Lab Manual on Stored Routines and Views (July 2020)

Allan O. Omondi,
Doctoral Fellow, Faculty of Information Technology,
Strathmore University, Nairobi, Kenya
aomondi [at] strathmore.edu

Objectives

By the end of this lab, you will be able to:

1. **Differentiate** between a traditional programming approach and a data science approach
2. **Differentiate** between data science and business intelligence
3. **Implement** an algorithm using SQL
4. **Create, view, and delete** a procedure
5. **Create, view, and delete** a function
6. **Create** and **delete** a view
7. **Invoke** a procedure
8. **Invoke** a function
9. **Use** an SQL view

Tools

1. MySQL DBMS (>=MySQL DBMS 8.0)
2. MySQL Workbench (>=MySQL Workbench 8.0)

Approximate Time Required

1 hour 45 minutes

Prerequisites

1. You should have installed MySQL DBMS (>= MySQL 8.0) and MySQL Workbench (>=MySQL Workbench 8.0). They are available via this link.
2. You should have imported the sample database called "classicmodels" into your database system. It is available via this link.

Narrative:

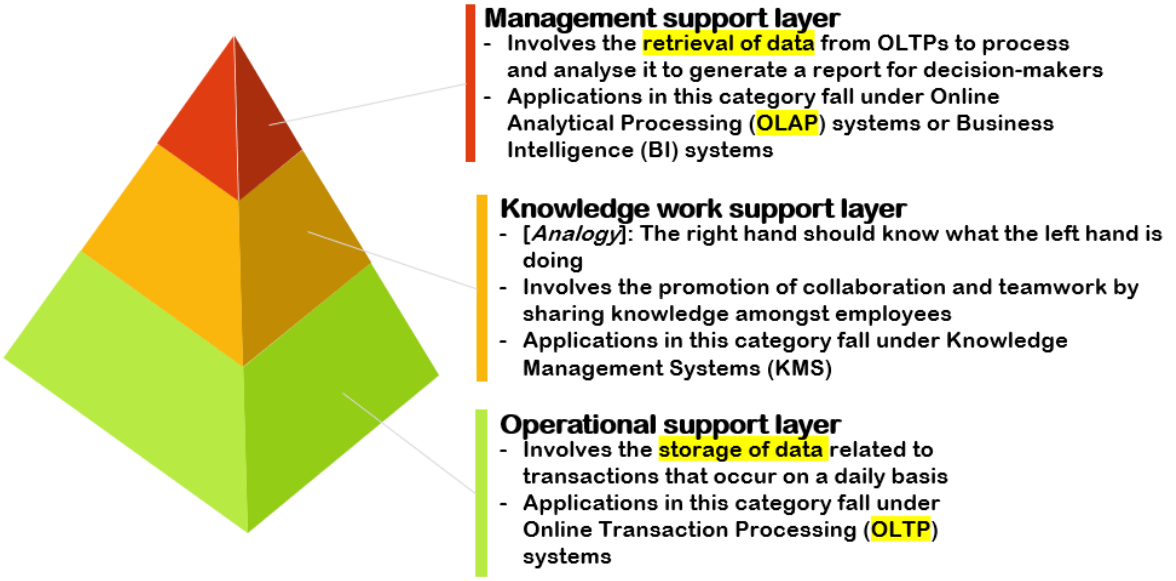
The "classicmodels" sample database contains data of a retail business. The retail business, in this case, is engaged in the sale of models of classic cars. It contains typical business data such as customers, products, product lines (categories of products), orders, order line items (details of an order), payments received, employees, and branch details.

A model of a P-51-D Mustang similar to product code "S18_2581" in the products relation

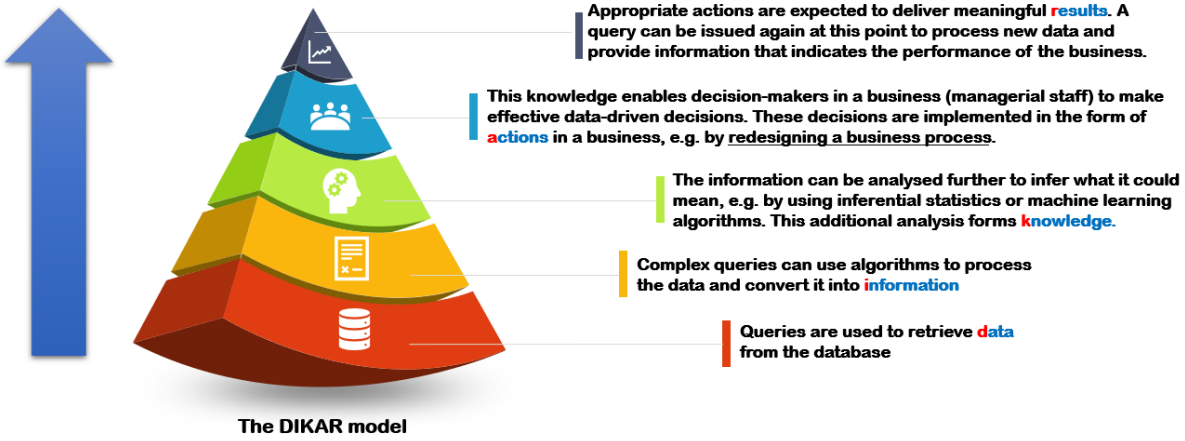
Recommended citation:
Omondi, A. O. (2020). *BBT 3104 – Advanced Database Systems – Lab Manual on Stored Routines and Views [PDF Lab Manual]*. Retrieved from Strathmore University eLearning website https://elearning.strathmore.edu/pluginfile.php/227099/mod_resource/content/2/5.a.Concept5of6-LabManual-StoredRoutinesAndViews.pdf

The Data, Information, Knowledge, Action, Result (DIKAR) Model

The storage of data and the retrieval of data are key functionalities that are required in the operational support and management support layers of a business information system respectively.



A DBA can create a complex query to retrieve data required to generate a report. This report may be required on a **routine** basis, e.g. at the end of every week or at the end of every quarter in the financial year. It is common to encounter the terms "End of Day Report", "End of Week Report", "End of Month Report", "End of Quarter Report", and "End of Year Report" in businesses. These reports are designed to retrieve **data** from the database, which is then processed to form **information**, then analysed further to form **knowledge**. The knowledge enables decision-makers in the business (e.g. managerial staff) to make effective data-driven decisions which are implemented in the form of **actions** in a business. Appropriate actions are expected to deliver meaningful **results**. This is known as the DIKAR model in the fields of business strategy and information management.



The DIKAR model

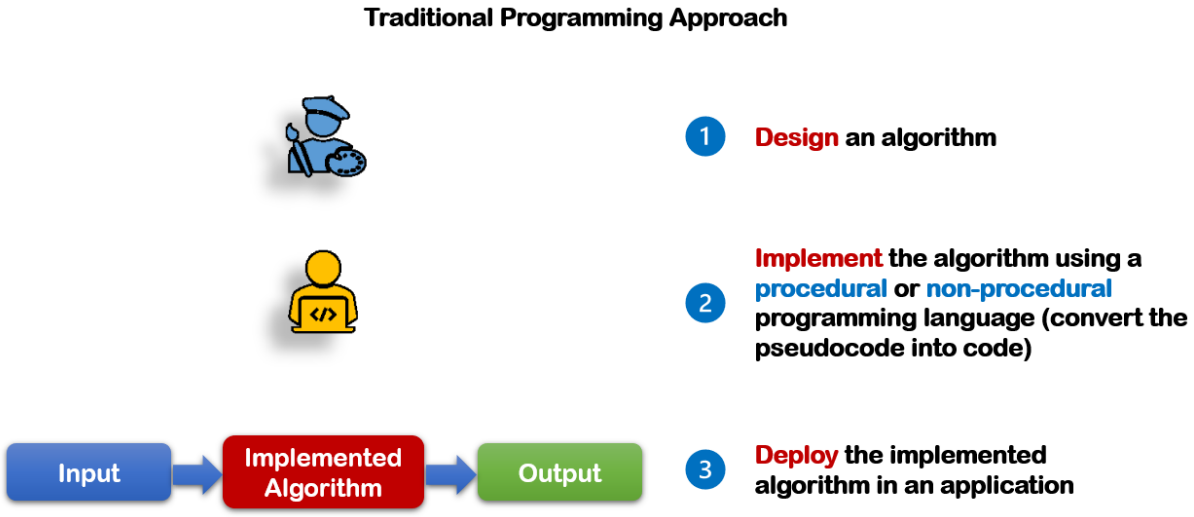
MySQL DBMS allows a DBA to store complex queries inside the database. Members of the database team can then edit or execute the complex query that is stored in a central location in the database whenever the data needs to be retrieved using the query. The database object that is used to store the complex query is called a **stored routine**. A stored routine can in turn be in the form of either a **procedure** or a **function.**

Traditional Programming Approach

Traditional programming can be grouped into either procedural approaches or non-procedural approaches. It is nowadays common to find traditional programming approaches being supplemented (not substituted) by data science which applies Machine Learning (ML). For example, a business can use a traditional programming approach to develop an e-Commerce web-based application (operational support layer) and supplement this with data science (management support layer). The data science component can in turn be used to design predictive models that can inform the business that if a customer sees product X near product Y on the e-Commerce application, then there is a high probability that they will buy both product X and product Y. This essentially associates product X to product Y.

Data science can, therefore, be used when a traditional programming approach is inadequate to fully implement a desired task. **Given the supplementary role that data science plays, then it is still important to learn about the traditional programming approach which relies on a relational database to store structured data.**

Traditional programming involves a sequence of instructions and control flow functions (if-else, do-while, case, etc.) that determine which instruction is executed when. In some informal contexts, this setup is referred to as an "if-this-then-that" (ifttt) setup. The following diagram shows the key steps involved.



Data Science Approach

Data science involves the execution of tasks that can analyse information further to convert it into knowledge. This knowledge can be disseminated at the knowledge work support layer and used to make decisions at the management support layer of business information systems. Examples of some of the analytical tasks that are relevant to a business include:

(i) **Clustering**: Identification of groups, e.g. market segmentation. An algorithm that can be used is the k-Means clustering algorithm. For example, a business can segment the East African market into smaller segments that enable it to customize their adverts and products to suit the needs of that smaller market segment, e.g. BlueBand margarine, produced by Unilever, can be customized to have a higher melting point for customers in Mtito Andei town (Makueni County) and a lower melting point for customers in Limuru town (Kiambu County). These groups can be identified by analysing historical data that shows a common trend amongst a group (cluster) of clients.

(ii) **Classification**: Grouping items into discrete classes and predicting which class an item belongs to. For example, understanding customers can enable a business to conduct direct marketing to a group of clients that have been classified as being in the group that has a high probability of purchasing the product being marketed to them. Note that classification is **predictive** unlike clustering which is **descriptive**. Algorithms that can be used include the Naïve Bayes, adaptive Bayes network, and the Support Vector Machine algorithm.

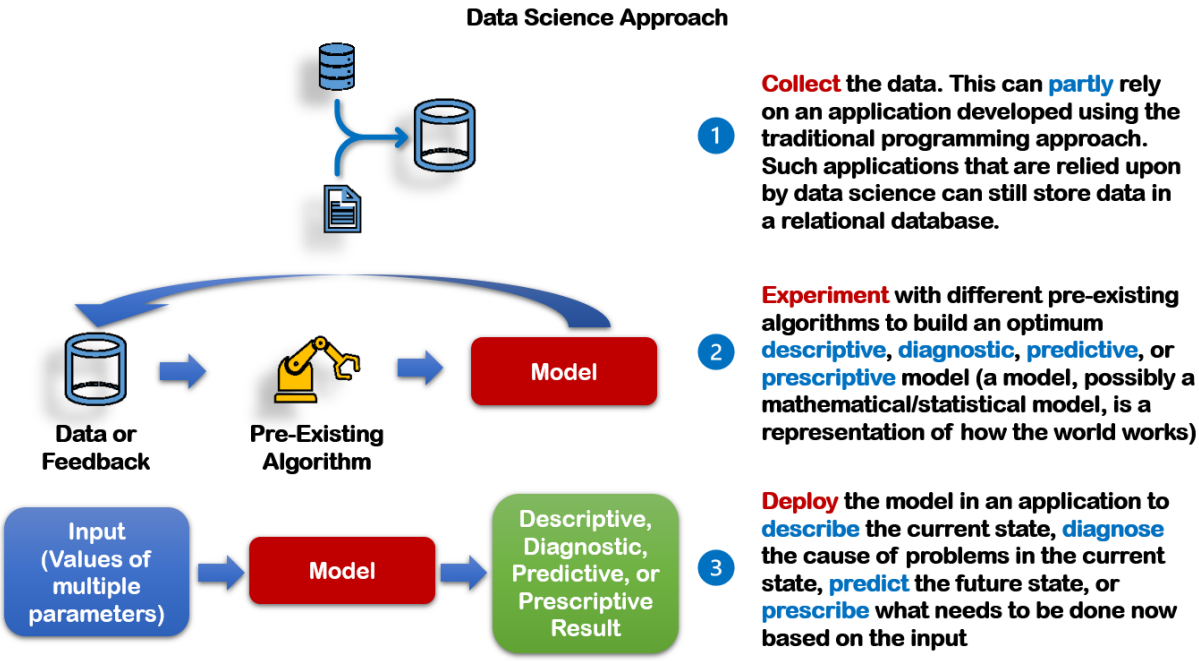
(iii) **Association**: Looking for relationships between variables, analysing "market baskets" or customer buying patterns, and identifying items that are likely to be purchased together, for example tomatoes and onions. The business can decide to place these products side by side to encourage the client to buy both. Algorithms that can be used to identify associations include A Priori, Eclat, and FP growth

Clustering, classification, and association require the **consideration of multiple variables**. For example, it is mandatory for multi-national companies to consider details like yesterday's exchange rate; external and internal economic changes in the country that issues the currency, among other variables, when predicting the foreign exchange rate (FOREX). The number of variables to consider can be in the **hundreds** and this can be overwhelming for a traditional programming approach that relies on structured data in normalized relations inside a relational database (a database that applies a relational data model). An if-this-then-that kind of setup, therefore, may not be able to consider all the hundreds of variables accurately. The data science approach offers a different perspective that does not rely on structured data in a relational database and on designing a customized algorithm. The following table shows other data models that can be used in a database apart from the relational data model.

Data Models

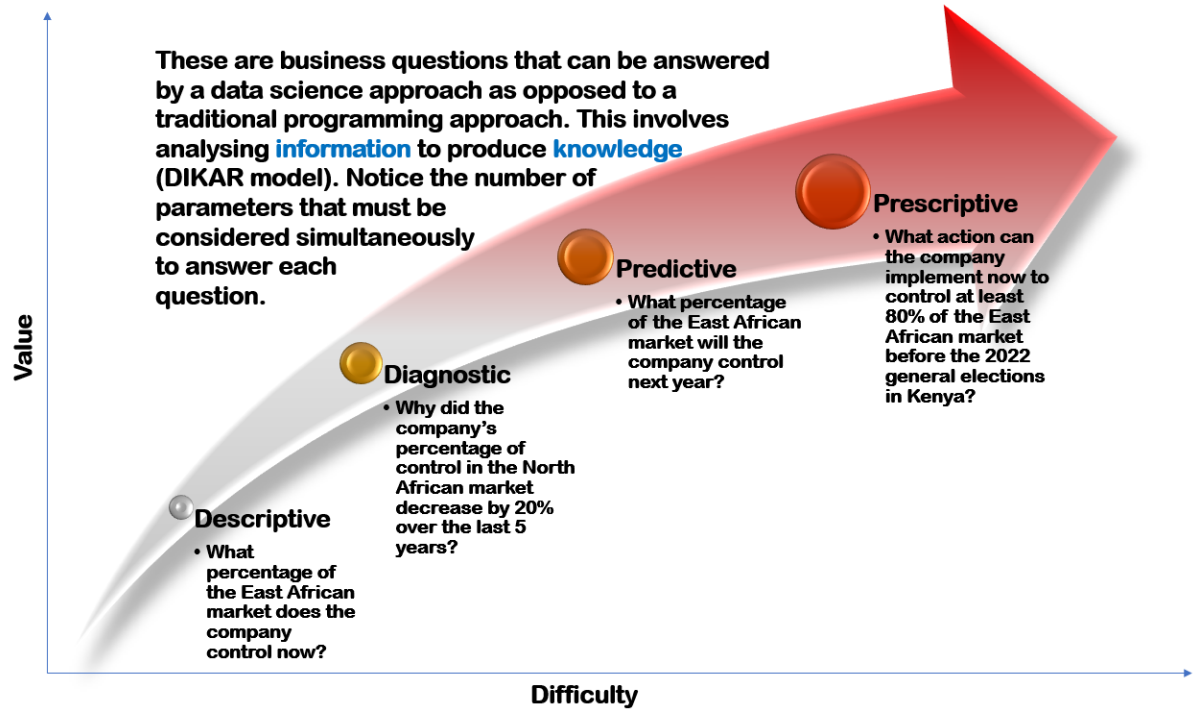
Data Model	Comment
Relational	Used in relational databases e.g. MySQL, Oracle, and Microsoft SQL Server. It is used to store structured data used by applications developed based on the traditional programming approach.
Key-Value	They are used to store unstructured and semi-structured data (data that is not well organized into numerous, distinct, and normalized relations that have rows and columns to organize the data). It is common to find these data models in NoSQL databases used in the analysis of big data and in real-time applications (which started off as web 2.0). Examples of NoSQL databases include: Couchbase Server, CouchDB, Neo4j, MongoDB, Apache Cassandra, and many more.
Graph	
Document	
Column-Family	
Array/Matrix	Provides an input to artificial intelligence and machine learning workloads
Hierarchical	Considered to be obsolete and are currently rare to find in a production server
Network	

The following diagram summarizes the data science approach.



The data science approach provides a different way to address the challenge of numerous variables. It requires significant amounts of historical data that is then given as input to existing algorithms (machine learning algorithms) e.g. k-Means clustering algorithm, Naïve Bayes, adaptive Bayes network, the Support Vector Machine algorithm, A Priori, Eclat, FP growth, amongst many other options. Here is a list of more algorithms: https://en.wikipedia.org/wiki/List_of_algorithms. This results in a model that can be used to process new and unforeseen data. The data scientist can then input the new data into the model (as opposed to an algorithm) which can then consider numerous variables simultaneously and produce a result.

Difference Between Business Intelligence and Data Science



The term "data science", as used in business, involves an interdisciplinary approach to make inferences from available business data. This data is usually in large quantities with hundreds or thousands of variables (columns) and complex. On the other hand, Business Intelligence (BI) helps to monitor the current state of business data to understand the historical performance of a business. While BI helps to interpret past data, data science can analyse past data to make predictions and prescriptions. BI is, therefore, mainly used for descriptive or diagnostic analytics whereas data science, which can handle many variables simultaneously, is mainly used for predictive or prescriptive analytics. Those who will choose the BI option, as opposed to the networking option, in BBIT 4th year will learn more about Business Intelligence.

SQL as a Non-Procedural Programming Language

SQL is a non-procedural (declarative) programming language. This means that it specifies what is to be done rather than how to do it. A DBA, therefore, uses SQL as a tool to express what is desired in terms of what is known, e.g. if it is known that the database has a relation that stores client's data, then a DBA can specify that he/she wants a report on the list of clients that are being served by a specific sales representative.

This prevents the customization of instructions on how a task is to be executed, therefore, SQL needs to be supplemented by a procedural programming language that can then implement the traditional programming approach. Those interested in a recap of different types of computer programming languages can refer to the online version of the Britannica Encyclopaedia on this link: https://www.britannica.com/technology/computer-programming-language Also, SQL does not offer an environment to test machine learning algorithms that can be used to build models. **These factors limit SQL to being used to support Business Intelligence applications in a traditional programming approach as opposed to being used for data science in a data science approach.**

This lab manual focuses on creating, listing, and deleting, and using stored routines (procedures and functions). The stored routines can eventually be used to retrieve data required for descriptive and diagnostic analytics in Business Intelligence (BBIT 4th year option). An even higher level of analytics (predictive and prescriptive analytics) can be implemented in data science based on data that can be retrieved using stored routines. Your ability to retrieve data using stored routines, therefore, forms a fundamental foundation for Business Intelligence (BI) and possibly data science in future. This knowledge is required to develop business information systems at the management support layer.

Advantages of Procedures

(i) **Reduce network traffic:** Stored procedures help to reduce the network traffic between applications and the database system because instead of sending multiple lengthy SQL statements, applications can send only the name and parameters of stored procedures.

(ii) **Centralize business logic in the database:** You can use the stored procedures to implement business logic that is reusable by multiple applications. The stored procedures help reduce the efforts of duplicating the same logic in many applications.

(iii) **Make the database more secure:** The database administrator can grant appropriate privileges to applications to allow them to access specific stored procedures without giving any privileges on the underlying tables. This implies that the application has a personalized account to connect to the database (not the root account).

Suppose that the business strategy committee at Classic Models identifies an **attractive market opportunity** in East Africa. This committee then retrieves historical data to conduct a SWOT analysis, that is, an analysis which aims to identify the Strengths, Weaknesses, Opportunities, and Threats that a business faces. The results of this analysis indicate that the business has the right **resources, competencies,** and **capabilities** to venture into the East African market. They also note that the business can make the right **compromises** that can enable it to venture into this new market. A decision is made by the Board of Directors (BOD) as informed by the business strategy committee. The decision is to **execute** the entry of Classic Models into the East African market with a lot of **discipline** and determination to succeed. They decide to setup their East African office in Nairobi.

STEP 1. Create a record for the Nairobi office

Execute the following command to create a record for the Nairobi office:

```
INSERT INTO `classicmodels`.`offices` (`officeCode`, `city`, `phone`, `addressLine1`, `addressLine2`, `country`, `postalCode`, `territory`) VALUES ('8', 'Nairobi', '+254 720 123 456', '16 5th Ngong Avenue, Upperhill', '5th Floor, Example Plaza', 'Kenya', '00202 KNH', 'EMEA');
```

STEP 2. Create records for 5 sales representatives who work in the Nairobi office

They then recruit 5 sales representatives to work in the Nairobi Office. Execute the following command to create records of the 5 employees:

```
INSERT INTO `classicmodels`.`employees` (`employeeNumber`, `lastName`, `firstName`, `extension`, `email`, `officeCode`, `reportsTo`, `jobTitle`) VALUES ('1703', 'Kiprono', 'John', 'x211', 'jkiprono@classicmodelcars.com', '8', '1056', 'Sales Rep');
INSERT INTO `classicmodels`.`employees` (`employeeNumber`, `lastName`, `firstName`, `extension`, `email`, `officeCode`, `reportsTo`, `jobTitle`) VALUES ('1704', 'Naliaka', 'Mary', 'x212', 'mnaliaka@classicmodelcars.com', '8', '1056', 'Sales Rep');
INSERT INTO `classicmodels`.`employees` (`employeeNumber`, `lastName`, `firstName`, `extension`, `email`, `officeCode`, `reportsTo`, `jobTitle`) VALUES ('1705', 'Mogaka', 'Andrew', 'x213', 'amogaka@classicmodelcars.com', '8', '1056', 'Sales Rep');
INSERT INTO `classicmodels`.`employees` (`employeeNumber`, `lastName`, `firstName`, `extension`, `email`, `officeCode`, `reportsTo`, `jobTitle`) VALUES ('1706', 'Shanyisa', 'Esther', 'x214', 'eshanyisa@classicmodelcars.com', '8', '1056', 'Sales Rep');
INSERT INTO `classicmodels`.`employees` (`employeeNumber`, `lastName`, `firstName`, `extension`, `email`, `officeCode`, `reportsTo`, `jobTitle`) VALUES ('1707', 'Oyier', 'Joshua', 'x215', 'joyier@classicmodelcars.com', '8', '1056', 'Sales Rep');
```

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1703	Kiprono	John	x211	jkiprono@classicmodelcars.com	8	1056	Sales Rep
1704	Naliaka	Mary	x212	mnaliaka@classicmodelcars.com	8	1056	Sales Rep
1705	Mogaka	Andrew	x213	amogaka@classicmodelcars.com	8	1056	Sales Rep
1706	Shanyisa	Esther	x214	eshanyisa@classicmodelcars.com	8	1056	Sales Rep
1707	Oyier	Joshua	x215	joyier@classicmodelcars.com	8	1056	Sales Rep

STEP 3. Create a relation to store the salaries of the employees in the Nairobi office

The Nairobi office requires a place to store the remuneration amount (salary) of each of its 5 employees. Create a table called “employees_salary” to record the amount of salary for each employee. This can be done by executing the following command:

```
CREATE TABLE `employees_salary` (
```

```
  `employeeNumber` int NOT NULL,
  `employees_salary_amount` decimal(9,2) NOT NULL DEFAULT '0.00',
  PRIMARY KEY (`employeeNumber`),
  UNIQUE KEY `IDX_employeeNumber_UNIQUE` (`employeeNumber`),
  CONSTRAINT `FK_1_employees_salary_TO_1_employees` FOREIGN KEY
(`employeeNumber`) REFERENCES `employees` (`employeeNumber`)
) ENGINE=InnoDB COMMENT='Records the salary of each employee'
```

STEP 4. Record the salary of each employee working in the Nairobi office

Execute the following statements to record the salary for each of the 5 employees based in Nairobi:

```
INSERT INTO `classicmodels`.`employees_salary` (`employeeNumber`,
`employees_salary_amount`) VALUES ('1703', '11000');
INSERT INTO `classicmodels`.`employees_salary` (`employeeNumber`,
`employees_salary_amount`) VALUES ('1704', '20000');
INSERT INTO `classicmodels`.`employees_salary` (`employeeNumber`,
`employees_salary_amount`) VALUES ('1705', '34000');
INSERT INTO `classicmodels`.`employees_salary` (`employeeNumber`,
`employees_salary_amount`) VALUES ('1706', '38500');
INSERT INTO `classicmodels`.`employees_salary` (`employeeNumber`,
`employees_salary_amount`) VALUES ('1707', '65000');
```

employeeNumber	employees_salary_amount
1703	11000.00
1704	20000.00
1705	34000.00
1706	38500.00
1707	65000.00

SUPPOSE that the Payroll Office has requested for your services as a Database Administrator to compute the individual income tax for each employee. This computation is based on the pre-determined Pay-As-You-Earn (PAYE) formula that is already being used manually by the Payroll Office. PAYE is the process by which a government (through its Revenue Authority) collects employees' income tax directly from the employer. It is the employer's statutory duty to deduct income tax from the remuneration of all employees who earn a monthly salary of more than KES. 11,135 in Kenya. In this scenario, the multi-national company (classic models) deducts income tax depending on the government rules, e.g. the residential employees (citizens) who work in Kenya may pay a different income tax from the residential employees who work in say, France or in the company's Japan office. The Government of Kenya stipulates that the following tax rates should be applied:

Monthly Taxable Income Range (KES)	**Taxable Income (KES)** *(i.e. the total amount in this income range)*	**Tax Rate (%)**	**Income Tax Charged**
0 - 12,298.99	$12{,}298 - 0 = 12{,}298.99$	10	$10\% \times 12{,}298.99$
12,299 - 23,885.99	$23{,}885.99 - 12{,}299 = 11{,}586.99$	15	$15\% \times 11{,}586.99$
23,886 - 35,472.99	$35{,}472.99 - 28{,}886 = 11{,}586.99$	20	$20\% \times 11{,}586.99$
35,473 - 47,059.99	$47{,}059.99 - 35{,}473 = 11{,}586.99$	25	$25\% \times 11{,}586.99$
47,060 and above	$< total\ taxable\ income > - 47{,}060 = x$	30	$30\% \times x$

For example, if an employee's taxable income is KES. 38,500.00 per month, then the total income tax the employee is expected to pay to the government is KES. 6,042.00 per month. This is computed as follows:

Monthly Taxable Income Range (KES)	Taxable Income (KES)	Tax Rate (%)	Income Tax Charged
0 - 12,298.99	12,298.99	10	1,229.80
12,299 - 23,885.99	11,586.99	15	1,738.05
23,886 - 35,472.99	11,586.99	20	2,317.40
35,473 - 47,059.99	3,027	25	756.75
47,060 and above	-	30	-
TOTAL	**≈38,500**		**6,042.00**

Through meetings between the Payroll Office and the IT Department's database team, the following algorithm has been stipulated as the formula used to calculate an individual employee's PAYE income tax in the Nairobi office.

Algorithm 1: PAYE (income tax) for employees working in the Kenyan office

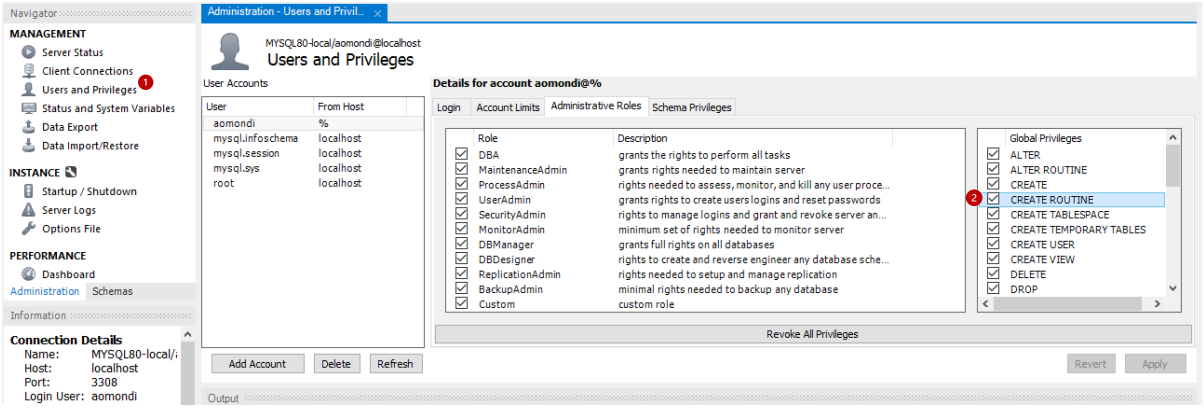
```
1:  function PAYE(employeeID)
2:      sal = salary of employeeID
3:      tax = 0.00
4:      if sal > 11,135 then
5:          if sal ≥ 12,298.99 then
6:              tax = tax + (0.1 * 12,298.99)
7:          else if sal < 12,298.99 then
8:              tax = tax + (0.1 * sal)
9:          end if
10:         if sal ≥ 23,885.99 then
11:             tax = tax + (0.15 * 11,586.99)
12:         else if sal < 23,885.99 and sal > 12,299 then
13:             tax = tax + (0.15 * (sal − 12,299))
14:         end if
15:         if sal ≥ 35,472.99 then
16:             tax = tax + (0.2 * 11,586.99)
17:         else if sal < 35,472.99 and sal > 23,886 then
18:             tax = tax + (0.2 * (sal − 23,886))
19:         end if
20:         if sal ≥ 47,059.99 then
21:             tax = tax + (0.25 * 11,586.99)
22:         else if sal < 47,059.99 and sal > 35,473 then
23:             tax = tax + (0.25 * (sal − 35,473))
24:         end if
25:         if sal ≥ 47,060 then
26:             tax = tax + (0.3 * (sal − 47,060))
27:         end if
28:         Return TOTAL monthly PAYE income tax of employeeID as tax
29:     end if
30: end function
```

STEP 5. Create a procedure called "PROC_PAYE_Kenya" to implement this algorithm

The stored procedure should accept the employee's ID as input and provide output as the tax that the individual employee is required to pay. Execute the following command to create the procedure:

```
DELIMITER \\
CREATE PROCEDURE `proc_PAYE_Kenya`(IN employeeID DOUBLE, OUT tax DOUBLE)
BEGIN
DECLARE sal DOUBLE;
SET tax = 0.00;
SELECT
    employees_salary_amount
INTO sal FROM
    employees_salary
WHERE
    emp_ID = employeeID;
if sal > 11135 THEN
            if sal >= 12289.99 THEN
                SET tax = tax + (0.1*12298.99);
            elseif sal < 12298.99 THEN
                SET tax = tax + (0.1 * sal);
            end if;
            if sal >= 23885.99 THEN
                SET tax = tax + (0.15*11586.99);
            elseif sal < 23885.99 AND sal > 12299 THEN
                SET tax = tax + (0.15 * (sal-12299));
            end if;
            if sal >= 35472.99 THEN
                SET tax = tax + (0.2*11586.99);
            elseif sal < 35472.99 AND sal > 23886 THEN
                SET tax = tax + (0.2 * (sal-23886));
            end if;
            if sal >= 47059.99 THEN
                SET tax = tax + (0.25*11586.99);
            elseif sal < 47059.99 AND sal > 35473 THEN
                SET tax = tax + (0.25 * (sal-35473));
            end if;
            if sal >= 47060 THEN
                SET tax = tax + (0.3*(sal - 47060));
            end if;
end if;
END\\
DELIMITER ;
```

- `DELIMITER \\` – Changes the delimiter symbol from the default semicolon (;) to two backslashes (\\). Once the procedure has been created, the delimiter is returned to a semicolon (;). Please refer to the lab manual on triggers for a detailed explanation of delimiters.
- `CREATE PROCEDURE` `proc_PAYE_Kenya` – This forms part of a DDL statement (a section of SQL) that allows you to define a database object called a procedure and to specify the name of the procedure. A DBA requires an account that has the "CREATE ROUTINE" privilege to create stored routines (both procedures and functions).



- `BEGIN END block` – The code in a procedure is placed between a BEGIN END block. The delimiter symbol is then placed after the END keyword to end the procedure.
- `(IN employeeID DOUBLE, OUT tax DOUBLE)` – A parameter can have one of three modes: IN, OUT, or INOUT. The syntax used to define a parameter is:

  ```
  [IN | OUT | INOUT] parameter_name datatype[(length)]
  ```

  In this case, `employeeID` is an IN parameter of the type `DOUBLE` whereas `tax` is an OUT parameter of the type `DOUBLE`.
  - `IN parameter`: IN is the default mode for parameters. An IN parameter is used to give input to a procedure. A procedure works on **a copy of** the IN parameter instead of on the original IN parameter. This means that the original value of an IN parameter is retained even after the procedure's execution ends.
  - `OUT parameter`: An OUT parameter is used to receive output from a procedure. The value of an OUT parameter can be changed inside the procedure. The new value of the OUT parameter is then passed back as output to the calling program.
  - `INOUT parameter`: A combination of IN and OUT parameters. This means that the calling program can pass an argument as input, and the procedure can modify the INOUT parameter, and pass the new value back to the calling program as output.
- `DECLARE sal DOUBLE;` – A variable is a named data object whose value can change during the execution of the procedure. Variables are used in procedures to hold immediate results. These variables are local to the stored procedure. Before using a variable, you must declare it. The syntax for declaring a variable in SQL is

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

- `SET tax = 0.00;` – The SET statement can be used to assign values to a variable once the variable has been declared. The syntax for the SET statement is:

  ```
  SET variable_name = value;
  ```

- `SELECT employees_salary_amount INTO sal FROM employees_salary WHERE emp_ID = employeeID;` – The SELECT INTO statement provides an alternative way of assigning values to a variable once the variable has been declared. In this case, the variable called `sal` is assigned the value of the employee's salary.

Variable Scope

A variable has its own scope that defines its lifetime. If you declare a variable inside a procedure, it will be out of scope when the END statement of the procedure is executed. When you declare a variable inside the BEGIN END block, it will be out of scope when the END statement of the BEGIN END block is executed. Being "out of scope" means that you can no longer access its value.

The format `@variable_name`, where the `variable_name` consists of alphanumeric characters is used to create a user-defined variable outside a procedure and outside any BEGIN END block. A user-defined variable defined by one client is not visible by other clients. In other words, a user-defined variable is session-specific. You can, however, refer to the variable during the session, for example:

```
SET @tax = 0.00;
CALL proc_PAYE_Kenya(1703,@tax);
SELECT @tax;
```

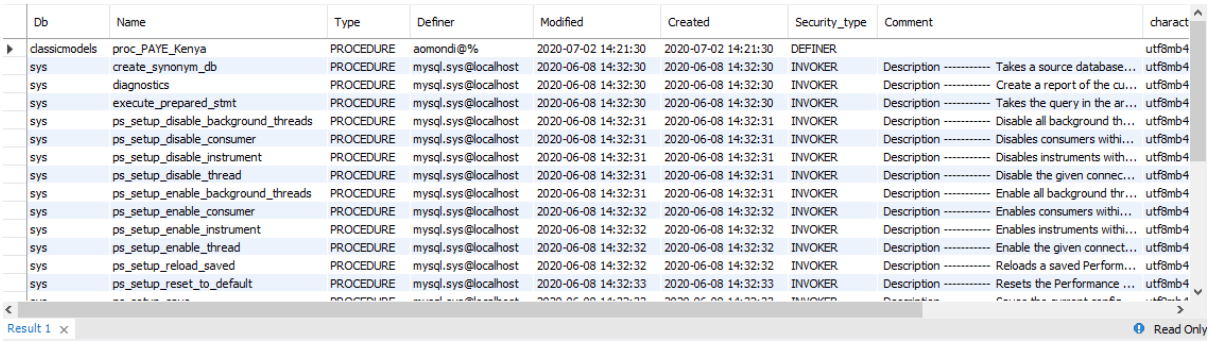
In this case, `SELECT @tax;` is used to display the value that is stored inside the variable @tax. The value will be maintained for as long as the user is logged in (i.e. during the session).

STEP 6. Show the created procedure

The SHOW PROCEDURE STATUS statement lists all the stored procedures in the database that you have a privilege to access. Execute the following command to confirm that the stored procedure has been created:

```
SHOW PROCEDURE STATUS;
```

You should get an output similar to (not identical to) the following:

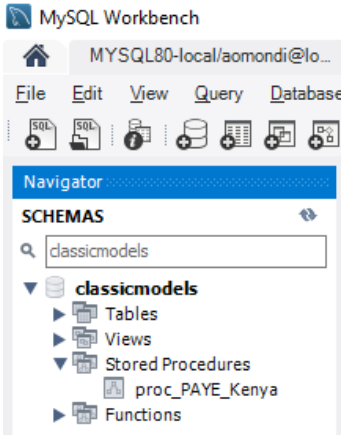


Db	Name	Type	Definer	Modified	Created	Security_type	Comment	charact
classicmodels	proc_PAYE_Kenya	PROCEDURE	aomondi@%	2020-07-02 14:21:30	2020-07-02 14:21:30	DEFINER		utf8mb4
sys	create_synonym_db	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:30	2020-06-08 14:32:30	INVOKER	Description ---------- Takes a source database...	utf8mb4
sys	diagnostics	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:30	2020-06-08 14:32:30	INVOKER	Description ---------- Create a report of the cu...	utf8mb4
sys	execute_prepared_stmt	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:30	2020-06-08 14:32:30	INVOKER	Description ---------- Takes the query in the ar...	utf8mb4
sys	ps_setup_disable_background_threads	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:31	2020-06-08 14:32:31	INVOKER	Description ---------- Disable all background th...	utf8mb4
sys	ps_setup_disable_consumer	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:31	2020-06-08 14:32:31	INVOKER	Description ---------- Disables consumers withi...	utf8mb4
sys	ps_setup_disable_instrument	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:31	2020-06-08 14:32:31	INVOKER	Description ---------- Disables instruments with...	utf8mb4
sys	ps_setup_disable_thread	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:31	2020-06-08 14:32:31	INVOKER	Description ---------- Disable the given connec...	utf8mb4
sys	ps_setup_enable_background_threads	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:31	2020-06-08 14:32:31	INVOKER	Description ---------- Enable all background thr...	utf8mb4
sys	ps_setup_enable_consumer	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:32	2020-06-08 14:32:32	INVOKER	Description ---------- Enables consumers withi...	utf8mb4
sys	ps_setup_enable_instrument	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:32	2020-06-08 14:32:32	INVOKER	Description ---------- Enables instruments withi...	utf8mb4
sys	ps_setup_enable_thread	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:32	2020-06-08 14:32:32	INVOKER	Description ---------- Enable the given connect...	utf8mb4
sys	ps_setup_reload_saved	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:32	2020-06-08 14:32:32	INVOKER	Description ---------- Reloads a saved Perform...	utf8mb4
sys	ps_setup_reset_to_default	PROCEDURE	mysql.sys@localhost	2020-06-08 14:32:33	2020-06-08 14:32:33	INVOKER	Description ---------- Resets the Performance ...	utf8mb4

Result 1 | Read Only

The first row shows the procedure "proc_PAYE_Kenya" that is inside the "classicmodels" database.

You can also view the procedures that have been created by using the GUI of MySQL Workbench under the "Navigator" tab.



Execute the following command to view the statements that were used to create the procedure:

```
SHOW CREATE PROCEDURE proc_PAYE_Kenya;
```

You should get the following output:

Procedure	sql_mode	Create Procedure	character_set_client	collation_connection	Database Collation

proc_PAYE_Kenya	STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTIO	CREATE DEFINER=`aomondi`@`%` PROCEDURE `proc_PAYE_Kenya`(IN employeeID DOUBLE, OUT tax DOUBLE)
		BEGIN
		DECLARE sal DOUBLE;
		SET tax = 0.00;
		SELECT
		employees_salary_amount
		INTO sal FROM
		employees_salary
		WHERE
		emp_ID = employeeID;
		if sal > 11135 THEN
		if sal >= 12289.99 THEN
		SET tax = tax + (0.1*12298.99);
		elseif sal < 12298.99 THEN
		SET tax = tax + (0.1 * sal);
		end if;
		if sal >= 23885.99 THEN
		SET tax = tax + (0.15*11586.99);
		elseif sal < 23885.99 AND sal > 12299
		THEN
			utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci

STEP 7. Call the stored procedure

The CALL statement allows you to invoke a stored procedure. The first time you invoke a stored procedure, the MySQL DBMS looks up the name of the procedure in the database catalogue, compiles the stored procedure's code, places the compiled version in a memory area known as cache, and then executes the stored procedure. If you invoke the same stored procedure in the same session again, MySQL executes the stored procedure from the cache without having to recompile it.

Execute the following commands to call the procedure and get John Kiprono's (employee number 1703) income tax:

```
SET @tax = 0.00;
CALL proc_PAYE_Kenya(1703,@tax);
SELECT @tax;
```

@tax
0

Execute the following commands to call the procedure and get Mary Naliaka's (employee number 1704) income tax:

```
SET @tax = 0.00;
CALL proc_PAYE_Kenya(1704,@tax);
```

```
SELECT @tax;
```

@tax
2385.049

Execute the following commands to call the procedure and get Andrew Mogaka's (employee number 1705) income tax:

```
SET @tax = 0.00;
CALL proc_PAYE_Kenya(1705,@tax);
SELECT @tax;
```

@tax
4990.7475

Execute the following commands to call the procedure and get Esther Shanyisa's (employee number 1706) income tax:

```
SET @tax = 0.00;
CALL proc_PAYE_Kenya(1706,@tax);
SELECT @tax;
```

@tax
6042.0955

Execute the following commands to call the procedure and get Joshua Oyier's (employee number 1707) income tax:

```
SET @tax = 0.00;
CALL proc_PAYE_Kenya(1707,@tax);
SELECT @tax;
```

@tax
13564.093

STEP 8. Create a function called "FUNC_PAYE_Kenya" to implement this algorithm

A few days later, the Payroll Office request for an easier way to get the income tax of all employees in the database using one command instead of a stored procedure that calculates the income tax for only one employee at a time. Create an SQL function called FUNC_PAYE_Kenya to compute the income tax of all the employees who work in the Nairobi office in Kenya. The function should be called only once to get the income tax for each of the 5 employees and its input should be the employee's salary. Execute the following command to implement this:

```
DELIMITER \\
CREATE FUNCTION `FUNC_PAYE_Kenya`(sal DOUBLE) RETURNS DOUBLE
DETERMINISTIC
BEGIN
DECLARE tax DOUBLE;
SET tax = 0.00;

if sal > 11135 THEN
            if sal >= 12298.99 THEN
                SET tax = tax + (0.1*12298.99);
            elseif sal < 12298.99 THEN
                SET tax = tax + (0.1 * sal);
            end if;
            if sal >= 23885.99 THEN
                SET tax = tax + (0.15*11586.99);
            elseif sal < 23885.99 AND sal > 12299 THEN
                SET tax = tax + (0.15 * (sal-12299));
            end if;
            if sal >= 35472.99 THEN
```

```
            SET tax = tax + (0.2*11586.99);
        elseif sal < 35472.99 AND sal > 23886 THEN
            SET tax = tax + (0.2 * (sal-23886));
        end if;
        if sal >= 47059.99 THEN
            SET tax = tax + (0.25*11586.99);
        elseif sal < 47059.99 AND sal > 35473 THEN
            SET tax = tax + (0.25 * (sal-35473));
        end if;
        if sal >= 47060 THEN
            SET tax = tax + (0.3*(sal - 47060));
        end if;
end if;
RETURN tax;
END\\
DELIMITER ;
```

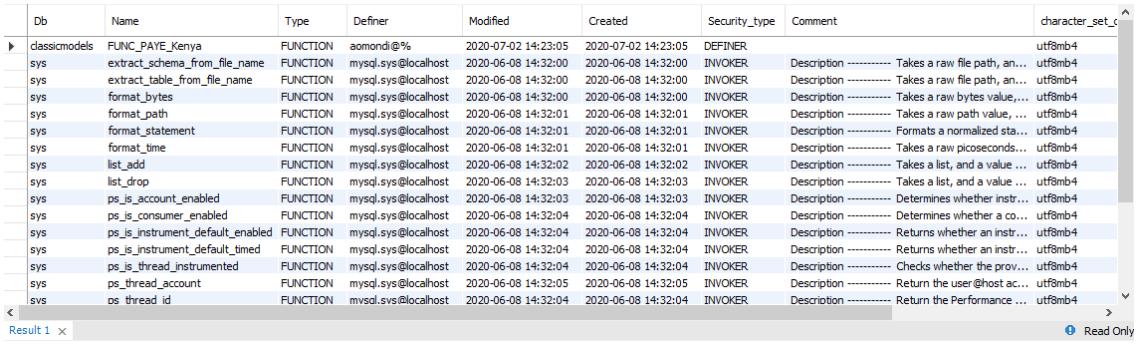
- **DETERMINISTIC** – A deterministic function always returns the same result for the same input parameters whereas a non-deterministic function returns different results for the same input parameters.

STEP 9. Show the created function

The SHOW FUNCTION STATUS statement lists all the functions in the database that you have a privilege to access. Execute the following command to confirm that the function has been created:

```
SHOW FUNCTION STATUS;
```

You should get an output similar to (not identical to) the following:

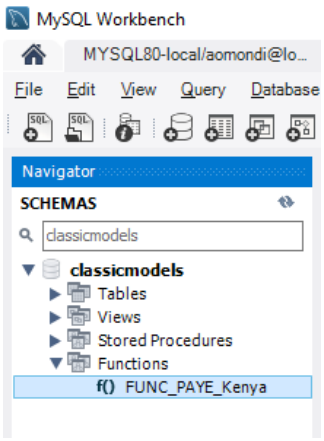


Db	Name	Type	Definer	Modified	Created	Security_type	Comment	character_set_c
classicmodels	FUNC_PAYE_Kenya	FUNCTION	aomondi@%	2020-07-02 14:23:05	2020-07-02 14:23:05	DEFINER		utf8mb4
sys	extract_schema_from_file_name	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:00	2020-06-08 14:32:00	INVOKER	Description ---------- Takes a raw file path, an...	utf8mb4
sys	extract_table_from_file_name	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:00	2020-06-08 14:32:00	INVOKER	Description ---------- Takes a raw file path, an...	utf8mb4
sys	format_bytes	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:00	2020-06-08 14:32:00	INVOKER	Description ---------- Takes a raw bytes value,...	utf8mb4
sys	format_path	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:01	2020-06-08 14:32:01	INVOKER	Description ---------- Takes a raw path value, ...	utf8mb4
sys	format_statement	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:01	2020-06-08 14:32:01	INVOKER	Description ---------- Formats a normalized sta...	utf8mb4
sys	format_time	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:01	2020-06-08 14:32:01	INVOKER	Description ---------- Takes a raw picoseconds...	utf8mb4
sys	list_add	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:02	2020-06-08 14:32:02	INVOKER	Description ---------- Takes a list, and a value ...	utf8mb4
sys	list_drop	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:03	2020-06-08 14:32:03	INVOKER	Description ---------- Takes a list, and a value ...	utf8mb4
sys	ps_is_account_enabled	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:03	2020-06-08 14:32:03	INVOKER	Description ---------- Determines whether instr...	utf8mb4
sys	ps_is_consumer_enabled	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:04	2020-06-08 14:32:04	INVOKER	Description ---------- Determines whether a co...	utf8mb4
sys	ps_is_instrument_default_enabled	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:04	2020-06-08 14:32:04	INVOKER	Description ---------- Returns whether an instr...	utf8mb4
sys	ps_is_instrument_default_timed	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:04	2020-06-08 14:32:04	INVOKER	Description ---------- Returns whether an instr...	utf8mb4
sys	ps_is_thread_instrumented	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:04	2020-06-08 14:32:04	INVOKER	Description ---------- Checks whether the prov...	utf8mb4
sys	ps_thread_account	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:05	2020-06-08 14:32:05	INVOKER	Description ---------- Return the user@host ac...	utf8mb4
sys	ps_thread_id	FUNCTION	mysql.sys@localhost	2020-06-08 14:32:04	2020-06-08 14:32:04	INVOKER	Description ---------- Return the Performance ...	utf8mb4

Result 1 | Read Only

The first row shows the function "FUNC_PAYE_Kenya" that is inside the "classicmodels" database.

You can also view the functions that have been created by using the GUI of MySQL Workbench under the "Navigator" tab.



Execute the following command to view the statements that were used to create the procedure:

```
SHOW CREATE FUNCTION FUNC_PAYE_Kenya;
```

You should get the following output:

Function	sql_mode	Create Function	character_set_client	collation_connection	Database Collation

FUNC_PAYE_Kenya	STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTIO	CREATE DEFINER=`aomondi`@`%` FUNCTION `FUNC_PAYE_Kenya`(sal DOUBLE) RETURNS double
		DETERMINISTIC
		BEGIN
		DECLARE tax DOUBLE;
		SET tax = 0.00;
		
		if sal > 11135 THEN
		if sal >= 12298.99 THEN
		SET tax = tax + (0.1*12298.99);
		elseif sal < 12298.99 THEN
		SET tax = tax + (0.1 * sal);
		end if;
		if sal >= 23885.99 THEN
		SET tax = tax + (0.15*11586.99);
		elseif sal < 23885.99 AND sal > 12299 THEN
		SET tax = tax + (0.15 * (sal-12299));
		end if;
		if sal >= 35472.99 THEN
		SET tax = tax + (0.2*11586.99);
			utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci

STEP 10. Call the function

Execute the following command to call the function and get the income tax for all the employees who work in the Nairobi office in Kenya:

```
SELECT
    classicmodels.employees_salary.employeeNumber AS 'Employee ID',
    classicmodels.employees.firstName AS 'First Name',
    classicmodels.employees.lastName AS 'Last Name',
    classicmodels.offices.city AS 'Office Location (City)',
    classicmodels.offices.country AS 'Office Location (Country)',
    classicmodels.offices.territory AS 'Office Location (Territory)',
    classicmodels.employees_salary.employees_salary_amount AS 'Taxable
Income (net salary after deducting NSSF contributions and retirement
savings)',
    FUNC_PAYE_KENYA(classicmodels.employees_salary.employees_salary_amount)
AS 'Income Tax Charged (Kenyan tax rates)'
FROM
    classicmodels.employees_salary
        JOIN
    classicmodels.employees ON employees_salary.employeeNumber =
employees.employeeNumber
        JOIN
    classicmodels.offices ON employees.officeCode = offices.officeCode
WHERE
    offices.officeCode = 8;
```

You should get the following result set as the output:

Employee ID	First Name	Last Name	Office Location (City)	Office Location (Country)	Office Location (Territory)	Taxable Income (net salary after deducting NSSF contributions and retirement savings)	Income Tax Charged (Kenyan tax rates)
1703	John	Kiprono	Nairobi	Kenya	EMEA	11000.00	0
1704	Mary	Naliaka	Nairobi	Kenya	EMEA	20000.00	2385.049
1705	Andrew	Mogaka	Nairobi	Kenya	EMEA	34000.00	4990.7475
1706	Esther	Shanyisa	Nairobi	Kenya	EMEA	38500.00	6042.0955
1707	Joshua	Oyier	Nairobi	Kenya	EMEA	65000.00	13564.093

STEP 11. Create a view called "VIEW_PAYE_Kenya" to implement this algorithm

Suppose that the Payroll Office is still not yet satisfied and has requested for an even simpler way to view the income tax of each of the employees working in the Nairobi office in Kenya. The database team finally decides to create for them a simple view which they can invoke by executing the following statement only: `SELECT * FROM VIEW_PAYE_Kenya;`

The result should have the following format. Notice the change of name in the 7th column which is titled "Taxable Income" instead of "Taxable Income (net salary after deducting NSSF contributions and retirement savings)".

VIEW_PAYE_Kenya

Employee ID	First Name	Last Name	Office Location (City)	Office Location (Territory)	Taxable Income	Income Tax Charged (Kenyan tax rates)
1703			*Nairobi*	*Kenya*		
1704			*Nairobi*	*Kenya*		
⋮			⋮	⋮		
1707			*Nairobi*	*Kenya*		

Execute the following command to create a view called “VIEW_PAYE_Kenya”:

```
CREATE VIEW `VIEW_PAYE_Kenya` AS
    SELECT
        classicmodels.employees_salary.employeeNumber AS 'Employee ID',
        classicmodels.employees.firstName AS 'First Name',
        classicmodels.employees.lastName AS 'Last Name',
        classicmodels.offices.city AS 'Office Location (City)',
        classicmodels.offices.country AS 'Office Location (Country)',
        classicmodels.offices.territory AS 'Office Location (Territory)',
        classicmodels.employees_salary.employees_salary_amount AS 'Taxable
Income',

FUNC_PAYE_KENYA(classicmodels.employees_salary.employees_salary_amount) AS
'Income Tax Charged (Kenyan tax rates)'
    FROM
        classicmodels.employees_salary
            JOIN
        classicmodels.employees ON employees_salary.employeeNumber =
employees.employeeNumber
            JOIN
        classicmodels.offices ON employees.officeCode = offices.officeCode
    WHERE
        offices.officeCode = 8;
```

STEP 12. Call the view

Execute the following command to make use of the view:

```
SELECT * FROM VIEW_PAYE_Kenya;
```

This should give you a simper way of retrieving the same data as you retrieved in STEP 10. The only difference is column 7 which has a shorter name (Taxable Income) according to the definition of the view.

Employee ID	First Name	Last Name	Office Location (City)	Office Location (Country)	Office Location (Territory)	Taxable Income	Income Tax Charged (Kenyan tax rates)
1703	John	Kiprono	Nairobi	Kenya	EMEA	11000.00	0
1704	Mary	Naliaka	Nairobi	Kenya	EMEA	20000.00	2385.049
1705	Andrew	Mogaka	Nairobi	Kenya	EMEA	34000.00	4990.7475
1706	Esther	Shanyisa	Nairobi	Kenya	EMEA	38500.00	6042.0955
1707	Joshua	Oyier	Nairobi	Kenya	EMEA	65000.00	13564.093

STEP 13. Delete the changes made to the database so far

Execute the following commands. These enable you to delete the procedure, the function, and the view:

```
DROP VIEW `classicmodels`.`VIEW_paye_kenya`;

DROP FUNCTION `classicmodels`.`FUNC_PAYE_Kenya`;

DROP PROCEDURE `classicmodels`.`proc_PAYE_Kenya`;

DROP TABLE `classicmodels`.`employees_salary`;

DELETE FROM `classicmodels`.`employees` WHERE (`employeeNumber` = '1703');
DELETE FROM `classicmodels`.`employees` WHERE (`employeeNumber` = '1704');
DELETE FROM `classicmodels`.`employees` WHERE (`employeeNumber` = '1705');
DELETE FROM `classicmodels`.`employees` WHERE (`employeeNumber` = '1706');
DELETE FROM `classicmodels`.`employees` WHERE (`employeeNumber` = '1707');

DELETE FROM `classicmodels`.`offices` WHERE (`officeCode` = '8');
```

This allows you, if you wish, to go through the lab manual from STEP 1 again.

This lab manual has gone through how to create a procedure and function and how to delete a procedure and function however, it has not gone through how to update an existing procedure. This is because, unfortunately, the MySQL DBMS does not support the editing of an existing procedure. This forces you to:

(i) View the statement that was used to create the procedure or function
(ii) Copy the statement to a different location, e.g. Notepad++
(iii) Edit the copied statement
(iv) Delete the current procedure or function
(v) Recreate the procedure or function using the edited statement

Further Reading and References

Computer programming language. (2019). In *Encyclopedia Britannica*. Encyclopædia Britannica, Inc. https://www.britannica.com/technology/computer-programming-language

MySQL Stored Function By Practical Examples. (n.d.). *MySQL Tutorial*. Retrieved 2 July 2020, from https://www.mysqltutorial.org/mysql-stored-function/

MySQL Stored Procedure Tutorial. (n.d.). *MySQL Tutorial*. Retrieved 2 July 2020, from https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx/

MySQL Views. (n.d.). Retrieved 2 July 2020, from https://www.mysqltutorial.org/mysql-views-tutorial.aspx

Oracle. (2020). *MySQL :: MySQL 8.0 Reference Manual: 24.2 Using Stored Routines*. https://dev.mysql.com/doc/refman/8.0/en/stored-routines.html

Oracle. (2020). *MySQL :: MySQL 8.0 Reference Manual: 24.5 Using Views*. https://dev.mysql.com/doc/refman/8.0/en/views.html