EIE 3112
SQL

(Part 3)

T. Connolly and C. Begg, "*Database Systems: A Practical Approach to Design, Implementation, and Management*," 6th Edition, Chapter 8, Pearson, 2015. (5th Edition is also fine)

http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx

You Will Learn

- How to use the SQL programming language
- Store Procedures in MySQL
- Exception Handling in MySQL
- How to use SQL cursors
- How to create triggers
- How to use triggers to enforce integrity constraints

SQL Programming Language

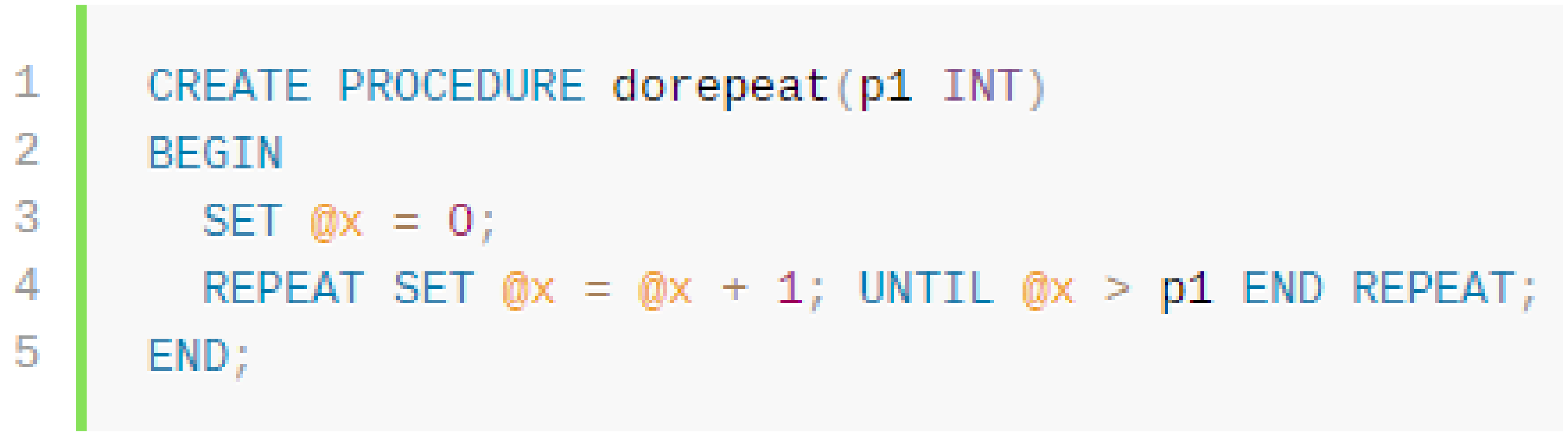
- Impedance mismatch
  - Mixing different programming paradigms
  - SQL is a declarative language (no if-then-else and for loop)
  - High-level language such as C is a procedural language
  - SQL and 3rd generation language (e.g., C++ and Java) use different models to represent data
- Solution:
  - Extend SQL to a full programming language

SQL Programming Language

- SQL/PSM (Persistent Stored Modules)
  - Extension of SQL
- PL/SQL (Procedural Language/SQL)
  - Oracle’s procedural extension to SQL

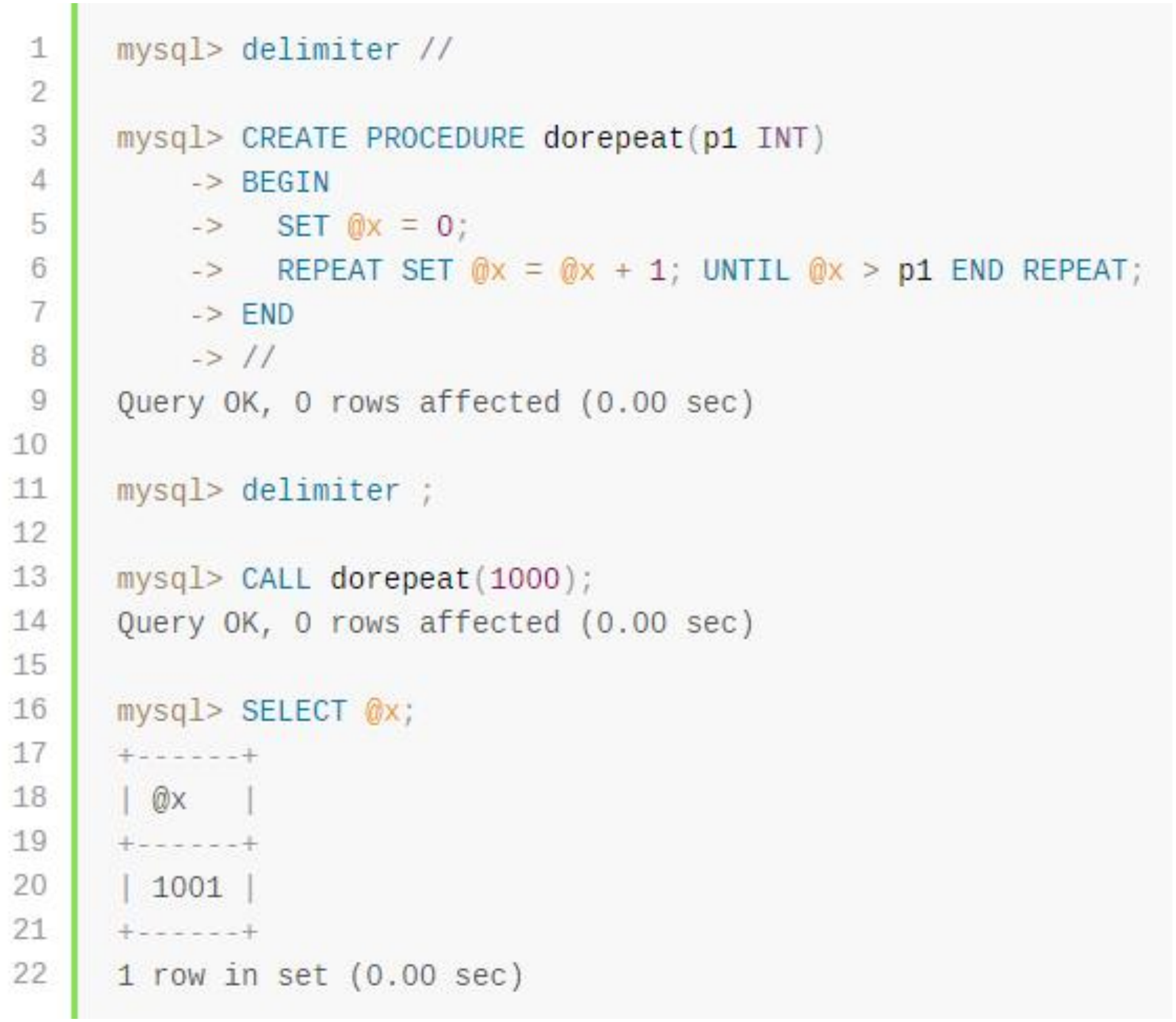
Defining Stored Procedure in MySQL

- https://dev.mysql.com/doc/refman/5.7/en/stored-programs-defining.html
- DELIMITER and PROCEDURE



```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
  REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END;
```

Defining Stored Procedure in MySQL



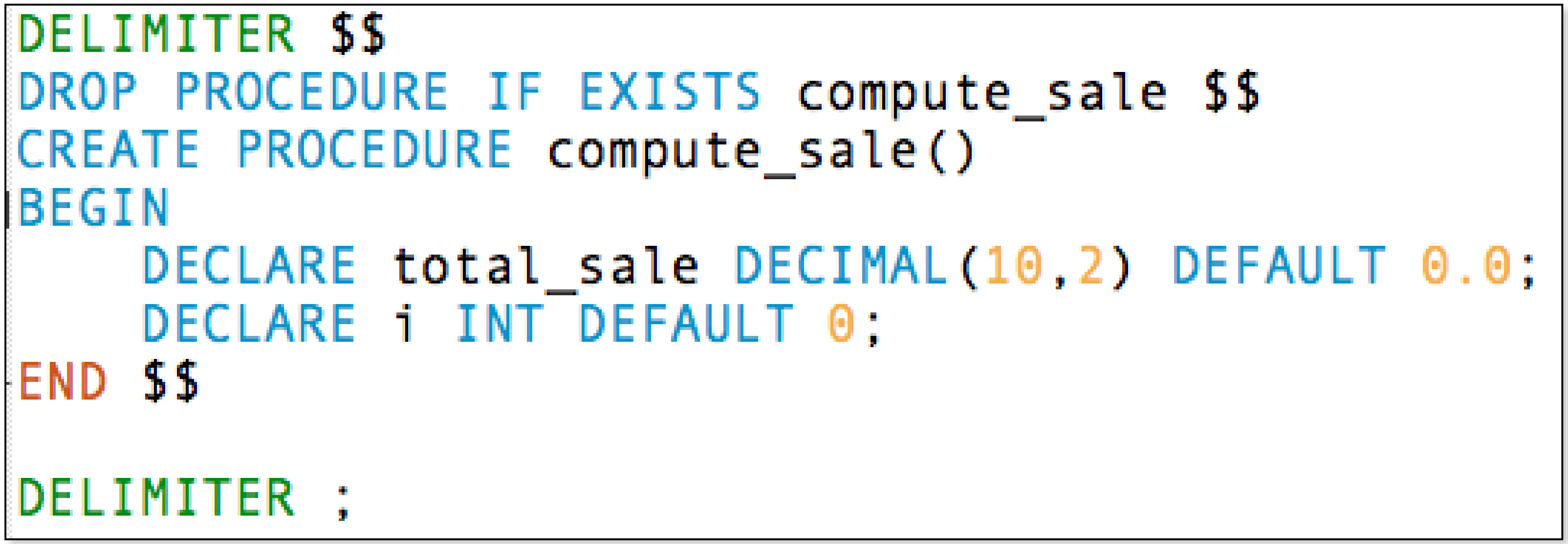
```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL dorepeat(1000);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x;
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

Declaration of Variables

- Variables and constant variables must be declared before they can be referenced

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

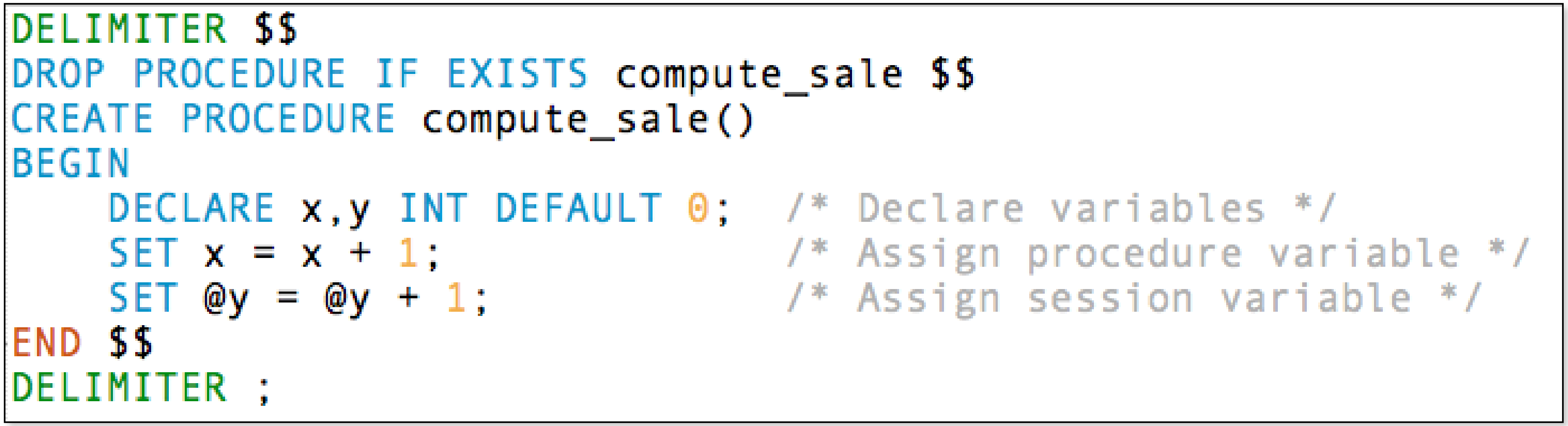
- Example in MySQL



```
DELIMITER $$
DROP PROCEDURE IF EXISTS compute_sale $$
CREATE PROCEDURE compute_sale()
BEGIN
    DECLARE total_sale DECIMAL(10,2) DEFAULT 0.0;
    DECLARE i INT DEFAULT 0;
END $$

DELIMITER ;
```

Assign Values to Variables

- Variables can be assigned in two ways:
  - Using the SET statement
  - Using an SQL SELECT or FETCH statement



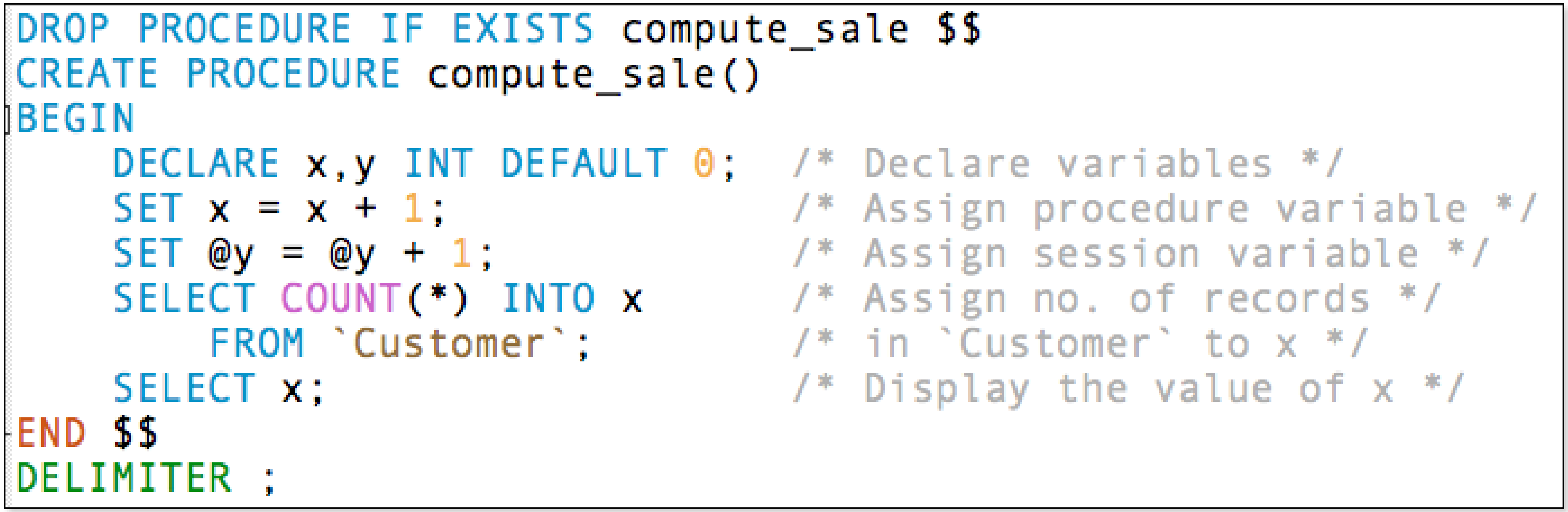
```
DELIMITER $$
DROP PROCEDURE IF EXISTS compute_sale $$
CREATE PROCEDURE compute_sale()
BEGIN
    DECLARE x,y INT DEFAULT 0;  /* Declare variables */
    SET x = x + 1;              /* Assign procedure variable */
    SET @y = @y + 1;            /* Assign session variable */
END $$
DELIMITER ;
```

Assign Values to Variables

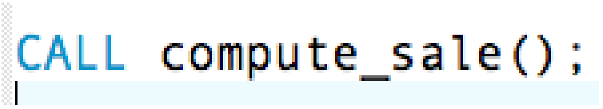
- Difference between procedure variable (y) and session variable (@y):
  - A session variable is a user-defined variable that starts with @, does not require declaration, can be used in any SQL query or statement, not visible to other sessions, and exists until the end of the current session.
  - The difference between a procedure variable and a session-specific user-defined variable is that procedure variable is reinitialized to NULL each time the procedure is called, while the session-specific variable is not.
  - The @ makes it a user-defined session variable. Otherwise it would be locally scoped variable (in a stored procedure)
  - The scope of this variable is the entire session. That means that while your connection with the database exists, the variable can still be used.

Assign Values to Variables

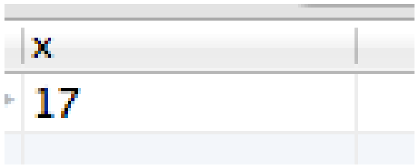
- Using an SQL SELECT or FETCH statement



```sql
DROP PROCEDURE IF EXISTS compute_sale $$
CREATE PROCEDURE compute_sale()
BEGIN
    DECLARE x,y INT DEFAULT 0;  /* Declare variables */
    SET x = x + 1;              /* Assign procedure variable */
    SET @y = @y + 1;            /* Assign session variable */
    SELECT COUNT(*) INTO x      /* Assign no. of records */
        FROM `Customer`;        /* in `Customer` to x */
    SELECT x;                   /* Display the value of x */
END $$
DELIMITER ;
```



```sql
CALL compute_sale();
```

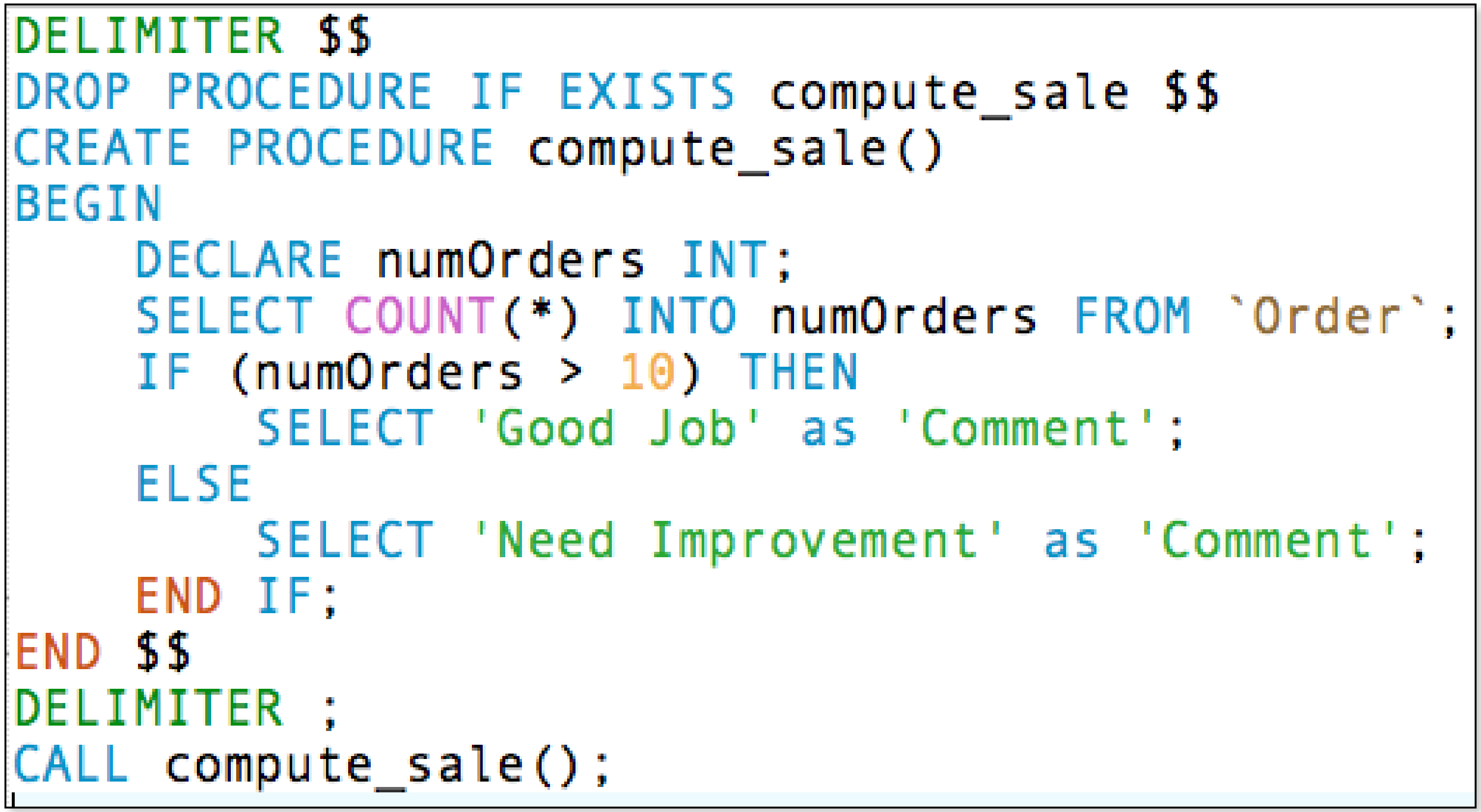


```
x
17
```

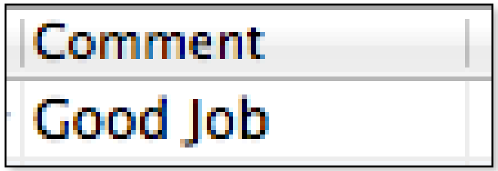
Control Statements

- Conditional IF statement
- Conditional CASE statement
- Iteration statement (LOOP)
- Iteration statement (WHILE and REPEAT)

Conditional IF Statements



```
DELIMITER $$
DROP PROCEDURE IF EXISTS compute_sale $$
CREATE PROCEDURE compute_sale()
BEGIN
    DECLARE numOrders INT;
    SELECT COUNT(*) INTO numOrders FROM `Order`;
    IF (numOrders > 10) THEN
        SELECT 'Good Job' as 'Comment';
    ELSE
        SELECT 'Need Improvement' as 'Comment';
    END IF;
END $$
DELIMITER ;
CALL compute_sale();
```



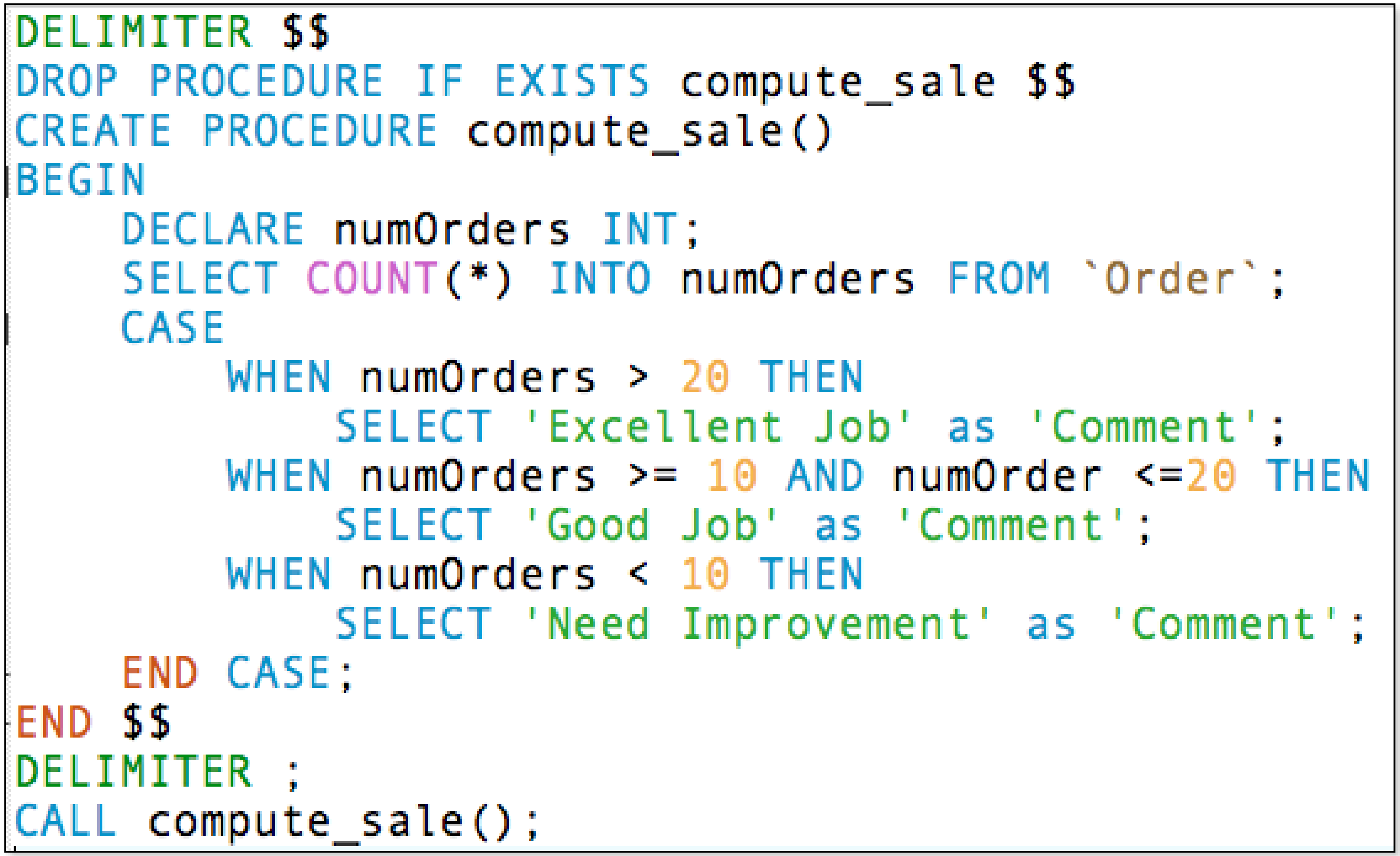
```
Comment
Good Job
```

Conditional CASE Statements

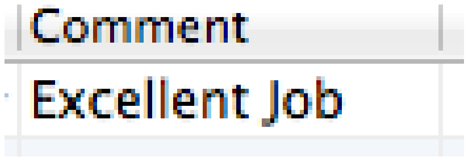
Syntax:

```
CASE  case_expression
   WHEN when_expression_1 THEN commands
   WHEN when_expression_2 THEN commands
   ...
   ELSE commands
END CASE;
```

Conditional CASE Statements

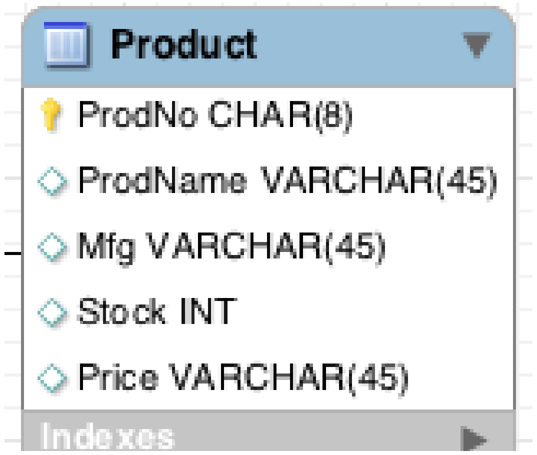


```
DELIMITER $$
DROP PROCEDURE IF EXISTS compute_sale $$
CREATE PROCEDURE compute_sale()
BEGIN
    DECLARE numOrders INT;
    SELECT COUNT(*) INTO numOrders FROM `Order`;
    CASE
        WHEN numOrders > 20 THEN
            SELECT 'Excellent Job' as 'Comment';
        WHEN numOrders >= 10 AND numOrder <=20 THEN
            SELECT 'Good Job' as 'Comment';
        WHEN numOrders < 10 THEN
            SELECT 'Need Improvement' as 'Comment';
    END CASE;
END $$
DELIMITER ;
CALL compute_sale();
```



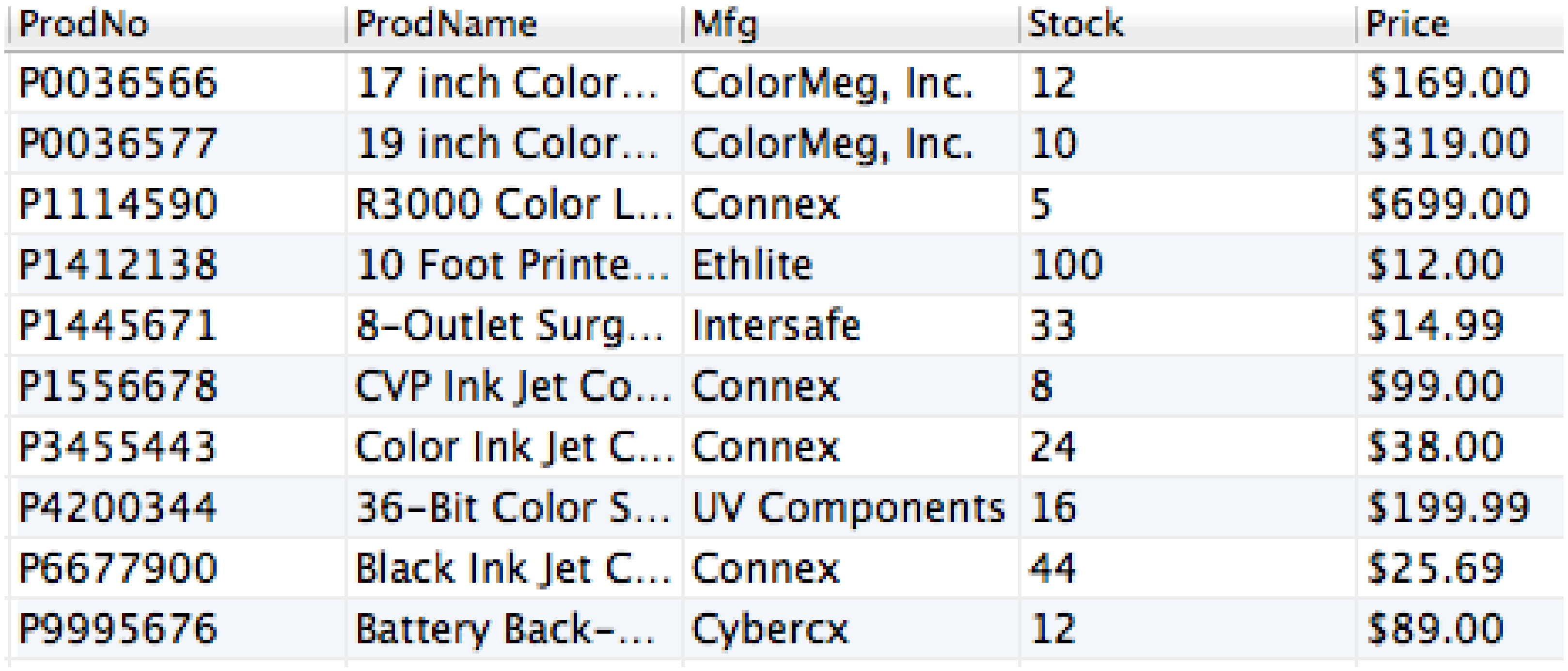
Comment
Excellent Job

Conditional CASE Statements



Product
- ProdNo CHAR(8)
- ProdName VARCHAR(45)
- Mfg VARCHAR(45)
- Stock INT
- Price VARCHAR(45)

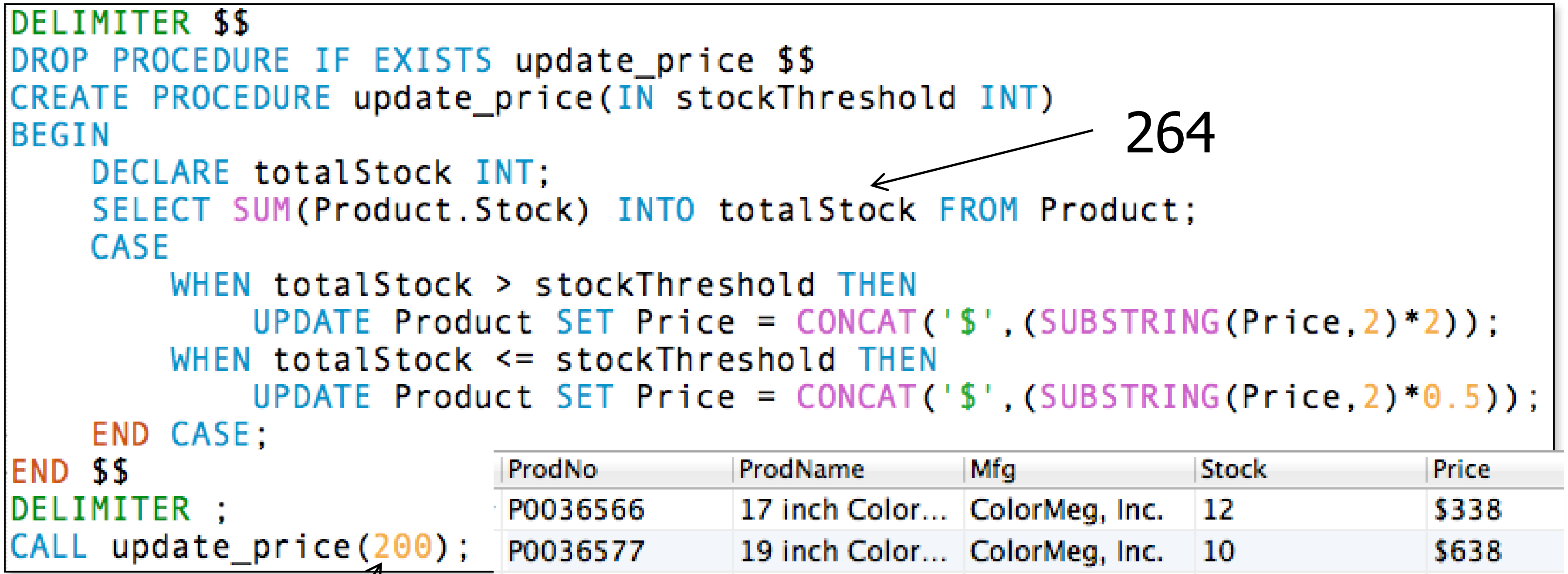
Indexes



ProdNo	ProdName	Mfg	Stock	Price
P0036566	17 inch Color…	ColorMeg, Inc.	12	$169.00
P0036577	19 inch Color…	ColorMeg, Inc.	10	$319.00
P1114590	R3000 Color L…	Connex	5	$699.00
P1412138	10 Foot Printe…	Ethlite	100	$12.00
P1445671	8-Outlet Surg…	Intersafe	33	$14.99
P1556678	CVP Ink Jet Co…	Connex	8	$99.00
P3455443	Color Ink Jet C…	Connex	24	$38.00
P4200344	36-Bit Color S…	UV Components	16	$199.99
P6677900	Black Ink Jet C…	Connex	44	$25.69
P9995676	Battery Back-…	Cybercx	12	$89.00

Conditional CASE Statements

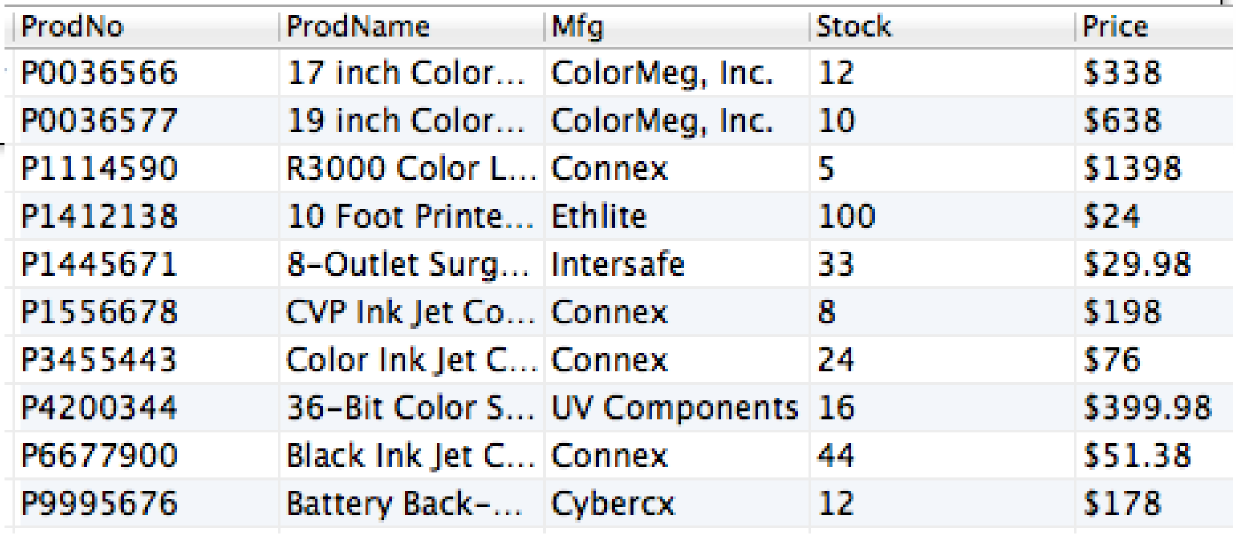
Double/half the price if the total stock is larger/smaller than 200



```
DELIMITER $$
DROP PROCEDURE IF EXISTS update_price $$
CREATE PROCEDURE update_price(IN stockThreshold INT)
BEGIN
    DECLARE totalStock INT;
    SELECT SUM(Product.Stock) INTO totalStock FROM Product;
    CASE
        WHEN totalStock > stockThreshold THEN
            UPDATE Product SET Price = CONCAT('$',(SUBSTRING(Price,2)*2));
        WHEN totalStock <= stockThreshold THEN
            UPDATE Product SET Price = CONCAT('$',(SUBSTRING(Price,2)*0.5));
    END CASE;
END $$
DELIMITER ;
CALL update_price(200);
```

264

Parameter

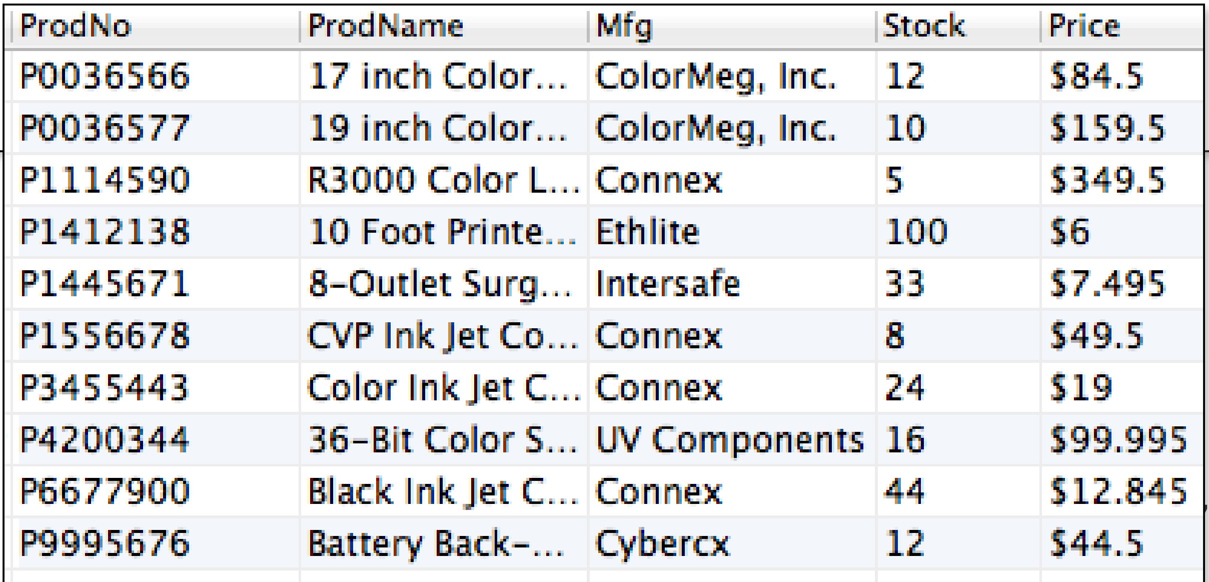


ProdNo	ProdName	Mfg	Stock	Price
P0036566	17 inch Color…	ColorMeg, Inc.	12	$338
P0036577	19 inch Color…	ColorMeg, Inc.	10	$638
P1114590	R3000 Color L…	Connex	5	$1398
P1412138	10 Foot Printe…	Ethlite	100	$24
P1445671	8-Outlet Surg…	Intersafe	33	$29.98
P1556678	CVP Ink Jet Co…	Connex	8	$198
P3455443	Color Ink Jet C…	Connex	24	$76
P4200344	36-Bit Color S…	UV Components	16	$399.98
P6677900	Black Ink Jet C…	Connex	44	$51.38
P9995676	Battery Back-…	Cybercx	12	$178

Conditional CASE Statements

Double/half the price if the total stock is larger/smaller than 500

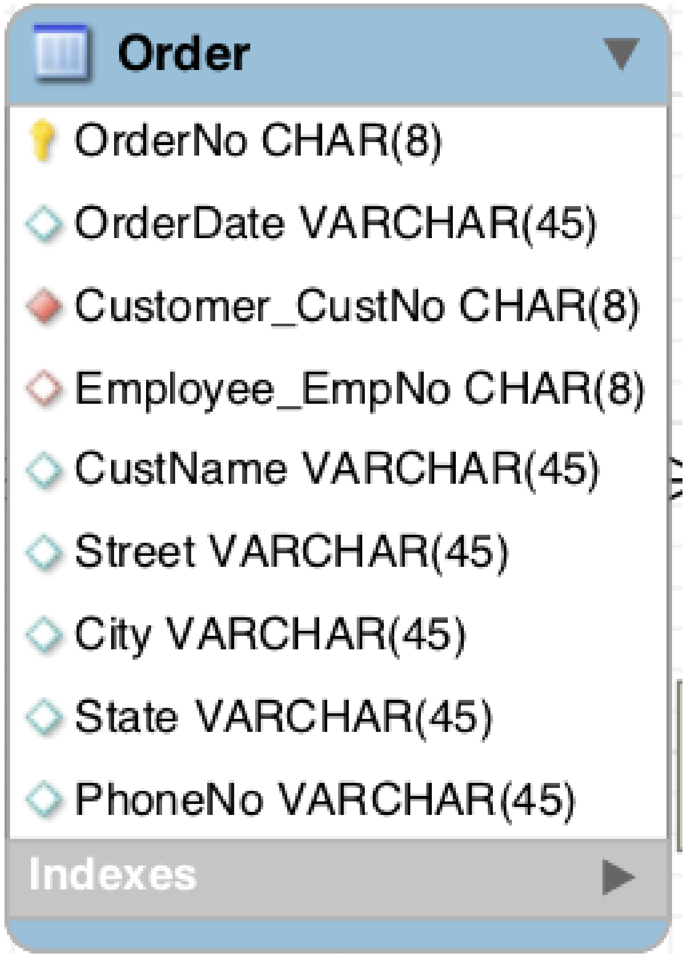
```
DELIMITER $$
DROP PROCEDURE IF EXISTS update_price $$
CREATE PROCEDURE update_price(IN stockThreshold INT)
BEGIN
    DECLARE totalStock INT;
    SELECT SUM(Product.Stock) INTO totalStock FROM Product;
    CASE
        WHEN totalStock > stockThreshold THEN
            UPDATE Product SET Price = CONCAT('$',(SUBSTRING(Price,2)*2));
        WHEN totalStock <= stockThreshold THEN
            UPDATE Product SET Price = CONCAT('$',(SUBSTRING(Price,2)*0.5));
    END CASE;
END $$
DELIMITER ;
CALL update_price(500);
```



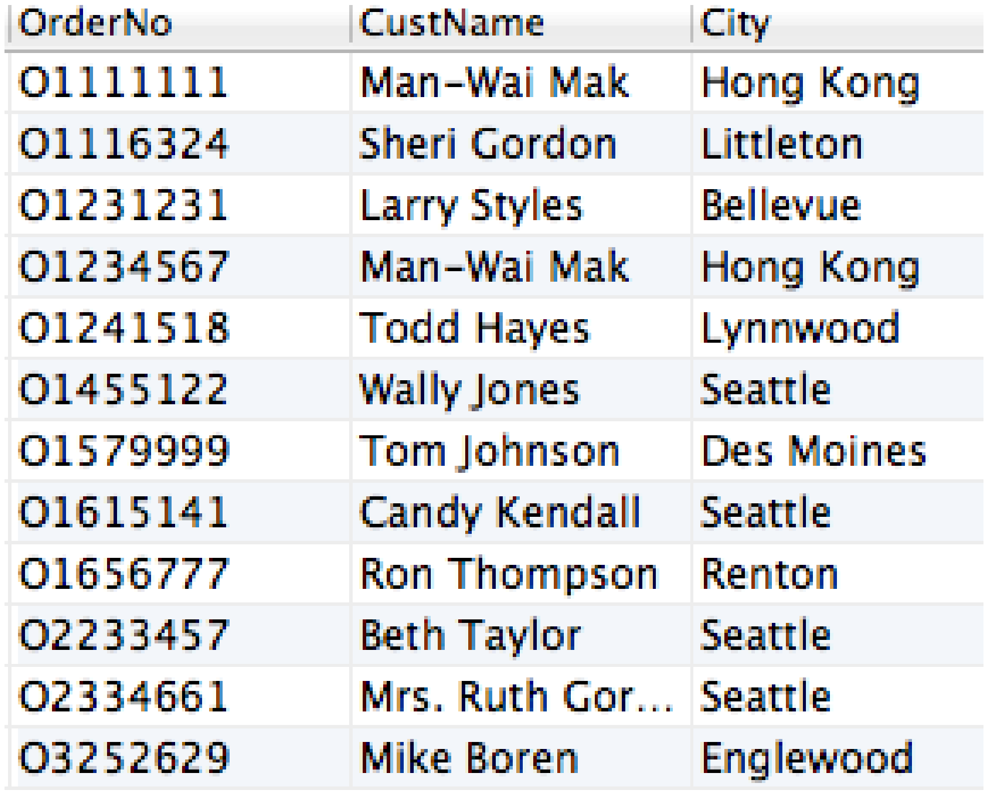
ProdNo	ProdName	Mfg	Stock	Price
P0036566	17 inch Color...	ColorMeg, Inc.	12	$84.5
P0036577	19 inch Color...	ColorMeg, Inc.	10	$159.5
P1114590	R3000 Color L...	Connex	5	$349.5
P1412138	10 Foot Printe...	Ethlite	100	$6
P1445671	8-Outlet Surg...	Intersafe	33	$7.495
P1556678	CVP Ink Jet Co...	Connex	8	$49.5
P3455443	Color Ink Jet C...	Connex	24	$19
P4200344	36-Bit Color S...	UV Components	16	$99.995
P6677900	Black Ink Jet C...	Connex	44	$12.845
P9995676	Battery Back-...	Cybercx	12	$44.5

Conditional CASE Statements

Return the number of days required for shipping out an order based on the recipient's city

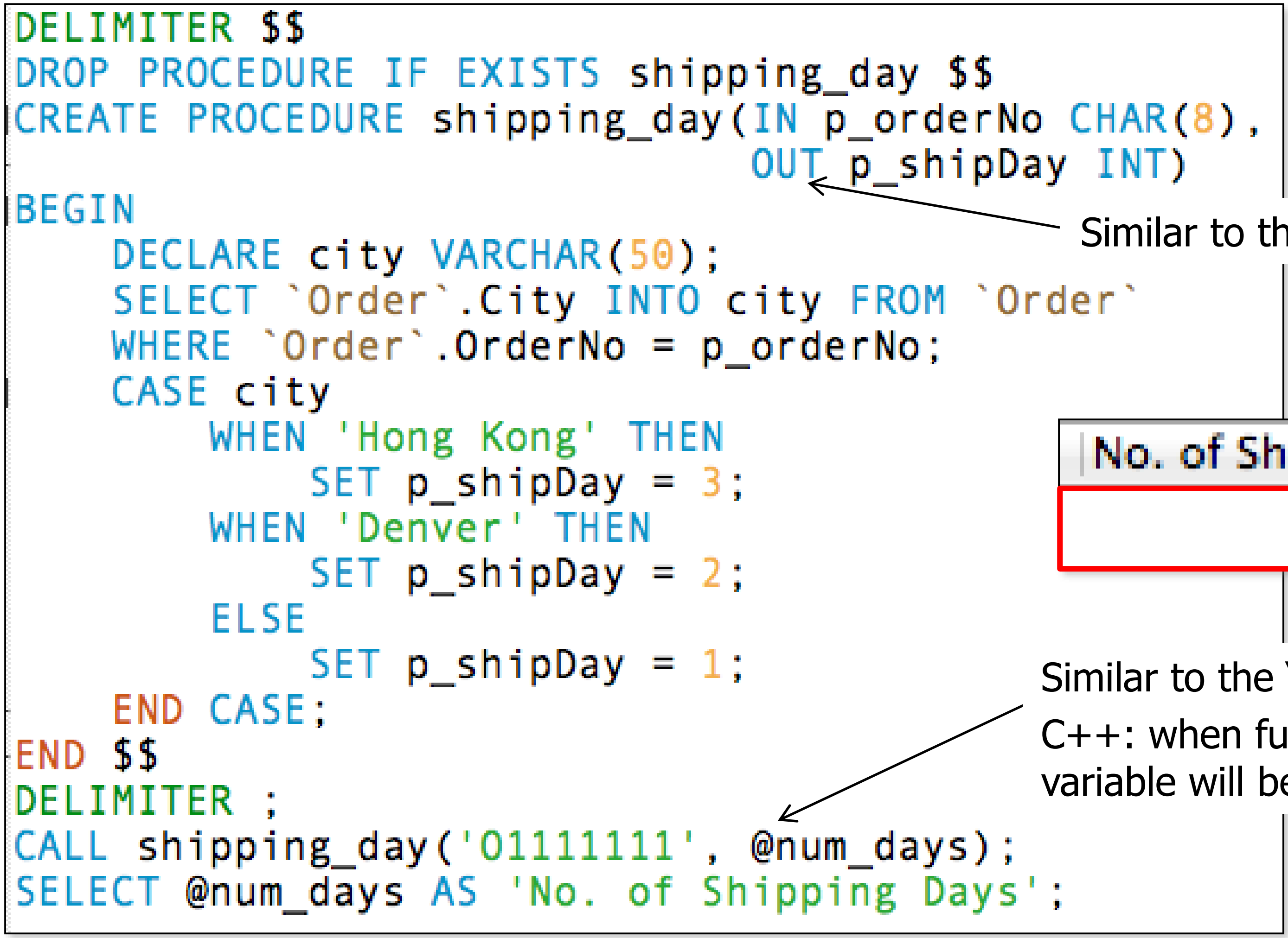


Order
OrderNo CHAR(8)
OrderDate VARCHAR(45)
Customer_CustNo CHAR(8)
Employee_EmpNo CHAR(8)
CustName VARCHAR(45)
Street VARCHAR(45)
City VARCHAR(45)
State VARCHAR(45)
PhoneNo VARCHAR(45)
Indexes



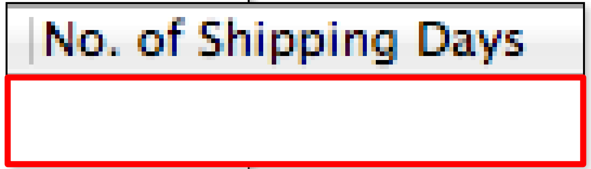
OrderNo	CustName	City
O1111111	Man-Wai Mak	Hong Kong
O1116324	Sheri Gordon	Littleton
O1231231	Larry Styles	Bellevue
O1234567	Man-Wai Mak	Hong Kong
O1241518	Todd Hayes	Lynnwood
O1455122	Wally Jones	Seattle
O1579999	Tom Johnson	Des Moines
O1615141	Candy Kendall	Seattle
O1656777	Ron Thompson	Renton
O2233457	Beth Taylor	Seattle
O2334661	Mrs. Ruth Gor...	Seattle
O3252629	Mike Boren	Englewood

Conditional CASE Statements



```
DELIMITER $$
DROP PROCEDURE IF EXISTS shipping_day $$
CREATE PROCEDURE shipping_day(IN p_orderNo CHAR(8),
                              OUT p_shipDay INT)
BEGIN
    DECLARE city VARCHAR(50);
    SELECT `Order`.City INTO city FROM `Order`
    WHERE `Order`.OrderNo = p_orderNo;
    CASE city
        WHEN 'Hong Kong' THEN
            SET p_shipDay = 3;
        WHEN 'Denver' THEN
            SET p_shipDay = 2;
        ELSE
            SET p_shipDay = 1;
    END CASE;
END $$
DELIMITER ;
CALL shipping_day('01111111', @num_days);
SELECT @num_days AS 'No. of Shipping Days';
```

Similar to the "*" notation in C++



```
No. of Shipping Days
```

Similar to the "&" notation in C++: when function returns, the variable will be updated

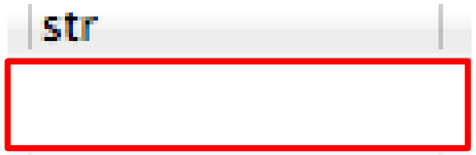
Iteration WHILE Statements

Syntax:

```
WHILE expression DO
   Statements
END WHILE
```

Example:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS while_loop $$
CREATE PROCEDURE while_loop()
BEGIN
    DECLARE x INT;
    DECLARE str VARCHAR(255);
    SET x = 1;
    SET str = '';
    WHILE x <= 5 DO
        SET str = CONCAT(str,x,',');
        SET x = x + 1;
    END WHILE;
    SELECT str;
END$$
DELIMITER ;
CALL while_loop();
```



```
str
```

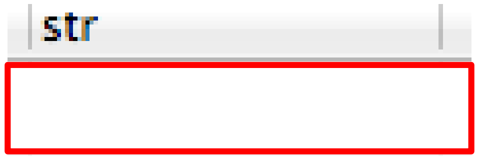
Iteration REPEAT Statements

Syntax:

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

Example:

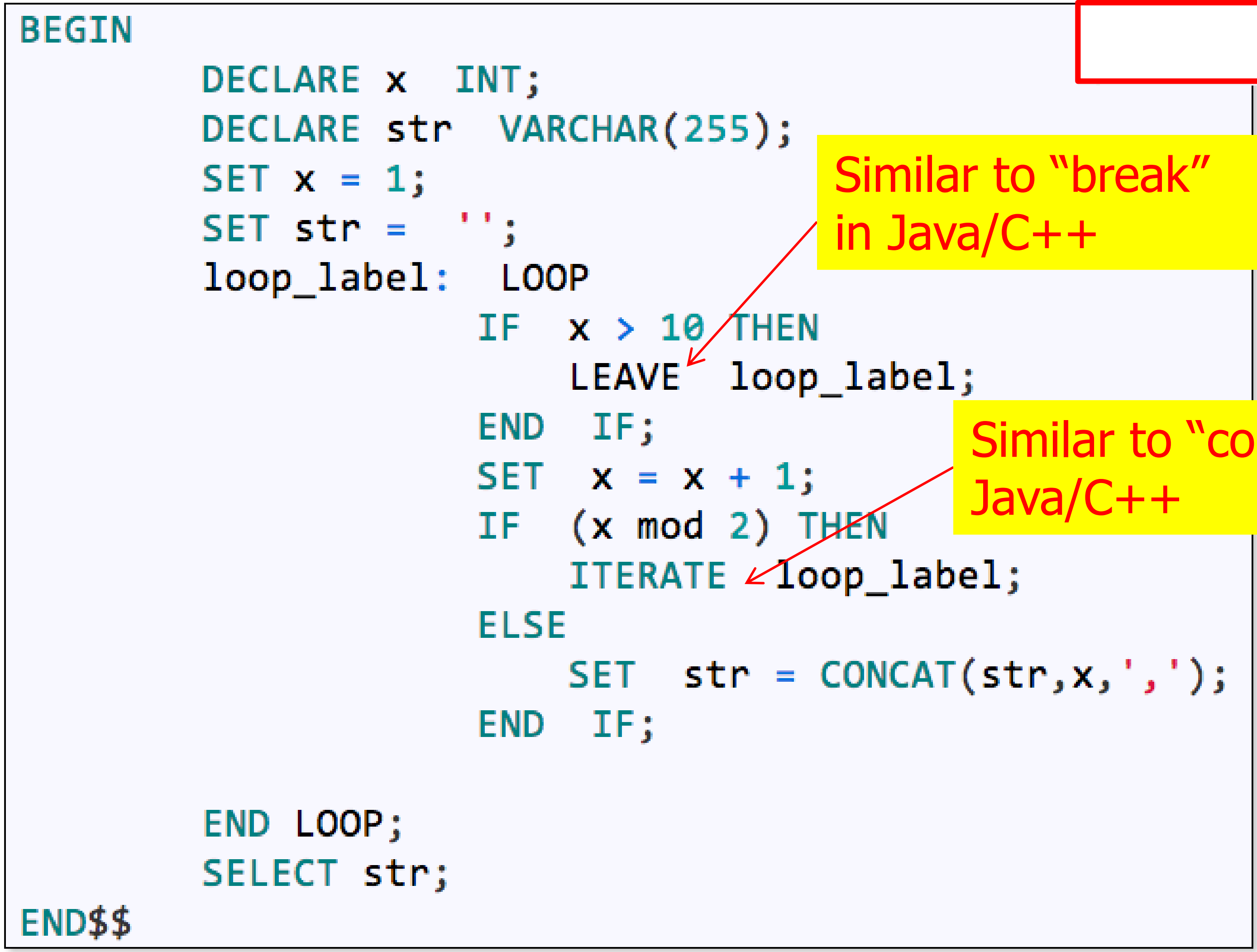
```
DELIMITER $$
DROP PROCEDURE IF EXISTS while_loop $$
CREATE PROCEDURE repeat_loop()
BEGIN
    DECLARE x INT;
    DECLARE str VARCHAR(255);
    SET x = 1;
    SET str = '';
    REPEAT
        SET str = CONCAT(str,x,',');
        SET x = x + 1;
    UNTIL x > 5
    END REPEAT;
    SELECT str;
END$$
DELIMITER ;
CALL repeat_loop();
```



```
str
```

Iteration LOOP Statements

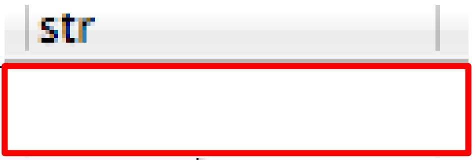
Example:



```
BEGIN
        DECLARE x  INT;
        DECLARE str  VARCHAR(255);
        SET x = 1;
        SET str =  '';
        loop_label:  LOOP
                     IF  x > 10 THEN
                         LEAVE  loop_label;
                     END  IF;
                     SET  x = x + 1;
                     IF  (x mod 2) THEN
                         ITERATE  loop_label;
                     ELSE
                         SET  str = CONCAT(str,x,',');
                     END  IF;

        END LOOP;
        SELECT str;
END$$
```

Similar to "break" in Java/C++

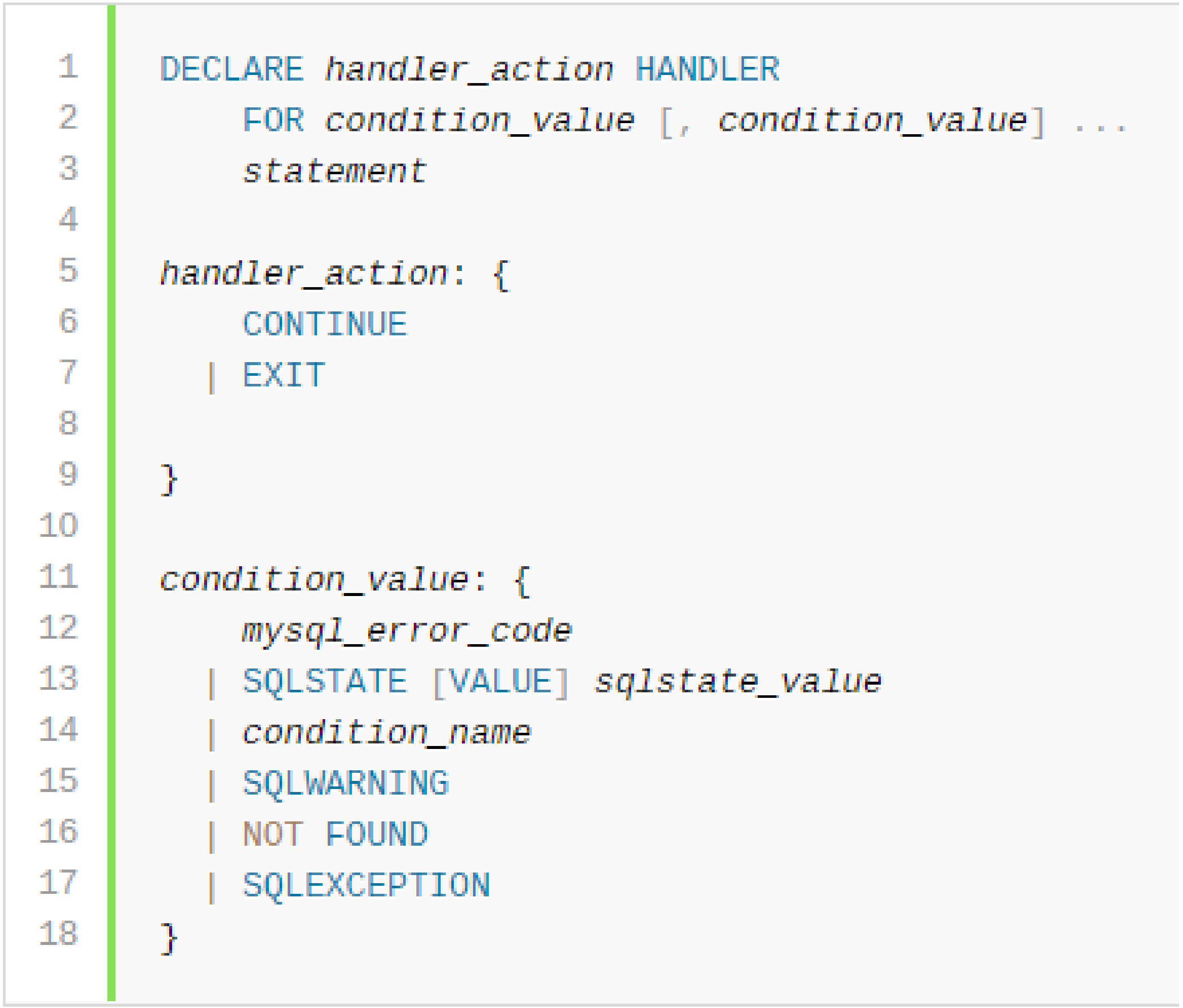
Similar to "continue" in Java/C++



```
str
```

Exception Handling in MySQL

Syntax:



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

handler_action: {
    CONTINUE
  | EXIT

}

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

Exception Handling in MySQL

Example 1:

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET has_error = 1;
```

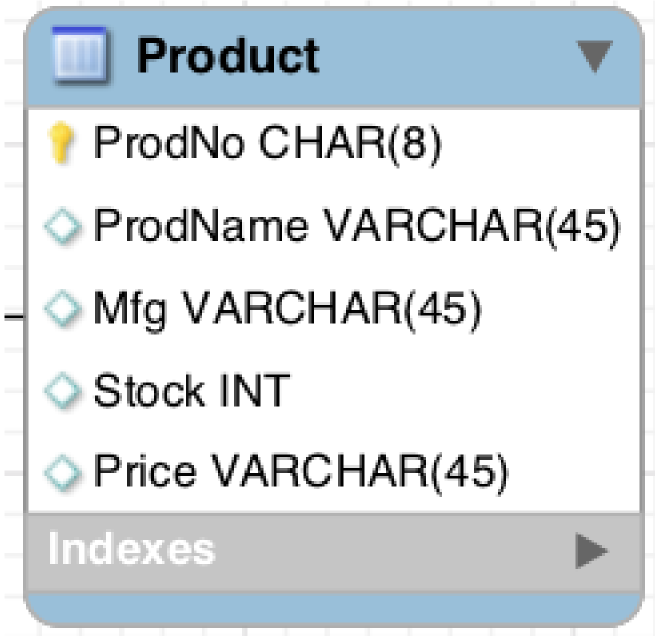
If an error occurs, set the value of the variable *has_error* to 1 and continue the execution.

Example 2:

```
DECLARE CONTINUE HANDLER FOR 1062
SELECT 'Error, duplicate key occurred';
```

Display an error message when ERROR 1062 occurs

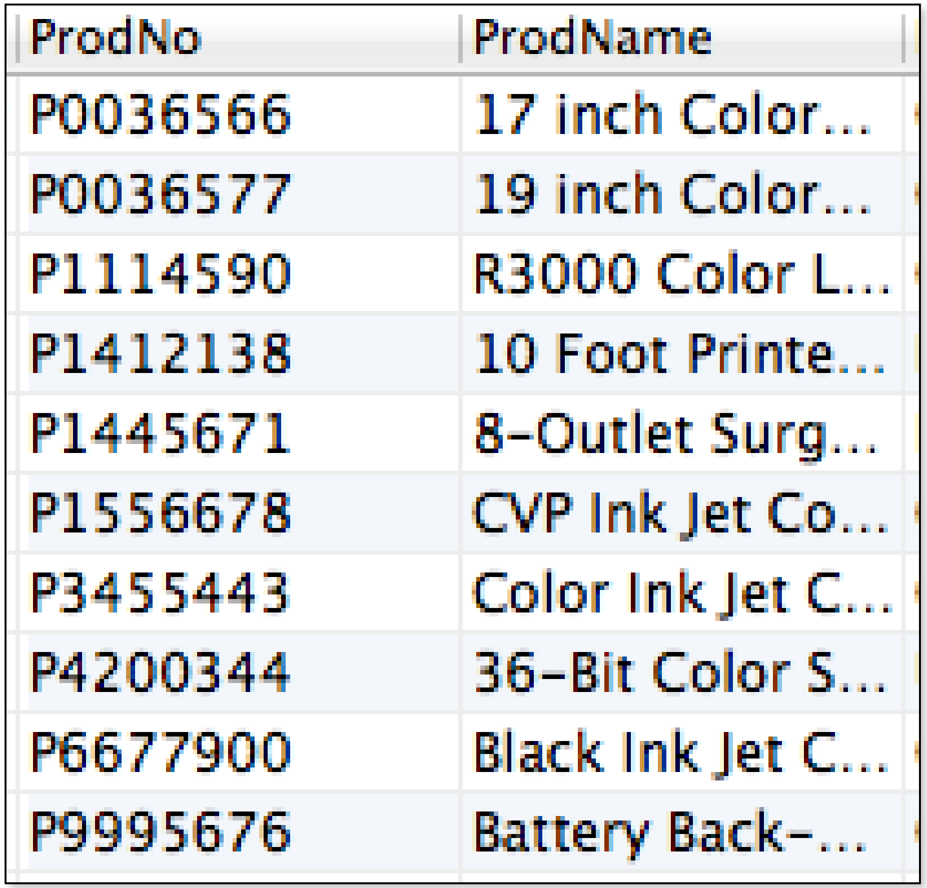
Exception Handling in MySQL



Product

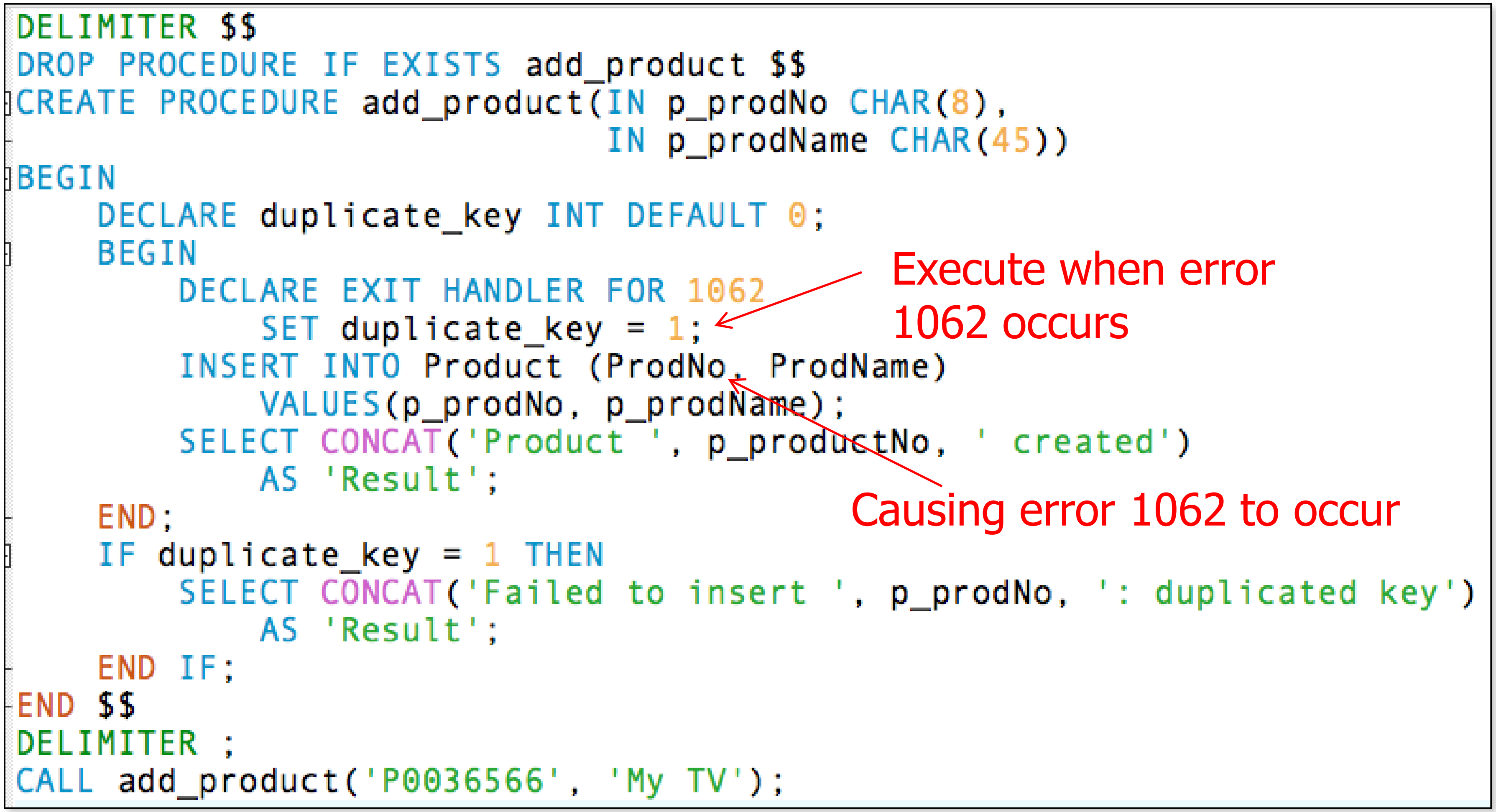
- ProdNo CHAR(8)
- ProdName VARCHAR(45)
- Mfg VARCHAR(45)
- Stock INT
- Price VARCHAR(45)

Indexes



ProdNo	ProdName
P0036566	17 inch Color...
P0036577	19 inch Color...
P1114590	R3000 Color L...
P1412138	10 Foot Printe...
P1445671	8-Outlet Surg...
P1556678	CVP Ink Jet Co...
P3455443	Color Ink Jet C...
P4200344	36-Bit Color S...
P6677900	Black Ink Jet C...
P9995676	Battery Back-...

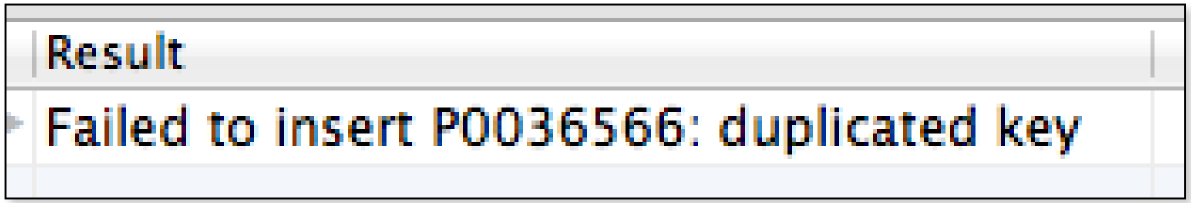
Exception Handling in MySQL



```
DELIMITER $$
DROP PROCEDURE IF EXISTS add_product $$
CREATE PROCEDURE add_product(IN p_prodNo CHAR(8),
                             IN p_prodName CHAR(45))
BEGIN
    DECLARE duplicate_key INT DEFAULT 0;
    BEGIN
        DECLARE EXIT HANDLER FOR 1062
            SET duplicate_key = 1;
        INSERT INTO Product (ProdNo, ProdName)
            VALUES(p_prodNo, p_prodName);
        SELECT CONCAT('Product ', p_productNo, ' created')
            AS 'Result';
    END;
    IF duplicate_key = 1 THEN
        SELECT CONCAT('Failed to insert ', p_prodNo, ': duplicated key')
            AS 'Result';
    END IF;
END $$
DELIMITER ;
CALL add_product('P0036566', 'My TV');
```

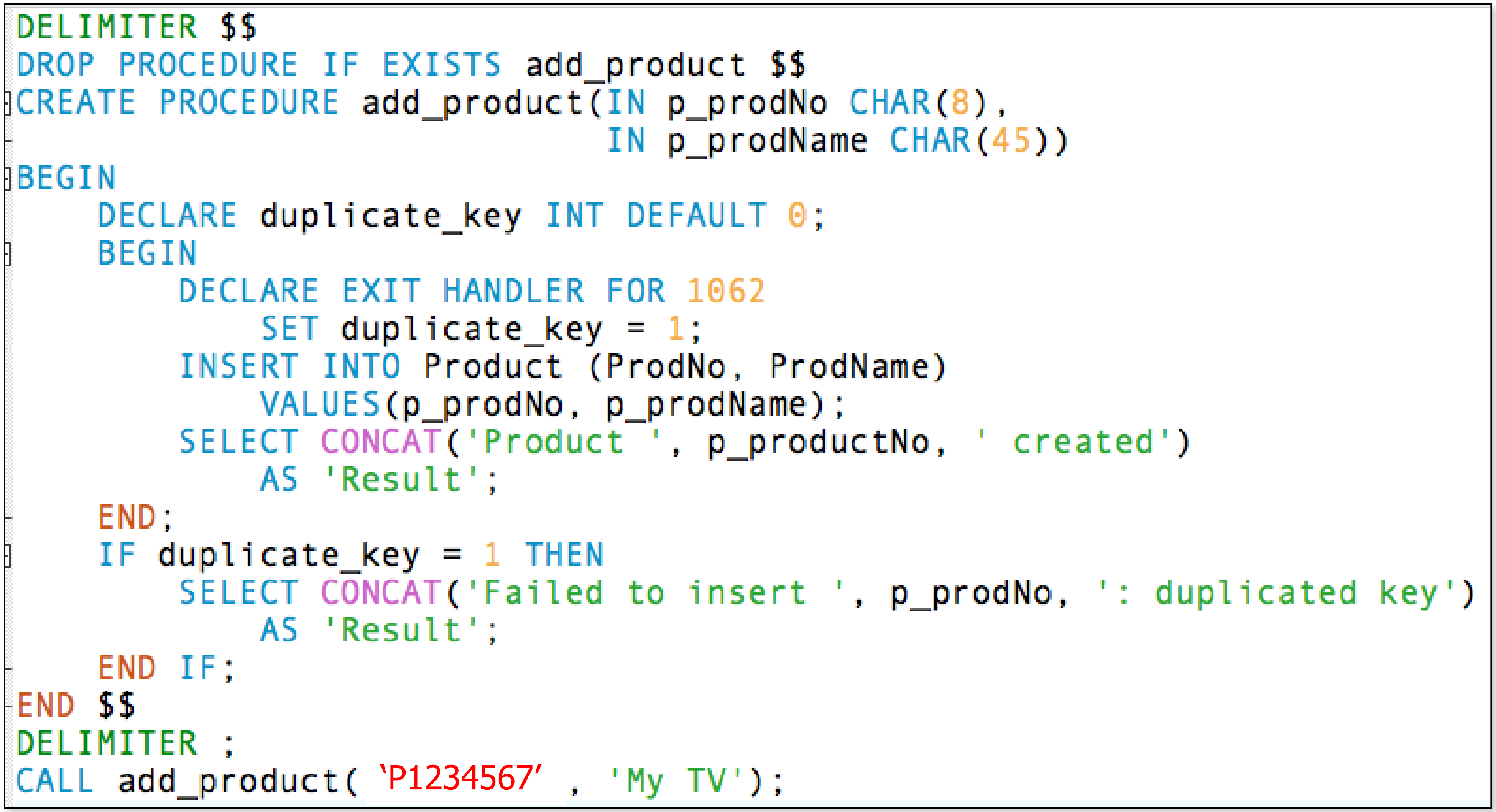
Execute when error 1062 occurs

Causing error 1062 to occur

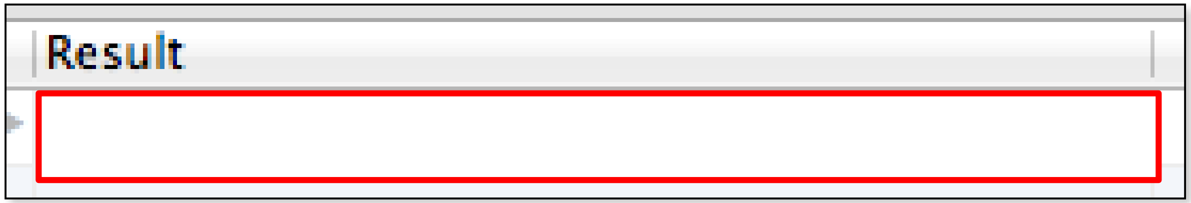


```
Result
Failed to insert P0036566: duplicated key
```

Exception Handling in MySQL



```
DELIMITER $$
DROP PROCEDURE IF EXISTS add_product $$
CREATE PROCEDURE add_product(IN p_prodNo CHAR(8),
                             IN p_prodName CHAR(45))
BEGIN
    DECLARE duplicate_key INT DEFAULT 0;
    BEGIN
        DECLARE EXIT HANDLER FOR 1062
            SET duplicate_key = 1;
        INSERT INTO Product (ProdNo, ProdName)
            VALUES(p_prodNo, p_prodName);
        SELECT CONCAT('Product ', p_productNo, ' created')
            AS 'Result';
    END;
    IF duplicate_key = 1 THEN
        SELECT CONCAT('Failed to insert ', p_prodNo, ': duplicated key')
            AS 'Result';
    END IF;
END $$
DELIMITER ;
CALL add_product( ‘P1234567’ , 'My TV');
```



Result

Cursors in MySQL

- Cursor
  - Allows the rows of a query result to be accessed one at a time
  - Must be declared and opened before use
  - Must be closed to deactivate it after it is no longer required
  - Read-only
  - Non-scrollable

Cursors in MySQL

- **Read only**: you cannot update data in the underlying table through the cursor.
- **Non-scrollable**: you can only fetch rows in the order determined by the SELECT statement. You cannot fetch rows in the reversed order. In addition, you cannot skip rows or jump to a specific row in the result set.

Working with MySQL Cursors

Step 1: Declare a cursor (after any variable declaration):

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

Step 2: Open a cursor

```
OPEN cursor_name;
```

Step 3: Use the FETCH statement to retrieve the next row pointed to by the cursor and move the cursor to the next row in the result set

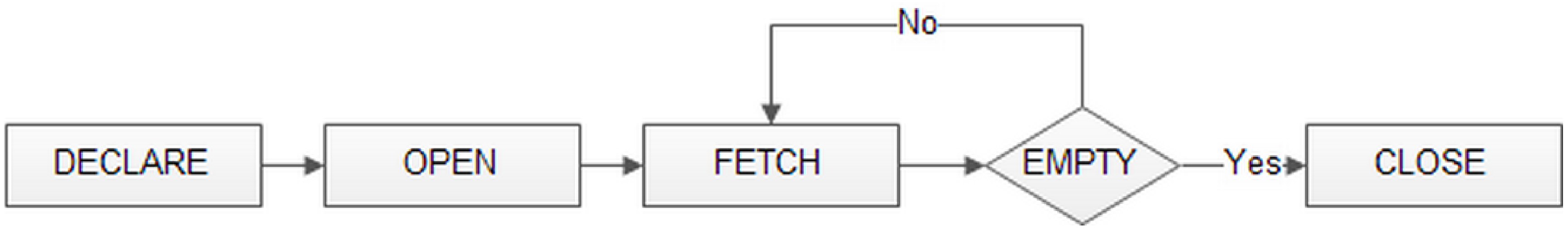
```
FETCH cursor_name INTO variables list;
```

Working with MySQL Cursors

Step 4: call the CLOSE statement to deactivate the cursor and release the memory associated with it

```
CLOSE cursor_name;
```

All Steps:



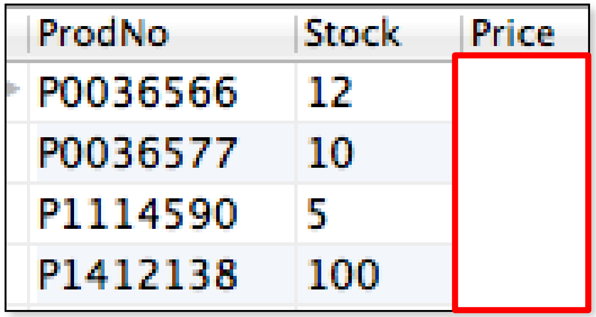
Example of MySQL Cursors

```
DELIMITER $$
CREATE PROCEDURE build_email_list (INOUT email_list varchar(4000))
BEGIN
    DECLARE v_finished INTEGER DEFAULT 0;
        DECLARE v_email varchar(100) DEFAULT "";

    -- declare cursor for employee email
    DECLARE email_cursor CURSOR FOR SELECT email FROM employees;

    -- declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_finished = 1;

    OPEN email_cursor;

    get_email: LOOP
        FETCH email_cursor INTO v_email;
        IF v_finished = 1 THEN
            LEAVE get_email;
        END IF;

        -- build email list
        SET email_list = CONCAT(v_email,";",email_list);
    END LOOP get_email;

    CLOSE email_cursor;
END$$
DELIMITER ;
```

Example of MySQL Cursors

```
DELIMITER $$
DROP PROCEDURE IF EXISTS change_price $$
CREATE PROCEDURE change_price(IN stockThreshold INT)
BEGIN
    DECLARE v_finished INTEGER DEFAULT 0;
    DECLARE v_prodNo CHAR(8);
    DECLARE v_stock INT DEFAULT 0;
    DECLARE product_cursor CURSOR FOR SELECT ProdNo, Stock FROM Product;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_finished = 1;

    OPEN product_cursor;
    get_product: LOOP
        FETCH product_cursor INTO v_prodNo, v_stock;
        IF v_finished = 1 THEN
            LEAVE get_product;
        END IF;
        CASE
            WHEN v_stock > stockThreshold THEN
                UPDATE Product SET Price = '$200' WHERE ProdNo = v_prodNo;
            WHEN v_stock <= stockThreshold THEN
                UPDATE Product SET Price = '$100' WHERE ProdNo = v_prodNo;
        END CASE;
    END LOOP get_product;

    CLOSE product_cursor;
END $$
DELIMITER ;
CALL change_price(10);
SELECT ProdNo, Stock, Price FROM Product;
```

Change price of product depending on stock



ProdNo	Stock	Price
P0036566	12	
P0036577	10	
P1114590	5	
P1412138	100	

Cursor.sql

Triggers

- Defines an action that the database should take when some events occur in the application.
- Triggers can be used for checking values to be inserted into a table or for calculating values involved in an update.
- A trigger is defined to activate when a statement inserts, updates, or deletes rows in the associated table. These row operations are called “trigger events”.
- Database trigger is a powerful tool for protecting the integrity of data

Syntax of Triggers

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }
```

`BEFORE INSERT` – activated before data is inserted into the table.

`AFTER INSERT` - activated after data is inserted into the table.

`BEFORE UPDATE` – activated before data in the table is updated.

`AFTER UPDATE` - activated after data in the table is updated.

`BEFORE DELETE` – activated before data is removed from the table.

`AFTER DELETE` – activated after data is removed from the table.

Creating Triggers in MySQL

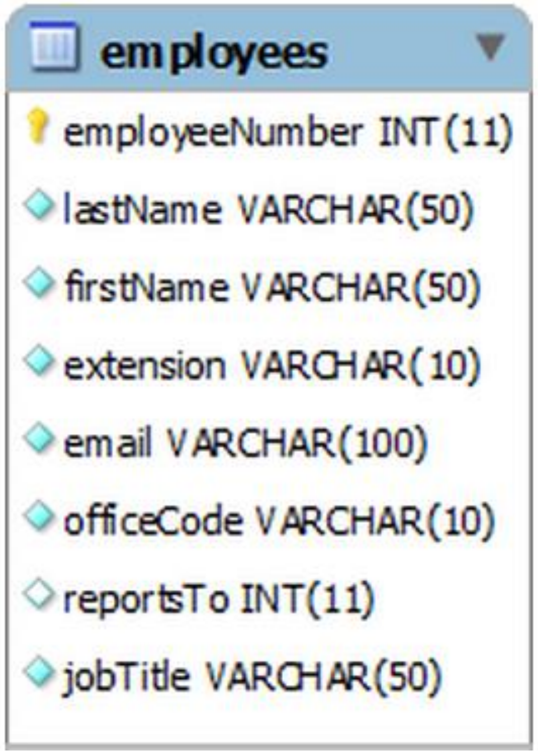
```
CREATE TRIGGER trigger_name trigger_time trigger_event
 ON table_name
 FOR EACH ROW
 BEGIN
 ...
 END
```

- The trigger name should follow the naming convention [trigger time]_[table name]_[trigger event], e.g. `before_employees_update`.
- Trigger event can be `INSERT`, `UPDATE` or `DELETE`. This event causes trigger to be invoked.
- A trigger must be associated with a specific table.
- The SQL statements are placed between BEGIN and END block.

Creating Triggers in MySQL

- The `OLD` keyword refers to the existing record before you change the data
- The `NEW` keyword refers to the new row after you change the data.

Example Triggers in MySQL



```
employees
employeeNumber INT(11)
lastName VARCHAR(50)
firstName VARCHAR(50)
extension VARCHAR(10)
email VARCHAR(100)
officeCode VARCHAR(10)
reportsTo INT(11)
jobTitle VARCHAR(50)
```

```sql
CREATE TABLE employees_audit (
    id int(11) NOT NULL AUTO_INCREMENT,
    employeeNumber int(11) NOT NULL,
    lastname varchar(50) NOT NULL,
    changedon datetime DEFAULT NULL,
    action varchar(50) DEFAULT NULL,
    PRIMARY KEY (id)
)
```

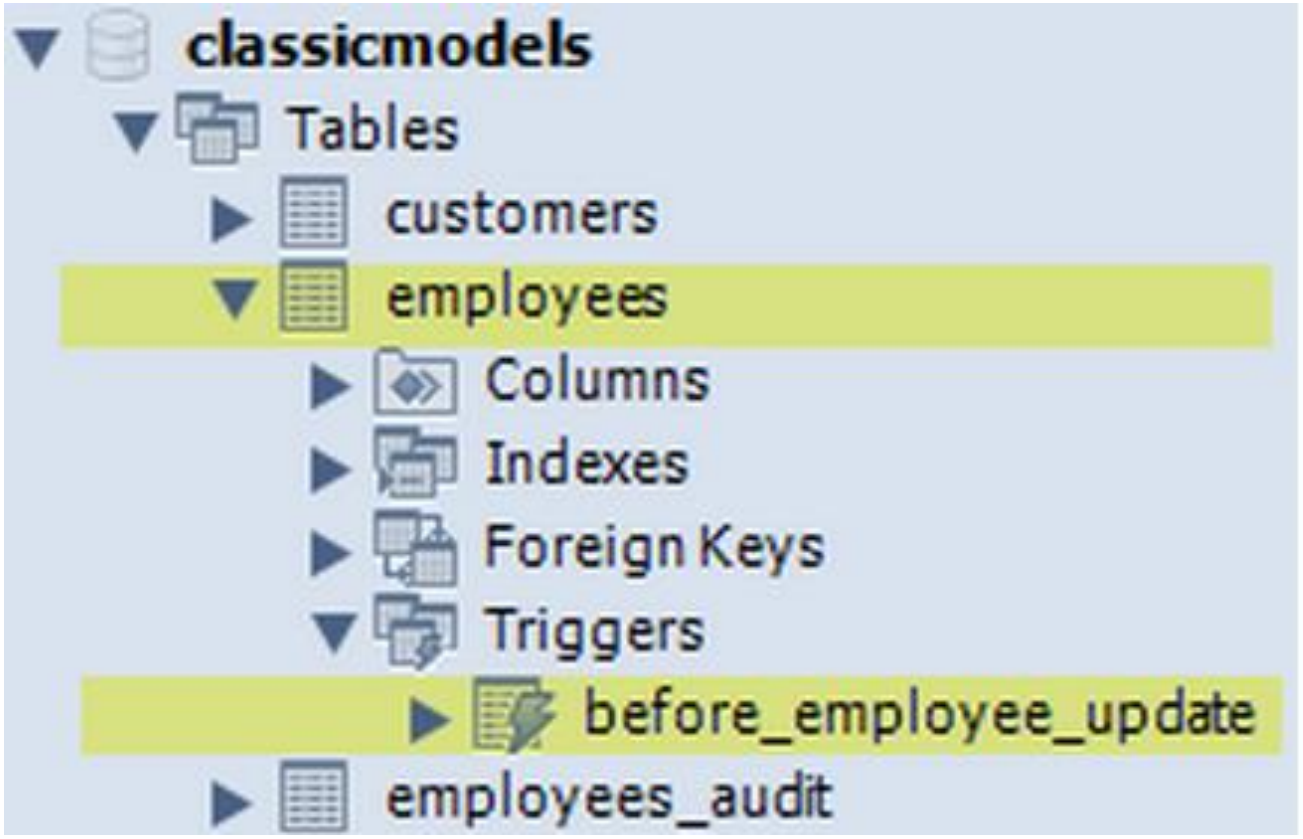
Example Triggers in MySQL

- Create a BEFORE UPDATE trigger to be invoked before a change is made to the employees table.

```
DELIMITER $$
CREATE TRIGGER before_employee_update
    BEFORE UPDATE ON employees
    FOR EACH ROW BEGIN

    INSERT INTO employees_audit
    SET action = 'update',
         employeeNumber = OLD.employeeNumber,
        lastname = OLD.lastname,
        changedon = NOW();
END$$
DELIMITER ;
```

Example Triggers in MySQL

- The schema of this example:



```
▼ classicmodels
    ▼ Tables
        ▶ customers
        ▼ employees
            ▶ Columns
            ▶ Indexes
            ▶ Foreign Keys
            ▼ Triggers
                ▶ before_employee_update
        ▶ employees_audit
```

Example Triggers in MySQL

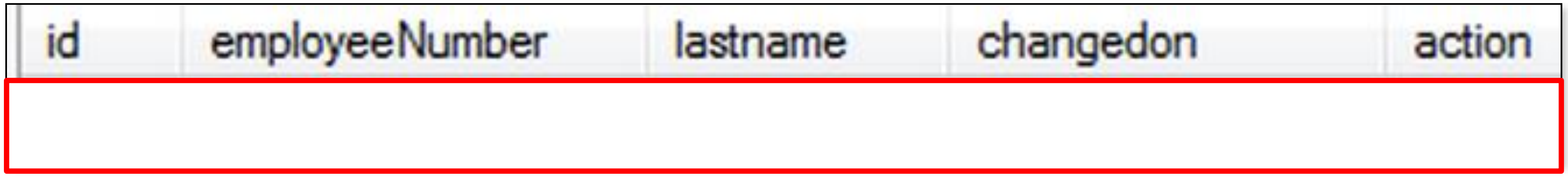
- Update an employee record to test if the trigger is really invoked.

```
UPDATE employees
SET lastName = 'Phan'
WHERE employeeNumber = 1056
```

Assume the lastName before update is 'Chan'.

- To check if the trigger was invoked by the UPDATE statement, we can query the `employees_audit` table by using the following query.

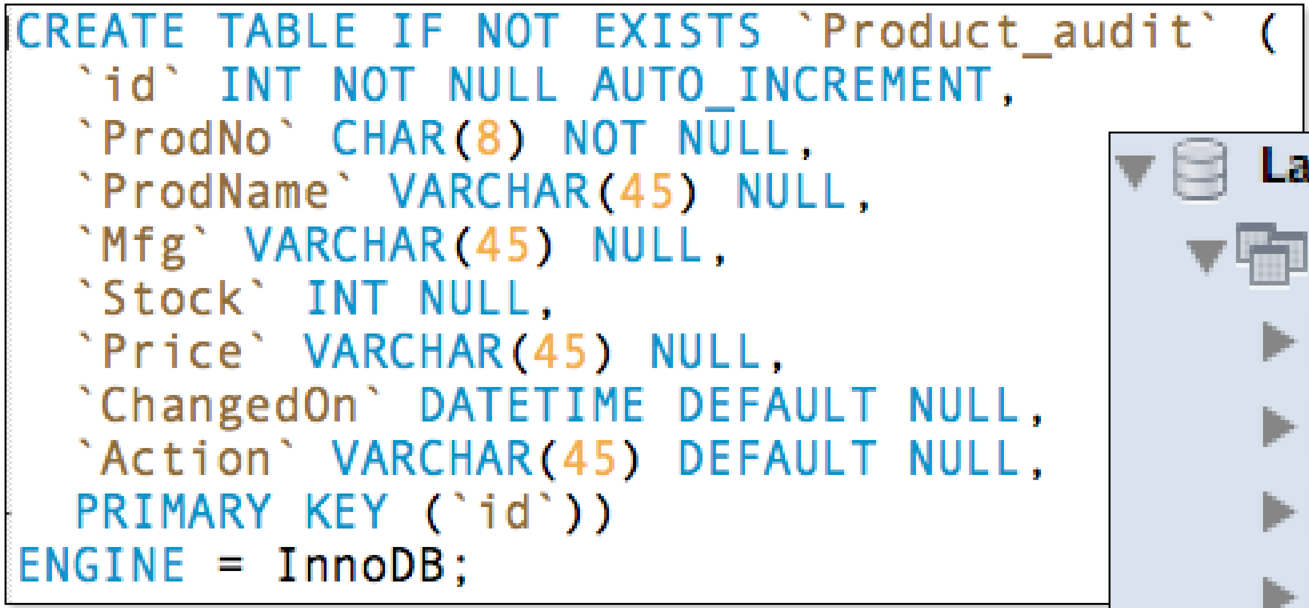
```
SELECT *
FROM employees_audit
```



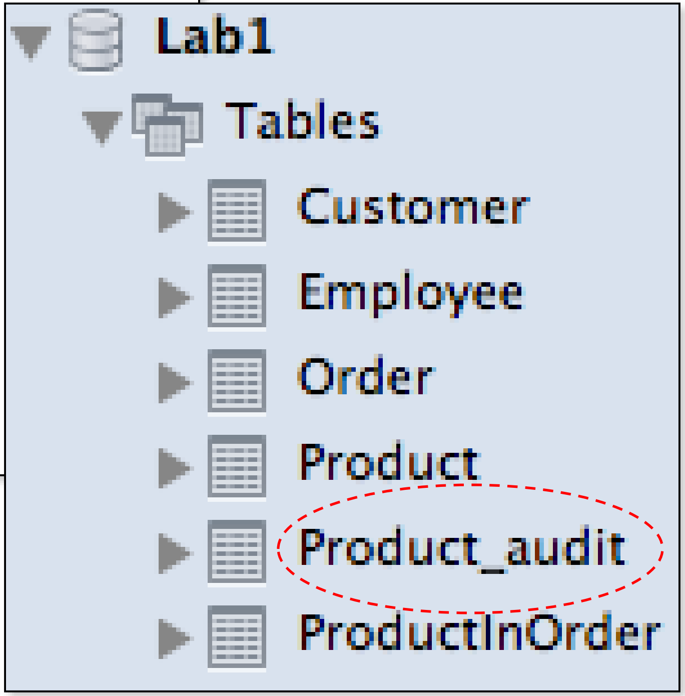
id	employeeNumber	lastname	changedon	action

Example 2: Trigger for Lab1

- Create a trigger for the `Product` table in Lab1.
- First, we create a `Product_audit` table



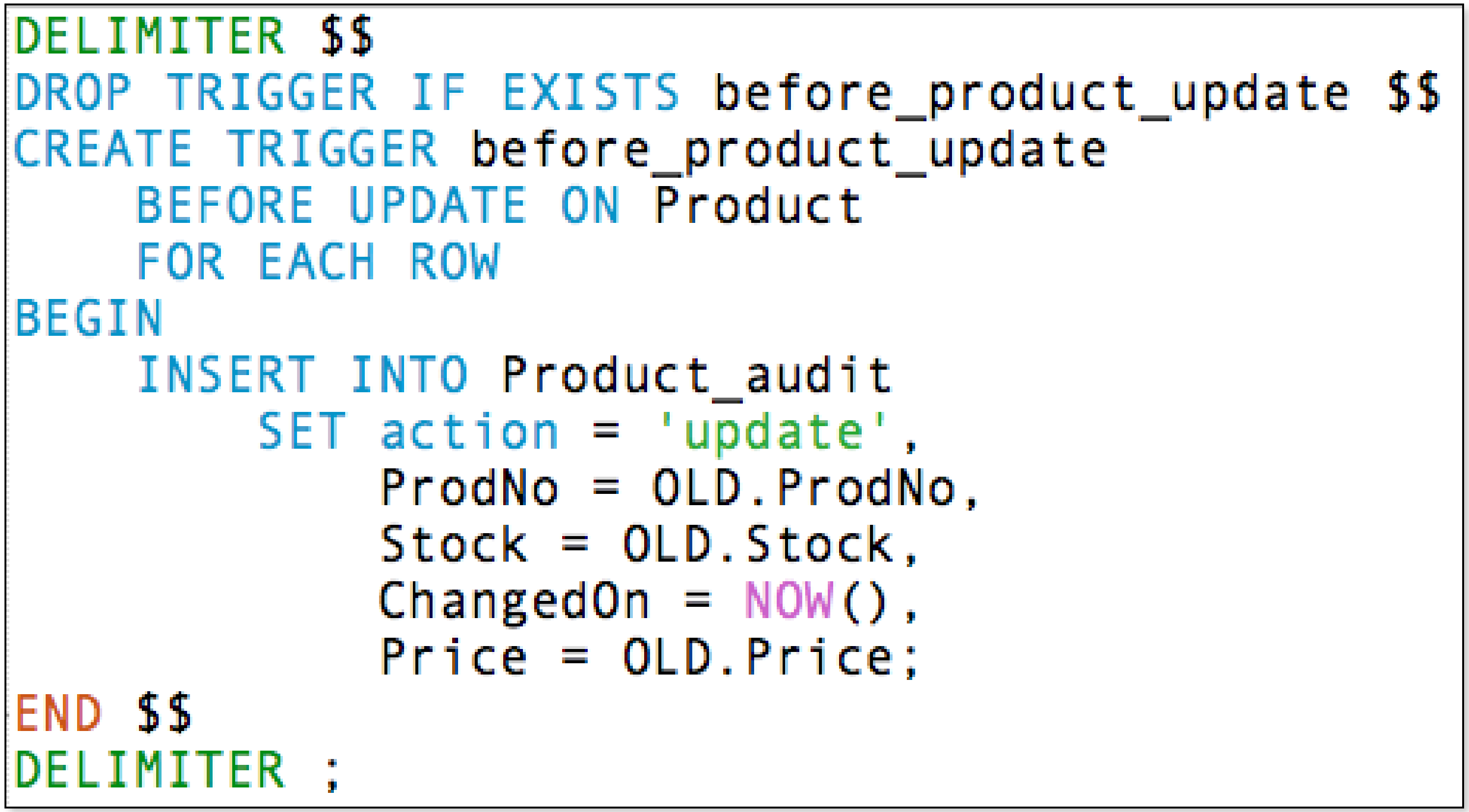
```
CREATE TABLE IF NOT EXISTS `Product_audit` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `ProdNo` CHAR(8) NOT NULL,
  `ProdName` VARCHAR(45) NULL,
  `Mfg` VARCHAR(45) NULL,
  `Stock` INT NULL,
  `Price` VARCHAR(45) NULL,
  `ChangedOn` DATETIME DEFAULT NULL,
  `Action` VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
```



```
Lab1
  Tables
    Customer
    Employee
    Order
    Product
    Product_audit
    ProductInOrder
```

Example 2: Trigger for Lab1

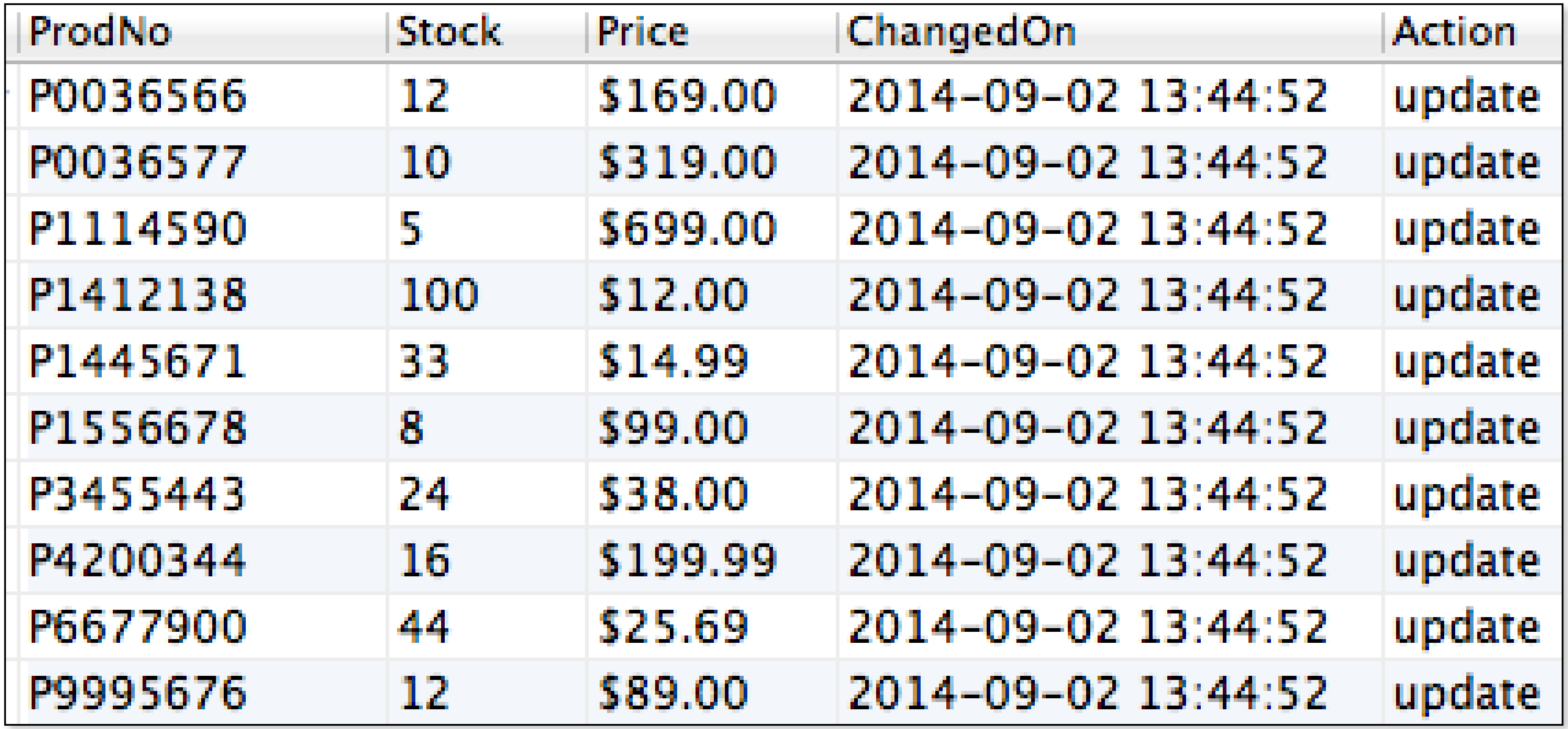
- Create a trigger for the `Product` table in Lab1.



```
DELIMITER $$
DROP TRIGGER IF EXISTS before_product_update $$
CREATE TRIGGER before_product_update
    BEFORE UPDATE ON Product
    FOR EACH ROW
BEGIN
    INSERT INTO Product_audit
        SET action = 'update',
            ProdNo = OLD.ProdNo,
            Stock = OLD.Stock,
            ChangedOn = NOW(),
            Price = OLD.Price;
END $$
DELIMITER ;
```

Example 2: Trigger for Lab1

- Execute the Cursor (`Cursor.sql`) we defined earlier.
- Then, list the content of `` `Product_audit` `` table



ProdNo	Stock	Price	ChangedOn	Action
P0036566	12	$169.00	2014-09-02 13:44:52	update
P0036577	10	$319.00	2014-09-02 13:44:52	update
P1114590	5	$699.00	2014-09-02 13:44:52	update
P1412138	100	$12.00	2014-09-02 13:44:52	update
P1445671	33	$14.99	2014-09-02 13:44:52	update
P1556678	8	$99.00	2014-09-02 13:44:52	update
P3455443	24	$38.00	2014-09-02 13:44:52	update
P4200344	16	$199.99	2014-09-02 13:44:52	update
P6677900	44	$25.69	2014-09-02 13:44:52	update
P9995676	12	$89.00	2014-09-02 13:44:52	update

Summary

- Stored Procedure: Add procedural programming constructs to SQL.
- Trigger: Very useful for checking data and for ensure data integrity.
- Cursor: Useful for accessing result sets one row at a time.
- Exception: Good for handling error caused by SQL operations