nom_login IDENTIFIED BY mot_de_passe;	SHOW GRANTS [FOR nom_login_ou_role]; SELECT * FROM information_schema.column_privileges;	REVOKE privilège ON [type_élément] type_domaine FROM nom_login;

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

GRANT GRANT OPTION ON [type_élément] type_domaine TO nom_login [IDENTIFIED BY mot_de_passe]; GRANT ... ON [type_élément] type_domaine TO nom_login [IDENTIFIED BY mot_de_passe] WITH GRANT OPTION;		

REQUEST DTL

Data Transaction Language (DTL) refers to the START TRANSACTION, SAVEPOINT, COMMIT and ROLLBACK [TO SAVEPOINT] statements
DTL is used to manage transactions (operations which include more instructions none of which can be executed if one of them fails).

- -- Déclaration des variables locales
- -- Déclaration des conditions
- -- Déclaration des curseurs
- -- Déclaration des gestionnaires d'erreur
- -- Début du traitement

SHOW variables WHERE variable_name LIKE '%autocommit%'; -- ou SELECT @@autocommit;	SET autocommit=OFF;	
START TRANSACTION; [request] COMMIT;		
SAVEPOINT jalon_name;	ROLLBACK TO jalon_name;	RELEASE SAVEPOINT jalon_name;
LOCK TABLES table1[, table2] {READ | WRITE}; {s} LOCK IN SHARE MODE; {s} FOR UPDATE;	SHOW OPEN TABLES FROM nom_db;	UNLOCK TABLES;
	SET [GLOBAL|SESSION] TRANSACTION	

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

	ISOLATION LEVEL {READ UNCOMMITTED|READ COMMITTED| REPEATABLE READ|SERIALIZABLE};	
SET @nom_variable = valeur; -- ou SET @nom_variable := valeur; -- à utiliser dans les requêtes SELECT col1, col2 INTO @col1, @col2 FROM table WHERE condition=1;	SELECT @nom_variable;	
PREPARE nom_requete_preparee FROM 'requête SQL' WHERE col=?;	EXECUTE nom_requete_preparee USING param;	DEALLOCATE PREPARE nom_requete_preparee;
DELIMITER $$ CREATE PROCEDURE nom_procedure([[IN\|OUT] [pi\|po\|pio]_name {dt}]) BEGIN {s} END $$ DELIMITER ;	CALL procedure_name([param]); SHOW CREATE PROCEDURE nom_procedure \G; SHOW PROCEDURE STATUS; -- ou SHOW PROCEDURE STATUS WHERE db='nom_db';	DROP PROCEDURE [IF EXISTS] nom_procedure;

CREATE FUNCTION nom_fonction() RETURNS type_donnée_retour
DETERMINISTIC
BEGIN
RETURN;
END;
	SELECT nom_fonction();
	
	SHOW CREATE FUNCTION nom_fonction \G;
	
	SHOW FUNCTION STATUS;
	-- ou
	SHOW FUNCTION STATUS WHERE db='nom_db';
	-- ou
	SELECT name, db, type FROM mysql.proc;
		DROP FUNCTION [IF EXISTS] nom_fonction;

DECLARE nom_variable type_variable [DEFAULT val_defaut];	SET nom_variable := valeur;	

IF condition(s) THEN
Bloc d'instruction(s)
END IF;
	IF condition(s) 1 THEN
	Bloc d'instruction(s) 1
	ELSEIF condition(s) 2 THEN
		CASE variable
		WHEN valeur 1 THEN
		instruction(s) 1

{?} : doit être remplacé:

cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

	Bloc d'instruction(s) 2
	ELSE
	Bloc d'instruction(s) 3
	END IF;
		WHEN valeur 2 THEN
		instruction(s) 2
		...
		ELSE -- équivalent du default en C
		instruction(s) n
		END CASE;

[label_de_boucle:]WHILE condition(s) DO
instruction(s)
END WHILE [label_de_boucle];
	[label_de_boucle:]REPEAT
	instruction(s)
	UNTIL condition(s)
	END REPEAT [label_de_boucle];
		[label_de_boucle:]LOOP
		instruction(s)
		IF l_compteur>l_max THEN
		LEAVE label_de_boucle;
		END IF;
		END LOOP [label_de_boucle];

DECLARE condition_test CONDITION FOR SQLSTATE '99999'; SIGNAL {SQLSTATE valeur_état|condition|condition_test} [SET {MESSAGE_TEXT|MYSQL_ERRNO|SCHEMA_NAME|TAB LE_NAME|...}[, ...]];	DECLARE {CONTINUE|EXIT|UNDO} HANDLER FOR {code_erreur|SQLSTATE [VALUE] etat_sql|nom_condition| SQLWARNING|NOT FOUND|SQLEXCEPTION}[,...] instruction(s);	
• RETURNED_SQLSTATE • MESSAGE_TEXT • MYSQL_ERRNO • CONSTRAINT_NAME • SCHEMA_NAME • TABLE_NAME • COLUMN_NAME • CURSOR_NAME	GET [CURRENT | STACKED] DIAGNOSTICS CONDITION numero_de_la_condition...	
DECLARE nom_du_curseur CURSOR FOR requête_sql; OPEN nom_du_curseur;	FETCH nom_du_curseur INTO var1[,var2,var3,...]; - ERROR 1329 (02000): No data - zero rows fetched, selected, or processed	CLOSE nom_du_curseur;
CREATE TRIGGER nom_trigger moment_trigger action_trigger	SHOW TRIGGERS;	DROP TRIGGER nom_trigger;

{?} : doit être remplacé:

cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

```
ON nom_table
FOR EACH ROW
BEGIN
instruction(s)
END;
```

```
SHOW CREATE TRIGGER nom_trigger;
```

```
SHOW VARIABLES LIKE 'event_scheduler';
-- généralement 'off' par défaut

SET GLOBAL event_scheduler:=ON;

CREATE EVENT [IF NOT EXISTS] nom_event
ON SCHEDULE [AI INTERVAL {v} {u}| EVERY
{v} {u}]
DO {s};
```

```
SHOW EVENTS;

SHOW CREATE EVENT nom_event;

SHOW PROCESSLIST;

ALTER EVENT nom_event
[ON SCHEDULE moment_planification]
[RENAME TO nouveau_nom_event]
[ENABLE | DISABLE]
[DO requête_sql];
```

```
DROP EVENT [IF EXISTS] nom_event;
```

{?} : doit être remplacé:
cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*

Mysql API

import

```
#include <mysql.h>
```

init

```
MYSQL mysql;
mysql_init(&mysql);
```

connect

```
MYSQL mysql;
MYSQL *connexion = NULL;
mysql_real_connect(
    &mysql,
    serveur,
    utilisateur,
    mot_passe,
    nom_db,
    num_port,
    NULL,
    0
);
```

query

```
mysql_query(connexion, {r});
```

close

```
mysql_close(connexion);
```

results

```
MYSQL_RES *resultats = NULL;
// Pour réaliser la lecture d'un enregistrement à la fois
resultats = mysql_use_result(connexion);
// Pour réaliser la lecture de tous les enregistrements à la fois
resultats = mysql_store_result(connexion);
```

nb de colonnes

```
mysql_num_fields(resultats);
```

nb de lignes

```
mysql_num_rows(resultats);
```

lignes affectées

```
mysql_affected_rows(connexion);
```

exploitation

```
MYSQL_ROW ligne;
ligne = mysql_fetch_row(resultats)
```

free

```
mysql_free_result(resultats);
```

{?} : doit être remplacé:

cd : condition *cp : op. de comparaison* *dt : datatype* *f : formats* *s : requête select* *u : units* *v : valeur*