

M02. BASES DE DADES

UF3. llenguatges SQL: DCL i extensió procedimental

GESTIÓ D'USUARIS I PRIVILEGIS

Quins privilegis han de
tenir?

GESTIÓ D'USUARIS I PRIVILEGIS

En un sistema informàtic les dades constitueixen un recurs valuós que ha d'estar controlat i gestionat estrictament.

Entenem per seguretat d'un sistema el conjunt de mecanismes de protecció enfront d'accessos no autoritzats, ja siguin intencionats o accidentals.

A més, si la informació fa referència a persones i s'emmagatzemen dades de naturalesa sensible ens caldrà saber quines són les obligacions legals que tenim.

GESTIÓ D'USUARIS I PRIVILEGIS

Quan utilitzem un sistema gestor de bases de dades (SGBD) per accedir a la informació emmagatzemada en una base de dades, primerament cal comprovar quines autoritzacions tenim sobre aquelles dades; d'això s'encarrega el component de seguretat de l'SGBD.

En un sistema d'informació (SI), les diferents aplicacions i usuaris de l'organització fan servir un únic conjunt de dades, anomenat base de dades corporativa, amb l'SGDB. D'una banda, això resol problemes de redundància, inconsistència i independència entre les dades i els programes i, de l'altra, fa que la seguretat esdevingui un dels problemes més importants en aquests entorns.

GESTIÓ D'USUARIS I PRIVILEGIS. SEGURETAT

- **Confidencialitat:** cal protegir l'ús de la informació per part de persones no autoritzades. Això implica que un usuari només ha de poder accedir a la informació per a la qual té autorització i que a partir d'aquesta informació no podrà inferir altra informació que es consideri secreta.
- **Integritat:** la informació s'ha de protegir de modificacions no autoritzades; això també inclou tant la inserció de dades falses com la destrucció de dades.
- **Disponibilitat:** la informació ha d'estar disponible en el moment que li faci falta a l'usuari.

GESTIÓ D'USUARIS I PRIVILEGIS. SEGURETAT

Definirem el concepte **amenança** com tot aquell agent hostil que, de manera casual o intencionada i utilitzant una tècnica especialitzada, pot revelar o modificar la informació gestionada pel sistema.

- Amenaces no fraudulent (accidents casuais)
- Amenaces fraudulent (violacions intencionades)

GESTIÓ D'USUARIS I PRIVILEGIS. SEGURETAT

- Amenaces no fraudulent (accidents casuais):
 - desastres naturals o accidentals: normalment són accidents que danyen el maquinari del sistema, com per exemple aquells produïts per terratrèmols, inundacions o foc
 - errors del sistema: corresponen a tots aquells errors accidentals en el maquinari o en el programari que poden conduir a accessos no autoritzats
 - errors humans: corresponen a aquelles errades involuntàries derivades de l'acció dels usuaris en introduir dades o utilitzar aplicacions que treballen sobre aquestes

GESTIÓ D'USUARIS I PRIVILEGIS. SEGURETAT

- Amenaces fraudulent: generen violacions intencionades i són causades per dos tipus d'usuaris diferents:
 - usuaris autoritzats que abusen dels seus privilegis
 - agents hostils o usuaris impropis que executen accions de vandalisme sobre el programari o el maquinari del sistema o també lectures o escriptures de dades

GESTIÓ D'USUARIS I PRIVILEGIS. SEGURETAT

Les violacions sobre una base de dades consisteixen en lectures, modificacions o esborraments incorrectes de les dades. Les conseqüències d'aquestes violacions es poden agrupar en tres categories:

1. Lectura inadequada d'informació. Causat per la lectura de dades per part d'usuaris no autoritzats mitjançant un accés intencionat o accidental. S'inclouen les violacions del secret derivades de les deduccions d'informació que es considera secreta.
2. Modificació impròpia de les dades. Correspon a totes les violacions de la integritat de les dades per tractaments o modificacions fraudulentos d'aquestes. Les modificacions impròpies no involucren necessàriament lectures no autoritzades, ja que les dades es poden falsificar sense ser llegides.
3. Denegació de serveis. Correspon a accions que puguin impedir que els usuaris accedeixin a les dades o utilitzin els recursos que tenen assignats.

GESTIÓ D'USUARIS I PRIVILEGIS. NIVELLS DE SEGURETAT

1. Sistema gestor de base de dades: pot ser que alguns usuaris de la base de dades solament tinguin accés a una part limitada de la base de dades. Pot ser que altres usuaris tant sols tinguin autorització per fer consultes però que no puguin modificar les dades. És responsabilitat de l'administrador de l'SGBD que no es violin aquestes restriccions d'autorització.
2. Sistema operatiu: independentment del nivell de seguretat assolit en l'SGBD la debilitat de la seguretat del sistema operatiu pot servir com a mitjà per a accessos no autoritzats a la base de dades.

GESTIÓ D'USUARIS I PRIVILEGIS. NIVELLS DE SEGURETAT

3. Xarxa: atès que gairebé tots els sistemes de bases de dades permeten l'accés remot mitjançant terminals o xarxes, la seguretat en el nivell de programari de la xarxa és tan important com la seguretat física, tant a Internet com en les xarxes privades de les empreses.
4. Físic: els llocs on estan ubicats els sistemes d'informació cal que estiguin adequadament protegits contra l'entrada d'intrusos.
5. Humà: els usuaris han d'estar degudament autoritzats per reduir la possibilitat que algun doni accés a intrusos a canvi de suborns o d'altres favors.

GESTIÓ D'USUARIS I PRIVILEGIS. IDENTIFICACIÓ I AUTENTICACIÓ

La primera acció que cal fer per assolir la seguretat d'un sistema d'informació és la capacitat de verificar la identitat dels usuaris. Aquest procés està format per dues parts:

- Identificació: implica la manera en què l'usuari proporciona la seva identitat al sistema (veure qui és). Segons els requisits operacionals, una identitat pot descriure un individu, més d'un individu, o un o més individus només durant un període de temps.
- Autenticació: és la manera en què un individu estableix la validesa de la seva identitat (verificar que l'usuari és qui diu que és).

GESTIÓ D'USUARIS I PRIVILEGIS. CONTROL D'ACCÉS

Definim **control d'accés** com el conjunt de funcions de l'SGBD per assegurar que els accessos al sistema estan d'acord amb les regles establertes per la política de protecció fixada pel model de negoci.

El control d'accés es pot considerar format per dos components:

- Polítiques d'accés: defineixen els principis pels quals s'autoritza un usuari o es denega l'accés específic a un objecte de la base de dades.
- Mecanismes de seguretat: formats per tots aquells procediments que s'aplicaran a les consultes amb l'objectiu que els usuaris compleixin els principis anteriors.

GESTIÓ D'USUARIS I PRIVILEGIS. AUDITORIA

L'auditoria correspon a un conjunt de mecanismes per saber qui ha fet què, és a dir, portar un registre de qui fa tots els canvis i consultes a la base de dades. Més que un mecanisme de seguretat és un mecanisme per detectar el culpable.

S'utilitza per als casos següents:

- La investigació d'una activitat sospitosa.
- El monitoratge d'activitats específiques de la base de dades.

GESTIÓ D'USUARIS I PRIVILEGIS. AUDITORIA

El sistema d'auditoria ha de permetre diferents formes d'utilització:

- Auditar sentències. L'auditoria indicarà quan i qui ha utilitzat un tipus de sentència correcta. Per exemple, auditar totes les insercions o esborraments.
- Auditar objectes. El sistema auditarà cada vegada que es faci una operació sobre un objecte determinat.
- Auditar sentències sobre objectes, una versió combinada de les dues anteriors.
- Auditar usuaris o grups.

La informació que s'acostuma a emmagatzemar quan es fa una tasca d'auditoria és el nom de l'usuari, l'identificador de la sessió, l'identificador del terminal, el nom de l'objecte al qual s'ha accedit, l'operació executada o intentada, el codi complet de l'operació, la data i l'hora.

GESTIÓ D'USUARIS I PRIVILEGIS. FUNCIONS DEL DBMS ADMIN

Les funcions de l'administrador de la base de dades inclouen:

1. Definició de l'esquema. L'administrador crea l'esquema original de la base de dades escrivint un conjunt d'instruccions de definició de dades.
2. Definició de l'estructura i del mètode d'accés. Referent al programari client emprat i les diferents activitats relacionades amb l'emmagatzematge i recuperació utilitzant diferents estàndards.
3. Modificació de l'esquema i l'organització física. Els administradors de la base de dades fan canvis en l'esquema i l'organització física per reflectir les necessitats canviants dins de l'organització, o per fer alteracions en l'organització física per millorar-ne el rendiment.

GESTIÓ D'USUARIS I PRIVILEGIS. FUNCIONS DEL DBMS ADMIN

4. Concessió d'autorització per a l'accés a les dades. La concessió de diferents tipus d'autorització permet a l'administrador de la base de dades determinar a quines parts de la base de dades pot accedir cada usuari: la informació d'autorització es manté en una estructura de l'esquema especial que el sistema de base de dades consulta quan s'intenta fer l'accés a les dades.

GESTIÓ D'USUARIS I PRIVILEGIS. FUNCIONS DEL DBMS ADMIN

5. Manteniment rutinari. Alguns exemples d'activitats rutinàries de manteniment de l'administrador són:
 - a. Còpia de seguretat periòdica de la base de dades (cinta o servidors remots) per prevenir la pèrdua de dades a causa de desastres naturals.
 - b. Verificació de l'espai lliure necessari al disc per a les operacions habituals i incrementar-lo en cas que sigui necessari.
 - c. Supervisió de les tasques que s'executen a la base de dades i verificar que el rendiment no es degrada per tasques molt costoses iniciades per alguns usuaris.

DEFINICIÓ I MANIPULACIÓ DE DADES

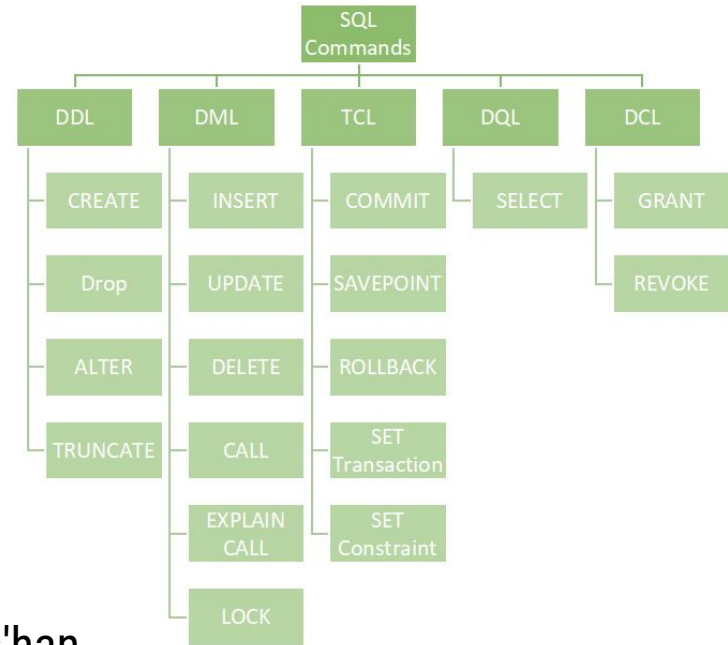
Sentències bàsiques DDL i
DML en MySQL

MYSQL SERVER. SQL STATEMENTS

1. DDL (Data Definition Language)
2. DQL (Data Query Language)
3. DML (Data Manipulation Language)
4. DCL (Data Control Language)
5. TCL (Transaction Control Language)

DCL (Data Control Language)

- GRANT, permet atorgar permisos.
- REVOKE, elimina els permisos que prèviament s'han concedit.



MYSQL SERVER. SETUP

1. Instal·lació:

```
$sudo apt update
```

```
$sudo apt install mysql-server
```

//opcional [securitzar](#) el server

2. Accés al MySQL server:

```
$mysql -u user -p
```

(ens demanarà introduir password)

3. Mostrar les bases de dades actuals

```
mysql> show databases;
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0,00 sec)
```

MYSQL SERVER. CREATING AND SELECTING A DATABASE

1. Creació de la database (base de dades)

```
CREATE DATABASE [IF NOT EXISTS] database_name  
[CHARACTER SET charset_name]  
[COLLATE collation_name];
```

Server Character Set
and Collation

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

2. Accedir a la database (base de dades)

```
mysql> use database_name;
```

3. Mostrar les taules de la database en ús

```
mysql> show tables;
```

MYSQL SERVER. CREATING A TABLE

4. Creació d'una taula

```
CREATE TABLE [IF NOT EXISTS] table_name(  
    column1 datatype,  
    column2 datatype,  
    column3 datatype  
);
```

```
CREATE TABLE pet (  
    name VARCHAR(20),  
    owner VARCHAR(20),  
    species VARCHAR(20),  
    sex CHAR(1),  
    birth DATE,  
    death DATE  
);
```

MYSQL SERVER. CREATING A TABLE

5. Visualització de l'estructura de la taula

```
mysql> describe table_name;
```

```
mysql> describe pet;
```

Field	Type	Null	Key	Default	Extra
name	varchar(20)	YES		NULL	
owner	varchar(20)	YES		NULL	
species	varchar(20)	YES		NULL	
sex	char(1)	YES		NULL	
birth	date	YES		NULL	
death	date	YES		NULL	

6 rows in set (0,01 sec)

MYSQL SERVER. CREATING A TABLE

6. Modificació d'una taula (afegir columnes)

```
ALTER TABLE table_name ADD column_name columntype;
```

```
ALTER TABLE pet ADD id int;  
ALTER TABLE pet ADD PRIMARY KEY (id);  
ALTER TABLE pet MODIFY COLUMN sex char NOT NULL;  
ALTER TABLE pet MODIFY COLUMN id INT auto_increment;  
ALTER TABLE pet DROP COLUMN sex;
```

MYSQL SERVER. CREATING A TABLE

7. Creació d'una taula (estructura general)

```
CREATE TABLE [IF NOT EXISTS] table_name(  
    column1 datatype NOT NULL AUTO_INCREMENT,  
    column2 datatype NOT NULL,  
    column3 datatype DEFAULT NULL,  
    PRIMARY KEY (column1),  
    FOREIGN KEY (column2) REFERENCES tbl_name (col3)  
);
```

[Definicions de la taula](#)

MYSQL SERVER. ADDING DATA TO A TABLE

8. Inserció de valors en una taula

```
INSERT INTO table_name [(col_name [, col_name] ...)]  
    { {VALUES | VALUE} (value_list) [, (value_list)] ... }
```

```
INSERT INTO pet (name, owner, species, sex, birth, date) VALUES  
('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

```
INSERT INTO pet VALUES  
('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

[Tipus d'insercions](#)

MYSQL SERVER. RETRIEVING DATA FROM A TABLE

9. Selecció de dades d'una o diverses taules

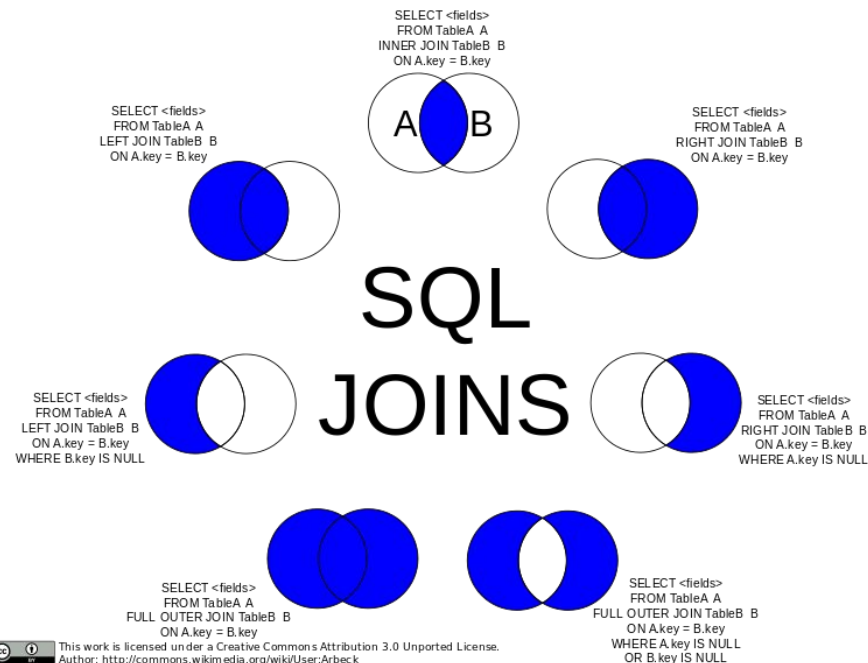
```
SELECT [(col_name [, col_name] ...)] FROM table_name  
{ WHERE... }
```

```
SELECT * FROM pet WHERE name = 'Bowser';  
SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')  
    OR (species = 'dog' AND sex = 'f');  
SELECT name, birth FROM pet ORDER BY birth DESC;
```

[Tipus de queries \(consultes\)](#)

MYSQL SERVER. RETRIEVING DATA FROM A TABLE

9. Selecció de dades d'una o diverses taules



MYSQL SERVER. RETRIEVING DATA FROM A TABLE

9. Selecció de dades d'una o diverses taules

INNER JOIN	<div><div>1</div><div>2</div><div>3</div></div>	INNER JOIN	<div><div>A</div><div>B</div><div>C</div></div>	=	<div><div>1</div><div>2</div></div> <div><div>B</div><div>A</div></div>	Only returns rows that meet the join condition
RIGHT OUTER JOIN	<div><div>1</div><div>2</div><div>3</div></div>	RIGHT OUTER JOIN	<div><div>A</div><div>B</div><div>C</div></div>	=	<div><div>1</div><div>2</div></div> <div><div>B</div><div>A</div><div>C</div></div>	Returns all rows from the table on the right side of JOIN and matched rows from the left side of the JOIN
LEFT OUTER JOIN	<div><div>1</div><div>2</div><div>3</div></div>	LEFT OUTER JOIN	<div><div>A</div><div>B</div><div>C</div></div>	=	<div><div>1</div><div>2</div><div>3</div></div> <div><div>B</div><div>A</div></div>	Returns all rows from the table on the left side of JOIN and matched rows from the right side of the JOIN
FULL OUTER JOIN	<div><div>1</div><div>2</div><div>3</div></div>	FULL OUTER JOIN	<div><div>A</div><div>B</div><div>C</div></div>	=	<div><div>1</div><div>2</div><div>3</div></div> <div><div>B</div><div>A</div><div>C</div></div>	Returns all rows from both sides even if join condition is not met
CROSS JOIN	<div><div>1</div><div>2</div><div>3</div></div>	CROSS JOIN	<div><div>A</div><div>B</div><div>C</div></div>	=	<div><div>1</div><div>1</div><div>1</div><div>2</div><div>2</div><div>2</div><div>3</div><div>3</div><div>3</div></div> <div><div>A</div><div>B</div><div>C</div><div>A</div><div>B</div><div>C</div><div>A</div><div>B</div><div>C</div></div>	Cartesian product between the two sides is a join but without a join condition. Returns all rows joined from both sides

MYSQL SERVER. UPDATING DATA IN A TABLE

10. Modificació de registres d'una taula

```
UPDATE [LOW_PRIORITY] [IGNORE] table_name  
    SET assignment_list  
    [WHERE where_condition]
```

```
UPDATE pet SET owner = 'Kerry' WHERE name = 'Fluffy';
```

Important! Cal indicar a quin registre es vol aplicar la modificació

[Actualització de registres](#)

MYSQL SERVER. DELETING DATA FROM A TABLE

11. Eliminació de registres d'una taula

```
DELETE [LOW_PRIORITY] [IGNORE]  
FROM table_name  
[WHERE where_condition]
```

```
DELETE FROM pet WHERE name = 'Fluffy';  
DELETE FROM pet;
```

[Eliminació de registres](#)

MYSQL SERVER. CONSTRAINTS

Podem aplicar les següents restriccions sobre les columnes de la taula:

- **NOT NULL** o **NULL**: Indica si la columna permet emmagatzemar valors nuls o no.
- **DEFAULT**: Permet indicar un valor inicial per defecte si no especifiquem cap en la inserció.
- **AUTO_INCREMENT**: Serveix per a indicar que és una columna autonumèrica. El seu valor s'incrementa automàticament en cada inserció d'una fila. Només s'utilitza en camps de tipus sencer.
- **UNIQUE KEY**: Indica que el valor de la columna és únic i no poden aparèixer dos valors iguals en la mateixa columna.
- **PRIMARY KEY**: Per a indicar que una columna o vàries són clau primària.
- **CHECK**: Ens permet realitzar restriccions sobre una columna. En les versions prèvies a MySQL 8.0 aquestes restriccions no s'aplicaven, només s'analitzava la sintaxi però eren ignorades pel sistema gestor de base de dades. A partir de la versió de MySQL 8.0 ja sí que s'apliquen les restriccions definides amb CHECK.

MYSQL SERVER. CONSTRAINTS

Restriccions a les Foreign Keys:

ON DELETE i **ON UPDATE**: Ens permeten indicar l'efecte que provoca l'esborrat o l'actualització de les dades que estan referenciats per claus alienes. Les opcions que podem especificar són les següents:

- **RESTRICT**: Impedeix que es puguin actualitzar o eliminar les files que tenen valors referenciats per claus alienes. És l'opció per defecte en MySQL.
- **CASCADE**: Permet actualitzar o eliminar les files que tenen valors referenciats per claus alienes.
- **SET NULL**: Assigna el valor NULL a les files que tenen valors referenciats per claus alienes.
- **NO ACTION**: És una paraula clau de l'estàndard SQL. En MySQL és equivalent a RESTRICT.
- **SET DEFAULT**: No és possible utilitzar aquesta opció quan treballem amb el motor d'emmagatzematge InnoDB. Es pot consultar més informació en la documentació oficial de [MySQL](https://dev.mysql.com/doc/refman/8.0/en/foreign-key-constraints.html).

PROGRAMACIÓ DE BASES DE DADES AMB GUIONS

Triggers, procediments i
funcions

PROGRAMACIÓ DE BASES DE DADES AMB GUIONS

Els **procedures** (procediments), **functions** (funcions) i **triggers** (disparadors) de MySQL són objectes que contenen codi SQL i s'emmagatzemen associats a una base de dades.



PROGRAMACIÓ DE BASES DE DADES AMB GUIONS

Procediment emmagatzemat

És un objecte que es crea amb la sentència `CREATE PROCEDURE` i s'invoca amb la sentència `CALL`. Un procediment pot tenir zero o molts paràmetres d'entrada i zero o molts paràmetres de sortida.

PROGRAMACIÓ DE BASES DE DADES AMB GUIONS

Funció emmagatzemada

És un objecte que es crea amb la sentència `CREATE FUNCTION` i s'invoca amb la sentència `SELECT` o dins d'una expressió. Una funció pot tenir zero o molts paràmetres d'entrada i sempre retorna un valor, associat al nom de la funció.

PROGRAMACIÓ DE BASES DE DADES AMB GUIONS

Trigger

És un objecte que es crea amb la sentència `CREATE TRIGGER` i ha d'estar associat a una taula. Un trigger s'activa quan ocorre un esdeveniment d'inserció, actualització o esborrat, sobre la taula a la qual està associat.

L'objectiu principal és executar una acció quan es produeix un event (succeeix alguna cosa).

AUTOMATITZACIÓ DE TASQUES

TRIGGERS

TRIGGERS

Un trigger és un objecte de la base de dades que està associat amb una taula i que s'activa quan té lloc un esdeveniment sobre la taula.

Els esdeveniments que poden tenir lloc sobre la taula són:

- **INSERT**: El *trigger* s'activa quan s'insereix una nova fila sobre la taula associada.
- **UPDATE**: El *trigger* s'activa quan s'actualitza una fila sobre la taula associada.
- **DELETE**: El *trigger* s'activa quan s'elimina una fila sobre la taula associada.

TRIGGERS

Quan s'executen?



INSERT (inserir) ↗ Before
↘ After

UPDATE (actualitzar) ↗ Before
↘ After

DELETE (eliminar) ↗ Before
↘ After

TRIGGERS

CREATE

[DEFINER = { user | CURRENT_USER }]

TRIGGER trigger_name

trigger_time trigger_event

ON tbl_name FOR EACH ROW

[trigger_order]

trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name

Valid SQL routine statement

TRIGGERS

DELIMITER

Per a definir un procediment emmagatzemat és necessari modificar temporalment el caràcter separador que s'utilitza per a delimitar les sentències SQL.

El caràcter separador que s'utilitza per defecte en SQL és el punt i coma (;). En els exemples que realitzarem en aquesta unitat utilitzarem els caràcters // per a delimitar les instruccions SQL, però és possible utilitzar qualsevol altre caràcter (\$\$, |).

TRIGGERS

Els triggers permeten l'accés als valors des de la taula amb finalitats de comparació utilitzant els nous i els antics. La disponibilitat dels modificadors depèn de l'esdeveniment d'activació que utilitzeu:

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

TRIGGERS

```
delimiter //  
CREATE TRIGGER article_ai AFTER INSERT ON article  
FOR EACH ROW  
    INSERT INTO log_article (codi_article, usuari, data)  
    VALUES (NEW.codi_article, CURRENT_USER(), NOW()); //  
delimiter ;
```

```
delimiter //  
CREATE TRIGGER article_au AFTER UPDATE ON article  
FOR EACH ROW  
    INSERT INTO log_article (codi_article, usuari, data, codi_article_old)  
    VALUES (NEW.codi_article, CURRENT_USER(), NOW(), OLD.codi_article); //  
delimiter ;
```

TRIGGERS

DECLARE

És possible declarar variables locals amb la paraula reservada **DECLARE** a funcions, procediments i triggers.

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

L'àmbit d'una variable local serà el bloc **BEGIN** i **END** del procediment o la funció on ha estat declarada.


```
DECLARE total INT UNSIGNED;
```

TRIGGERS

Estructures de control: IF-THEN-ELSE

```
IF search_condition THEN statement_list  
  [ELSEIF search_condition THEN statement_list] ...  
  [ELSE statement_list]  
END IF
```

```
BEGIN  
  IF NEW.nota < 0 THEN  
    set NEW.nota = 0;  
  ELSEIF NEW.nota > 10 THEN  
    set NEW.nota = 10;  
  END IF;  
END
```



el bloc BEGIN...END
s'utilitza per a
escriure
sentències
compostes

TRIGGERS

```
delimiter //
```

```
CREATE TRIGGER article_ai AFTER INSERT ON article  
FOR EACH ROW  
BEGIN
```

```
    DECLARE total_productes INT;  
    DECLARE total_quantitat INT;
```

```
    SELECT COUNT(*)  
    INTO total_productes  
    FROM article;
```

```
    SELECT SUM(preu*quantitat)  
    INTO total_quantitat  
    FROM article;
```

```
    IF total_productes > 0 THEN  
        INSERT INTO stats_article(id_accio, total_productes, total_quantitat, data)  
        VALUES(NULL, total_productes, total_quantitat, NOW());  
    END IF;
```

```
END;  
//
```

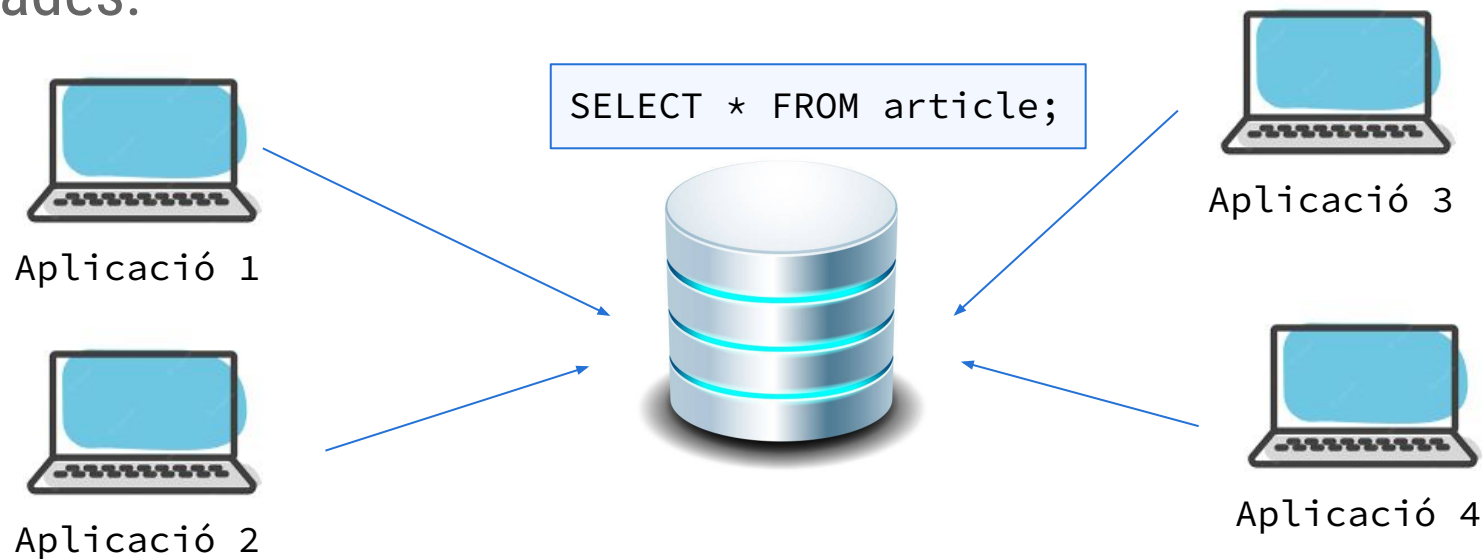
```
delimiter ;
```

AUTOMATITZACIÓ DE
TASQUES

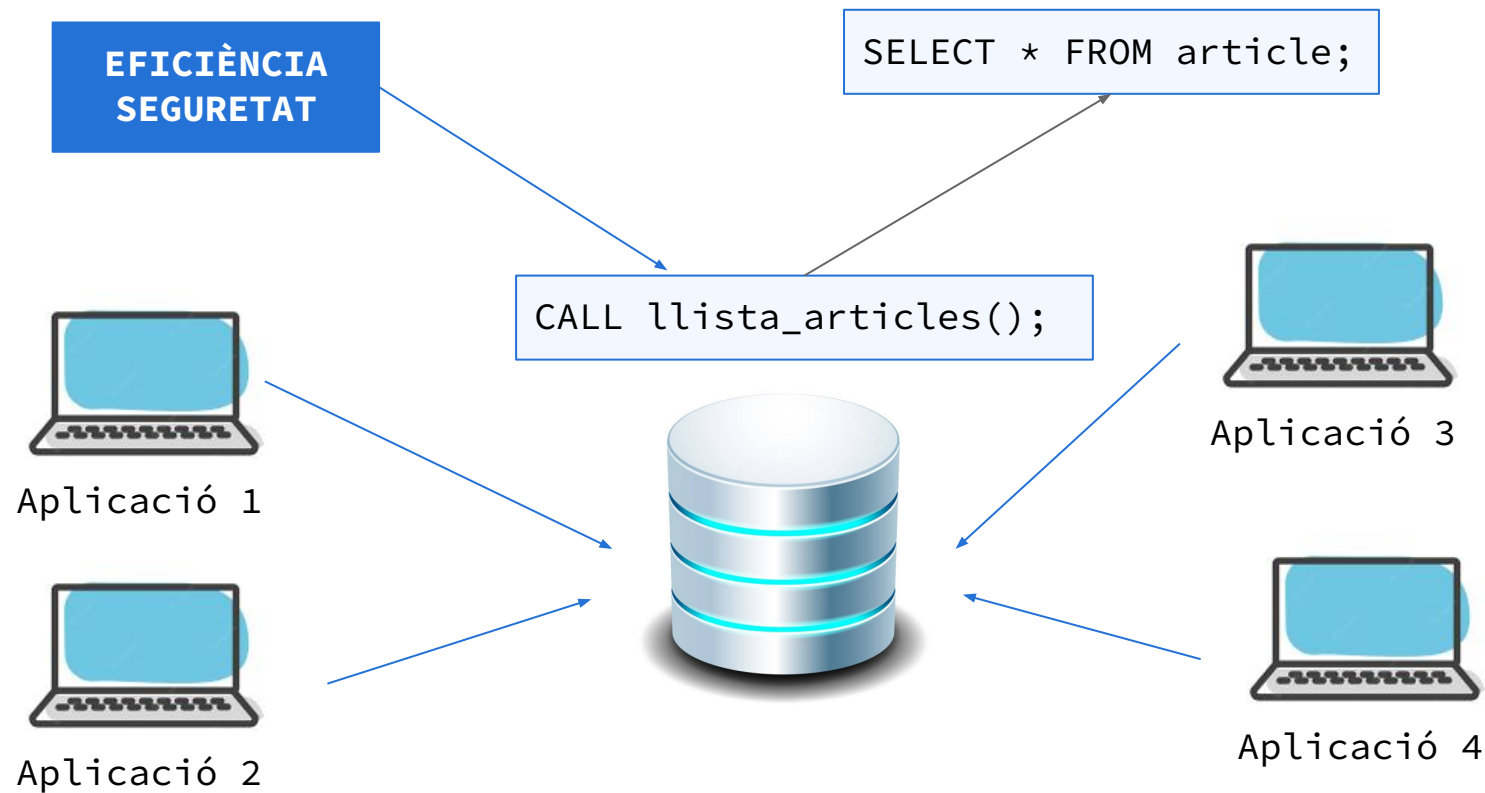
PROCEDURES

PROCEDURES

Un *procedure* (procediment emmagatzemat) és un conjunt d'instruccions SQL que s'emmagatzema associat a una base de dades.



PROCEDURES



PROCEDURES

Hi ha 4 tipus de *procedures* en MySQL:

- sense paràmetres
- amb paràmetres d'entrada
- amb paràmetres de sortida
- amb paràmetres d'entrada/sortida

PROCEDURES

Sense paràmetres

Un procediment sense paràmetres no rep cap valor d'entrada o emet una sortida indirectament. Es crida simplement amb el seu nom de procediment seguit dels parèntesi () (sense cap paràmetre dins). S'utilitza per a consultes senzilles.

```
delimiter //  
CREATE PROCEDURE llista_articles()  
SELECT * FROM article;//  
  
CALL llista_articles();
```

PROCEDURES

Amb paràmetres d'entrada

S'indiquen mitjançant la paraula reservada `IN` davant del nom del paràmetre. Aquests paràmetres no poden canviar el seu valor dins del procediment, és a dir, quan el procediment finalitzi aquests paràmetres tindran el mateix valor que tenien quan es va fer l'anomenada al procediment.

PROCEDURES

Amb paràmetres de sortida

S'indiquen posant la paraula reservada `OUT` davant del nom del paràmetre. Aquests paràmetres canvien el seu valor dins del procediment. Quan es fa l'anomenada al procediment comencen amb un valor inicial i quan finalitza l'execució del procediment poden acabar amb un altre valor diferent.

PROCEDURES

Amb paràmetres d'entrada/sortida

És una combinació dels tipus `IN` i `OUT`. Aquests paràmetres s'indiquen posant la paraula reservada IN/OUT davant del nom del paràmetre.

PROCEDURES

CREATE

```
[DEFINER = { user | CURRENT_USER }]
PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...] routine_body
```

proc_parameter:

```
[ IN | OUT | INOUT ] param_name type
```

func_parameter:

```
param_name type
```

type:

```
Any valid MySQL data type
```

characteristic:

```
COMMENT 'string'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
```

routine_body:

```
Valid SQL routine statement
```

PROCEDURES

```
delimiter //  
CREATE PROCEDURE llistar_productes(IN proc_codi VARCHAR(6))  
BEGIN  
    SELECT *  
    FROM article  
    WHERE article.codi_article = proc_codi;  
END //  
delimiter ;
```

PROCEDURES

```
delimiter //  
CREATE PROCEDURE max_preus(OUT m_preu FLOAT)  
BEGIN  
    SELECT MAX(preu)  
    INTO m_preu  
    FROM article;  
END //  
delimiter ;
```

```
CALL max_preus(@m_preu);  -- assigna el valor al paràmetre  
SELECT @m_preu;           -- mostra el valor del paràmetre
```

FUNCTIONS

Una *function* (funció emmagatzemada) és un conjunt d'instruccions SQL que s'emmagatzema associada a una base de dades.

És un objecte que es crea amb la sentència `CREATE FUNCTION` i s'invoca amb la sentència `SELECT` o dins d'una expressió. Una funció pot tenir zero o molts paràmetres d'entrada i sempre retorna un valor, associat al nom de la funció.

AUTOMATITZACIÓ DE
TASQUES

FUNCTIONS

FUNCTIONS

Tots els paràmetres són d'entrada, per tant, no serà necessari utilitzar la paraula reservada `IN` davant del nom dels paràmetres. Sempre retornarà un valor de sortida, associat al nom de la funció.

En la definició de la capçalera de la funció cal definir el tipus de dada que retorna amb la paraula reservada `RETURNS` i en el cos de la funció s'ha d'incloure la paraula reservada `RETURN`, per a retornar el valor de la funció.

FUNCTIONS

CREATE

```
[DEFINER = { user | CURRENT_USER }]
FUNCTION sp_name ([func_parameter[,...]])
RETURNS type
[characteristic ...] routine_body
```

func_parameter:

```
param_name type
```

type:

Any valid MySQL data type

characteristic:

```
COMMENT 'string'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
```

routine_body:

Valid SQL routine statement

FUNCTIONS

Característiques

- **DETERMINISTIC**: Indica que la funció sempre retorna el mateix resultat quan s'utilitzen els mateixos paràmetres d'entrada.
- **NOT DETERMINISTIC**: Indica que la funció no sempre retorna el mateix resultat, encara que s'utilitzin els mateixos paràmetres d'entrada. Aquesta és l'opció que se selecciona per defecte quan no s'indica una característica de manera explícita.
- **CONTAINS SQL**: Indica que la funció conté sentències SQL, però no conté sentències de manipulació de dades. Alguns exemples de sentències SQL que poden aparèixer en aquest cas són operacions amb variables (Ej: `SET @x = 1`) o ús de funcions de MySQL (Ej: `SELECT NOW();`) entre altres. Però en cap cas apareixeran sentències d'escriptura o lectura de dades.
- **NO SQL**: Indica que la funció no conté sentències SQL.
- **READS SQL DATA**: Indica que la funció no modifica les dades de la base de dades i que conté sentències de lectura de dades, com la sentència `SELECT`.
- **MODIFIES SQL DATA**: Indica que la funció sí que modifica les dades de la base de dades i que conté sentències com `INSERT`, `UPDATE` o `DELETE`.

FUNCTIONS

Per a poder crear una funció en MySQL és necessari indicar almenys una d'aquestes tres característiques:

- DETERMINISTIC
- NO SQL
- READS SQL DATA

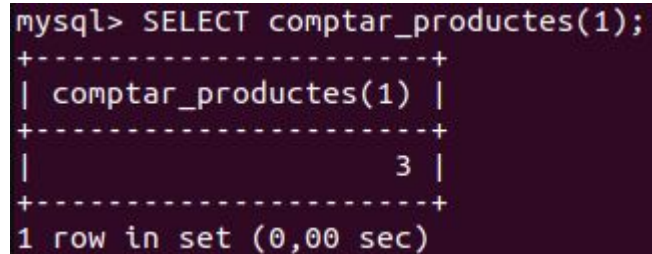
Si no s'indica almenys una d'aquestes característiques, obtindrem el següent missatge d'error:

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,  
or READS SQL DATA in its declaration and binary logging is enabled  
(you *might* want to use the less safe log_bin_trust_function_creators  
variable)
```

FUNCTIONS

```
delimiter //  
CREATE FUNCTION comptar_productes(codi_tipus INT(4))  
RETURNS INT  
DETERMINISTIC  
BEGIN  
    DECLARE total INT;  
    SET total = (  
        SELECT COUNT(*)  
        FROM article  
        WHERE article.codi_tipus = codi_tipus);  
    RETURN total;  
END //  
delimiter ;
```

```
SELECT comptar_productes(1);
```



A terminal window with a dark purple background showing the execution of the MySQL function. The prompt is 'mysql>'. The command is 'SELECT comptar_productes(1);'. The output is a table with one row and one column, showing the value '3'. Below the table, it says '1 row in set (0,00 sec)'.

```
mysql> SELECT comptar_productes(1);  
+-----+  
| comptar_productes(1) |  
+-----+  
|                      3 |  
+-----+  
1 row in set (0,00 sec)
```

AUTOMATITZACIÓ DE TASQUES

ELEMENTS COMUNS

ELEMENTS COMUNS

Els elements que es poden utilitzar a les tasques automatitzades són els següents:

- BEGIN ... END bloc
- DECLARE statement
- Flow control statements
 - Conditionals
 - Iteratives

ESTRUCTURES DE CONTROL

Instruccions condicionals

Hi ha dues estructures de control condicionals:

- *If-Then-Else*
- *Case*

ESTRUCTURES DE CONTROL CONDICIONALES

If-Then-Else statement

```
IF search_condition THEN statement_list  
    [ELSEIF search_condition THEN statement_list] ...  
    [ELSE statement_list]  
END IF
```

ESTRUCTURES DE CONTROL CONDICIONALS

CASE statement

```
CASE case_value
  WHEN when_value THEN statement_list
  [WHEN when_value THEN statement_list] ...
  [ELSE statement_list]
END CASE
```

```
CASE
  WHEN search_condition THEN statement_list
  [WHEN search_condition THEN statement_list] ...
  [ELSE statement_list]
END CASE
```


ESTRUCTURAS DE CONTROL CONDICIONALES

```
CASE customerCountry
  WHEN 'USA' THEN
    SET pShipping = '2-day Shipping';
  WHEN 'Canada' THEN
    SET pShipping = '3-day Shipping';
  ELSE
    SET pShipping = '5-day Shipping';
END CASE;
```

```
delimiter //

CREATE PROCEDURE GetCustomerShipping(
  IN pCustomerNumber INT,
  OUT pShipping VARCHAR(50)
)
BEGIN
  DECLARE customerCountry VARCHAR(100);

  SELECT country INTO customerCountry
  FROM customers
  WHERE customerNumber = pCustomerNumber;

  CASE customerCountry
    WHEN 'USA' THEN
      SET pShipping = '2-day Shipping';
    WHEN 'Canada' THEN
      SET pShipping = '3-day Shipping';
    ELSE
      SET pShipping = '5-day Shipping';
  END CASE;
END //

delimiter ;
```

ESTRUCTURAS DE CONTROL CONDICIONALES

CASE

```
WHEN waitingDay = 0 THEN
    SET pDeliveryStatus = 'On Time';
WHEN waitingDay >= 1 AND waitingDay < 5 THEN
    SET pDeliveryStatus = 'Late';
WHEN waitingDay >= 5 THEN
    SET pDeliveryStatus = 'Very Late';
ELSE
    SET pDeliveryStatus = 'No Information';
END CASE;
```

```
delimiter //

CREATE PROCEDURE GetDeliveryStatus(
    IN pOrderNumber INT,
    OUT pDeliveryStatus VARCHAR(100)
)
BEGIN
    DECLARE waitingDay INT DEFAULT 0;
    SELECT
        DATEDIFF(requiredDate, shippedDate)
    INTO waitingDay
    FROM orders
    WHERE orderNumber = pOrderNumber;

    CASE
        WHEN waitingDay = 0 THEN
            SET pDeliveryStatus = 'On Time';
        WHEN waitingDay >= 1 AND waitingDay < 5 THEN
            SET pDeliveryStatus = 'Late';
        WHEN waitingDay >= 5 THEN
            SET pDeliveryStatus = 'Very Late';
        ELSE
            SET pDeliveryStatus = 'No Information';
        END CASE;
    END //

delimiter ;
```

ESTRUCTURES DE CONTROL

Instruccions iteratives

Hi ha tres estructures de control iteratives:

- *While*
- *Repeat*
- *Loop*

ESTRUCTURES DE CONTROL ITERATIVES

While

```
[begin_label:] WHILE search_condition DO  
    statement_list  
END WHILE [end_label]
```

ESTRUCTURAS DE CONTROL ITERATIVAS

```
BEGIN
```

```
    DECLARE x INT;
```

```
    DECLARE string_value varchar(20);
```

```
    SET x = 1;
```

```
    SET string_value = "";
```

```
    WHILE x <= 5 DO
```

```
        SET string_value = CONCAT(string_value,x," , ");
```

```
        SET x = x + 1;
```

```
    END WHILE;
```

```
    SELECT string_value;
```

```
END
```

ESTRUCTURES DE CONTROL ITERATIVES

Repeat

```
[begin_label:] REPEAT  
    statement_list  
UNTIL search_condition  
END REPEAT [end_label]
```

ESTRUCTURAS DE CONTROL ITERATIVAS

BEGIN

DECLARE x INT;

DECLARE string_value varchar(20);

SET x = 1;

SET string_value = "";

REPEAT

SET string_value = CONCAT(string_value,x," , ");

SET x = x + 1;

UNTIL x > 5

END REPEAT;

SELECT string_value;

END

ESTRUCTURES DE CONTROL ITERATIVES

Loop

```
[begin_label:] LOOP  
    statement_list  
END LOOP [end_label]
```


ESTRUCTURAS DE CONTROL ITERATIVAS

BEGIN

DECLARE x INT;

DECLARE string_value varchar(20);

SET x = 1;

SET string_value = "";

label_loop: LOOP

IF x > 5 THEN

LEAVE label_loop;

END IF;

SET string_value = CONCAT(string_value,x,",");

SET x = x + 1;

END LOOP;

SELECT string_value;

END

ESTRUCTURES DE CONTROL ITERATIVES

ITERATE = CONTINUE THE LOOP BUT
ESCAPE AT SOME CONDITION

LEAVE = TERMINATE THE RUNNING LOOP

```
BEGIN
    DECLARE x INT;
    DECLARE string_value varchar(20);

    SET x = 1;
    SET string_value = "";

    label_loop: LOOP
        IF x >= 20 THEN
            LEAVE label_loop;
        END IF;

        SET x = x + 1;

        IF (x mod 2) THEN
            ITERATE label_loop;
        ELSE
            SET string_value = CONCAT(string_value,x,",");
        END IF;
    END LOOP;

    SELECT string_value;
END//
```

AUTOMATITZACIÓ DE
TASQUES

TRACTAMENT D'ERRORS

TRACTAMENT D'ERRORS

```
DECLARE handler_action HANDLER
    FOR condition_value [, condition_value] ...
    statement
```

```
handler_action: {
    CONTINUE
    | EXIT
    | UNDO
}
```

```
condition_value: {
    mysql_error_code
    | SQLSTATE [VALUE] sqlstate_value
    | condition_name
    | SQLWARNING
    | NOT FOUND
    | SQLEXCEPTION
}
```

Les accions possibles que podem seleccionar com handler_action són:

- **CONTINUE**: L'execució del programa continua.
- **EXIT**: Acaba l'execució del programa.
- **UNDO**: No està suportat en MySQL.

TRACTAMENT D'ERRORS

```
DECLARE CONTINUE HANDLER FOR 1051  
BEGIN  
    -- body of handler  
END;
```

S'executarà quan es produeixi l'error 1051

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'  
BEGIN  
    -- body of handler  
END;
```

S'executarà quan la variable SQLSTATE tingui el valor 42S02

```
DECLARE CONTINUE HANDLER FOR SQLWARNING  
BEGIN  
    -- body of handler  
END;
```

S'executarà quan SQLSTATE tingui el valor 01

TRACTAMENT D'ERRORS

```
CREATE TABLE usuari (id_usuari INT, PRIMARY KEY (id_usuari));
```

```
DELIMITER //
```

```
CREATE PROCEDURE handlerdemo ()
```

```
BEGIN
```

```
    DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SELECT 'Aquí es  
produeix error';
```

```
    INSERT INTO usuari VALUES (1);
```

```
    SELECT 'Missatge després del primer insert';
```

```
    INSERT INTO usuari VALUES (1);
```

```
    SELECT 'Missatge després del segon insert';
```

```
    INSERT INTO usuari VALUES (2);
```

```
    SELECT 'Missatge després del tercer insert';
```

```
END //
```

```
DELIMITER ;
```

```
CALL handlerdemo();
```

TRACTAMENT D'ERRORS

```
CREATE TABLE usuari (id_usuari INT, PRIMARY KEY (id_usuari));

DELIMITER //
CREATE PROCEDURE handlerdemo ()
BEGIN
    DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'Aquí es produeix
error';
    INSERT INTO usuari VALUES (1);
    SELECT 'Missatge després del primer insert';
    INSERT INTO usuari VALUES (1);
    SELECT 'Missatge després del segon insert';
    INSERT INTO usuari VALUES (2);
    SELECT 'Missatge després del tercer insert';
END //

DELIMITER ;
CALL handlerdemo();
```

AUTOMATITZACIÓ DE
TASQUES

CURSORS

CURSORS

Un cursor és una estructura de control que serveix per a processar fila a fila els resultats d'una consulta.

```
DECLARE cursor_name CURSOR FOR select_statement
```



Declaració del cursor

```
OPEN cursor_name
```



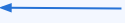
Obertura del cursor

```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ..
```



Obtenció de cada fila

```
DECLARE CONTINUE HANDLER FOR NOT FOUND ..
```



Tractament d'errors

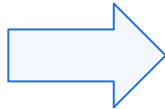
```
CLOSE cursor_name
```



Tancament del cursor

AUTOMATITZACIÓ DE TASQUES. CURSORS

```
CREATE PROCEDURE curdemo()  
BEGIN  
    DECLARE done INT DEFAULT FALSE;  
    DECLARE a CHAR(16);  
    DECLARE b, c INT;  
    DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;  
    DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;  
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;  
  
    OPEN cur1;  
    OPEN cur2;  
  
    read_loop: LOOP  
        FETCH cur1 INTO a, b;  
        FETCH cur2 INTO c;  
        IF done THEN  
            LEAVE read_loop;  
        END IF;  
        IF b < c THEN  
            INSERT INTO test.t3 VALUES (a,b);  
        ELSE  
            INSERT INTO test.t3 VALUES (a,c);  
        END IF;  
    END LOOP;  
  
    CLOSE cur1;  
    CLOSE cur2;  
END;
```



```
DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;  
DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;  
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;  
  
OPEN cur1;  
OPEN cur2;  
  
read_loop: LOOP  
    FETCH cur1 INTO a, b;  
    FETCH cur2 INTO c;  
    IF done THEN  
        LEAVE read_loop;  
    END IF;  
    IF b < c THEN  
        INSERT INTO test.t3 VALUES (a,b);  
    ELSE  
        INSERT INTO test.t3 VALUES (a,c);  
    END IF;  
END LOOP;  
  
CLOSE cur1;  
CLOSE cur2;
```