MySQL
Stored-Procedures & Triggers

Presented by
Victor Matos
Cleveland State University

Exporting a MySQL Database

mysqldump -uroot -padmin --databases company > company-dump.sql
Importing a MySQL Database

```
mysql -u root -padmin < company-dump.sql
```

Company Database
Some MySQL Data Types

**Text Data Types**
- **CHAR()** fixed from 0 to 255 characters long
- **VARCHAR()** variable from 0 to 255 characters long
- **TEXT** maximum length of 65535 chars
- **LONGTEXT** maximum length of 4294967295 characters

**Numeric Data Types**
- **INT()** -2147483648 to 2147483647 SIGNED or 0 to 4294967295 UNSIGNED
- **DOUBLE(),** large number with a floating decimal point
- **DECIMAL(),** DOUBLE stored as a string, allowing for a fixed decimal point

**Date Data Types**
- **DATE** YYYY-MM-DD
- **DATETIME** YYYY-MM-DD HH:MM:SS
- **TIMESTAMP** YYYYMMDDHHMMSS
- **TIME** HH:MM:SS

---

Defining & Setting Variables

```sql
DECLARE var1, var2 INT DEFAULT 0;
DECLARE str1 VARCHAR(50);
DECLARE today DATE DEFAULT CURRENT_DATE;
DECLARE v1, v2, v3 DOUBLE(10,2);

set var1 = 101;
set str1 = 'Hello world';
set v1 = 19.99;
```
MySQL Server-Side Programming

Assignment – Simple Statements

```
set var1 = 123.99;
set str1 = ‘Hello world’;
```

Blocks – Multiple Statements

```
BEGIN
  statement(s);
  . . .
END
```

Conditional Operator

```
if condition then
  statement1;
else
  statement2;
End if;
```
**MySQL Server-Side Programming**

**While Loop**
declare var1 int default 1;
myloop1:
while (var1 <= 10) do
    select var1;
    if var1 > 3 then
        leave myloop1;
    end if;
    set var1 = var1 + 1;
end while;
select 'adios';

**MySQL Server-Side Programming**

**Repeat Loop**
declare var1 int default 1;
myloop1: repeat
    select var1;
    if var1 > 3 then
        leave myloop1;
    end if;
    set var1 = var1 + 1;
until var1 > 5 end repeat;
select 'adios';
MySQL Server-Side Programming

Loop

```sql
declare var1 int default 1;
myloop1: loop
  select var1;
  if var1 > 3 then
    leave myloop1;
  end if;
  set var1 = var1 + 1;
end loop myloop1;
select 'adios';
```

Flowchart:
- **Statement1**
- **Condition**
  - **false**
  - **true**
- **Statement2**

MySQL Server-Side Programming

Functions
Named blocks of code that accept zero or more parameters, do some work, and return a single result.

Procedures
Void functions that use **IN, OUT, and INOUT** parameters to pass data in and out of the method.

Triggers
Methods associated to tables and their maintenance operations. Typically used to enforce business rules.
Defining Stored Functions

```sql
DELIMITER $$
CREATE FUNCTION `function-name` ( parameter TYPE )
RETURNS output-type
BEGIN
    statements;
    return some_value;
END $$
```

**INVOKING A FUNCTION**

```sql
set @result = functionName ( argumentValue );
select @result;
```

---

**Stored Functions**

```sql
-- Function returns the full-name of given employee
DROP FUNCTION IF EXISTS getFullName;
DELIMITER $$
CREATE FUNCTION `companyxyz`.`getFullName` ( empSSN INT)
RETURNS VARCHAR(100)
BEGIN
    declare fullName varchar(100);
    select concat(fname, ' ', lname) into fullName
    from employee where ssn = empSSN;
    if ROW_COUNT() = 0 then
        return 'n.a.';
    else
        return fullName;
    end if;
END
```
Stored Functions

Click here to execute the script

![Code snippet for creating and testing a stored function](image)

---

** TESTING THE STORED-FUNCTION **

use companyXYZ;

set @result = getfullName(123456789); select @result;

set @result = getfullName(123); select @result;

select getfullname(ssn) from employee;
### Stored Functions

**MySQL Comparison Functions & Operators**

- Between
- COALESCE()
- NULL Safe equal to operator (<>)
- Equal operator(=)
- Greater than or equal operator(>=)
- Greater than operator(>)
- GREATEST()
- IN()
- INTERVAL()
- IS NOT NULL
- IS NOT
- IS NULL
- IS

- IS NULL()
- LEAST()
- LESS THAN OR EQUAL OPERATOR(<=)
- LESS THAN OPERATOR(<)
- LIKE
- NOT BETWEEN AND
- NOT EQUAL OPERATOR(<>),!=)
- NOT IN()
- NOT LIKE
- STRCMP()


### Stored Functions

**MySQL Logical Operators**

- **And operator**
- **Not operator**
- **Or operator**
- **Xor operator**

**MySQL Flow Functions**

- Case operator
- IF()
- IFNULL()
- NULLIF()

### Stored Functions

#### Some MySQL String Functions

<table>
<thead>
<tr>
<th>ASCII</th>
<th>INSTR</th>
<th>ORD</th>
<th>STRCMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIN</td>
<td>LCASE</td>
<td>POSITION</td>
<td>SUBSTR</td>
</tr>
<tr>
<td>BIT_LENGTH</td>
<td>LEFT</td>
<td>QUOTE</td>
<td>SUBSTRING_INDEX</td>
</tr>
<tr>
<td>CHAR_LENGTH</td>
<td>LENGTH</td>
<td>REGEXP</td>
<td>SUBSTRING</td>
</tr>
<tr>
<td>CHAR</td>
<td>LIKE</td>
<td>REPEAT</td>
<td>TRIM</td>
</tr>
<tr>
<td>CHARACTER_LENGTH</td>
<td>LOAD_FILE</td>
<td>REPLACE</td>
<td>UCASE</td>
</tr>
<tr>
<td>CONCAT_WS</td>
<td>LOCATE</td>
<td>REVERSE</td>
<td>UNHEX</td>
</tr>
<tr>
<td>CONCAT</td>
<td>LOWER</td>
<td>RIGHT</td>
<td>UPPER</td>
</tr>
<tr>
<td>ELT</td>
<td>LPAD</td>
<td>RLIKE</td>
<td></td>
</tr>
<tr>
<td>EXPORT_SET</td>
<td>LTRIM</td>
<td>NOT RLIKE</td>
<td></td>
</tr>
<tr>
<td>FIELD</td>
<td>MAKE_SET</td>
<td>RPAD</td>
<td></td>
</tr>
<tr>
<td>FIND_IN_SET</td>
<td>MID</td>
<td>RTRIM</td>
<td></td>
</tr>
<tr>
<td>FORMAT</td>
<td>NOT LIKE</td>
<td>SOUNDEX</td>
<td></td>
</tr>
<tr>
<td>HEX</td>
<td>NOT REGEXP</td>
<td>SOUNDS_LIKE</td>
<td></td>
</tr>
<tr>
<td>INSERT</td>
<td>OCTET_LENGTH</td>
<td>SPACE</td>
<td></td>
</tr>
</tbody>
</table>


### Stored Functions

#### Some MySQL Math Functions

<table>
<thead>
<tr>
<th>ABS()</th>
<th>DIVISION</th>
<th>RAND()</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS()</td>
<td>EXP()</td>
<td>ROUND()</td>
</tr>
<tr>
<td>ASIN()</td>
<td>FLOOR()</td>
<td>SIGN()</td>
</tr>
<tr>
<td>ATAN2()</td>
<td>LN()</td>
<td>SIN()</td>
</tr>
<tr>
<td>ATAN()</td>
<td>LOG()</td>
<td>SQRT()</td>
</tr>
<tr>
<td>CEIL()</td>
<td>LOG2()</td>
<td>TAN()</td>
</tr>
<tr>
<td>CEILING()</td>
<td>LOG10()</td>
<td>TRUNCATE()</td>
</tr>
<tr>
<td>CONV()</td>
<td>MOD()</td>
<td>FORMAT()</td>
</tr>
<tr>
<td>COS()</td>
<td>OCT()</td>
<td></td>
</tr>
<tr>
<td>COT()</td>
<td>PI()</td>
<td></td>
</tr>
<tr>
<td>CRC32()</td>
<td>POW()</td>
<td></td>
</tr>
<tr>
<td>DEGREES()</td>
<td>POWER()</td>
<td></td>
</tr>
<tr>
<td>DIV</td>
<td>RADIANS()</td>
<td></td>
</tr>
</tbody>
</table>

Stored Functions

**Some Date/Time Functions**

<table>
<thead>
<tr>
<th>Name</th>
<th>DAYNAME()</th>
<th>DAYOFMONTH()</th>
<th>MONTH()</th>
<th>TIMESTAMPDIFF()</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDDATE()</td>
<td>DAYOFWEEK()</td>
<td>DAYOFYEAR()</td>
<td>MONTNAME()</td>
<td>TO_DAYS()</td>
</tr>
<tr>
<td>ADDTIME()</td>
<td>EXTRACT()</td>
<td>FROM_DAYS()</td>
<td>PERIOD_ADD()</td>
<td>TO_SECONDS()</td>
</tr>
<tr>
<td>CONVERT_TZ()</td>
<td>FROM_UNIXTIME()</td>
<td>GET_FORMAT()</td>
<td>PERIOD_DIFF()</td>
<td>UNIX_TIMESTAMP()</td>
</tr>
<tr>
<td>CURDATE()</td>
<td>HOUR()</td>
<td>LOCALTIME()</td>
<td>QUARTER()</td>
<td>UTC_DATE()</td>
</tr>
<tr>
<td>CURRENT_DATE()</td>
<td>LAST_DAY</td>
<td>LOCALTIMESTAMP()</td>
<td>SEC_TO_TIME()</td>
<td>UTC_TIMESTAMP()</td>
</tr>
<tr>
<td>CURRENT_TIME()</td>
<td>SECOND()</td>
<td>MAKEDATE()</td>
<td>SUBDATE()</td>
<td>WEEK()</td>
</tr>
<tr>
<td>CURRENT_TIMESTAMP()</td>
<td>STR_TO_DATE()</td>
<td>MAKETIME()</td>
<td>SUBTIME()</td>
<td>WEEKDAY()</td>
</tr>
<tr>
<td>CURTIME()</td>
<td>STR_TO_DATE()</td>
<td>MICROSECOND()</td>
<td>TIMESTAMP()</td>
<td>WEEKEND()</td>
</tr>
<tr>
<td>DATE_ADD()</td>
<td>TIME_FORMAT()</td>
<td>LOCALTIME()</td>
<td>TIMESTAMPADD()</td>
<td>YEAR()</td>
</tr>
<tr>
<td>DATE_FORMAT()</td>
<td>LOCALTIMESTAMP()</td>
<td>MAKETIME()</td>
<td>TIME()</td>
<td>YEARWEEK()</td>
</tr>
<tr>
<td>DATE_SUB()</td>
<td>MAKEDATE()</td>
<td>MICROSECOND()</td>
<td>TIMEDIFF()</td>
<td></td>
</tr>
<tr>
<td>DATE()</td>
<td>MAKETIME</td>
<td>MINUTE()</td>
<td>TIMESTAMP()</td>
<td></td>
</tr>
<tr>
<td>DATEDIFF()</td>
<td>MINUTE()</td>
<td>TIME()</td>
<td>TIMESTAMPADD()</td>
<td></td>
</tr>
</tbody>
</table>

**MySQL Comparison Functions & Operators**

<table>
<thead>
<tr>
<th>Between</th>
<th>IS NULL()</th>
<th>LEAST()</th>
</tr>
</thead>
<tbody>
<tr>
<td>COALESCE()</td>
<td>LESS THAN OR EQUAL OPERATOR(&lt;=)</td>
<td>LESS THAN OPERATOR(&lt;)</td>
</tr>
<tr>
<td>NULL</td>
<td>Safe equal to operator (&lt;&gt;='')</td>
<td>NOT BETWEEN AND</td>
</tr>
<tr>
<td>Safe equal to operator (&lt;&gt;='')</td>
<td>NOT EQUAL OPERATOR&lt;&gt;(&gt;)</td>
<td>NOT IN()</td>
</tr>
<tr>
<td>Equal operator(=)</td>
<td>NOT EQUAL OPERATOR&lt;&gt;(&gt;,!=)</td>
<td>NOT LIKE</td>
</tr>
<tr>
<td>Greater than or equal operator(&gt;=)</td>
<td>NOT IN()</td>
<td>NOT LIKE</td>
</tr>
<tr>
<td>Greater than operator(&gt;)</td>
<td>NOT IN()</td>
<td>NOT LIKE</td>
</tr>
<tr>
<td>GREATEST()</td>
<td>NOT IN()</td>
<td>NOT LIKE</td>
</tr>
<tr>
<td>IN()</td>
<td>NOT IN()</td>
<td>NOT LIKE</td>
</tr>
<tr>
<td>INTERVAL()</td>
<td>NOT IN()</td>
<td>NOT LIKE</td>
</tr>
<tr>
<td>IS NOT NULL</td>
<td>NOT IN()</td>
<td>NOT LIKE</td>
</tr>
<tr>
<td>IS NOT</td>
<td>NOT IN()</td>
<td>NOT LIKE</td>
</tr>
<tr>
<td>IS NULL</td>
<td>NOT IN()</td>
<td>NOT LIKE</td>
</tr>
</tbody>
</table>


Stored Functions

Some Compression & Encryption Functions

- AES_DECRYPT()
- AES_ENCRYPT()
- COMPRESS()
- DECODE()
- DES_DECRYPT()
- DES_ENCRYPT()
- ENCODE()
- ENCRYPT()
- CRC8()
- MD5()
- OLD_PASSWORD()
- PASSWORD()
- SHA1()
- UNCOMPRESS()
- UNCOMPRESSED_LENGTH()

Some Bit Functions

- BIT_COUNT
- BITWISE AND
- INVERT BITS
- BITWISE OR
- BITWISE XOR
- Left shift
- Right shift

Reference: http://www.w3resource.com/mysql/mysql-functions-and-operators.php

Stored Procedures

Creating a Procedure

```
DELIMITER $$

CREATE PROCEDURE `procedure_name` ( [[IN | OUT | INOUT] parameter1 TYPE] )
BEGIN
  statements;
END $$
```

Calling a Procedure

```
call myproc1();

call getEmpAddress ( 123456789, @empAddress );
```
**GOAL:** simple PROCEDURE - Greeting earthlings!!

```sql
drop procedure if exists p1;
DELIMITER $$
CREATE PROCEDURE `p1`() 
BEGIN
    select 'Hello world';
END $$
```

**GOAL:** Invoke procedure - Pass IN parameters to procedure

```sql
drop procedure if exists p2;
DELIMITER $$
CREATE PROCEDURE `p2`(IN var1 varchar(100)) 
BEGIN
    select var1;
    select concat('hello ', var1, ' world');
END $$
```
**Stored Procedures**

---

**GOAL: Defining & Using variables**

DELIMITER $$
CREATE PROCEDURE `p3`() 
BEGIN 
    DECLARE var1, var2 INT DEFAULT 0; 
    DECLARE str1 VARCHAR(50); 
    DECLARE today DATE DEFAULT CURRENT_DATE; 
    DECLARE v1, v2, v3 DOUBLE(10,2); 
    set var1 = 101; 
    set str1 = 'Hello world'; 
    set v1 = 19.99; 
    set @sessionVar1 = 'Hello world'; 
    set @sessionVar1 = 123; 
    select concat('Value of v1 is ', v1, ', Today is: ', today, ' session variable: ', @sessionVar1); 
END$$
---

**GOAL: IF-THEN-ELSE  compare male/female count**

DELIMITER $$
CREATE PROCEDURE `companyxyz`.`p4`() 
BEGIN 
    declare ladiesCounter int; 
    declare gentCounter int; 
    declare difference int; 
    select count(*) INTO ladiesCounter from employee where sex = 'F'; 
    select count(*) INTO gentCounter from employee where sex = 'M'; 
    if ( ladiesCounter > gentCounter ) then 
      select concat('Ladies rule! ', ladiesCounter, ' to ', gentCounter); 
    else 
      begin 
        set difference = gentCounter - ladiesCounter; 
        select concat(difference, ' more gentlemen than ladies'); 
      end; 
    end if; 
END$$
---
Stored Procedures

-- GOAL: passing INPUT parameters and using them in SQL statements

DELIMITER $$
CREATE PROCEDURE `companyxyz`.`p5` (IN empSSN varchar(10))
BEGIN
    declare sumHours double(10,2);
    declare countProjects int;
    set sumHours = 0;
    set countProjects = 0;
    select sum(hours) into sumHours from works_on where essn = empSSN;
    select count(*) into countProjects from works_on where essn = empSSN;
    select concat('Hours ', sumHours, ', Total prj: ', countProjects);
END

DELIMITER $$/

STORED PROCEDURES

-- GOAL: using OUTPUT parameters in a Procedure

DELIMITER $$
CREATE DEFINER=`csuperson`@`localhost`
PROCEDURE `p6` (IN empSSN varchar(10), OUT fullName varchar(100))
BEGIN
    select concat(fname, ' ', lname) into fullName
    from employee
    where ssn = empSSN;
END
DELIMITER $$
Stored Procedures

DELIMITER $$
CREATE PROCEDURE getSupervisorAndSpouse (IN empSsn int,
                                           OUT supervisor varchar(100),
                                           OUT spouse varchar(100))
BEGIN
  set supervisor = "none";
  set spouse = "none";

  select concat(s.fname, ' ', s.lname) into supervisor
  from employee e, employee s
  where (e.superssn = s.ssn) and (e.ssn = empssn);

  if ( ROW_COUNT() = 0 ) then
    LEAVE procBody;
  end if;

  select d.dependent_name into spouse
  from dependent d
  where d.relationship = 'Spouse'
  and d.essn = empssn;
END $$

DELIMITER $$
CREATE PROCEDURE `companyxyz`.`p7` (IN max INT)
BEGIN
  declare var1 int;
  declare buffer varchar(100);

  set var1 = 0;
  set buffer = ""
  WHILE var1 < max DO
    set buffer = concat(buffer, '\n var1=', var1, '");
    set var1 = var1 + 1;
  END WHILE;

  select buffer;
END $$
MySQL Cursors

• A cursor is a type of pointer built into MySQL to provide sequential access (*one row at a time*) to tables produced by SQL statements.

• **Why cursors?**
  • Remember, SQL query results are *always* presented to the user in the form of a table.
  • Cursors facilitate the navigation of those answer tables allowing the *fetching* of individual rows.
MySQL Implicit Cursors

- Depending on the cursor’s life cycle and management code there are two types:
  
  - **Implicit** cursors (controlled by MySQL on execution of DML-maintenance or select ... into statements)
  
  - **Explicit** cursors (created with a select statement)

```
-- IMPLICIT CURSORS and the ROW_COUNT() function
DELIMITER $$
CREATE PROCEDURE `companyxyz`.`p9` ()
BEGIN
  update employee set salary = salary + 1 where ssn = 0;
  set @recordsAffected = ROW_COUNT();
  select concat('1.Records affected: ', @recordsAffected);
  if @recordsAffected = 0 then
    select ('No EMPLOYEE record was updated');
  end if;
  update employee set salary = salary - 1 where ssn > 0;
  select concat('2.Records affected: ', Row_Count());
END $$
```
MySQL Explicit Cursors

• Explicit cursors require the use of OPEN, FETCH, and CLOSE statements.

• Explicit cursors are defined in function of SQL select-statements. For example:
  DECLARE mycursor CURSOR FOR select fname from employee;

• The FETCH command moves to the next available records and transfer its data to the corresponding receiving variables.

You may check the cursor’s end-of-data condition through the following listener:

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_data = true;

  when not found is triggered the handler changes (SET) the control variable

---

-- GOAL: using EXPLICIT CURSORS

DELIMITER $$
CREATE PROCEDURE `p8`() BEGIN
  DECLARE empName varchar(100);
  DECLARE empSalary DOUBLE(10,2);
  DECLARE no_more_data BOOLEAN DEFAULT false;

  DECLARE cursor1 CURSOR FOR select fname, salary from employee;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_data = true;

  DROP TEMPORARY TABLE IF EXISTS `myLog`;
  CREATE TEMPORARY TABLE `myLog` (`myline` TEXT);

  OPEN cursor1;
  myLoop: WHILE (no_more_data = false) DO
    FETCH cursor1 INTO empName, empSalary;
    IF (no_more_data ) THEN
      CLOSE cursor1;
      LEAVE myloop;
      END IF;
      insert into `myLog` values( concat("n", empName, ",", empSalary ));
    END WHILE;
  select * from `mylog`;
END $$
Consider the following SQL statement

```sql
select e.ssn, e.lname,
     (select group_concat(d.dependent_name)
         from Dependent d
         where d.essn = e.ssn
     ) as 'Dependents'
from Employee e;
```

This convenient operator could be used instead of a nested-loop solution with an outer-cursor producing employee records and an inner-cursor concatenating the names of corresponding family members.

---

### Transaction Processing

**START TRANSACTION, COMMIT, and ROLLBACK Syntax**

```sql
START TRANSACTION [WITH CONSISTENT SNAPSHOT]
BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

These statements provide control over use of transactions:

- **START TRANSACTION** or **BEGIN** start a new transaction
- **COMMIT** commits the current transaction, making its changes permanent
- **ROLLBACK** rolls back the current transaction, canceling its changes
- **SET autocommit** disables/enables the default autocommit mode for the current session
Transaction Processing

Example

```sql
start transaction;
delete from employee where ssn > 0;
select * from employee;
rollback;
select * from employee where ssn > 0;
```

MySQL Triggers

- Triggers are useful in enforcing database policies and business rules.
- A trigger is a server-side procedure-like object that is associated with a table, and is activated when a particular event occurs for the table.
- Triggers are linked to maintenance of database tables and are invoked or fired on DELETE, UPDATE, INSERT operations.
- Triggers CANNOT include the COMMIT / ROLLBACK statements.
- In general, triggers appear to execute quietly without the user even knowing of their existence.
**MySQL Triggers. Example**

```sql
Insert into works_on(Essn,Pno, Hours) values (333445555, 22, 45);

CREATE TRIGGER upto4projects
BEFORE INSERT
ON company.works_on
FOR EACH ROW
BEGIN
...
END;
```

**Business Rule:**
Do not allow employees to work on more than 4 projects.

**WORKS_ON table**

<table>
<thead>
<tr>
<th>ESSN</th>
<th>PNO</th>
<th>HOURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>1</td>
<td>32.5</td>
</tr>
<tr>
<td>123456789</td>
<td>2</td>
<td>7.5</td>
</tr>
<tr>
<td>666884444</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>453453453</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>453453453</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>333445555</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>333445555</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>333445555</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>333445555</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>999887777</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>999887777</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>987897987</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>987897987</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td>987654321</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>987654321</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>888665555</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>

See complete solution later

---

**MySQL Triggers**

**Uses of triggers**

- Provide sophisticated auditing / logging
- Prevent invalid transactions
- Enforce referential integrity
- Enforce data integrity
- Enforce complex business rules
- Enforce complex security authorizations
MySQL Triggers

Consulting Database Dictionary

SHOW TRIGGERS [{FROM | IN} db_name]
[LIKE 'pattern' | WHERE expr]

SHOW TRIGGERS lists the triggers currently defined for tables in a database (the default database unless a FROM clause is given).

Example:

show triggers from companyXYZ like 'works_on';

Abbreviated Trigger Syntax

CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
[ BEFORE | AFTER ]
trigger_event ON tbl_name
FOR EACH ROW trigger_body
BEGIN
   Body-of-the-trigger;
END;

INSERT
DELETE
UPDATE
MySQL Triggers

**EXAMPLE1.** An individual job assignment should not exceed 40 hours. Correct assignments exceeding this limit by resetting them to 40.

```sql
use companyxyz;
drop trigger if exists max40hours;
delimiter $$
CREATE DEFINER=`csuperson`@`localhost`
TRIGGER `companyxyz`.`max40hours`
BEFORE INSERT ON companyxyz.works_on
FOR EACH ROW
BEGIN
    if ( NEW.hours > 40 ) then
        set NEW.hours = 40;
    end if;
END;
$$
```

MySQL Triggers

**Row-Level Triggers**

A **row trigger** is fired each time a record in the associated table is affected by the triggering statement.

MySQL only implements row-level triggers
(no statement-level as in Oracle)

For example, if an UPDATE statement modifies multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement.

- If a triggering statement affects no rows, a row trigger is not run.
- Row triggers are useful if the code in the trigger action depends on rows that are affected (use NEW. OLD. prefixes).
MySQL Triggers

BEFORE Triggers
BEFORE triggers run the trigger action before the triggering statement is run. This type of trigger is commonly used in the following situations:

- When the trigger action determines whether the triggering statement should be allowed to complete. Using a BEFORE trigger for this purpose, you can eliminate unnecessary processing of the triggering statement and its eventual rollback in cases where an exception is raised in the trigger action.

- To derive specific column values before completing a triggering INSERT or UPDATE statement.

AFTER Triggers
AFTER triggers run the trigger action after the triggering statement is run.

-- GOAL: Understanding ROW-LEVEL Triggers (Part-1)
-- ---------------------------------------------------------------------------
use companyxyz;
drop trigger if exists spyTrigger;
delimiter $$
CREATE TRIGGER `companyxyz`.`spyTrigger` BEFORE UPDATE ON companyxyz.works_on FOR EACH ROW BEGIN
  insert into mylog values (concat(`Old`.essn, ' ', `Old`.pno, ' ', `Old`.Hours),
                           concat(`New`.essn, ' ', `New`.pno, ' ', `New`.Hours));
END;
$$
MySQL Triggers

---- GOAL: Understanding ROW-LEVEL Triggers (Part 2) ----

drop table if exists myLog;
create table myLog (oldValues text, newValues text);
update works_on set hours = hours - 1;
select * from mylog;

MySQL Triggers

Throwing Exceptions – Sending SIGNALS

SIGNAL is the MySQL way to “throw” an error.

SIGNAL provides error information to a handler, to an outer portion of the application, or to the client.

The generic SQLSTATE value, uses '45000', which means “unhandled user-defined exception.”

Example:

SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'Problem - blah... ';

26
MySQL Triggers

Exception Handlers – Catching SIGNALs

- A HANDLER specifies a code-fragment that deals with one or more conditions (roughly equivalent to a Java catch statement).
- If one of these conditions occurs, the specified statement executes.
- The statement can be a simple assignment such as `SET var_name = value`, or a compound statement written using `BEGIN` and `END`.

Example:

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '45000' BEGIN
  insert into mylog values ('Insert operation failed', now());
END;
```

Example2: Employee’s max number of assignments is 4. Reject >4!

```
use companyxyz;
drop trigger if exists upto4projects;
delimiter $$
CREATE DEFINER=`csuperson`@`localhost` TRIGGER `companyxyz`.`upto4projects` BEFORE INSERT ON companyxyz.works_on FOR EACH ROW
BEGIN
  declare counter integer;
  declare customMessage varchar(100);
  select count(*) into counter from works_on where essn = NEW.essn;
  if ( counter > 3 ) then
    set customMessage = concat('ERROR Excessive work load SSN=', NEW.essn, ' current project-count: ', counter);
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = customMessage ;
end if;
END;
$$
```
MySQL Triggers

**Example 2:** Employee's max number of assignments is 4. Reject >4!

Testing the trigger!

Before inserting:

```
insert into works_on values (333445555, 10, 25);  
```

Trigger is fired, business rule is enforced, and new record is NOT inserted!

```
189 14:02:16 insert into works_on values (333445555, 25, 10)
Error Code: 1644 ERROR Exceeds work load SSN=333445555 current project count: 4
```

---

MySQL Triggers

**Example 2:** Employee’s max number of assignments is 4. Reject >4!

Again - Testing the trigger!

Before inserting:

```
insert into works_on values (333445555, 10, 40);
```

---

DELIMITER $$

CREATE PROCEDURE `companyxyz`.p10()

BEGIN

DECLARE CONTINUE HANDLER FOR SQLSTATE '45000' BEGIN 

insert into mylog values('Insert operation failed', NOW()); 
END;

insert into mylog values('before Insert', NOW());

insert into works_on values (333445555, 10, 40);

insert into mylog values('after Insert ok', NOW());

END
```
EXAMPLE 3

The following trigger is fired when a new dependent record is inserted.

- If the record corresponds to a new born child (< 1 year) several gifts are ordered for the baby.

- If the new entry is not related to an existing employee the insertion is rejected (referential integrity).

- Assume there is a logging table MYLOG(c1,c2)

MySQL Triggers

use companyxyz;
drop trigger if exists humanResources;
delimiter $$
CREATE TRIGGER `companyxyz`.`humanResources`
BEFORE INSERT ON companyxyz.dependent FOR EACH ROW
BEGIN
    declare mName varchar(20);
    declare months int;
    select Lname into mName from employee where ssn = NEW.Essn;
    set months = TIMESTAMPDIFF(MONTH, NEW.Bdate, now() );
    if (months < 12 ) then
       insert into Mylog values ( concat(now(), ' Personnel Dept--->' ),
                                concat('Send NEW_BABY greeting card to ', mName ) );
       insert into Mylog values ( concat(now(), ' Marketing Dept.--->' ),
                                concat('Mail $100 company stock to ', mName ) );
       insert into mylog values ( concat(now(), ' Purchasing Dpt.--->' ),
                                concat('Order one-year diapers for ', mName ) );
    end if;
END;
$$
MySQL Triggers

Testing humanResource Trigger

-- This insertion is valid and will be accepted
-- all the gifts will be ordered for the new baby (change bdate!).

```
insert into dependent values
(123456789, 'Jose', 'M',
 date_add(NOW(), INTERVAL 7 DAY),
 'Son' );
```

-- This insertion is valid and will be accepted
-- but not a baby.

```
insert into dependent values
(123456789, 'Peter', 'M', '2001-7-04', 'Son' );
```

MySQL Triggers

Your turn → How to enforce REFERENTIAL INTEGRITY?

Try a ROW LEVEL trigger fixing the CASCADE/NULLIFY/MANDATORY retention modes of EMPLOYEE and DEPENDENT.

Extend the results for dealing with job assignments and other meaningful relationships in the database.
MySQL Triggers

Enforcing Referential-Integrity: DEPENDENT.ESSN → EMPLOYEE.SSN

use companyxyz;
drop trigger if exists referential;
delimiter $$
CREATE DEFINER=`csuperson`@`localhost`
TRIGGER `companyxyz`.referential
BEFORE INSERT ON companyxyz.dependent FOR EACH ROW
BEGIN
  declare empCounter INT;
  declare customMessage varchar(100);
  -- is there a PARENT employee with the given Soc. Sec. Numb. ?
  select count(*) into empCounter from employee where ssn = NEW.essn;
  if empCounter = 0 then
    set customMessage = concat( 'ERROR - Referential Integrity violation. ESSN=’,
                                 NEW.essn, ', does not exist in EMPLOYEE table' );
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = customMessage ;
  end if;
END
$$

Your turn ...

--- RULE1:
Only CSUPERSON is allowed to change the EMPLOYEE table.
Changes in salary cannot be larger than 15%.

--- RULE2:
Changes to WORKS_ON are accepted only from Mon-Fri between working hours 8:00 AM. and 6:00 PM. Reject otherwise.
MySQL Triggers

--- BUSINESS RULE 1: [1 of 2]

Only CSUPERSON is allowed to change the salary field of the EMPLOYEE table. Changes in salary cannot be larger than 15%.

use companyxyz;
drop trigger if exists onlyCsupersonChangesSalary;

DELIMITER $$
CREATE TRIGGER `companyxyz`.`onlyCsupersonChangesSalary` BEFORE UPDATE ON companyxyz.employee FOR EACH ROW BEGIN
    declare userName varchar(50);
declare msg varchar(100);
    -- userName holds a values such as 'csuperson@localhost'
    set userName = substring_index(user(), '@', 1);

    -- only CSUPERSON is allowed to operate on EMPLOYEE
    if userName <> 'csuperson' then
        set msg = concat(username, ' not authorized! worked on ', OLD.ssn);
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
    end if;

    -- salary cannot jump by more than 15%
    if (NEW.salary > 1.15 * OLD.salary) then
        set msg = concat('Invalid raise >15% for: ', OLD.ssn, ' OLD salary ', OLD.salary, ', NEW salary ', NEW.salary);
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
    end if;
END

"MySQL Triggers"

--- BUSINESS RULE 1: [2 of 2]

Only CSUPERSON is allowed to change the salary field of the EMPLOYEE table. Changes in salary cannot be larger than 15%.

--- only CSUPERSON is allowed to operate on EMPLOYEE
if userName <> 'csuperson' then
    set msg = concat(username, ' not authorized! worked on ', OLD.ssn);
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
end if;

-- salary cannot jump by more than 15%
if (NEW.salary > 1.15 * OLD.salary) then
    set msg = concat('Invalid raise >15% for: ', OLD.ssn, ' OLD salary ', OLD.salary, ', NEW salary ', NEW.salary);
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
end if;
MySQL Triggers

-- BUSINESS RULE1:
Only CSUPERSON is allowed to change the salary field of the EMPLOYEE table. Changes in salary cannot be larger than 15%.

TESTING RULE1
(login as user: root )
use companyxyz;
update employee set salary = 1.0*salary where ssn=123456789;
select @@error_count;
show errors;

TESTING RULE1
(login as user: csuperson)
use companyxyz;
update employee set salary = 1.25 * salary where ssn=123456789;
select @@error_count;
show errors;

MySQL Triggers

-- RULE2: 1 of 4
Changes to PROJECT table are accepted only from Mon-Fri between working hours 8:00 AM and 6:00 PM. Reject otherwise.

use companyxyz;
drop trigger if exists projectBusinessHoursUpdate;

DELIMITER $$
CREATE TRIGGER `companyxyz`.`projectBusinessHoursUpdate`
BEFORE UPDATE ON companyxyz.project FOR EACH ROW
BEGIN
  call checkBusinessHours();
END;
$$

Triggers are allowed to call procedures and functions (with/without parameters)
MySQL Triggers

-- RULE2:
2 of 4
Changes to PROJECT table are accepted only from Mon-Fri between working hours 8:00 AM. and 6:00 PM. Reject otherwise.

use companyxyz;
drop trigger if exists projectBusinessHoursUpdate;

DELIMITER $$
CREATE TRIGGER `companyxyz`.`projectBusinessHoursUpdate`
BEFORE INSERT ON companyxyz.project FOR EACH ROW
BEGIN
    call checkBusinessHours();
END;
$$

MySQL Triggers

-- RULE2:
3 of 4
Changes to PROJECT table are accepted only from Mon-Fri between working hours 8:00 AM. and 6:00 PM. Reject otherwise.

use companyxyz;
drop trigger if exists projectBusinessHoursUpdate;

DELIMITER $$
CREATE TRIGGER `companyxyz`.`projectBusinessHoursUpdate`
BEFORE DELETE ON companyxyz.project FOR EACH ROW
BEGIN
    call checkBusinessHours();
END;
$$
MySQL Triggers

-- RULE2: Changes to PROJECT table are accepted only from Mon-Fri between working hours 8:00 AM. and 6:00 PM. Reject otherwise.

drop procedure if exists checkBusinessHours;
delimiter $$;
CREATE PROCEDURE `companyxyz`.`checkBusinessHours` ()
BEGIN
    declare dd int;
declare hh int;
    -- day: 1 = Sunday, 2 = Monday, ..., 7 = Saturday
    set dd = dayofweek(now());
    -- hours 0-24 military
    set hh = hour(now());
    -- exclude SATURDAY and SUNDAY as well as hour not in 9a-5p
    if (dd = 1 or dd = 7) or ( hh < 9 or hh > 17 ) then
        signal sqlstate '45000' SET MESSAGE_TEXT =
            'Changes to PROJECT must be applied on business hours Mo-Fr 9a-5p';
    end if;
END;
$$

MySQL Triggers

-- TESTING RULE2: Changes to PROJECT table are accepted only from Mon-Fri between working hours 8:00 AM. and 6:00 PM. Reject otherwise.

update project set pname = pname;
-- RULE3: (Bonus Problem)
A new employee cannot make more money than his/her manager. If this is the case, throw an exception.

use companyxyz;
drop trigger if exists salaryNotHigherThanManagersSalary;

DELIMITER $$
CREATE TRIGGER `companyxyz`.`salaryNotHigherThanManagersSalary`
BEFORE UPDATE ON companyxyz.Employee FOR EACH ROW
BEGIN
    declare mgrSalary double;
    declare mgrSsn int;
    declare msg varchar(100);
    select ssn, salary into mgrSsn, mgrSalary from employee where ssn = NEW.ssn;
    if ( mgrSalary < NEW.salary ) then
        set msg = concat ( NEW.ssn, ' makes ', NEW.salary, ' which is higher than manager-salary ', mgrSalary );
        SIGNAL sqlstate '45000' SET MESSAGE_TEXT = msg;
    end if;
END;
$$
MySQL Triggers

Your Turn

Enforce the following business rule:

At all times a department MUST have a manager. In case the current manager is deleted, assign his/her immediate supervisor to act as the interim manager.

If no interim could be found, refuse the request to delete the employee.
MySQL – PHP

- PHP is an Open-Source, general-purpose interpreted, server-side scripting language designed for web development. It has limited Object-Oriented capabilities (single-inheritance, single-constructor, ...)

- PHP code is embedded into HTML pages using the tags
  
  ```html
  <HTML>
  ...
  <?php
  PHP_code_goes_here ...
  ?>
  ...
  </HTML>
  ```

- PHP syntax is similar to C, C#, Java.

- PHP is particularly favored in applications accessing relational databases.

---

Example1.php

```html
<html>
<head>
  <title>Example 1</title>
</head>
<body>
  <?php
    $msg = "<H1>Hello World</H1>"
    echo $msg . "<p>Bye</p>";
  ?>
</body>
</html>
```

Hello World

Bye
Example2.php  FOR-LOOP, IF-ELSE

```php
<html>
<head>
<title>Example 2</title>
</head>
<body>
<?php
for ($i=0; $i < 7; $i++) {
    echo "<b>" . $i . "</b> ";
    if ($i % 2 == 0) {
        echo "EVEN NUMBER <br>";
    } else {
        echo "ODD NUMBER " . $i . " <br>";
    }
}?
</body>
</html>
```

Example3A.php  ARRAYS

```php
<html>
<head>
<title>Example 3A</title>
</head>
<body>
<H1>Days of the Week</H1>
<?php
$days = array("Sun","Mon","Tue","Wed","Thu","Fri","Sat");
$dayslength = count($days);

for($i=0; $i<$dayslength; $i++)
{
    echo "<br>" . $i . " - " . $days[$i];
}
?>
</body>
</html>
```
MySQL – PHP

Example3b.php  ARRAYS / foreach

```php
<html>
<head>
<title>Example 3B</title>
</head>
<body>
  <h1>Days of the Week</h1>
  <?php
  $days = array("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");
  $i = 0;
  foreach ($days as $aDay )
  {
    echo "<br>" . $i++ . " - " . $aDay;
  }
  ?>
</body>
</html>
```

MySQL – PHP

Example4.php  ASSOCIATIVE ARRAYS

```php
<html>
<head>
<title>Example 4</title>
</head>
<body>
  <h1>Calories</h1>
  <?php
  //associative array
  $calories = array( "Coca-Cola" => "142",
                    "BigMac" => "550",
                    "Large Apple" => "110");
  echo "<p> Coca-Cola " . $calories["Coca-Cola"];
  echo "<p> Large Apple " . $calories["Large Apple"];
  echo "<p> BigMac " . $calories["BigMac"];
  ?>
</body>
</html>
```
**Example5.php**  
**TABLES**

```php
<html>
<head>
<title>Example 5</title>
</head>
<body>
    <h1>Tables – Calories – Associative Arrays</h1>
    $calories = array("Coca-Cola"=>"142", "BigMac"=>"550", "Large Apple"=>"110");
    echo "<table border='1'>";
    echo "<tr>
    <th>Coca-Cola</th>
    <th>".".$calories["Coca-Cola"]"></th>
    </tr>
    echo "<tr>
    <th>BigMac</th>
    <th>".".$calories["BigMac"]"></th>
    </tr>
    echo "<tr>
    <th>Large Apple</th>
    <th>".".$calories["Large Apple"]"></th>
    </tr>
    echo "</table>";
</body>
</html>
```

**Example7.php**  
**FUNCTIONS**

```php
<html>
<head>
<title>Example 7</title>
</head>
<body>
    <h1>Functions</h1>
    <?php
    function add($v1, $v2)
    {
        $result = $v1 + $v2;
        return $result;
    }
    $a = 11;
    $b = 22;
    echo "(".".$a."+".".$b." ") = " . add($a, $b);
    ?>
</body>
</html>
```
```php
<?php
class Person
{
    // class variables
    private $name, $phone;
    // constructor
    public function __construct($newName, $newPhone)
    {
        $this->name = $newName;
        $this->phone = $newPhone;
    }
    // accessors + user-defined methods
    public function setName($newval)
    {
        $this->name = $newval;
    }
    public function getName()
    {
        return $this->name;
    }
    public function setPhone($newval)
    {
        $this->phone = $newval;
    }
    public function getPhone()
    {
        return $this->phone;
    }
    public function showPerson()
    {
        return $this->name . ' ' . $this->phone;
    }
}
// Create two objects, set data, get values, show objects
echo "<br><H3>creating a person</H3>";
$p = new Person('Daenerys', '555-1234');
echo "<br> UserName: " . $p->getName();
echo "<br> Password: " . $p->getPhone();
$p->setPhone("555-7777");
echo "<br> UserName: " . $p->getName();
echo "<br> Password: " . $p->getPhone();
$p = new Person('Tyrion');
echo "<br> UserName: " . $p->getName();
echo "<br> Password: " . $p->getPhone();
?>
```

```
<html>
<head>
<title>Example 8A</title>
</head>
<body>

<H1> Forms (Posting)</H1>
<form action="example8b.php" method="post">
<br> UserName: <input type="text" name="userName">
<br> Password: <input type="password" name="pwd">
<br> <input type="submit">
</form>

</body>
</html>
```

When the user pushes the Submit button, each variable name and value included in the <form> is inserted in the associative array $_POST[...]

```
**MySQL – PHP**

Example8b.php  
**FORMS - POST/GET**

```html
<html>
<head>
  <title>Example 8B</title>
</head>
<body>
  <H1>Forms</H1>
  <br>
  Your user name is: <?php echo $_POST["userName"]; ?>
  <br>
  Your password is: <?php echo $_POST["pwd"]; ?>
</body>
</html>
```

$_POST[…] is an associative array holding all the <key,value> pairs included in the submitted portion <FORM>…</FORM>

---

Example8c.php  
**FORMS - REQUEST**

```html
<html>
<head>
  <title>Example 8C</title>
</head>
<body>
  <H1>Forms</H1>
  <br>
  Your user name is: <?php echo $_REQUEST["userName"]; ?>
  <br>
  Your password is: <?php echo $_REQUEST["pwd"]; ?>
</body>
</html>
```

$_REQUEST[…] is an alternative associative array holding all the <key,value> pairs included in $_POST, $_GET, and $_COOKIE.
<?php
    echo "<H1> Contents of \$_POST: </H1>";
    foreach ($POST as $k => $v) {
        echo "<br> \$k = \$v ";
    }
?>

You may access each <key,value> of the collections: \$_POST, \$_GET, \$_REQUEST, \$_SERVER and \$_COOKIE

print_r(…)
called 'print human readable' command.
A cookie is a small expiring file send by the server to the user's computer. Cookies are *embedded* into the requested page.

**Create a Cookie**

```php
setcookie("cookieName", "cookieValue", expirationTime);
```

**Get a Cookie**

```php
$var = $_COOKIE["cookieName"];
```

**Destroy a Cookie**

```php
setcookie("cookieName", "cookieValue", time() - 1);
```

or

```php
if (isset($_COOKIE['cookieName']))
    unset($_COOKIE['cookieName']);
```

```html
<html>
<head> 
<title>Example 8A</title>
</head>
<body>
<H1>Forms (Posting)</H1>
<!-- each cookie has: name, value, expiration time (sec) -->
<?php
    setcookie("Dragon1", "Viserion", time()+60);
    setcookie("Dragon2", "Rhaegal", time()+60);
    setcookie("Dragon3", "Drogon", time()+60);
    setcookie("SongOfFireAndIceBooks", "A Game of Thrones, A Clash of Kings, A Storm Swords, A Feast for Crows, A Dance with Dragons, The Winds of Winter, A Dream of Spring", time()+60);
    setcookie("CounterAccess", "107", time()+60);
?>
<form action="example8b.php" method="post">
    <br>
    UserName: <input type="text" name="userName">
    <br>
    Password: <input type="password" name="pwd">
    <br>
    <input type="submit">
</form>
</body>
</html>
```
**MySQL – PHP**

### Example8e.php  SHOW COOKIES

```html
<html>
<head>
  <title>Example 8B</title>
</head>
<body>
  <br>
  Your user name is: <?php echo $_REQUEST["userName"];?>
  <br>
  Your password is: <?php echo $_REQUEST["pwd"];?>
  <br>
  <?php
    $value = $_COOKIE["Dragon3"]; 
    echo "Dragon3= ". $value;
    echo "<br>Contents of \$_COOKIE: </H1>";
    foreach ($_COOKIE as $k => $v) {
      echo "<br> $k = $v ";
    }
  ?>
</body>
</html>
```

---

### Example9a.php  SESSION

The HTTP protocol is memory-less (stateless).

The $_SESSION array is used to create and store application variables. The array is destroyed when the session ends.

When the session starts a unique identifier is given to the user. This ID is associated to her $_SESSION array which can be accessed by other of her pages.
A normal HTML website will not pass data from one page to another. In other words, all information is forgotten when a new page is loaded. This makes it quite a problem for tasks like a shopping cart, which requires data (the user’s selected product) to be remembered from one page to the next.

**php sessions - overview**

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping cart items, etc). However, this session information is temporary and is usually deleted very quickly after the user has left the website that uses sessions. It is important to ponder if the sessions’ temporary storage is applicable to your website. If you require a more permanent storage you will need to find another solution, like a MySQL database. Sessions work by creating a unique identification (UID) number and storing variables based on this ID. This helps to prevent two users’ data from getting confused with one another when visiting the same webpage.

**Example9a.php**

```php
DATABASE CONNECTION
```

Making a connection with a database server can be done as follows:

```
mysqli_connect(host, username, password, dbname);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>Optional. Either a host name or an IP address</td>
</tr>
<tr>
<td>username</td>
<td>Optional. The MySQL user name</td>
</tr>
<tr>
<td>password</td>
<td>Optional. The password to log in with</td>
</tr>
<tr>
<td>dbname</td>
<td>Optional. The default database to be used when performing queries</td>
</tr>
</tbody>
</table>
SYNTACTICAL STYLES

PROCEDURAL STYLE
$link = mysqli_connect($host, $username, $password, $dbname);
...
mysqli_query($link, $sqlstatement);

OBJECT STYLE
$mysqli = @new mysqli($host, $username, $password, $dbname);
...
$mysqli->query($sqlstatement);

Warning
Do not use the deprecated mysql API, instead try the ‘improved version’ mysqli (or the PDO PHP-Data-Objects) API. The next examples use mysqli calls.

DATABASE CONNECTION

Connecting csuperson/euclid to companyXYZ running on localhost.

```php
<?php
// Create connection object. Args: host, username, password, dbname);
$link = mysqli_connect("137.148.1.66","csuperson","euclid","companyXYZ");

// Check connection
if (mysqli_connect_errno($link))
{
    echo "<br> Failed to connect to MySQL<br> " . mysqli_connect_error($link);
} else {
    echo "<br> Successfully connected to MySQL ";
    // do some work here ...
    // close connection ...
    mysqli_close($link);
}
?>
```
Example11.php  CREATE NEW DATABASE

```php
<?php
// Create connection-args: host, username, password;
$link = mysqli_connect("localhost","csuperson","euclid");

// Check connection
if (mysqli_connect_errno($link))
{
    echo "<br> Failed to connect to MySQL: " . mysqli_connect_error($link);
} else {
    echo "<br> Successfully connected to MySQL ";

    // create database now ...
    $sql = "CREATE DATABASE myTestDb; ";
    if (mysqli_query($link,$sql))
    {
        echo "<br> Database myTestDb created successfully";
    } else {
        echo "<br> Error creating myTestDb database: " . mysqli_error($link);
    }
    // close connection ...
    mysqli_close($link);
}
?>
```

Example12.php  CREATE TABLE

```php
<?php
// Create connection-args: host, username, password, dbname);
$link = mysqli_connect("localhost","csuperson","euclid","mytestdb");

// Check connection
if (mysqli_connect_errno($link))
{
    echo "<br> Failed to connect to MySQL: " . mysqli_connect_error($link);
    exit;
} else {
    echo "<br> Successfully connected to MySQL ";

    // create table now ...
    $sql = "CREATE TABLE myTempTable ( col1 INT primary key, col2 VARCHAR(100)); ";
    if (mysqli_query($link,$sql))
    {
        echo "<br> Success. myTempTable created"
    } else {
        echo "<br> Problems creating myTempTable " . mysqli_error($link);
    }
    // close connection ...
    mysqli_close($link);
?>
```
Example 13.php

```
<?php
// Create connection-args: host, username, password, dbname);
$link = mysqli_connect("localhost","csuperson","euclid","MyTestDb");

// Check connection
if (mysqli_connect_errno($link)) {
    echo "<br> Failed to connect to MySQL: " . mysqli_connect_error($link);
    exit;
} else {
    echo "<br> Successfully connected to MySQL ";
}

// create table now ...
$sql = "INSERT INTO myTempTable VALUES (111, 'AAA');";
// Execute query
if (mysqli_query($link, $sql)) {
    echo "<br> Record inserted: " . mysqli_affected_rows($link);
} else {
    echo "<br> Problems inserting record<br>" . mysqli_error($link);
}
// close connection ...
mysqli_close($link);
?>
```

Example 13B.php

```
<?php
// Create connection-args: host, username, password, dbname);
$link = mysqli_connect("localhost","csuperson","euclid","MyTestDB");

// Check connection
if (mysqli_connect_errno($link)) {
    echo "<br> Failed to connect to MySQL: " . mysqli_connect_error($link);
} else {
    echo "<br> Successfully connected to MySQL ";
}

// create table now ...
$sql = "INSERT INTO myTempTable VALUES (222, 'BBB'); " . "INSERT INTO myTempTable VALUES (333, 'CCC');";
// Execute query
if (mysqli_multi_query($link, $sql)) {
    echo "<br> Success. Record inserted: " . mysqli_affected_rows($link);
} else {
    echo "<br> Problems inserting record" . mysqli_error($link);
}
// close connection ...
mysqli_close($link);
?>
```
Example14.php  UPDATE RECORD

```php
<?php
// Create connection-args: host, username, password, dbname);
$link = mysqli_connect("localhost","csuperson","euclid","mytestdb");

// Check connection
if (mysqli_connect_errno($link)) {
    echo "<br> Failed to connect to MySQL: " . mysqli_connect_error($link);
} else {
    echo "<br> Sucessfully connected to MySQL ";
}

// create table now ...
$sql = "UPDATE myTempTable SET col1=col1+1000 where col2> 'AAA'; ";
// Execute query
if ( mysqli_query($link, $sql) ) {
    echo "<br> Success. Records updated " . mysqli_affected_rows($link);  
} else {
    echo "<br> Problems updating table " . mysqli_error($link);  
}

// close connection ...
mysqli_close($link);
?>
```

Example15.php  DELETE RECORD

```php
<?php
// Create connection-args: host, username, password, dbname);
$link = mysqli_connect("localhost","csuperson","euclid","mytestDb");

// Check connection
if (mysqli_connect_errno($link)) {
    echo "<br> Failed to connect to MySQL: " . mysqli_connect_error($link);
} else {
    echo "<br> Sucessfully connected to MySQL ";
}

// create table now ...
$sql = "DELETE FROM myTempTable WHERE col2='CCC'; ";
// Execute query
if ( mysqli_query($link, $sql) ) {
    echo "<br> Success. Records deleted " . mysqli_affected_rows($link);  
} else {
    echo "<br> Problems deleting record " . mysqli_error($link);  
}

// close connection ...
mysqli_close($link);
?>
```
<?php
$link = mysqli_connect("localhost","csuperson","euclidx");
if (!$link) { die('Could not connect: ' . mysqli_connect_error()); }
mysqli_select_db($link, 'mytestDb');
$resultSet = mysqli_query($link, "SELECT * FROM mytempTable; ");

while ($row = mysqli_fetch_array($resultSet)) {
    echo "<tr>",
    echo "<td>" . $row['col1'] . "</td>",
    echo "<td>" . $row['col2'] . "</td>",
    echo "</tr>";
}

mysqli_close($link);
$link->close();
?>

MySQL – PHP

Example16.php  FETCHING a RESULTSET

MySQL – PHP

Warning: Using plain SQL queries is dangerous!!!

SQL injection is a form of computer attack in which correctly formatted SQL queries are intercepted and modified by attaching extra pieces of code. When the altered query is executed it will produce results different from those intended in the original request.

EXAMPLE
"select * from table where id =" idValue "";

This statement is vulnerable. The last ‘ symbol could be changed to something like: ‘ or true to produce

"select * from table where id =" idValue " or true;"

This new expression retrieves (to the chagrin of the programmer) all records from table (instead of the one row she intended).

A mitigation strategy is to use prepared or parametized queries;
Prepared Statements
A prepared statement or a parameterized statement is used to execute the same statement repeatedly with high efficiency.

Basic workflow
The prepared statement execution consists of two stages: prepare and execute.
1. At the prepare stage a statement template is sent to the database server.
2. The server performs a syntax check and initializes server internal resources for later use.

The MySQL server supports using anonymous, positional placeholder with ?.

Taken from:

Example17.php

```php
<?php
$link = mysqli_connect("localhost", "csuperson", "euclid", "mytestdb");

// test connection
if (mysqli_connect_errno()) {
    echo "<h1>Connect failed</h1>" . mysqli_connect_error();
    exit();
}

// create a prepared statement
$stmt = mysqli_prepare($link,
    "SELECT col1, col2 FROM mytempTable where col1 >= ? and col2 >= ?;" );
if (!$stmt) {
    echo "<h1>Statement failed</h1>" . mysqli_error($link);
    exit;
}

// bind parameters to local variables-constants. The type specification
// 'is' indicates supplied args are: (i)nteger, (s)tring respectively
mysqli_stmt_bind_param($stmt, 'is', $arg1, $arg2);
$arg1 = 111;
$arg2 = 'AAA';
```
Example 17.php
PREPARED STATEMENT

```php
// execute query
mysqli_stmt_execute($stmt);
// bind sql result to local variable
mysqli_stmt_bind_result($stmt, $localVar1, $localVar2);
while (mysqli_stmt_fetch($stmt)) {
    echo "COL1: $localVar1 COL2: $localVar2 ";
}
// close prepared statement
mysqli_stmt_close($stmt);
mysqli_close($link);
```

MySQL – PHP
Example 18.php
CALLING STORED PROCEDURE

```php
<?php
// $link = mysqli_connect("localhost", "csuperson", "euclid", "mytestdb");
$link = mysqli_connect("mysql.informatos.org", "csuperson", "euclid", "informatos_mytestdb");
print_r($link);

// test connection
if (mysqli_connect_errno()) {
    echo "<h1>Connect failed</h1>" . mysqli_connect_error();
    exit();
}

// create a prepared statement
$query = "CALL getCol2Proc( ?, @arg2 )";
$stmt = mysqli_prepare($link, $query);
if (!$stmt) {
    echo "<h1>Statement failed</h1>" . mysqli_error($link);
    exit;
}

// bind parameter(s) to local variables-constants
// type spec 'i' indicates: (i)nteger [(s)tring, (d)ouble, ...]
mysqli_stmt_bind_param($stmt, 'i', $arg1);
$arg1 = 111;
```
Example18.php CALLING STORED PROCEDURE

Assume the following stored procedure has been already created in MySQL database. You supply the value of col1 and it returns the corresponding col2.

```sql
CREATE PROCEDURE getCol2Proc (IN col1Param INT,
                               OUT col2Value VARCHAR(100) )
BEGIN
    select count(*) into @counter from informatos_mytemptable
    where col1 = col1Param;
    set col2Value = 'n.a.';
    if (@counter = 1) then
        select col2 into @result from informatos_mytemptable
        where col1 = col1Param;
        set col2Value = @result;
    end if;
    select col2Value;
END
```
Assume the following stored function has been already created in MySQL database. The function works on myTempTable. It accepts a key value(col1) and returns its corresponding col2-value or 'n.a.' if the key is not found.

```sql
CREATE FUNCTION getCol2 (col1Param INT) RETURNS varchar(100)
BEGIN
    select count(*) into @counter from informatos_mytemptable where col1 = col1Param;

    if (@counter = 1) then
        select col2 into @col2Value from informatos_mytemptable where col1 = col1Param;
        return @col2Value;
    end if;
    return 'n.a.';
END
```

```php
<?php
$link = mysqli_connect("localhost", "csuperson", "euclid", "mytestdb");
// test connection
if (mysqli_connect_errno()) {
    echo "<h1>Connect failed</h1>" . mysqli_connect_error();
    exit();
}

// create a prepared statement to reach the stored function
$query = "select getCol2( ? ) " ;
$stmt = mysqli_prepare($link, $query);
if (!$stmt) {
    echo "<h1>Statement failed</h1>" . mysqli_error($link);
    exit;
}
// bind parameters to local variables-constants. type spec
// 'i' indicates the supplied argument is an (i)nteger
mysqli_stmt_bind_param($stmt, 'i', $arg1);
$arg1 = 111;
// execute query
mysqli_stmt_execute($stmt);
```
Example19.php  CALLING A STORED FUNCTION

```php
// bind SQL result to local variable
mysqli_stmt_bind_result($stmt, $localVar1);

while (mysqli_stmt_fetch($stmt)) {
    echo "<br> Result: $localVar1 ";
}

// close prepared statement
mysqli_stmt_close($stmt);

mysqli_close($link);
```

Example20a.php  COMPANY DATABASE

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=Cp1252">
<title>Company Page1</title>
</head>
<body>
<h1>Selecting Employees by Gender (M/F)</h1>
<form action="example20b.php" method=post>
    Enter your name here:
    <br><input type="text" name="yourname">
    Enter required GENDER value (M/F)
    <br><input type="text" name="gender">
    <p>
        <input type="submit" name="submit" value="Retrieve by Gender">
    </p>
</form>
</body>
</html>
```
```php
<?php

// Example20b.php COMPANY DATABASE

echo "<h1>Company ‐ Page 2</h1>"

echo "<br>
This came with $_POST: <br>
" . print_r($_POST); 

// $link = mysqli_connect("localhost", "csuperson", "euclid", "companyxyz");
$link = mysqli_connect("mysql.informatos.org", "csuperson", "euclid", "companyxyz");

// test connection
if (mysqli_connect_errno()) {
    echo "<h1>Connect failed</h1>" . mysqli_connect_error();
    exit();
}

// create a prepared statement
$stmt = mysqli_prepare($link, 'SELECT ssn, fname, lname, salary FROM employee where sex = ? ;');
if (!$stmt) {
    echo "<h1>Statement failed</h1>" . mysqli_error($link);
    exit;
}

// bind parameter to local variable
// type spec 'is' indicates: (i)nteger, (s)tring input types
mysqli_stmt_bind_param($stmt, 's', $arg1);
$arg1 = $_POST['gender'];

// execute query
mysqli_stmt_execute($stmt);

// bind sql result to local variables
mysqli_stmt_bind_result($stmt, $ssn, $fname, $lname, $salary);

echo "<table border='1'>
<tr style='background-color:#FFFF00'>
<th>SSN</th>
<th>First Name</th>
<th>Last Name</th>
<th>Salary</th>
</tr>

while (mysqli_stmt_fetch($stmt)) {
    echo "<tr>
    <td> $ssn </td>
    <td> $fname </td>
    <td> $lname </td>
    <td> $salary </td>
    </tr>
    
    " . $ssn . " <td> $fname <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $ssn . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    " . $fname . " <td>
    " . $lname . " <td>
    " . $salary . " <td>
    
    echo "</table>";
// close prepared statement
mysqli_stmt_close($stmt);
mysqli_close($link);
?>
```
APPENDIX A. Install MySQL DBMS & WorkBench

All tools at once
Use the link http://dev.mysql.com/downloads/ to reach the download-site for the Windows version of MySQL server and tools. Just download and execute the installer.

After installation is completed you should see the following components: Workbench, Server, Connectors...

APPENDIX B. Install PHP Development Environment

Step1. PHP Engine – Windows Platform
Download from this link: http://windows.php.net/download/ the most current version of PHP for Windows (pick the Non-Thread-Safe version).

Run the installer:
(a) set installation folder to: c:/PHP and
(b) choose IISFast-cgi server mode.
(c) For additional setup information: http://php.net/manual/en/install.windows.iis7.php
APPENDIX B. Install PHP Development Environment

Step 2. NETBEANS IDE
A recommended IDE for PHP programming is Netbeans available from: https://netbeans.org/downloads/.
Choose either the version “ALL” or just “PHP” development version.

After Netbeans is installed set a reference to the PHP interpreter. Click the toolbar Tool > Options > PHP. On the ‘PHPS Interpreter’ textbox enter: c:/PHP/php.exe

APPENDIX B. Install PHP Development Environment (cont)

Click on the Debugging tab. You should see the following entries. Press OK button to close it, you are done setting PHP & Netbeans.
APPENDIX B. Install PHP Development Environment

Step 3. Test the Platform

Run Netbeans, from the tool-bar follow the sequence:
File > New Project > PHP > PHP Application > Next.
A project called PhpProject1 is initiated. At this point you should see the following:

Click Next

APPENDIX B. Install PHP Development Environment

Step 3. Test the Platform

The next screen is the Run Configuration panel on which you indicate the paths to the web server and virtual directory on which the application will be run.
1. Check mark the option box ‘Copy files from Sources…’
2. In the ‘Copy to folder’ textbox enter the path to your IIS server directory holding the php files (in our example C:\inetpub\wwwroot\PhpProject1)
3. Click Finish.
APPENDIX B. Install PHP Development Environment

Step 3. Test the Platform

The layout of an HTML document nesting a PHP fragment is shown. Enter the statement: `phpinfo();` Click the Execute button (green Play button on tool bar)

On your browser you should see a screen similar to the one below.
APPENDIX B. Install PHP Development Environment

Step 4. OPTIONAL – Add Debugging Capabilities to NetBeans-PHP

The debugger is an optional 3rd-party component (a single .dll file) that you could download from: http://xdebug.org/download.php

1. Move the XDEBUG (.dll) file to the c:\PHP\ folder
2. Add to the c:\PHP\php.ini control file the following code fragment (append it at the end of the .ini file). Save the file.

```ini
[xdebug]
zend_extension="C:/PHP/php_xdebug-2.2.2-5.3-vc9-nts.dll"
xdebug.remote_enable=on
xdebug.remote_handler=dbgp
xdebug.remote_host=localhost
xdebug.remote_port=9000
xdebug.idekey="netbeans-xdebug"
```

3. Re-boot your machine for all these changes to take effect.

---

Step 5. OPTIONAL – Test Debugging Features

Click on a line number to set a BREAKPOINT. Press the RUN-DEBUGGER key to test your app. Observer the navigation controls added to the tool-bar.
Install Eclipse Eclipse IDE for Java EE Developers download it from: http://www.eclipse.org/downloads/

From the tool-bar do this: Help > Install New Software ...

In the ‘Work with:’ textbox enter ‘all available sources’. Enter ‘PHP’ in the textbox below.

Wait for responses to be displayed. Look for Php Development Tools (PDT). Next> Accept Licensing > Finish.

Create a new Php Project.

Using Windows-Explorer create a folder called ‘c:\inetpub\wwwroot\MyPhpSite’

In Eclipse create new PHP project MyProject1. Choose ‘Create project at existing location…’, in the Directory enter ‘c:\inetpub\wwwroot\MyPhpSite’. Click on Finish.


NOTE: If your browser doesn’t support frames, you need to remove all the <frameset> <frame> tags.
APPENDIX C. Eclipse & PHP

Running your code.
From tool-bar click on drop-down arrow for ‘Run Configurations’. In the Name textbox enter: MyPhpSite. Select the ‘Server’ tab. In the File textbox enter “/MyPhpSite/myPage1.php”. Apply > Close. Hit the Run button to test the app.

Activate Debugging
APPENDIX D. MySQL Error ....

Safe Operations:
13:02:06 delete from tableName...

Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To disable safe mode, toggle the option in Tool Bar > Edit > Preferences -> SQL Queries and reconnect. 0.000 sec

APPENDIX E. WAMP Server

WAMP SERVER
This Windows-based package contains Apache Server, PHP, and MySQL in a single installation. It is available from: http://www.wampserver.com/en/
### Example18.php  rowID

**Warning:** MySQL does not offer a rowid function similar to the one found in Oracle (and other DBMS). Here is a workaround for this problem (the following query retrieves the first n records – similar to `select..from..limit 0,n;`)

```sql
select col1, col2, @rowid:=@rowid+1 as rowid
from informatos_mytemptable, (select @rowid:=0) as dummyTable
where @rowid < n;
```