|  |
| --- |
| Department of Software Engineering  Mehran University of Engineering and Technology, Jamshoro |

|  |  |  |  |
| --- | --- | --- | --- |
| Course: SWE324 - Data Warehousing and Data Mining | | | |
| Instructor | Rabeea Jaffari | **Practical/Lab No.** | 04 |
| Date |  | **CLOs** | CLO-4: P3 & P4 |
| Signature |  | **Assessment Score** | 1 Mark |

|  |  |
| --- | --- |
| Topic | To familiar with OLTP system reporting |
| Objectives | * To learn report generation in OLTP systems |

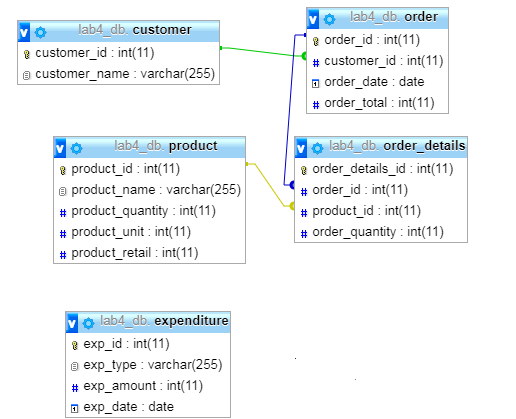
|  |
| --- |
| Lab Discussion: Theoretical concepts and Procedural steps |

**OLTP SYSTEM REPORTS:**  An OLTP systemreport is the formatted result of OLTP system queries and contains useful data for decision-making and analysis and is usually used by managers to get a high-level summary from the transactional data. Most good business applications contain a built-in reporting tool; this is simply a front-end interface that calls or runs back-end OLTP queries that are formatted for easy application usage. However, in case of absence of such tool, one can create his/her own reports by writing all the desired queries, grouping them together in a stored procedure (so that all of them can be executed immediately together) and displaying the returned results in a table. The user can then process the tabular data in other ways (filter/sort, create charts, export to Excel, crosstab, etc.)

**TOOL:** Although GUI based reporting tools available with most OLTP systems (DBMSs) can be used, this lab focuses on creating customized reports from queries in stored procedures and generating tabular results from them to be displayed in the reports.   
**SQLYog** MYSQL DMBS will be used to achieve the task.

**REPORTING SCENARIO:**

Suppose you are the owner of a sales business (a shop for instance) and you have been given the task to formulate a monthly report of your business and deliver it to the manager. The ERD for the sales business scenario is as follows:



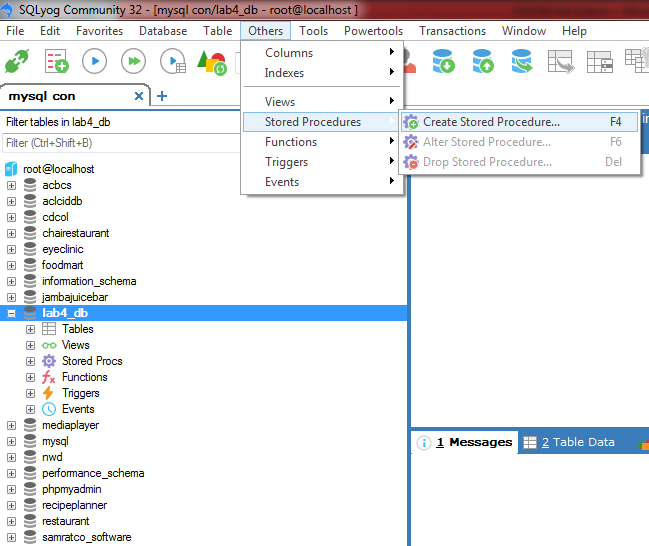
A description of the scenario is as follows:

Customers place orders at a specified date, an order may encompass several products and their prescribed quantities along with the order\_total and order\_date. A product is purchased from any supplier (not included here) using “product\_unit” price and is sold to the customer using the “product\_retail” price.

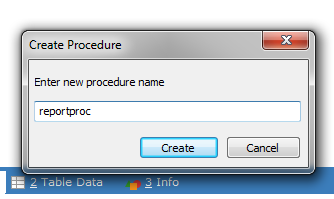
Apart from the product purchasing, the sales business also handles some other expenditures such as transport charges, electricity bills and so on.

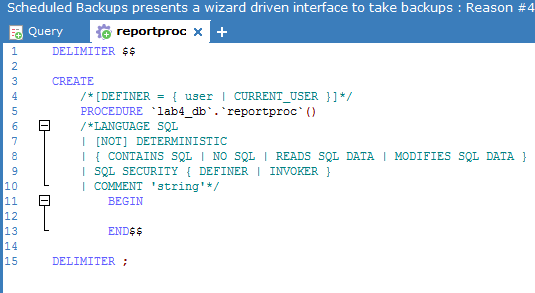
**STORED PROCEDURE:** A stored procedure (also termed proc, storp, sproc, StoPro, StoredProc, StoreProc, sp, or SP) is a set of [Structured Query Language (SQL)](https://searchsqlserver.techtarget.com/definition/SQL) statements with an assigned name, which is stored in an OLTP system as a group, so it can be reused and shared by multiple programs. These stored procedures can be useful when creating customized reports, as they can house all the queries, store them in variables and then generate tabular results from them. To create a sample stored procedure in SQLYOG to calculate total expenditure for a month using the above scenario, follow the steps below:

1. Select the database for which you want to create a stored procedure, click “Others” menu on the task bar and then within the “Stored procedures” category click “Create new stored procedure”.

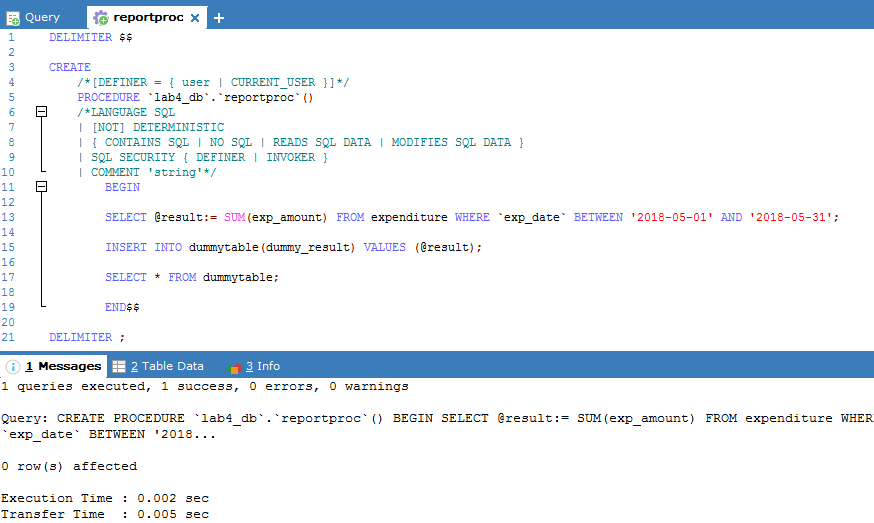


1. Give any suitable name to your stored procedure and a default stored procedure template will be created for you as shown in the figure below:



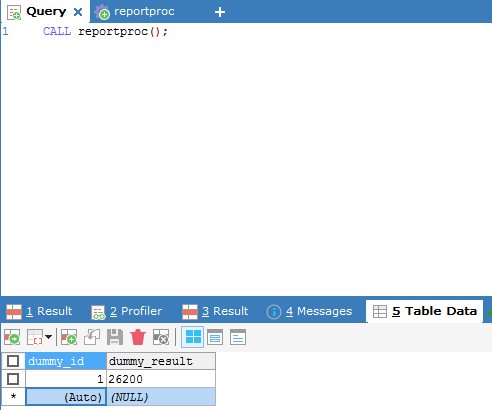


1. Begin writing all your queries within the “BEGIN” and “END” section of the stored procedure. To save the result of the query in a variable (use @ notation for a variable), use the SELECT @varname:= query notation shown below:



It can be seen from the screenshot above that the stored procedure first calculates the total expenditure of the month for the sales business (by using the SUM group function), stores it in the “@result” variable, passes the variable as a value to be inserted in the temporary report “dummy table” to show the results and then finally displays the result from the dummy table using SELECT statement at last.

1. This stored procedure can be called using a simple “CALL reportproc()” syntax to execute all the statements together and display the final tabular results.



|  |
| --- |
| Lab Tasks |
| Submission Date: 23-04-19 |

Generate an OLTP system report which displays the following results:

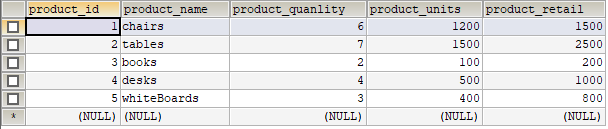
1. **Total sales in a month** (Use only order table).
2. **Profit/loss in a month** (Use product (to account for purchasing costs), expenditure as well as order tables).
3. **Highest selling product of the month** (By highest sold quantity)
4. **Lowest selling product of the month** (By lowest sold quantity)

**Hint:** Create a report table to hold all the above results from queries after they are executed in the stored procedure and then create a stored procedure in the similar manner as shown above.

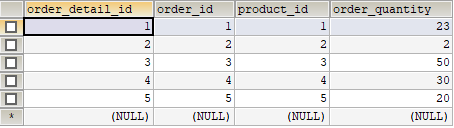
**Solution:**

**Tables in sqlyog:**

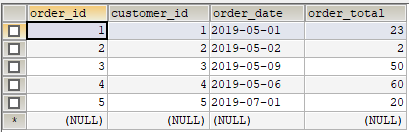
**Product table:**

****

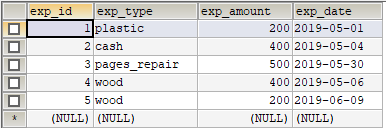
**Order\_details table:**

****

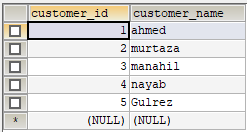
**Order table:**

****

**Expenditure table:**

****

**Customer table**

****

**Executed Queries for all above tasks:**

1)SELECT SUM(exp\_amount) FROM expenditure WHERE `exp\_date` LIKE '2019-05-01' AND '2019-05-30';



2)SELECT SUM(order\_total) AS total\_sale FROM `order` WHERE `order\_date` LIKE '2019-05%';



3)SELECT (SUM(product.product\_retail)-(SUM(product.product\_units)+SUM(expenditure.exp\_amount))) AS Profit\_Loss FROM

product product INNER JOIN expenditure expenditure ON product.product\_id= expenditure.exp\_id WHERE expenditure.exp\_date IN

(SELECT order.order\_date FROM `order` `order` HAVING order.order\_date BETWEEN '2019-05-01' AND '2019-05-30');



4)SELECT product.product\_name FROM product product INNER JOIN order\_details order\_details ON

product.product\_id=(SELECT order\_details.product\_id WHERE order\_details.order\_quantity=

(SELECT MAX(order\_details.order\_quantity) FROM order\_details order\_details));



5)SELECT product.product\_name FROM product product INNER JOIN order\_details order\_details ON

product.product\_id=(SELECT order\_details.product\_id WHERE order\_details.order\_quantity=

(SELECT MIN(order\_details.order\_quantity) FROM order\_details order\_details))

INSERT INTO result(`type`) VALUES ('highest\_sale\_product'),('lowest\_sale\_product');



**Report of all above task:**

DELIMITER $$

CREATE

/\*[DEFINER = { user | CURRENT\_USER }]\*/

PROCEDURE `reportdb`.`report`()

/\*LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

| COMMENT 'string'\*/

BEGIN

SELECT @total\_expend:=SUM(exp\_amount) AS total\_expend FROM expenditure WHERE `exp\_date` LIKE '2019-05-01' AND '2019-05-30';

INSERT INTO result(total\_expend) VALUES (@total\_expend);

SELECT @total\_sale:=SUM(order\_total) AS total\_sale FROM `order` WHERE `order\_date` LIKE '2019-05%';

INSERT INTO result(total\_sale) VALUES (@total\_sale);

SELECT @profit\_loss:=(SUM(product.product\_retail)-(SUM(product.product\_units)+SUM(expenditure.exp\_amount))) AS Profit\_Loss FROM

product product INNER JOIN expenditure expenditure ON product.product\_id= expenditure.exp\_id WHERE expenditure.exp\_date IN

(SELECT order.order\_date FROM `order` `order` HAVING order.order\_date BETWEEN '2019-05-01' AND '2019-05-30');

INSERT INTO result(profit\_loss) VALUES (@profit\_loss);

SELECT \* FROM result;

SELECT @low\_sale\_product:=product.product\_name FROM product product INNER JOIN order\_details order\_details ON

product.product\_id=(SELECT order\_details.product\_id WHERE order\_details.order\_quantity=(SELECT MIN(order\_details.order\_quantity)

FROM order\_details order\_details));

INSERT INTO result(lowest\_sale\_product) VALUES (@low\_sale\_product);

SELECT \* FROM result;

SELECT @high\_sale\_product:=product.product\_name FROM product product INNER JOIN order\_details order\_details ON

product.product\_id=(SELECT order\_details.product\_id WHERE order\_details.order\_quantity=

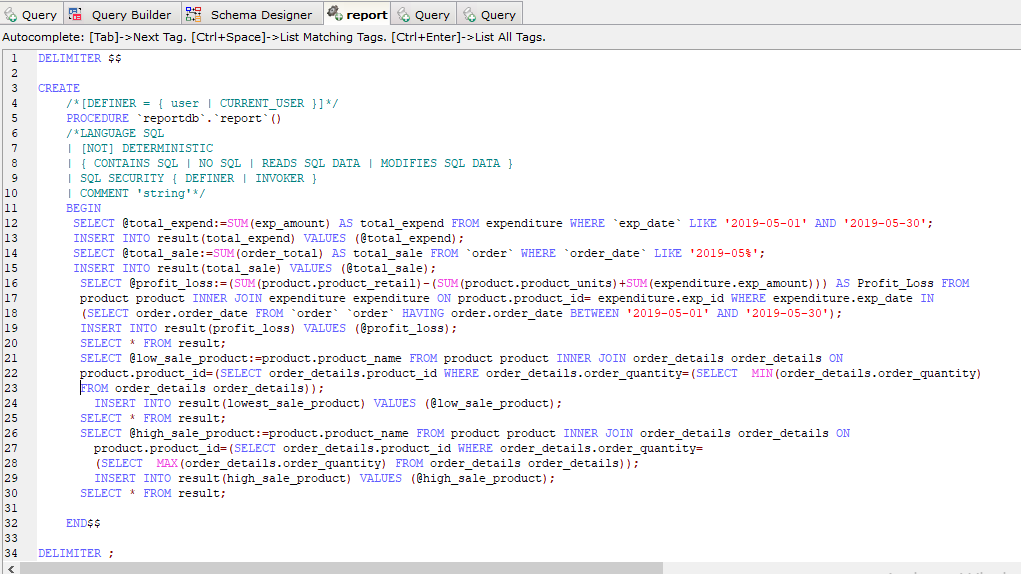
(SELECT MAX(order\_details.order\_quantity) FROM order\_details order\_details));

INSERT INTO result(high\_sale\_product) VALUES (@high\_sale\_product);

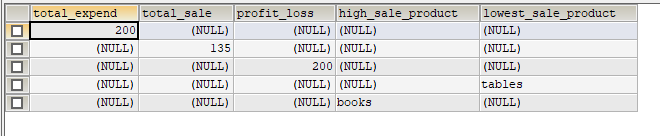
SELECT \* FROM result;

END$$

DELIMITER ;

****

**Result:**

****