MySQL,
The Comprehensive Course
الدورة الشاملة

إعداد وتقديم: د. عبدالناصر ضياف

12

برمجة MySQL

استدراك بعض المواضيع مثل فرض مبدأ الذرية Atomicity على مجموعة من العمليات والمعروفة بـ Transaction، والتعرف على مفهوم الجدول المؤقت Temporary Table، بالإضافة إلى فائدة الأعمدة المولَّدة Generated Columns.

الدخول إلى عالم البرمجة الإجرائية وأدواتها التي يقدمها نظام MySQL

MySQL Transactions

- لفهم ماهية الTransaction في MySQL، دعنا نأخذ مثالا لإضافة أمر مبيعات جديد في نموذج قاعدة البيانات classic models. وهذه هي الخطوات:
  - أولاً، استعلم عن أحدث رقم أمر مبيعات من جدول الطلبات واستخدم رقم أمر المبيعات التالي كرقم أمر المبيعات الجديد
  - بعد ذلك، قم بإدراج أمر مبيعات جديد في جدول الطلبات
  - ثم احصل على رقم أمر المبيعات المدرج حديثًا
  - بعد ذلك، قم بإدراج أصناف أمر المبيعات الجديدة في جدول تفاصيل الطلب مع رقم أمر المبيعات
  - أخيرًا، حدد البيانات من جداول الطلبات وتفاصيل الطلب لتأكيد التغييرات
- الآن، تخيل ماذا سيحدث لبيانات أمر المبيعات إذا فشلت خطوة أو أكثر من الخطوات المذكورة أعلاه لسبب أو لآخر مثل قفل الجدول، فإذا فشلت خطوة إضافة عناصر الطلب إلى جدول تفاصيل الطلب، فسيكون لديك أمر مبيعات فارغ.
- هذا هو السبب في أن معالجة Transactions أتت للإنقاذ.
- تسمح لك MySQL Transaction بتنفيذ مجموعة من العمليات للتأكد من أن قاعدة البيانات لا تحتوي أبدًا على نتائج جزئية للعمليات. في حال فشل عملية جزئية أو أكثر، يحدث التراجع ROLL BACK لاستعادة قاعدة البيانات إلى حالتها الأصلية. أما في حال النجاح الكامل، يتم تثبيت COMMIT تأثيرات العمليات بأكملها في قاعدة البيانات.

MySQL Transaction Statements

- تزودنا MySQL بالتعليمة المهمة التالية للتحكم في transactions:
  - لبدء transaction، يمكننا استخدام تعليمة البدء `START TRANSACTION` أو `BEGIN` `[WORK]`
  - لتثبيت تأثيرات الtransaction الحالي وجعل تغييراتها دائمة، يمكننا استخدام عبارة `COMMIT` `[WORK]`
  - للتراجع عن الtransaction الحالي والغير مثبت وإلغاء تغييراته، يمكننا استخدام عبارة `ROLLBACK` `[WORK]`

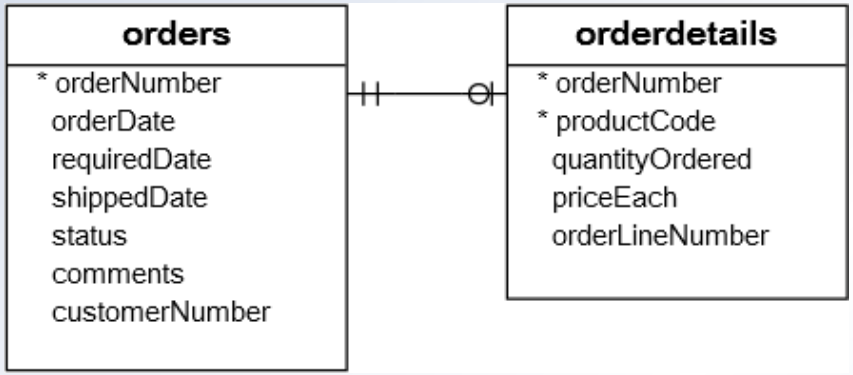
- لتعطيل أو تمكين وضع التثبيت التلقائي لتأثيرات العمليات، يمكنك استخدام عبارة `SET autocommit`

  ← **تنبيه** autocommit ليس له تأثير داخل الtransaction

  - الوضع الطبيعي في MySQL أنه يقوم تلقائيا وآنيًا بتثبيت التغييرات بشكل دائم على قاعدة البيانات
  - لإجبار MySQL على عدم تثبيت التغييرات تلقائيًا، نستخدم التعليمة التالية:
    - `SET autocommit = 0;` or `SET autocommit = OFF;`
  - بالمقابل،، إذا أردنا إعادة تفعيل التثبيت التلقائي للتغييرات نستخدم التالي:
    - `SET autocommit = 1;` or `SET autocommit = ON;`

مثال حول MySQL Transaction

- سنستخدم جدول الطلبيات وتفاصيل الطلبيات من نموذج قاعدة البيانات classicmodels للتوضيح:



- **مثال عن تثبيت التغييرات COMMIT**
- توضيح خطوات إنشاء طلبية جديدة:

1) نبدأ transaction باستخدام تعليمة START TRANSACTION
2) نجلب آخر رقم طلبية من جدول الطلبيات حتى نستخدم الذي يليه كرقم طلبية جديد
3) نقوم بإدراج طلبية جديدة في جدول الطلبيات orders
4) نقوم بإدراج عناصر الطلبية في جدول تفاصيل الطلبيات orderdetails
5) وأخيرًا، نقوم بتثبيت تغييرات الtransaction باستخدام عبارة COMMIT

- للتحقق: يمكننا جلب البيانات من الجدولين المعنيين للتحقق من الطلبية الجديدة

مثال حول MySQL Transaction

```
START TRANSACTION;

SELECT @orderNumber:=MAX(orderNUmber)+1 FROM orders;

INSERT INTO orders(orderNumber, orderDate, requiredDate,
   shippedDate, status, customerNumber)
VALUES(@orderNumber, '2005-05-31', '2005-06-10', '2005-06-11',
   'In Process', 145);

INSERT INTO orderdetails(orderNumber, productCode,
   quantityOrdered, priceEach, orderLineNumber)
VALUES(@orderNumber,'S18_1749', 30, '136', 1),
      (@orderNumber,'S18_2248', 50, '55.09', 2);

COMMIT;

--للتحقق
SELECT a.orderNumber, orderDate, requiredDate, shippedDate,
   status, comments, customerNumber, orderLineNumber,
   productCode, quantityOrdered, priceEach
FROM orders a INNER JOIN orderdetails b USING (orderNumber)
WHERE a.ordernumber = @orderNumber;
```

مثال حول MySQL Transaction

- **مثال عن التراجع عن تثبيت التغييرات ROLLBACK**

1) لنقم ببدء transaction
2) نحذف جميع الطلبيات من جدول الطلبيات
3) من خلال دخول آخر، نتفحص محتوى الجدول
4) من ثم نتراجع عن تثبيت التغييرات ونتفحص محتوى الجدول

الجداول المؤقتة في MySQL

Temporary Tables in MySQL

- في MySQL، الجدول المؤقت هو نوع خاص من الجداول يسمح لك بتخزين مجموعة نتائج مؤقتا، والتي يمكنك إعادة استخدامها عدة مرات في نفس الجلسة session
- يتم إنشاء الجداول المؤقتة بشكل سريع وتستخدم عادةً لإجراء عمليات حسابية معقدة أو تخزين النتائج الوسيطة أو معالجة مجموعات فرعية من البيانات أثناء تنفيذ استعلام أو سلسلة من الاستعلامات.
- متى نحتاج استخدام الجدول المؤقت؟
  - عندما يكون الاستعلام عن البيانات التي تتطلب عبارة SELECT واحدة مستحيلًا
  - عندما يكون جلب النتائج في مرحلة واحدة مكلفًا
- يحتوي جدول MySQL المؤقت على الخصائص التالية:
  - يتم إنشاء جدول مؤقت باستخدام عبارة CREATE TEMPORARY TABLE
  - يمكننا استخدام عبارة DROP TABLE لإزالة جدول مؤقت بشكل صريح عندما لم نعد تستخدمه
  - يقوم MySQL بإزالة الجدول المؤقت تلقائيًا عند انتهاء الجلسة أو إنهاء الاتصال
  - الجدول المؤقت متاح فقط ويمكن الوصول إليه من قبل العميل client الذي قام بإنشائه
  - يمكن للعملاء المختلفين إنشاء جداول مؤقتة بنفس الاسم دون التسبب في حدوث أي أخطاء
  - يمكن أن يكون للجدول المؤقت نفس اسم الجدول العادي في قاعدة البيانات إلا أن المؤقت يحجب العادي
    - احذر هنا من اعادة الاتصال التلقائي لأنه غير منظور مع غياب التمييز بين المؤقت والعادي
    - لذا تجنب تشابه الأسماء نهائيا وتمسك باستخدام DROP TEMPORARY TABLE بدلا عن DROP TABLE

أمثلة على الجداول المؤقتة في MySQL

بناء وتعبئة جدول مؤقت:

```
CREATE TEMPORARY TABLE credits(
  customerNumber INT PRIMARY KEY,
  creditLimit DEC(10, 2)
);


INSERT INTO credits(customerNumber, creditLimit)
SELECT
  customerNumber,
  creditLimit
FROM
  customers
WHERE
  creditLimit > 0;
```

أمثلة على الجداول المؤقتة في MySQL

بناء جدول مؤقت بتركيبة ناتجة عن تعليمة استعلام:

```
CREATE TEMPORARY TABLE top_customers
SELECT p.customerNumber,
       c.customerName,
       SUM(p.amount) sales
FROM payments p
INNER JOIN customers c ON c.customerNumber = p.customerNumber
GROUP BY p.customerNumber
ORDER BY sales DESC
LIMIT 10;


SELECT
    customerNumber,
    customerName,
    sales
FROM
    top_customers
ORDER BY sales;
```

customerNumber	customerName	sales
146	Saveley & Henriot, Co.	130305.35
321	Corporate Gift Ideas Co.	132340.78
276	Anna's Decorations, Ltd	137034.22
187	AV Stores, Co.	148410.09
323	Down Under Souveniers, Inc	154622.08
148	Dragon Souveniers, Ltd.	156251.03
151	Muscle Machine Inc	177913.95
114	Australian Collectors, Co.	180585.07
124	Mini Gifts Distributors Ltd.	584188.24
141	Euro+ Shopping Channel	715738.98

الأعمدة المولّدة أو المشتقة في MySQL

Generated Columns in MySQL

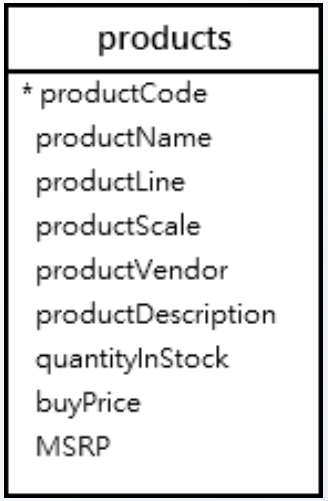
- منذ MySQL 5.7 تم إضافة ميزة جديدة تسمى العمود المولَّد Generated Column. يتم حساب البيانات الموجودة في هذه الأعمدة بناءً على أعمدة أخرى من خلال تعبيرات محددة مسبقًا.
- بشكل أساسي، العمود المولّد هو عمود يتم حسابه بواسطة تعبير expression، بدلاً من تزويده بشكل صريح.
- **الشكل النحوي:**

```
<column_name> <data_type> GENERATED ALWAYS AS (expression)
   [VIRTUAL | STORED]
```

- **افتراضي أم مخزّن Virtual vs Stored:**
  - عند إنشاء عمود مولّد، يمكننا تحديد ما إذا كان يجب أن يكون افتراضيًا أو مخزّنًا باستخدام الكلمة الأساسية VIRTUAL أو STORED. إذا لم تحدد أي شيء، فسيكون العمود افتراضيًا.
  - يتم حساب العمود الافتراضي في وقت الاستعلام ولا يتم حفظه. وهذا يعني أن الحساب قد يستغرق وقتًا أطول، لكنه لا يؤثر على الحجم الإجمالي للجدول.
  - من ناحية أخرى، يتم حساب العمود المخزن أثناء إدراج البيانات أو تحديثها وحفظها، تمامًا مثل العمود العادي. وهذا يعني أنه قد يكون من الأسرع استرداد البيانات من عمود مخزن، ولكنها ستشغل مساحة أكبر على القرص.
- **حالات الاستخدام الشائعة للأعمدة المولّدة:**
  - اجراء بعض الحسابات للحاجة لها في التقارير مثلا
  - استخلاص بيانات مفتاحية من عمود JSON
  - دعم قيود سلامة البيانات بشكل أو بآخر

أمثلة على الأعمدة المولَّدة في MySQL

إضافة عمود في جدول المنتجات products يوفر قيمة المخزون من الصنف:




```
ALTER TABLE products
  ADD COLUMN stockValue DOUBLE
  GENERATED ALWAYS
  AS (buyprice*quantityinstock) STORED;
```

```
SELECT
    productName,
    stockValue
FROM
    products;
```

productName	stock_value
1969 Harley Davidson Ultimate Chopper	387209.73
1952 Alpine Renault 1300	720126.90
1996 Moto Guzzi 1100i	457058.75
2003 Harley-Davidson Eagle Drag Bike	508073.64
1972 Alfa Romeo GTA	278631.36
1962 LanciaA Delta 16V	702325.22
1968 Ford Mustang	6483.12
2001 Ferrari Enzo	345940.21
1958 Setra Bus	123004.10
2002 Suzuki XREO	662501.19
1969 Corvair Monza	615600.84

ماذا لو كنا نريد فرض فعالية صف واحد ضمن مجموعة من الصفوف؟

تمهيد

- هناك عدة طرق مختلفة لحماية قاعدة البيانات من الفساد:
  - نوع البيانات لكل عمود
  - المفتاح الأساسي وقيود التفرّد الأخرى
  - قيود التكامل المرجعي
- إحدى الفوائد الرئيسية للقيام بكل هذا في قاعدة البيانات هي أنه لا يوجد "باب خلفي" لقاعدة البيانات
- والفائدة الأخرى هي أن الكيانات البرمجية المخزنة (StoredProcedure, Function, Trigger, Event) تعمل على الخادم، مما يضمن الالتزام بالشروط والقيود ويوفر موارد الشبكة من حيث المرور Traffic

أساليب أخرى

- كبديل،،، يمكن أن تتم كل الاتصالات بقاعدة البيانات من خلال التطبيق التي صممت من أجله
- حيث يبدأ التطبيق بامتلاك البيانات ويكون مسؤولا على فرض الشروط والقيود
- لن يمر وقت حتى تصبح تلك البيانات ذات أهمية لتطبيقات أخرى
- وهذا يعني أن تلك التطبيقات لابد أن تأتي عبر واجهة التطبيق الرئيس بطريقة أو بأخرى بحيث يتم تنفيذ قواعد العمل مرة واحدة فقط
- أو،،، يستوجب نشر قواعد العمل والموافقة عليها من قبل جميع مستخدمي البيانات لتجنب التلف

- يعتمد SQL Server على T-SQL، وتعتمد Oracle على PL/SQL لواجهتها البرمجية لقواعد البيانات
- بالرغم من أن المفاهيم البرمجة متشابهة جدًا في معظم نظم قواعد البيانات العلائقية،،، إلا أن الشكل النحوي أو النمطي للتعليمات قد يكون مختلفًا من نظام لآخر

دوال ومؤثرات التحكم المضمّنة
Inline Control Flow Functions & Operators

MySQL have several Control Flow Functions listed as below:

1) CASE (Also in Oracle, SQLServer, and others)
2) IF
3) IFNULL (NVL() in Oracle, ISNULL() in SQLServer)
4) NULLIF (Also in Oracle, SQLServer, and others)

Control function returns value based on each row processed by the query executed.

دوال ومؤثرات التحكم المضمّنة
Inline Control Flow Functions & Operators

1) CASE

We can say this is just like the switch case in programming languages.

Sample Queries:

```
// Below will return zero
SELECT CASE 0 WHEN 0 THEN 'zero' WHEN 1 THEN 'one' ELSE 'no one' END;

// Below will return true
SELECT CASE WHEN 5>2 THEN 'true' ELSE 'false' END;
```

دوال ومؤثرات التحكم المضمّنة
Inline Control Flow Functions & Operators

2) IF

This function takes the three parameters as expressions, if expression one is true then it will return second parameter otherwise it will return third parameter.

Sample Queries:

```
SELECT IF(expr1,expr2,expr3);

// return yes
SELECT IF(1<5,'yes','no');
```

دوال ومؤثرات التحكم المضمّنة
Inline Control Flow Functions & Operators

3) IFNULL

This function takes two parameters as expressions. And if expression one is not null then it will return expression one otherwise it will return expression two.

Sample Queries:

```
SELECT IFNULL(expr1, expr2);

// Return 5
SELECT IFNULL(5,0);

// Return 10
SELECT IFNULL(NULL,10);
```

دوال ومؤثرات التحكم المضمّنة
Inline Control Flow Functions & Operators

4) NULLIF

This function takes two parameters as expressions. It will return `NULL` if expr1=expr2 return `TRUE` otherwise it will return expr1.

Sample Queries:

```
SELECT NULLIF(expr1,expr2);

// Return NULL
SELECT NULLIF(5,5);

// Return 10
SELECT NULLIF(10,4);
```

الإجراءات المخزنة Stored Procedures

- **ما هو الإجراء المخزن؟**

- الستورد بروسيجر في MySQL هو عبارة عن مجموعة من التعليمات البرمجية المكتوبة بلغة SQL تكون مخزنة في نظام قاعدة البيانات وتهدف لإنجاز عمل معين أو الحصول على نتيجة معينة و حيث بالإمكان استدعاؤها (تنفيذها) من خلال تطبيقات خارجية أو من خلال برامج مخزنة أخرى.

- فهو يمثل طريقة لتخزين وتنفيذ منطق العمل Business Logic مباشرة من داخل نظام قاعدة البيانات.

- Recursive Stored Procedure (الستورد بروسيجر الراجع أو الذاتي) هو الذي يستدعي نفسه.

مميزات الإجراءات المخزنة Advantages of SPs

- Independency يمنح استقلالية لمنطق العمل عن لغة البرمجة أو بيئة التشغيل.
- Security يمنح مستوى جديد من الأمن من خلال تقديم بيانات الجداول عن طريقه.
- Performance يزيد من مستوى الأداء للتطبيقات.
- التقليل من حركة النقل بين التطبيق ونظام قاعدة البيانات.
- Reusability الرفع من مستوى إعادة الاستخدام باعتبار إمكانية الاستدعاء من كيانات أكبر منه.
- Maintainability سهولة التصحيح والتطوير باعتبار المركزية وعدم الحاجة لبيئة تطوير IDE أو مكتبات Libraries

عيوب الإجراءات المخزنة Disadvantages of SPs

- Performance ضخامة منطق العمل المخزّن وكثرة الاستدعاءات له تأثيرات سلبية على مستوى أداء نظام قاعدة البيانات باعتباره غير مصمم أساسا لهذا العمل.

- بساطة تركيبة الإجراء المخزن وتراكيب البيانات المتاحة يجعل من الصعب برمجة منطق عمل معقد.

- إدارة الإجراءات المخزنة ليس بالأمر السهل لافتقارها لأي نموذج تركيبي كنموذج البرمجة الشيئية OOP على سبيل المثال.

- الإفتقار لأدوات التدقيق والتتبع Debuggers فبالتالي يصعب كشف مواقع الأخطاء وتصحيحها.

MySQL Stored Procedure Syntax

We are going to develop a simple stored procedure named GetUsers() to help you get familiar with the syntax.

The GetUsers() stored procedure selects all users from the 'Users' table.

The *CREATE PROCEDURE* statement is used to create procedure in MySQL

Launch the mySQL client tool and type the following commands:

```
DELIMITER//
        CREATE  PROCEDURE  GetUsers()
        BEGIN
        SELECT  *  FROM  User;
        END
        DELIMITER ;
```

- We use the *CREATE PROCEDURE* statement to create a new stored procedure.we specify the name of stored procedure after the *CREATE PROCEDURE* statement.
- The code - section between *BEGIN* and *END* is called the body of the stored procedure in mysql.

Calling Stored Procedures

In order to call a stored procedure, you use the following SQL command:

```
CALL    STORED_PROCEDURE_NAME
```

You use the *CALL* statement to call a stored procedures e.g. to call the GetUsers stored procedure, use the following statement:

```
CALL    GetUsers();
```

Delete A Stored Procedures

If you want to delete a procedure you use the DROP PROCEDURE statement

Syntax:

```
DROP  PROCEDURE    IF EXISTS    procedure_name
```

MySQL Stored Procedure Variables

A variable is a named data object whose value can change during the stored procedure execution.

You must declare a variable before you can use it.

You use the *DECLARE* statement to declare a variable inside a stored procedure.

```
DECLARE    variable_name   datatype(size)    DEFAULT  default_name ;
```

We declared two INT variables x and y, and set their default values to zero

You use the SET statement to assign a variable another value

```
DECLARE    x,  y      INT    DEFAULT   0
SET x = 10 ;
```

The value of the x variable is 10 after the assignment.

Parameters

The parameters make the stored procedure more flexible and useful.In MySQL, a parameter has one of three modes *IN*, *OUT* or *INOUT*.

- *IN* : is the default mode.
- *OUT*: the value of an *OUT* parameter can be changed inside the stored procedure and its new value is passed back to the calling program.
- *INOUT* : an *INOUT* parameter is the combination of *IN* parameter and *OUT* parameter. It means that the calling program may pass the argument, and the stored procedure can modify the *INOUT* parameter and pass the new value back to the calling program.

How you can define parameters within a stored procedure

CREATE PROCEDURE proc () : Parameter list is empty

CREATE PROCEDURE proc (*IN* varname DATA-TYPE) :one input parameter.IN is optional because parameters are IN by default

CREATE PROCEDURE proc (*OUT* varname DATA-TYPE) : one output parameter.

CREATE PROCEDURE proc (*INOUT*varname DATA-TYPE):one parameter which is both input and outpu.

IN Example

```
DELIMITER//
        CREATE PROCEDURE   proc_in (IN    var1    INT)
        BEGIN
              SELECT   var1+2   AS  result;
        END//
```

OUT Example

```
DELIMITER//
        CREATE PROCEDURE   proc_out (OUT   name   VARCHAR(100))
        BEGIN
              SET   name = 'sandeep kumar nehra';
        END//
```

INOUT Example

```
DELIMITER//
        CREATE PROCEDURE   proc_inout (INOUT   var1   INT(4), IN   var2  INT(4))
        BEGIN
              SET    var1 = var1+var2;
        END//
```

MySQL If Statement

The MySQL IF statement allows you to execute a set of SQL statement based on a certain condition or value of an expression. An expression can return three value FALSE TRUE or NULL.

Mysql if statement syntax :

```
IF if_expression THEN commands
   [ELSEIF elseif_expression THEN commands]
   [ELSE commands]
END IF;
```

If the *if_expression* evaluates to TRUE the commands in the IF branch will execute.If it evaluates to FALSE, MySQL will check the *elseif_expression* and execute the commands in ELSEIF branch if the *elseif_expression* evaluates to TRUE.

MySQL If Statement Example :-

```
DELIMITER $$
        CREATE   PROCEDURE  GetLevel(
        IN      u_customerNumber  int(10),
        OUT    u_customerLevel     varchar(10))
    BEGIN
        DECLARE   creditlim  double;
                SELECT    creditlimit  INTO  creditlim
                FROM      customers
                WHERE     customerNumber = u_customerNumber;
            IF  creditlim  > 50000   THEN
               SET  u_customerLevel  = 'PLATINUM' ;
            ELSEIF  (creditlim <= 50000  AND  creditlim >= 10000)  THEN
               SET  u_customerLevel  = 'GOLD' ;
            ELSEIF  creditlim < 10000  THEN
               SET  u_customerLevel  = 'SILVER' ;
            END IF;
            END$$
```

MySQL CASE statement

The MySQL CASE statement makes the code more readable and efficient.
The *CASE* statements in mysql can be divided into two form simple and searched.

Mysql CASE statement syntax

```
CASE    case_expression
        WHEN   when_expression   THEN   commands
        .....
        ELSE  commands
END  CASE;
```

You use the simple *CASE* statement to check the value of an expression against a set of unique values.

```
CASE
        WHEN  condition_statement    THEN  commands
        WHEN  condition_statement    THEN  commands
        .....
        ELSE  commands
END  CASE;
```

```
DELIMITER$$
    CREATE  PROCEDURE GetShipping(
            IN      u_customerNumber   int(10),
            OUT   u_shipping             varchar(100))
    BEGIN
            DECLARE  customerCountry   varchar(30);
            SELECT   country  INTO  customerCountry
            FROM     customers
            WHERE    customerNumber  = u_customerNumber;
             CASE    customerCountry
                    WHEN   'UAE'    THEN
                           SET     u_shipping = '3-day shipping';
                    WHEN   'USA'    THEN
                           SET     u_shipping = '4-day shipping';
                    ELSE
                           SET     u_shipping = '6-day shipping';
     END CASE;
END$$
```

MySQL WHILE Loop

MySQL provides loop statement that allow you to execute a block of SQL code repeatedly based on a condition.There are three loop statements in MySQL :*WHILE*, *REPEAT* and *LOOP*.

The *WHILE* loop is called as pre-test loop.

Syntax of *WHILE* loop:

```
WHILE       expression     DO
            statements
END  WHILE
```

Mysql - *WHILE* loops checks the expression at the beginning of each iteration.If the expression evaluates to *TRUE*,MySQL will executes statements between *WHILE* and *END WHILE* until the expression evaluates to *FALSE*.

MySQL REPEAT Loop

The *REPEAT* loop statement is known as post-test loop.

Syntax of REPEAT loop :

```
REPEAT      Statements;
UNTIL     expression
END   REPEAT
```

In this loop the all statement executes first,and then it evaluates the *expression*.If the *expression* evaluates to TRUE, MySQL executes the ***statements*** repeatedly until the ***expression*** evaluates to FALSE.

while loop

```
DELIMITER $$
CREATE  PROCEDURE   WhileLoop()
      BEGIN
              DECLARE   x      INT;
              DECLARE   str    VARCHAR(100);
              SET x = 1;
              SET str = '' ;
  WHILE   x <= 10   DO
                    SET  str  =  CONCAT(str,x,',');
                    SET  x    =  x + 1;
  END WHILE;
  SELECT  str;
  END$$
```

repeat loop

```
DELIMITER $$
CREATE  PROCEDURE   WhileLoop()
     BEGIN
            DECLARE   x      INT;
            DECLARE   str    VARCHAR(100);
            SET x = 1;
            SET str = '' ;
  REPEAT
                  SET  str  =  CONCAT(str,x,',');
                  SET  x    =  x + 1;
  UNTIL  x >=10
  END REPEAT;
  SELECT  str;
  END$$
  DELIMITER;
```

MySQL Cursor

The mysql cursor is used to handle a result-set inside a stored procedure.A cursor is used to iterate through a set of rows returned by a query and process each row.

You can use MySQL cursors in stored procedures, stored functions and triggers.

MySQL is read only, non-scrollable and asensitive.

- **Read Only** : Cursor are not updatable.
- **Not Scrollable**: Cursors can be traversed only in one direction, forward, and you can't skip records from fetching.
- **Asensitive**: Once open, the cursor will not reflect changes in its source tables.In fact, MySQL does not guarantee the cursor will be updated, so you can't rely on it.

The *DECLARE* statement is used to define the cursor

Syntax:

```
DECLARE    cursor_name      CURSOR  FOR    SELECT_statement;
```

A cursor in mysql must always be associated with a *SELECT* statement.

The cursor declaration must be after any variable declaration

If you declare a cursor before variables declaration statement, MySQL will generate an error.

Next you open the cursor by using the *OPEN* statement.The open statement initializes the result-set for the cursor.

Syntax:

```
OPEN    cursor_name ;
```

The *FETCH* statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result-set.

Mysql cursor syntax:

```
FETCH    cursor_name    INTO   variables list;
```

You use the *CLOSE* statement to deactivate the cursor and release the memory associated with it.

Syntax:

```
CLOSE    cursor_name ;
```

Note : If you use MySQL cursor, you must also declare a NOT FOUND handler to handle the situation when the cursor could not find any row.Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set.When the cursor reaches the end of the result-set, it will not be able to get the data, and a condition is raised.The handler is used to handle this condition.

Syntax:

```
DECLARE  CONTINUE  HANDLER    FOR  NOT  FOUND     SET    finished=1 ;
```

MySQL cursor

```
DELIMITER//
    CREATE PROCEDURE 'cursor_work' (OUT param INT)
BEGIN
    DECLARE a, b, c INT;
    DECLARE cur1 CURSOR FOR SELECT col1 FROM table1;
    DECLARE  CONTINUE HANDLER FOR NOT FOUND SET b = 10 ;
            OPEN cur1 ;
            SET b = 0 ;
            SET c = 0 ;
            WHILE b = 0    DO
                    FETCH cur1 INTO a ;
                    IF   b = 0  THEN
                        SET c = c+a;
                    END IF;
            END WHILE;
            CLOSE cur1;
            SET param = c;
END//
```

MySQL Trigger

Definition :-A trigger or database trigger is a stored program that is executed automatically to respond to a specific event associated with table e.g. insert, update or delete.It is powerful tool for protecting the integrity of the data in you MySQL databases.It is useful to operations such as auditing and logging.

Note :- MySQL triggers are activated by SQL statement only.

A SQL trigger is a set of SQL statements stored in the database catalog.trigger is a special type of stored procedure this is called automatically when a data modification event is made against a table.

advantages of using SQL triggers

- SQL triggers provide an alternative way to check the integrity of data
- SQL triggers can catch errors in business logic in the database layer
- SQL triggers are very useful to audit the changes of data in tables

disadvantages of using SQL triggers

- SQL triggers may increase the overhead of the database server
- SQL triggers only can provide an extended validation and they ca't replace all the validations

In MySQL, a trigger is a set of SQL statements that is invoked automatically when a change is made to the data on the associated table.

Trigger can be invoked either before or after the data is changed by INSERT, UPDATE or DELETE statements.

The MySQL allows you to define maximum triggers for each table.

- *BEFORE INSERT* - activated before data is inserted into the table
- *BEFORE UPDATE* - activated before data in the table is updated
- *BEFORE DELETE* - activated before data is removed from the table
- *AFTER INSERT* - activated after data is inserted into the table
- *AFTER UPDATE* - activated after data in the table is updated
- *AFTER DELETE* - activated after data is removed from the table

Note :- when you use a statement that makes changes to the table but does not use *DELETE*, *INSERT* or *UPDATE* statement, the trigger is not invoked.

mySQL trigger limitations

- MySQL triggers can't call a stored procedure or stored function
- MySQL triggers can't use dynamic SQL statements
- MySQL triggers can't use *SHOW, LOAD DATA, LOAD TABLE, BACKUP DATABASE, RESTORE, RETURN* statements

To create MySQL trigger you can use the CREATE TRIGGER statement.

```
CREATE TRIGGER trigger_name trigger_time trigger_event
        ON table_name
        FOR EACH ROW
        BEGIN
                                        trigger_body
        END
```

Note : MySQL triggers are activated by SQL statements only.

The SQL statements are placed between *BEGIN* and *END* block

The OLD and NEW keywords are very handy.the *NEW* keyword refers to the new row after you change the data and The OLD keyword refers to the existing record before you change the data.

Triggers for a table are also dropped if you drop the table.

You cannot associate a trigger with a *TEMPORARY* table or view.

Triggers names exist in the schema namespace, meaning that all triggers must have unique names within a schema.Triggers in different schemas can have the same name.

You cannot have two triggers for a table that have the same activation time and activation event

MySQL Trigger Example

The AFTER UPDATE trigger is almost identical:

```
DELIMITER $$
CREATE TRIGGER 'blog_after_insert' AFTER INSERT
ON 'blog'
FOR EACH ROW BEGIN
    IF NEW.deleted THEN
                SET @changetype = 'DELETE' ;
            ELSE
                SET @changetype = 'NEW';
            END IF;
 INSERT INTO audit (blog_id,changetype) VALUES (NEW.id, @changetype);
    END$$
    DELIMITER;
```

MySQL DROP Trigger

To remove a trigger, you can use DROP TRIGGER statement

```
DROP TRIGGER table_name.trigger_name;
```

Validating MySQL data entry with triggers:

```
signal sqlstate '45000' set message_text = 'My Error Message';
```

State 45000 is a generic state representing "unhandled user-defined exception".

الدوال المعرّفة Functions in MySQL

- الدوال التي نقوم بتعريفها تعمل كما لو كانت دوال تتبع نظام إدارة قواعد البيانات
- لا تختلف عن نظيراتها في لغات البرمجة فهي لها معرّف وتستقبل بارامترات وتقوم بإرجاع قيمة واحدة فقط
- الشكل النحوي لها:

```
CREATE FUNCTION <function_name> (<arg1> <type1>, [<arg2> <type2>
[,…]) RETURNS <return type> [[NOT] DETERMINISTIC]
BEGIN
    ...
END
```

  - الحتمية DETERMINISTIC تعني أن المخرجات مرتبطة بشكل صارم كنتيجة لقيم البارامترات.
  - **انتبه:** الاعتماد على قيم غير ساكنة كالتاريخ الحالي أو القيمة العشوائية تجعل من الدالة غير حتمية NOT DETERMINISTIC حتى ولو ظهرت غير ذلك

- تستخدم تعليمة RETURN لإنهاء تنفيذ الدالة وتمرير النتيجة خارجًا
- **لا** يسري على الدوال ما يسري على الإجراءات فيما يخص وضع البارامتر IN, OUT, INOUT

CREATE PROCEDURE & CREATE FUNCTION

```
CREATE
    [DEFINER = user]
    PROCEDURE [IF NOT EXISTS] sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = user]
    FUNCTION [IF NOT EXISTS] sp_name ([func_parameter[,...]])
    RETURNS type
    [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
```

MySQL Scheduled Events الأحداث المجدولة

```
CREATE
    [DEFINER = user]
    EVENT
    [IF NOT EXISTS]
    event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    DO event_body;

schedule: {
    AT timestamp [+ INTERVAL interval] ...
  | EVERY interval
    [STARTS timestamp [+ INTERVAL interval] ...]
    [ENDS timestamp [+ INTERVAL interval] ...]
}

interval:
    quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
              WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
              DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

MySQL Scheduled Events الأحداث المجدولة

- يمكننا تخصيص إعدادات مختلفة لـ***schedule*** مثل:
- التشغيل لمرة واحدة عند تاريخ ووقت معين:

```
AT 'YYYY-MM-DD HH:mm:ss'
AT '2023-12-01 00:00:00'
```

- التشغيل لمرة واحدة بعد انقضاء فترة محددة:

```
AT CURRENT_TIMESTAMP + INTERVAL n [HOUR|MONTH|WEEK|DAY|MINUTE]
AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
```

- تشغيل على فترات زمنية محددة إلى الأبد:

```
EVERY n [HOUR|MONTH|WEEK|DAY|MINUTE]
EVERY 1 DAY
```

- تشغيل على فترات زمنية محددة خلال فترة محددة:

```
EVERY n [HOUR|MONTH|WEEK|DAY|MINUTE] STARTS date ENDS date
EVERY 1 DAY STARTS CURRENT_TIMESTAMP + INTERVAL 1 WEEK ENDS
'2024-01-01 00:00:00'
```

MySQL Scheduled Events الأحداث المجدولة

- يتم حذف الحدث المجدول تلقائيًا بمجرد انتهاء صلاحية جدولته (ON COMPLETION NOT PRESERVE)
  - لمنع ذلك السلوك والمحافظة على الحدث المجدول نستخدم ON COMPLETION PRESERVE
- **مثال**: أرشفة الطلبات المسحوبة:

```
DELIMITER $$
CREATE EVENT deleteWithdrawnApps
ON SCHEDULE EVERY 1 WEEK STARTS '2023-05-01 00:00:00' DO
BEGIN
  INSERT INTO archive
     SELECT * FROM applicant WHERE withdrawn=1;
  DELETE FROM applicant WHERE withdrawn=1;
END $$
DELIMITER ;
```

لتعطيل الحدث يمكننا فعل التالي:

```
ALTER EVENT deleteWithdrawnApps DISABLE;
```

Dynamic SQL

- قد نحتاج لإدارة بيانات بجدول أو أعمدة قد لا تكون ساكنة
- MySQL تقدم لنا مجموعة أدوات لتكوين استعلام ديناميكي وترجمته أو تجميعه وتنفيذه بالإضافة لتحرير الموارد التي يستغلها
- `SET, PREPARE, EXECUTE [USING], and {DEALLOCATE|DROP} PREPARE`
- مثال:

```
CREATE PROCEDURE `dynamic`(in tableName varchar(40))
BEGIN
   SET @statement = concat('SELECT count(*) FROM ', tableName, ' INTO @count');
   PREPARE stmt FROM @statement;
   EXECUTE stmt;
   SELECT concat('Count was: ', @count, ' FROM table: ', tableName);
   DEALLOCATE PREPARE stmt;
end
```