CL203 - Database Systems Lab

Lab#12 – SP & Triggers

Stored Procedures

A stored procedure is a named collection of procedural and SQL statements. Stored procedures are stored in the database. One of the major advantages of stored procedures is that they can be used to encapsulate and represent business transactions.

To create a stored procedure, you use the following syntax:

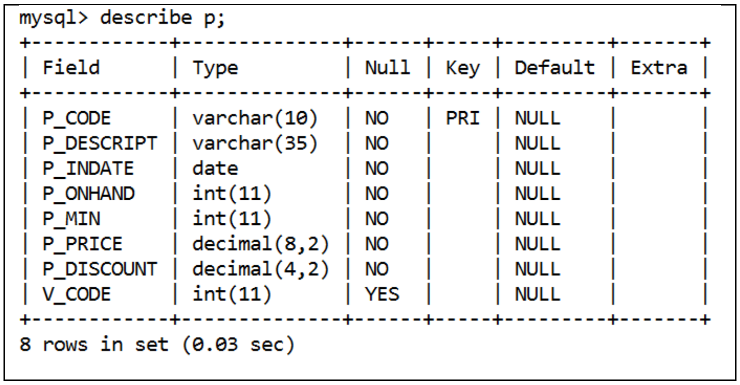
```
CREATE OR REPLACE PROCEDURE procedure_name [([IN/OUT] argument data-
type, … )]
   BEGIN
   SQL statements;
   ...
END;
```

Note the following important points about stored procedures and their syntax:

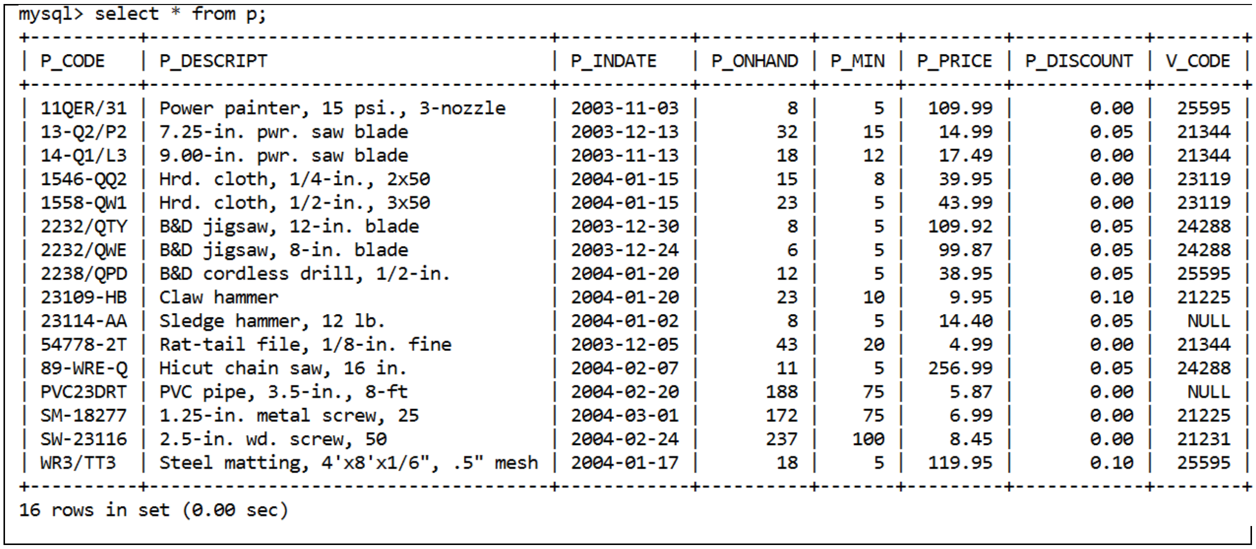
- Argument specifies the parameters that are passed to the stored procedure. A stored procedure could have zero or more arguments or parameters.
- IN/OUT indicates whether the parameter is for input, output, or both.
- data-type is one of the procedural SQL data types used in the RDBMS. The data types normally match those used in the RDBMS table creation statement.

Example 1:

Consider the following table *(script available on slate->resources/scripts/sales_co.sql)*:



```
mysql> describe p;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| P_CODE     | varchar(10)  | NO   | PRI | NULL    |       |
| P_DESCRIPT | varchar(35)  | NO   |     | NULL    |       |
| P_INDATE   | date         | NO   |     | NULL    |       |
| P_ONHAND   | int(11)      | NO   |     | NULL    |       |
| P_MIN      | int(11)      | NO   |     | NULL    |       |
| P_PRICE    | decimal(8,2) | NO   |     | NULL    |       |
| P_DISCOUNT | decimal(4,2) | NO   |     | NULL    |       |
| V_CODE     | int(11)      | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
8 rows in set (0.03 sec)
```



```
mysql> select * from p;
+----------+-------------------------------------+------------+----------+-------+---------+------------+--------+
| P_CODE   | P_DESCRIPT                          | P_INDATE   | P_ONHAND | P_MIN | P_PRICE | P_DISCOUNT | V_CODE |
+----------+-------------------------------------+------------+----------+-------+---------+------------+--------+
| 11QER/31 | Power painter, 15 psi., 3-nozzle    | 2003-11-03 |        8 |     5 |  109.99 |       0.00 |  25595 |
| 13-Q2/P2 | 7.25-in. pwr. saw blade             | 2003-12-13 |       32 |    15 |   14.99 |       0.05 |  21344 |
| 14-Q1/L3 | 9.00-in. pwr. saw blade             | 2003-11-13 |       18 |    12 |   17.49 |       0.00 |  21344 |
| 1546-QQ2 | Hrd. cloth, 1/4-in., 2x50           | 2004-01-15 |       15 |     8 |   39.95 |       0.00 |  23119 |
| 1558-QW1 | Hrd. cloth, 1/2-in., 3x50           | 2004-01-15 |       23 |     5 |   43.99 |       0.00 |  23119 |
| 2232/QTY | B&D jigsaw, 12-in. blade            | 2003-12-30 |        8 |     5 |  109.92 |       0.05 |  24288 |
| 2232/QWE | B&D jigsaw, 8-in. blade             | 2003-12-24 |        6 |     5 |   99.87 |       0.05 |  24288 |
| 2238/QPD | B&D cordless drill, 1/2-in.         | 2004-01-20 |       12 |     5 |   38.95 |       0.05 |  25595 |
| 23109-HB | Claw hammer                         | 2004-01-20 |       23 |    10 |    9.95 |       0.10 |  21225 |
| 23114-AA | Sledge hammer, 12 lb.               | 2004-01-02 |        8 |     5 |   14.40 |       0.05 |   NULL |
| 54778-2T | Rat-tail file, 1/8-in. fine         | 2003-12-05 |       43 |    20 |    4.99 |       0.00 |  21344 |
| 89-WRE-Q | Hicut chain saw, 16 in.             | 2004-02-07 |       11 |     5 |  256.99 |       0.05 |  24288 |
| PVC23DRT | PVC pipe, 3.5-in., 8-ft             | 2004-02-20 |      188 |    75 |    5.87 |       0.00 |   NULL |
| SM-18277 | 1.25-in. metal screw, 25            | 2004-03-01 |      172 |    75 |    6.99 |       0.00 |  21225 |
| SW-23116 | 2.5-in. wd. screw, 50               | 2004-02-24 |      237 |   100 |    8.45 |       0.00 |  21231 |
| WR3/TT3  | Steel matting, 4'x8'x1/6", .5" mesh | 2004-01-17 |       18 |     5 |  119.95 |       0.10 |  25595 |
+----------+-------------------------------------+------------+----------+-------+---------+------------+--------+
16 rows in set (0.00 sec)
```

To illustrate stored procedures, assume that you want to create a procedure (PRC_PROD_DISCOUNT) to assign an additional 5 percent discount for all products when the quantity on hand is more than or equal to twice the minimum quantity.

```
CREATE PROCEDURE PRG_PROD()

BEGIN

UPDATE P

SET P_DISCOUNT = P_DISCOUNT*0.05

WHERE P_ONHAND >= P_MIN*2;

END
```

To execute the stored procedure, you must use the following syntax:

call procedure_name[(parameter_list)];

In this case we will write *call prg_prod();* to execute the procedure.

Example 2:

Using the product table again:

```
CREATE PROCEDURE PRG_AVG_PRICE(out avg_price decimal)

BEGIN

SELECT AVG(P_PRICE) INTO avg_price FROM P;

END
```

In order to execute the procedure write:

call prg_avg_price(@out);

and then:

SELECT @out;

To print the output.

Triggers

A trigger is procedural SQL code that is automatically invoked by the RDBMS upon the occurrence of a given data manipulation event. It is useful to remember that:

- A trigger is invoked before or after a data row is inserted, updated, or deleted.
- A trigger is associated with a database table.
- Each database table may have one or more triggers.
- A trigger is executed as part of the transaction that triggered it.

In order to explain triggers let us create an example database for a blogging application. Two tables are required:

1. \`blog\`: stores a unique post ID, the title, content, and a deleted tag.
2. \`audit\`: stores a basic set of historical changes with a record ID, the blog post ID, the change type (NEW, EDIT or DELETE) and the date/time of that change.

The following SQL creates the \`blog\` and indexes the deleted column:

```
CREATE TABLE `blog` (
`id` mediumint(8) unsigned NOT NULL AUTO_INCREMENT,
`title` text,
`content` text,
`deleted` tinyint(1) unsigned NOT NULL DEFAULT '0',
PRIMARY KEY (`id`),
KEY `ix_deleted` (`deleted`)
)
```

The following SQL creates the `audit` table. All columns are indexed and a foreign key is defined for audit.blog_id which references blog.id. Therefore, when we physically DELETE a blog entry, it's full audit history is also removed.

```
CREATE TABLE `audit` (
`id` mediumint(8) unsigned NOT NULL AUTO_INCREMENT,
`blog_id` mediumint(8) unsigned NOT NULL,
`changetype` enum('NEW','EDIT','DELETE') NOT NULL,
`changetime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
PRIMARY KEY (`id`),
KEY `ix_blog_id` (`blog_id`),
KEY `ix_changetype` (`changetype`),
KEY `ix_changetime` (`changetime`),
CONSTRAINT `FK_audit_blog_id` FOREIGN KEY (`blog_id`) REFERENCES `blog`
(`id`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

When a record is INSERTed into the blog table, we want to add a new entry into the audit table containing the blog ID and a type of 'NEW' (or 'DELETE' if it was deleted immediately).

When a record is UPDATEd in the blog table, we want to add a new entry into the audit table containing the blog ID and a type of 'EDIT' or 'DELETE' if the deleted flag is set.

Note that the changetime field will automatically be set to the current time.

Each trigger requires:

1. A unique name. It is preferred to use a name which describes the table and action, e.g. blog_before_insert or blog_after_update.
2. The table which triggers the event. A single trigger can only monitor a single table.

3. When the trigger occurs. This can either be BEFORE or AFTER an INSERT, UPDATE or DELETE. A BEFORE trigger must be used if you need to modify incoming data. An AFTER trigger must be used if you want to reference the new/changed record as a foreign key for a record in another table.
4. The trigger body; a set of SQL commands to run. Note that you can refer to columns in the subject table using OLD.col_name (the previous value) or NEW.col_name (the new value). The value for NEW.col_name can be changed in BEFORE INSERT and UPDATE triggers.

The basic trigger syntax is:

```
CREATE
TRIGGER `event_name` BEFORE/AFTER INSERT/UPDATE/DELETE
ON `database`.`table`
FOR EACH ROW BEGIN
      -- trigger body
      -- this code is applied to every
      -- inserted/updated/deleted row
END;
```

We require two triggers — AFTER INSERT and AFTER UPDATE on the blog table. It’s not necessary to define a DELETE trigger since a post is marked as deleted by setting its deleted field to true.

Our trigger body requires a number of SQL commands separated by a semi-colon (;). To create the full trigger code we must change delimiter to something else such as $$.

Our AFTER INSERT trigger can now be defined. It determines whether the deleted flag is set, sets the @changetype variable accordingly, and inserts a new record into the audit table:

```
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 ;
```

The AFTER UPDATE trigger is almost identical:

```
DELIMITER $$

CREATE
      TRIGGER `blog_after_update` AFTER UPDATE
      ON `blog`
      FOR EACH ROW BEGIN
            IF NEW.deleted THEN
                  SET @changetype = 'DELETE';
            ELSE
                  SET @changetype = 'EDIT';
            END IF;

            INSERT  INTO  audit  (blog_id,  changetype)  VALUES  (NEW.id,
@changetype);
END $$

DELIMITER ;
```

Let us see what happens when we insert a new post into our blog table:

```
INSERT INTO blog (title, content) VALUES ('Article One', 'Initial text.');
```

Check both the blog and the audit table.

Now let us update our blog text:

```
UPDATE blog SET content = 'Edited text' WHERE id = 1;
```

Check the blog and audit tables again.

Finally, let us mark the post as deleted:

```
UPDATE blog SET deleted = 1 WHERE id = 1;
```

The `audit` table is updated accordingly and we have a record of when changes occurred.