SWE4203: DATABASES

ASSIGNMENT 2: IMPLEMENTED DATABASE SOLUTION FOR PLUG-INS COMPANY

TO BE SUBMITTED TO: ANINA JOHN

RENUKA NYAYADHISH

WORK DONE BY: LES PAUL RANALAN

2230727

TABLE OF CONTENTS

TABLE OF FIGURES

ENTITY RELATIONSHIP (ER) DIAGRAM

It is a graphical representation or a data modeling technique that is used when designing a database and describes what are the entities with its attributes and the relationships between entities in a database system. Below are the different components of an ER diagram and their use cases.

Entity – represents a real-world object that exists. Example: customer.

Attribute – it describes a characteristic or a property of an entity. Example: name of customer.

Relationship – Represents a connection between two or more entities. Example: one order is for one customer, and one customer can place multiple orders.

Cardinality – describes the number of instances of one entity that can be associated with the other entities in a relationship. This is displayed via line symbols or simply “1”, “N”, “M”. Below are the different cardinality relationships found in an ER diagram.

- **One-to-one (1→1)** – one instance of an entity is associated with only one instance of another entity.
- **One-to-many (1→N)** – one instance of an entity is associated with many instances of another entity.
- **Many-to-many (N→N)** – many instances of an entity are associated with many instances of another entity.

Primary Key – unique identifier for an entity that is used to distinguish it from other entities. This is represented by underlining the attribute that make up a primary key, or simply putting “PK” after the attribute name.

Foreign Key – it is used to reference the primary key to another table. This is represented by connecting the entities together, dash-underlining the attribute that make up the foreign key, or simply putting “FK” after the attribute name.

Note that the ER diagram represents only the relationship between entities. If database design is to be created with references to ER diagram, the database would be subject to a few changes when normalization is applied.

ER DIAGRAM OF PLUG-INS COMPANY

Below is the updated ER diagram of Plug-Ins company that was based on the last assignment. A few changes were made to correct and improve the ER model such as implementing a new entity, Sales, together with their corresponding attributes. Other changes made to the ER diagram are minor such as renaming attribute names, removing one attribute and moving it to another entity, etc.

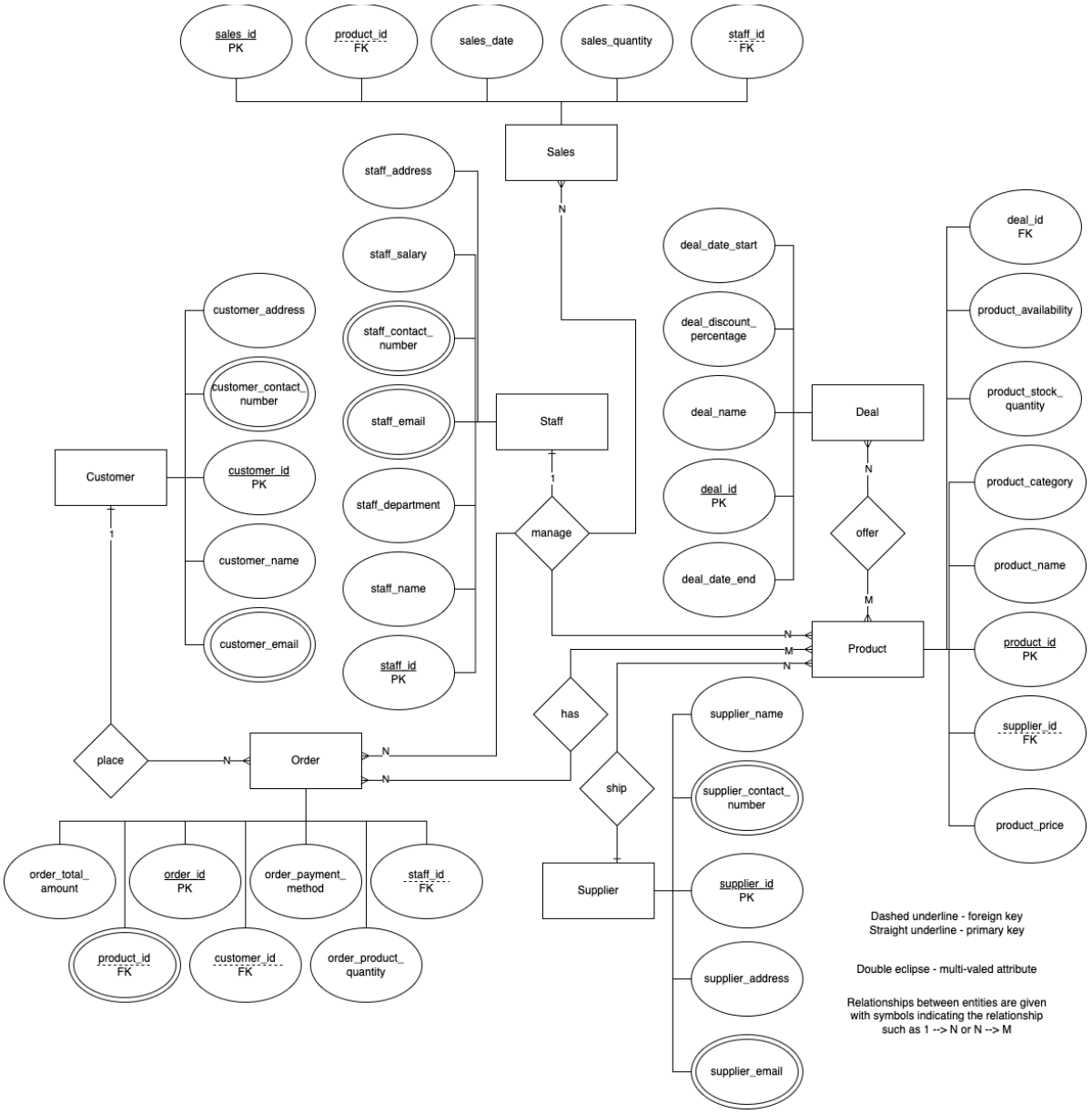


Figure 1: Updated ER diagram of Plug-Ins Company

TABLE CREATION

When creating tables in the database, the below syntax must be followed:

```
CREATE TABLE < TABLE NAME >(
    < column1 NAME > < column1 DATA TYPE >,
    < column2 NAME > < column2 DATA TYPE >, …
);
```

For the table creation of Plug-Ins Company, there needs to be 14 tables after normalization up to 3NF which are given below with the code corresponding to that table, along with that are the column names and which data types are being utilized.

CUSTOMER

```
CREATE TABLE customer(
    customer_id INT,
    customer_name VARCHAR(55)
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 2: Table structure of customer table

STAFF

```
CREATE TABLE staff(
    staff_id INT,
    staff_name VARCHAR(55),
    staff_department VARCHAR(55),
    staff_salary INT
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**staff_department**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**staff_salary**	int(11)			Yes	*NULL*		

Figure 3: Table structure of staff table

SUPPLIER

```
CREATE TABLE supplier(
    supplier_id INT,
    supplier_name VARCHAR(55)
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**supplier_id**	int(11)			Yes	*NULL*		
2	**supplier_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 4: Table structure of supplier table

EMIRATE

```
CREATE TABLE emirate(
    emirate_id INT(1),
    emirate_name VARCHAR(55)
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**email_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**supplier_id**	int(11)			Yes	*NULL*		
5	**email_address**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 5: Table structure of emirate table

ADDRESS

```
CREATE TABLE address(
    address_id INT,
    customer_id INT,
    staff_id INT,
    supplier_id INT,
    emirate_id INT,
    area VARCHAR(55),
    street VARCHAR(55)
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**address_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**supplier_id**	int(11)			Yes	*NULL*		
5	**emirate_id**	int(11)			Yes	*NULL*		
6	**area**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
7	**street**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 6: Table structure of address table

CONTACT NUMBER

```
CREATE TABLE contact_number(
    contact_number_id INT,
    customer_id INT,
    staff_id INT,
    supplier_id INT,
    contact_number VARCHAR(13)
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**contact_number_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**supplier_id**	int(11)			Yes	*NULL*		
5	**contact_number**	varchar(13)	utf8mb4_general_ci		Yes	*NULL*		

Figure 7: Table structure of contact number table

EMAIL

```
CREATE TABLE email(
    email_id INT,
    customer_id INT,
    staff_id INT,
    supplier_id INT,
    email_address VARCHAR(55)
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**email_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**supplier_id**	int(11)			Yes	*NULL*		
5	**email_address**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 8: Table structure of email table

PRODUCT

```
CREATE TABLE product(
    product_id INT,
    product_name VARCHAR(55),
    product_category VARCHAR(55),
    product_price FLOAT,
    product_stock_quantity INT,
    product_availability VARCHAR(3)
);

CREATE TABLE product_supplier(
    serial_id INT,
    product_id INT,
    supplier_id INT
);

CREATE TABLE product_deal(
    serial_id INT,
    product_id INT,
    deal_id INT
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**product_id**	int(11)			Yes	*NULL*		
2	**product_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**product_category**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**product_price**	float			Yes	*NULL*		
5	**product_stock_quantity**	int(11)			Yes	*NULL*		
6	**product_availability**	varchar(3)	utf8mb4_general_ci		Yes	*NULL*		

Figure 9: Table structure of product table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**serial_id**	int(11)			Yes	*NULL*		
2	**product_id**	int(11)			Yes	*NULL*		
3	**deal_id**	int(11)			Yes	*NULL*		

Figure 10: Table structure of product deal table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**serial_id**	int(11)			Yes	*NULL*		
2	**product_id**	int(11)			Yes	*NULL*		
3	**supplier_id**	int(11)			Yes	*NULL*		

Figure 11: Table structure of product supplier table

DEAL

```
CREATE TABLE deal(
    deal_id INT,
    deal_name VARCHAR(55),
    deal_discount_percentage INT(3),
    deal_start DATE,
    deal_end DATE
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**deal_id**	int(11)			Yes	*NULL*		
2	**deal_name**	varchar(55)	utf8mb4_general_ci		No	*None*		
3	**deal_discount_percentage**	int(3)			No	*None*		
4	**deal_start**	date			No	*None*		
5	**deal_end**	date			No	*None*		

Figure 12: Table structure of deal table

ORDER

```
CREATE TABLE order_(
    order_id INT,
    customer_id INT,
    staff_id INT,
    order_total_price FLOAT,
    order_payment_method VARCHAR(55),
    order_date DATE
);

CREATE TABLE order_items(
    serial_id INT,
    order_id INT,
    product_id INT,
    order_product_quantity INT
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**order_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**order_total_price**	float			Yes	*NULL*		
5	**order_payment_method**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
6	**order_date**	date			Yes	*NULL*		

Figure 13: Table structure of order table

SALES

```
CREATE TABLE sales(
    sales_id INT,
    product_id INT,
    staff_id INT,
    sales_quantity INT,
    sales_date DATE
);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**sales_id**	int(11)			Yes	*NULL*		
2	**product_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**sales_quantity**	int(11)			Yes	*NULL*		
5	**sales_date**	date			Yes	*NULL*		

Figure 14: Table structure of sales table

DATA ENTRY

When inserting a row in the database, INSERT INTO syntax must be used. Below are two formats of the syntax. The upper syntax is inserting values into specific columns while the bottom syntax is for inserting values in all columns without the need to specify the columns. Note that inserting values requires the values to be in the same order as the column format.

```
INSERT INTO < table name >(
    < column name1,
    column name2,
    ... >
)
VALUES
     (value1, value2);
```

```
INSERT INTO < table name >
VALUES
     (value1, value2);
```

EMIRATE

SYNTAX:

```
INSERT INTO emirate
VALUES(1, "Sharjah");

INSERT INTO emirate
VALUES(2, "Dubai");

INSERT INTO emirate
VALUES(3, "Abu Dhabi");

INSERT INTO emirate
VALUES(4, "Ajman");

INSERT INTO emirate
VALUES(5, "Umm Al Quwain");

INSERT INTO emirate
VALUES(6, "Ras Al Khaimah");

INSERT INTO emirate
VALUES(7, "Fujairah");
```

OUTPUT:

emirate_id	emirate_name
1	Sharjah
2	Dubai
3	Abu Dhabi
4	Ajman
5	Umm Al Quwain
6	Ras Al Khaimah
7	Fujairah

Figure 15: Output of INSERT INTO emirate syntax

CUSTOMER

SYNTAX:

```
INSERT INTO customer (customer_name)
VALUES
    ('Les Paul Ranalan'),
    ('John Smith'),
    ('Jane Doe'),
    ('Mike Johnson'),
    ('Sarah Lee');
```

OUTPUT:

customer_id	customer_name
1000	Les Paul Ranalan
1001	John Smith
1002	Jane Doe
1003	Mike Johnson
1004	Sarah Lee

Figure 16: Output of INSERT INTO customer syntax

STAFF

SYNTAX:

```
INSERT INTO staff (
    staff_name,
    staff_department,
    staff_salary
)
VALUES
    ('Mark Johnson', 'Marketing', 5000),
    ('Emma Davis', 'Sales', 6000),
    ('Oliver Taylor', 'IT', 7000),
    ('Isabella Martinez', 'Finance', 8000),
    ('Lucas Garcia', 'HR', 5500),
    ('Sophie Clark', 'Marketing', 4500),
    ('Liam Baker', 'Sales', 6500),
    ('Chloe Wong', 'IT', 7500),
    ('Ethan Scott', 'Finance', 8500),
    ('Ava Lee', 'HR', 6000);
```

OUTPUT:

staff_id	staff_name	staff_department	staff_salary
2000	Mark Johnson	Marketing	5000
2001	Emma Davis	Sales	6000
2002	Oliver Taylor	IT	7000
2003	Isabella Martinez	Finance	8000
2004	Lucas Garcia	HR	5500
2005	Sophie Clark	Marketing	4500
2006	Liam Baker	Sales	6500
2007	Chloe Wong	IT	7500
2008	Ethan Scott	Finance	8500
2009	Ava Lee	HR	6000

Figure 17: Output of INSERT INTO staff syntax

SUPPLIER

SYNTAX:

```
INSERT INTO supplier (supplier_name)
VALUES
    ('ABC Company'),
    ('XYZ Inc.'),
    ('123 Corporation'),
    ('Acme Co.'),
    ('Best Suppliers Ltd.');
```

OUTPUT:

supplier_id	supplier_name
3000	ABC Company
3001	XYZ Inc.
3002	123 Corporation
3003	Acme Co.
3004	Best Suppliers Ltd.

Figure 18: Output of INSERT INTO supplier syntax

ADDRESS

SYNTAX:

```
INSERT INTO address (customer_id, emirate_id, area, street)
VALUES
     (1000, 1, 'Majaz 2', 'Jamal Abdul Nasser'),
     (1001, 2, 'Jumeirah 1', 'Al Wasl Road'),
     (1002, 3, 'Al Bateen', 'Khalifa Bin Shakhbout Street'),
     (1003, 4, 'Al Rashidiya', 'Mohammed Bin Zayed Road'),
     (1004, 7, 'Al Khalidiyah', 'Electra Street');
```

```
INSERT INTO address (staff_id, emirate_id, area, street)
VALUES
     (2000, 1, 'Al Majaz 3', 'Buheira Corniche'),
     (2001, 2, 'Al Barsha 1', 'Sheikh Zayed Road'),
     (2002, 3, 'Tourist Club Area', 'Al Muroor Road'),
     (2003, 4, 'Al Hamidiyah', 'Al Rashidiya Road'),
     (2004, 5, 'Al Salamah', 'Sheikh Zayed Bin Sultan Street'),
     (2005, 6, 'Al Rams', 'Sheikh Mohammed Bin Salem Road'),
     (2006, 7, 'Al Hayl', 'Masafi-Dibba Road'),
     (2007, 1, 'Al Taawun', 'Al Taawun Road'),
     (2008, 2, 'Mirdif', 'Emirates Road'),
     (2009, 3, 'Khalifa City A', 'Abu Dhabi - Al Ain Road');
```

```
INSERT INTO address (supplier_id, emirate_id, area, street)
VALUES
     (3000, 1, 'Al Majaz 1', 'Corniche Road'),
     (3001, 2, 'Dubai Investment Park', 'Dubai - Abu Dhabi
Road'),
     (3002, 3, 'Musaffah Industrial Area', 'Street 10'),
     (3003, 4, 'Al Jurf Industrial Area', 'Sheikh Mohammed Bin
Zayed Road'),
     (3004, 5, 'Al Ittihad Road', 'Opposite UAQ Municipality');
```

OUTPUT:

address_id	customer_id	staff_id	supplier_id	emirate_id	area	street
1	1000	NULL	NULL	1	Majaz 2	Jamal Abdul Nasser
2	1001	NULL	NULL	2	Jumeirah 1	Al Wasl Road
3	1002	NULL	NULL	3	Al Bateen	Khalifa Bin Shakhbout Street
4	1003	NULL	NULL	4	Al Rashidiya	Mohammed Bin Zayed Road
5	1004	NULL	NULL	7	Al Khalidiyah	Electra Street
6	NULL	2000	NULL	1	Al Majaz 3	Buheira Corniche
7	NULL	2001	NULL	2	Al Barsha 1	Sheikh Zayed Road
8	NULL	2002	NULL	3	Tourist Club Area	Al Muroor Road
9	NULL	2003	NULL	4	Al Hamidiyah	Al Rashidiya Road
10	NULL	2004	NULL	5	Al Salamah	Sheikh Zayed Bin Sultan Street
11	NULL	2005	NULL	6	Al Rams	Sheikh Mohammed Bin Salem Road
12	NULL	2006	NULL	7	Al Hayl	Masafi-Dibba Road
13	NULL	2007	NULL	1	Al Taawun	Al Taawun Road
14	NULL	2008	NULL	2	Mirdif	Emirates Road
15	NULL	2009	NULL	3	Khalifa City A	Abu Dhabi - Al Ain Road
16	NULL	NULL	3000	1	Al Majaz 1	Corniche Road
17	NULL	NULL	3001	2	Dubai Investment Park	Dubai - Abu Dhabi Road
18	NULL	NULL	3002	3	Musaffah Industrial Area	Street 10
19	NULL	NULL	3003	4	Al Jurf Industrial Area	Sheikh Mohammed Bin Zayed Road
20	NULL	NULL	3004	5	Al Ittihad Road	Opposite UAQ Municipality

Figure 19: Output of INSERT INTO address syntax

Note that only one entity ID is allowed per row, thus the NULL value. This is the result of the CHECK constraint. When trying to insert two or more IDs using GUI in phpMyAdmin, it would result in error.

CONTACT NUMBER

SYNTAX:

```
INSERT INTO contact_number (customer_id, contact_number)
VALUES
    (1000, '+971501234567'),
    (1001, '+971509876543'),
    (1002, '+971502345678'),
    (1003, '+971503456789'),
    (1004, '+971508765432');
```

```
INSERT INTO contact_number (staff_id, contact_number)
VALUES
     (2000, '+971501234567'),
     (2001, '+971501234568'),
     (2002, '+971501234569'),
     (2003, '+971501234570'),
     (2004, '+971501234571'),
     (2005, '+971501234572'),
     (2006, '+971501234573'),
     (2007, '+971501234574'),
     (2008, '+971501234575'),
     (2009, '+971501234576');
```

```
INSERT INTO contact_number (supplier_id, contact_number)
VALUES
     (3000, '+97124567890'),
     (3001, '+97124567891'),
     (3002, '+97124567892'),
     (3003, '+97124567893'),
     (3004, '+97124567894');
```

OUTPUT:

contact_number_id	customer_id	staff_id	supplier_id	contact_number
1	1000	NULL	NULL	+971501234567
2	1001	NULL	NULL	+971509876543
3	1002	NULL	NULL	+971502345678
4	1003	NULL	NULL	+971503456789
5	1004	NULL	NULL	+971508765432
6	NULL	2000	NULL	+971501234567
7	NULL	2001	NULL	+971501234568
8	NULL	2002	NULL	+971501234569
9	NULL	2003	NULL	+971501234570
10	NULL	2004	NULL	+971501234571
11	NULL	2005	NULL	+971501234572
12	NULL	2006	NULL	+971501234573
13	NULL	2007	NULL	+971501234574
14	NULL	2008	NULL	+971501234575
15	NULL	2009	NULL	+971501234576
16	NULL	NULL	3000	+97124567890
17	NULL	NULL	3001	+97124567891
18	NULL	NULL	3002	+97124567892
19	NULL	NULL	3003	+97124567893
20	NULL	NULL	3004	+97124567894

Figure 20: Output of INSERT INTO contact number syntax

Note that only one entity ID is allowed per row, thus the NULL value. This is the result of the CHECK constraint. When trying to insert two or more IDs using GUI in phpMyAdmin, it would result in error.

EMAIL

SYNTAX:

```
INSERT INTO email (customer_id, email_address)
VALUES
  (1000, 'les.paul@gmail.com'),
  (1001, 'john.smith@yahoo.com'),
  (1002, 'jane.doe@outlook.com'),
  (1003, 'mike.johnson@protonmail.com'),
  (1004, 'sarah.lee@gmail.com');
```

```
INSERT INTO email (staff_id, email_address)
VALUES
  (2000, 'mark.johnson@plug-ins.com'),
  (2001, 'emma.davis@plug-ins.com'),
  (2002, 'oliver.taylor@plug-ins.com'),
  (2003, 'isabella.martinez@plug-ins.com'),
  (2004, 'lucas.garcia@plug-ins.com'),
  (2005, 'sophie.clark@plug-ins.com'),
  (2006, 'liam.baker@plug-ins.com'),
  (2007, 'chloe.wong@plug-ins.com'),
  (2008, 'ethan.scott@plug-ins.com'),
  (2009, 'ava.lee@plug-ins.com');
```

```
INSERT INTO email (supplier_id, email_address)
VALUES
     (3000, 'abc.company@mycompanydomain.com'),
     (3001, 'xyz.inc@suppliersdomain.com'),
     (3002, '123.corporation@corpdomain.com'),
     (3003, 'acme.co@acmedomain.com'),
     (3004, 'best.suppliers.ltd@bestsuppliersdomain.com');
```

OUTPUT:

email_id	customer_id	staff_id	supplier_id	email_address
1	1000	NULL	NULL	les.paul@gmail.com
2	1001	NULL	NULL	john.smith@yahoo.com
3	1002	NULL	NULL	jane.doe@outlook.com
4	1003	NULL	NULL	mike.johnson@protonmail.com
5	1004	NULL	NULL	sarah.lee@gmail.com
6	NULL	2000	NULL	mark.johnson@plug-ins.com
7	NULL	2001	NULL	emma.davis@plug-ins.com
8	NULL	2002	NULL	oliver.taylor@plug-ins.com
9	NULL	2003	NULL	isabella.martinez@plug-ins.com
10	NULL	2004	NULL	lucas.garcia@plug-ins.com
11	NULL	2005	NULL	sophie.clark@plug-ins.com
12	NULL	2006	NULL	liam.baker@plug-ins.com
13	NULL	2007	NULL	chloe.wong@plug-ins.com
14	NULL	2008	NULL	ethan.scott@plug-ins.com
15	NULL	2009	NULL	ava.lee@plug-ins.com
16	NULL	NULL	3000	abc.company@mycompanydomain.com
17	NULL	NULL	3001	xyz.inc@suppliersdomain.com
18	NULL	NULL	3002	123.corporation@corpdomain.com
19	NULL	NULL	3003	acme.co@acmedomain.com
20	NULL	NULL	3004	best.suppliers.ltd@bestsuppliersdomain.com

Figure 21: Output of INSERT INTO email syntax

Note that only one entity ID is allowed per row, thus the NULL value. This is the result of the CHECK constraint. When trying to insert two or more IDs using GUI in phpMyAdmin, it would result in error.

DEAL

SYNTAX:

```
INSERT INTO deal (deal_name, deal_discount_percentage,
deal_start, deal_end)
VALUES
     ('Ramadan Deals', 10, '2023-3-22', '2023-4-22'),
     ('Summer Sale', 15, '2023-6-1', '2023-6-30'),
     ('Back to School Deals', 20, '2023-8-1', '2023-8-31'),
     ('Halloween Sale', 25, '2023-10-1', '2023-11-1'),
     ('Black Friday Deals', 30, '2023-11-22', '2023-11-30');
```

OUTPUT:

deal_id	deal_name	deal_discount_percentage	deal_start	deal_end
5000	Ramadan Deals	10	2023-03-22	2023-04-22
5001	Summer Sale	15	2023-06-01	2023-06-30
5002	Back to School Deals	20	2023-08-01	2023-08-31
5003	Halloween Sale	25	2023-10-01	2023-11-01
5004	Black Friday Deals	30	2023-11-22	2023-11-30

Figure 22: Output of INSERT INTO deal syntax

PRODUCT

SYNTAX:

```
INSERT INTO product (product_name, product_category,
product_price, product_stock_quantity, product_availability)
VALUES
     ('Motospeed CK61', 'Peripherals', 99.99, 100, 'YES'),
     ('Logitech G502', 'Peripherals', 69.99, 150, 'YES'),
     ('Razer DeathAdder', 'Peripherals', 49.99, 200, 'YES'),
     ('Corsair K95', 'Peripherals', 149.99, 2, 'YES'),
     ('SteelSeries Rival 600', 'Peripherals', 79.99, 175,
'YES'),
     ('HyperX Cloud II', 'Headsets', 99.99, 100, 'YES'),
     ('Sennheiser HD 599', 'Headsets', 199.99, 50, 'YES'),
     ('Bose QuietComfort 35 II', 'Headsets', 299.99, 4, 'YES'),
     ('Samsung 970 EVO', 'Storage', 149.99, 75, 'YES'),
     ('WD Blue SN550', 'Storage', 99.99, 100, 'YES'),
     ('Fujifilm Instax Mini 9', 'Cameras', 99.99, 0, 'NO'),
     ('Kindle Paperwhite', 'Tablet', 499.99, 0, 'NO'),
     ('BenQ GW2780', 'Monitor', 699.99, 1, 'YES');
```

OUTPUT:

product_id	product_name	product_category	product_price	product_stock_quantity	product_availability
4000	Motospeed CK61	Peripherals	99.99	100	YES
4001	Logitech G502	Peripherals	69.99	150	YES
4002	Razer DeathAdder	Peripherals	49.99	200	YES
4003	Corsair K95	Peripherals	149.99	2	YES
4004	SteelSeries Rival 600	Peripherals	79.99	175	YES
4005	HyperX Cloud II	Headsets	99.99	100	YES
4006	Sennheiser HD 599	Headsets	199.99	50	YES
4007	Bose QuietComfort 35 II	Headsets	299.99	4	YES
4008	Samsung 970 EVO	Storage	149.99	75	YES
4009	WD Blue SN550	Storage	99.99	100	YES
4010	Fujifilm Instax Mini 9	Cameras	99.99	0	NO
4011	Kindle Paperwhite	Tablet	499.99	0	NO
4012	BenQ GW2780	Monitor	699.99	1	YES

Figure 23: Output of INSERT INTO product syntax

PRODUCT DEAL

SYNTAX:

```
INSERT INTO product_deal (product_id, deal_id)
VALUES
    (4000, 5001),
    (4001, 5002),
    (4002, 5004);
```

OUTPUT:

serial_id	product_id	deal_id
1	4000	5001
2	4001	5002
3	4002	5004

Figure 24: Output of INSERT INTO product deal syntax

PRODUCT SUPPLIER

SYNTAX:

```
INSERT INTO product_supplier (product_id, supplier_id)
VALUES
    (4000, 3000),
    (4001, 3003),
    (4002, 3000),
    (4003, 3002),
    (4004, 3001),
    (4005, 3004),
    (4006, 3000),
    (4007, 3004),
    (4008, 3003),
    (4009, 3001),
    (4010, 3002),
    (4011, 3001),
    (4012, 3004);
```

OUTPUT:

serial_id	product_id	supplier_id
1	4000	3000
2	4001	3003
3	4002	3000
4	4003	3002
5	4004	3001
6	4005	3004
7	4006	3000
8	4007	3004
9	4008	3003
10	4009	3001
11	4010	3002
12	4011	3001
13	4012	3004

Figure 25: Output of INSERT INTO product supplier syntax

ORDER

SYNTAX:

```
INSERT INTO order_ (customer_id, staff_id, order_total_price,
order_payment_method, order_date)
VALUES
    (1000, 2001, 299.99, 'Credit Card', '2023-04-01'),
    (1001, 2003, 49.99, 'Cash', '2023-04-02'),
    (1003, 2006, 149.99, 'Credit Card', '2023-05-05'),
    (1004, 2008, 399.99, 'Debit Card', '2023-05-08'),
    (1002, 2005, 199.99, 'Debit Card', '2023-06-10'),
    (1004, 2007, 99.99, 'Credit Card', '2023-06-15'),
    (1000, 2002, 149.99, 'Debit Card', '2023-06-20'),
    (1001, 2004, 449.99, 'Credit Card', '2023-07-25'),
    (1002, 2009, 199.99, 'Cash', '2023-07-27'),
    (1003, 2005, 99.99, 'Debit Card', '2023-08-30'),
    (1004, 2007, 249.99, 'Credit Card', '2023-08-31'),
    (1002, 2006, 149.99, 'Cash', '2023-09-10'),
    (1001, 2009, 199.99, 'Credit Card', '2023-09-11'),
    (1000, 2004, 299.99, 'Debit Card', '2023-09-15');
```

OUTPUT:

order_id	customer_id	staff_id	order_total_price	order_payment_method	order_date
10000	1000	2001	299.99	Credit Card	2023-04-01
10001	1001	2003	49.99	Cash	2023-04-02
10002	1003	2006	149.99	Credit Card	2023-05-05
10003	1004	2008	399.99	Debit Card	2023-05-08
10004	1002	2005	199.99	Debit Card	2023-06-10
10005	1004	2007	99.99	Credit Card	2023-06-15
10006	1000	2002	149.99	Debit Card	2023-06-20
10007	1001	2004	449.99	Credit Card	2023-07-25
10008	1002	2009	199.99	Cash	2023-07-27
10009	1003	2005	99.99	Debit Card	2023-08-30
10010	1004	2007	249.99	Credit Card	2023-08-31
10011	1002	2006	149.99	Cash	2023-09-10
10012	1001	2009	199.99	Credit Card	2023-09-11
10013	1000	2004	299.99	Debit Card	2023-09-15

Figure 26: Output of INSERT INTO order syntax

ORDER ITEMS

SYNTAX:

```
INSERT INTO order_items (order_id, product_id,
order_product_quantity)
VALUES
  (10002, 4005, 1),
  (10000, 4000, 2),
  (10001, 4001, 3),
  (10003, 4007, 2),
  (10004, 4009, 1),
  (10005, 4006, 3),
  (10006, 4001, 1),
  (10007, 4008, 3),
  (10008, 4004, 6),
  (10009, 4001, 3),
  (10010, 4006, 3),
  (10011, 4003, 1),
  (10012, 4012, 2),
  (10013, 4010, 1);
```

OUTPUT:

serial_id	order_id	product_id	order_product_quantity
1	10002	4005	1
2	10000	4000	2
3	10001	4001	3
4	10003	4007	2
5	10004	4009	1
6	10005	4006	3
7	10006	4001	1
8	10007	4008	3
9	10008	4004	6
10	10009	4001	3
11	10010	4006	3
12	10011	4003	1
13	10012	4012	2
14	10013	4010	1

Figure 27: Output of INSERT INTO product items syntax

SALES

SYNTAX:

```
INSERT INTO sales (product_id, staff_id, sales_quantity,
sales_date)
VALUES
     (4000, 2000, 5, '2023-04-01'),
     (4001, 2001, 10, '2023-04-02'),
     (4002, 2002, 7, '2023-05-03'),
     (4003, 2003, 2, '2023-05-04'),
     (4004, 2004, 8, '2023-05-05'),
     (4005, 2005, 12, '2023-06-06'),
     (4006, 2006, 3, '2023-07-07'),
     (4007, 2007, 6, '2023-08-08'),
     (4008, 2008, 9, '2023-09-09'),
     (4009, 2009, 4, '2023-10-10'),
     (4000, 2005, 3, '2023-11-11'),
     (4002, 2003, 5, '2023-12-12'),
     (4004, 2001, 2, '2024-01-13'),
     (4006, 2009, 6, '2024-02-14'),
     (4008, 2007, 8, '2024-03-15'),
     (4001, 2002, 7, '2024-04-16'),
     (4003, 2004, 1, '2024-05-17'),
     (4005, 2006, 4, '2024-06-18'),
     (4007, 2008, 10, '2024-07-19'),
     (4009, 2000, 9, '2024-08-20');
```

OUTPUT:

sales_id	product_id	staff_id	sales_quantity	sales_date
1	4000	2000	5	2023-04-01
2	4001	2001	10	2023-04-02
3	4002	2002	7	2023-05-03
4	4003	2003	2	2023-05-04
5	4004	2004	8	2023-05-05
6	4005	2005	12	2023-06-06
7	4006	2006	3	2023-07-07
8	4007	2007	6	2023-08-08
9	4008	2008	9	2023-09-09
10	4009	2009	4	2023-10-10
11	4000	2005	3	2023-11-11
12	4002	2003	5	2023-12-12
13	4004	2001	2	2024-01-13
14	4006	2009	6	2024-02-14
15	4008	2007	8	2024-03-15
16	4001	2002	7	2024-04-16
17	4003	2004	1	2024-05-17
18	4005	2006	4	2024-06-18
19	4007	2008	10	2024-07-19
20	4009	2000	9	2024-08-20

Figure 28: Output of INSERT INTO sales syntax

IMPLEMENTING INTEGRITY CONSTRAINTS

When dealing with integrity constraints, it is important to know ALTER TABLE syntax as it is a way to add constraints to specific columns. There are two syntaxes below, they do the same thing but just different formats.

```
ALTER TABLE < table name >
    MODIFY < column name > < column data type > < constraint >;
```

```
ALTER TABLE < table name >
    ADD CONSTRAINT < constraint syntax >;
```

PRIMARY KEY

The PRIMARY KEY constraint was utilized to uniquely identify a row of each table. This is important in terms of consistency and integrity.

Below is the syntax for applying PRIMARY KEY constraint on each of the tables.

```
ALTER TABLE customer
    MODIFY customer_id INT PRIMARY KEY;

ALTER TABLE staff
    MODIFY staff_id INT PRIMARY KEY;

ALTER TABLE supplier
    MODIFY supplier_id INT PRIMARY KEY;

ALTER TABLE emirate
    MODIFY emirate_id INT PRIMARY KEY;

ALTER TABLE address
    MODIFY address_id INT PRIMARY KEY;

ALTER TABLE contact_number
    MODIFY contact_number_id INT PRIMARY KEY;

ALTER TABLE email
    MODIFY email_id INT PRIMARY KEY;

ALTER TABLE product
    MODIFY product_id INT PRIMARY KEY;

ALTER TABLE product_deal
     MODIFY serial_id INT PRIMARY KEY;

ALTER TABLE product_supplier
     MODIFY serial_id INT PRIMARY KEY;

ALTER TABLE deal
    MODIFY deal_id INT PRIMARY KEY;

ALTER TABLE order_
    MODIFY order_id INT PRIMARY KEY;

ALTER TABLE order_items
    MODIFY serial_id INT PRIMARY KEY;
```

```
ALTER TABLE sales
    MODIFY sales_id INT PRIMARY KEY;
```

AUTO INCREMENT

The AUTO INCREMENT constraint was utilized to automatically provide a unique identifier to each row without the need to type in the number specifically.

Below is the syntax for applying AUTO INCREMENT in each table.

```
ALTER TABLE customer
    MODIFY customer_id INT AUTO_INCREMENT,
    AUTO_INCREMENT=1000;

ALTER TABLE staff
    MODIFY staff_id INT AUTO_INCREMENT,
    AUTO_INCREMENT=2000;

ALTER TABLE supplier
    MODIFY supplier_id INT AUTO_INCREMENT,
    AUTO_INCREMENT=3000;

ALTER TABLE emirate
    MODIFY emirate_id INT AUTO_INCREMENT;

ALTER TABLE address
    MODIFY address_id INT AUTO_INCREMENT;

ALTER TABLE contact_number
    MODIFY contact_number_id INT AUTO_INCREMENT;

ALTER TABLE email
    MODIFY email_id INT AUTO_INCREMENT;

ALTER TABLE product
    MODIFY product_id INT AUTO_INCREMENT,
    AUTO_INCREMENT=4000;

ALTER TABLE product_deal
     MODIFY serial_id INT AUTO_INCREMENT;

ALTER TABLE product_supplier
     MODIFY serial_id INT AUTO_INCREMENT;

ALTER TABLE deal
    MODIFY deal_id INT AUTO_INCREMENT,
    AUTO_INCREMENT=5000;
```

```
ALTER TABLE order_
    MODIFY order_id INT AUTO_INCREMENT,
    AUTO_INCREMENT=10000;

ALTER TABLE order_items
     MODIFY serial_id INT AUTO_INCREMENT;

ALTER TABLE sales
    MODIFY sales_id INT AUTO_INCREMENT;
```

NOT NULL

The NOT NULL constraint was utilized in majority of the columns of all tables as it is never ideal or advisable to enter empty or null values in a column unless it is required in a specific use.

Below is the syntax for applying NOT NULL constraint on each of the tables.

```
ALTER TABLE customer
    MODIFY customer_name VARCHAR(55) NOT NULL;

ALTER TABLE staff
    MODIFY staff_name VARCHAR(55) NOT NULL,
    MODIFY staff_department VARCHAR(55) NOT NULL,
    MODIFY staff_salary INT NOT NULL;

ALTER TABLE supplier
    MODIFY supplier_name VARCHAR(55) NOT NULL;

ALTER TABLE emirate
    MODIFY emirate_name varchar(55) NOT NULL;

ALTER TABLE address
    MODIFY emirate_id INT NOT NULL,
    MODIFY area VARCHAR(55) NOT NULL,
    MODIFY street VARCHAR(55) NOT NULL;

ALTER TABLE contact_number
    MODIFY contact_number VARCHAR(13) NOT NULL;

ALTER TABLE email
    MODIFY email_address VARCHAR(55) NOT NULL;

ALTER TABLE product
    MODIFY product_name VARCHAR(55) NOT NULL,
    MODIFY product_category VARCHAR(55) NOT NULL,
    MODIFY product_price FLOAT NOT NULL,
    MODIFY product_stock_quantity INT NOT NULL,
    MODIFY product_availability VARCHAR(3) NOT NULL;

ALTER TABLE product_deal
     MODIFY product_id INT NOT NULL,
     MODIFY deal_id INT NOT NULL;

ALTER TABLE product_supplier
```

```
     MODIFY product_id INT NOT NULL,
     MODIFY supplier_id INT NOT NULL;

ALTER TABLE deal
    MODIFY deal_name VARCHAR(55)NOT NULL,
    MODIFY deal_discount_percentage INT(3) NOT NULL,
    MODIFY deal_start DATE NOT NULL,
    MODIFY deal_end DATE NOT NULL;


ALTER TABLE order_
    MODIFY customer_id INT NOT NULL,
    MODIFY staff_id INT NOT NULL,
    MODIFY order_total_price FLOAT NOT NULL,
    MODIFY order_payment_method VARCHAR(55) NOT NULL,
    MODIFY order_date DATE NOT NULL;

ALTER TABLE order_items
     MODIFY order_id INT NOT NULL,
     MODIFY product_id INT NOT NULL,
     MODIFY order_product_quantity INT NOT NULL;

ALTER TABLE sales
    MODIFY product_id INT NOT NULL,
    MODIFY staff_id INT NOT NULL,
    MODIFY  sales_quantity INT NOT NULL,
    MODIFY sales_date DATE NOT NULL;
```

FOREIGN KEY

The FOREIGN KEY constraint was used to link two tables together, making it easier to work with data. ON UPDATE CASCADE and ON DELETE CASCADE modifiers was also added to make sure that if a parent record is deleted or modified, it also deletes or modifies all child records associating to that record; this is important in terms of consistency and integrity.

Below is the syntax for applying FOREIGN KEY constraint on the specific tables.

```
ALTER TABLE address
    ADD CONSTRAINT FOREIGN KEY(emirate_id) REFERENCES
emirate(emirate_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY(customer_id) REFERENCES
customer(customer_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY(supplier_id) REFERENCES
supplier(supplier_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY(staff_id) REFERENCES
staff(staff_id) ON UPDATE CASCADE ON DELETE CASCADE;

ALTER TABLE contact_number
     ADD CONSTRAINT FOREIGN KEY(customer_id) REFERENCES
customer(customer_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY(supplier_id) REFERENCES
supplier(supplier_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY(staff_id) REFERENCES
staff(staff_id) ON UPDATE CASCADE ON DELETE CASCADE;

ALTER TABLE email
     ADD CONSTRAINT FOREIGN KEY(customer_id) REFERENCES
customer(customer_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY(supplier_id) REFERENCES
supplier(supplier_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY(staff_id) REFERENCES
staff(staff_id) ON UPDATE CASCADE ON DELETE CASCADE;

ALTER TABLE product_deal
     ADD CONSTRAINT FOREIGN KEY (product_id) REFERENCES
product(product_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY (deal_id) REFERENCES
deal(deal_id);

ALTER TABLE product_supplier
```

```
     ADD CONSTRAINT FOREIGN KEY (product_id) REFERENCES
product(product_id) ON UPDATE CASCADE ON DELETE CASCADE,
     ADD CONSTRAINT FOREIGN KEY(supplier_id) REFERENCES
supplier(supplier_id) ON UPDATE CASCADE ON DELETE CASCADE;

ALTER TABLE order_
     ADD CONSTRAINT FOREIGN KEY(customer_id) REFERENCES
customer(customer_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY(staff_id) REFERENCES
staff(staff_id) ON UPDATE CASCADE ON DELETE CASCADE;

ALTER TABLE order_items
     ADD CONSTRAINT FOREIGN KEY(order_id) REFERENCES
order_(order_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY(product_id) REFERENCES
product(product_id) ON UPDATE CASCADE ON DELETE CASCADE;

ALTER TABLE sales
     ADD CONSTRAINT FOREIGN KEY(product_id) REFERENCES
product(product_id) ON UPDATE CASCADE ON DELETE CASCADE,
    ADD CONSTRAINT FOREIGN KEY(staff_id) REFERENCES
staff(staff_id) ON UPDATE CASCADE ON DELETE CASCADE;
```

UNIQUE

The UNIQUE constraint was not utilized in this database as primary keys and foreign keys were sufficient to make a row unique with its respectable ID.

DEFAULT

The DEFAULT constraint was not utilized in this database as there is really no applicable use for it.

CHECK

The CHECK constraint was utilized primarily only on the EMAIL, ADDRESS, AND CONTACT NUMBER tables as all CUSTOMER, STAFF, and SUPPLIER details are all stored in each table. Through CHECK constraint, only one ID can only be assigned per row.

Example: Only customer ID is assigned to “Dubai”, and not staff ID or supplier ID.

Below is the syntax for applying CHECK constraint for the specific tables.

```
ALTER TABLE address
     ADD CONSTRAINT CHECK
         (
             customer_id IS NOT NULL AND staff_id IS NULL AND
supplier_id IS NULL OR customer_id IS NULL AND staff_id IS NOT
NULL AND supplier_id IS NULL OR customer_id IS NULL AND
staff_id IS NULL AND supplier_id IS NOT NULL
         );

ALTER TABLE contact_number
    ADD CONSTRAINT CHECK
    (
        customer_id IS NOT NULL AND staff_id IS NULL AND
supplier_id IS NULL OR customer_id IS NULL AND staff_id IS NOT
NULL AND supplier_id IS NULL OR customer_id IS NULL AND
staff_id IS NULL AND supplier_id IS NOT NULL
    );

ALTER TABLE email
     ADD CONSTRAINT CHECK
         (
             customer_id IS NOT NULL AND staff_id IS NULL AND
supplier_id IS NULL OR customer_id IS NULL AND staff_id IS NOT
NULL AND supplier_id IS NULL OR customer_id IS NULL AND
staff_id IS NULL AND supplier_id IS NOT NULL
         );

ALTER TABLE
    product ADD CONSTRAINT
CHECK
    (
        product_availability IN('YES', 'NO')
    );
```

SQL QUERY

It is a command written in SQL that retrieves data from a database. This is where SELECT syntax is mostly utilized. Below is the SELECT syntax.

```
SELECT
    < column1,
    column2,
    ... >
FROM
    < table name >
WHERE
    < condition >;
```

JOIN syntax can also be used with SELECT syntax to accurately join two or more tables, making it easier to work with data especially linked tables. Below is the SELECT syntax with the JOIN syntax used together.

```
SELECT
    < column1 >,
    < column2 >,
    < ... >
FROM
    < table1 >
JOIN < table2 >  ON < table1.column_name > = <
table2.column_name >;
```

Note that SELECT query can be summarized via views using the CREATE VIEW syntax. Below is the syntax for both.

```
CREATE VIEW < view_name > AS
SELECT
    < column1,
    column2,
    ... >
FROM
    < table name >
WHERE
    < condition >;
```

In some cases, INSERT syntax is also used to add a new data in the database. Below is the syntax mentioned in the previous chapters.

```
INSERT INTO < table name >(
    < column name1,
    column name2,
    ... >
)
VALUES
     (value1, value2);
```

```
INSERT INTO < table name >
VALUES
     (value1, value2);
```

Another case of SQL query is the UPDATE syntax, wherein it changes the value of the specified record. Below is the syntax for UPDATE.

```
UPDATE
    < table name >
SET
    < column1 > = < new_value >,
    < ... >
WHERE
    < condition >;
```

The DELETE syntax is for removing a record from a database. Below is the syntax for DELETE command.

```
DELETE
FROM
    < table name >
WHERE
    < column > = < value >;
```

1. ADDING PRODUCT

SYNTAX:

```
INSERT INTO product(
    product_name,
    product_category,
    product_price,
    product_stock_quantity
)
VALUES(
    'GeForce GTX 1660',
    'Graphics Card',
    699.99,
    20
);
```

OUTPUT:

product_id ▼ 1	product_name	product_category	product_price	product_stock_quantity
4013	GeForce GTX 1660	Graphics Card	699.99	20

Figure 29: Output of syntax when adding a product

Note that the product ID has the AUTO_INCREMENT constraint starting at 4000.

2. UPDATING PRICE OF PRODUCT

SYNTAX:

```
UPDATE
    product
SET
    product_price = 749.99
WHERE
    product_id = 4013;
```

OUTPUT:

product_id ▼ 1	product_name	product_category	product_price	product_stock_quantity
4013	GeForce GTX 1660	Graphics Card	749.99	20

Figure 30: Output of syntax updating price of product

Note that the previous price was 699.99 on product ID, 4010, as seen on previous queries.

3. CHECKING PRODUCT AVAILABILITY

SYNTAX:

```
SELECT
    *
FROM
    product
WHERE
    product_availability = 'YES';
```

OUTPUT:

product_id	product_name	product_category	product_price	product_stock_quantity	product_availability
4000	Motospeed CK61	Peripherals	99.99	100	YES
4001	Logitech G502	Peripherals	69.99	150	YES
4002	Razer DeathAdder	Peripherals	49.99	200	YES
4003	Corsair K95	Peripherals	149.99	2	YES
4004	SteelSeries Rival 600	Peripherals	79.99	175	YES
4005	HyperX Cloud II	Headsets	99.99	100	YES
4006	Sennheiser HD 599	Headsets	199.99	50	YES
4007	Bose QuietComfort 35 II	Headsets	299.99	4	YES
4008	Samsung 970 EVO	Storage	149.99	75	YES
4009	WD Blue SN550	Storage	99.99	100	YES
4012	BenQ GW2780	Monitor	699.99	1	YES

Figure 31: Output of syntax when checking product availability

As seen in the output figure above, product ID 4010 and 4011 are not present as those products have 0 stock quantities.

4. PLACE ORDER WITH PRODUCT

SYNTAX:

```
INSERT INTO order_(order_id, customer_id, staff_id,
order_total_price, order_payment_method, order_date)
VALUES
     (10014, 1001, 2007, 799.99, 'Cash', '2023-4-18');

INSERT INTO order_items(order_id, product_id,
order_product_quantity)
VALUES
     (10014, 4013, 1);
```

OUTPUT:

order_id ▼ 1	customer_id	staff_id	order_total_price	order_payment_method	order_date
10014	1001	2007	799.99	Cash	2023-04-18

Figure 32: Output of order table when placing order with product

serial_id ▼ 1	order_id	product_id	order_product_quantity
15	10014	4013	1

Figure 33: Output of order items table when placing order with product

5. CANCEL ORDER

SYNTAX:

```
DELETE
FROM
    order_
WHERE
    order_id = 10014;
```

OUTPUT:

order_id	customer_id	staff_id	order_total_price	order_payment_method	order_date
10000	1000	2001	299.99	Credit Card	2023-04-01
10001	1001	2003	49.99	Cash	2023-04-02
10002	1003	2006	149.99	Credit Card	2023-05-05
10003	1004	2008	399.99	Debit Card	2023-05-08
10004	1002	2005	199.99	Debit Card	2023-06-10
10005	1004	2007	99.99	Credit Card	2023-06-15
10006	1000	2002	149.99	Debit Card	2023-06-20
10007	1001	2004	449.99	Credit Card	2023-07-25
10008	1002	2009	199.99	Cash	2023-07-27
10009	1003	2005	99.99	Debit Card	2023-08-30
10010	1004	2007	249.99	Credit Card	2023-08-31
10011	1002	2006	149.99	Cash	2023-09-10
10012	1001	2009	199.99	Credit Card	2023-09-11
10013	1000	2004	299.99	Debit Card	2023-09-15

Figure 34: Output of order table when cancelling order

serial_id	order_id	product_id	order_product_quantity
1	10002	4005	1
2	10000	4000	2
3	10001	4001	3
4	10003	4007	2
5	10004	4009	1
6	10005	4006	3
7	10006	4001	1
8	10007	4008	3
9	10008	4004	6
10	10009	4001	3
11	10010	4006	3
12	10011	4003	1
13	10012	4012	2
14	10013	4010	1

Figure 35: Output of order items table when cancelling order

As seen in the figures, order ID 10014 was deleted, both in order table and order items table because of the ON DELETE CASCADE constraint wherein if any changes were to happen to the main primary key, it would also make changes to the foreign keys in other tables.

6. COUNT NUMBER OF ORDERS FOR SPECIFIC DATE RANGE

SYNTAX:

```
SELECT
    *
FROM
    order_
WHERE
    order_date BETWEEN '2023-4-1' AND '2023-5-31';
```

OUTPUT:

order_id	customer_id	staff_id	order_total_price	order_payment_method	order_date
10000	1000	2001	299.99	Credit Card	2023-04-01
10001	1001	2003	49.99	Cash	2023-04-02
10002	1003	2006	149.99	Credit Card	2023-05-05
10003	1004	2008	399.99	Debit Card	2023-05-08

Figure 36: Output of syntax when counting orders for specific date range

As seen in the above figure, only orders between April 1st, 2023, and May 31st, 2023, are shown. The number of orders for this query is only four.

7. RETRIEVE CUSTOMER DETAILS

SYNTAX:

```
SELECT
    customer.customer_id,
    customer_name,
    email_address,
    contact_number,
    emirate_id,
    area,
    street
FROM
    customer
    JOIN email ON customer.customer_id = email.customer_id
    JOIN address ON customer.customer_id = address.customer_id
    JOIN contact_number ON customer.customer_id =
contact_number.customer_id
WHERE
    customer.customer_id = 1000;
```

OUTPUT:

customer_id	customer_name	email_address	contact_number	emirate_id	area	street
1000	Les Paul Ranalan	les.paul@gmail.com	+971501234567	1	Majaz 2	Jamal Abdul Nasser

Figure 37: Output of syntax when retrieving customer details

As seen above, using the SELECT syntax together with the JOIN syntax, it is possible to retrieve information of customer ID 1000.

8. CHECK ORDER DETAILS ON SPECIFIC DATE, PAYMENTS RECEIVED & ITS REQUIREMENTS

SYNTAX:

```
SELECT
    order_.order_id,
    customer_id,
    staff_id,
    product_id,
    order_product_quantity,
    order_total_price,
    order_payment_method,
    order_date
FROM
    order_
     JOIN order_items ON order_items.order_id = order_.order_id
WHERE
    order_date = '2023-6-10' AND order_payment_method = 'Debit
Card';
```

OUTPUT:

order_id	customer_id	staff_id	product_id	order_product_quantity	order_total_price	order_payment_method	order_date
10004	1002	2005	4009	1	199.99	Debit Card	2023-06-10

Figure 38: Output of syntax when checking order details via specific date, payments received, etc.

As seen in the above figure, the syntax specifies to only show orders on June 1st, 2023, and with payment method of only Debit Card. The output shows only one order.

9. GENERATE REPORT OF PRODUCTS LESS THAN FIVE

SYNTAX:

```
CREATE VIEW products_less_than_5 AS
SELECT
    *
FROM
    product
WHERE
    product_stock_quantity < 5;
```

OUTPUT:

product_id	product_name	product_category	product_price	product_stock_quantity
4003	Corsair K95	Peripherals	149.99	2
4007	Bose QuietComfort 35 II	Headsets	299.99	4
4010	Fujifilm Instax Mini 9	Cameras	99.99	0
4011	Kindle Paperwhite	Tablet	499.99	0
4012	BenQ GW2780	Monitor	699.99	1

Figure 39: Output of syntax when generating report of products less than 5

As seen in the above figure, the syntax specifies to only show products that have less than 5 stock quantity. The output shows 5 products.

10. GENERATE MONTHLY TOTAL SALES REPORT

SYNTAX:

```
CREATE VIEW sales_monthly_report AS
SELECT
    *
FROM
    sales
WHERE
    MONTH(sales_date) = MONTH(CURRENT_DATE()) AND
    YEAR(sales_date) = YEAR(CURRENT_DATE());
```

OUTPUT:

sales_id	product_id	staff_id	sales_quantity	sales_date
1	4000	2000	5	2023-04-01
2	4001	2001	10	2023-04-02

Figure 40: Output of syntax when generating report of monthly sales

As seen in the above figures, the syntax uses `MONTH(CURRENT_DATE)` and `YEAR(CURRENT_DATE)` to retrieve the current date of the system. It is then compared to the sales date column, and finally executes the query. The output shows two sales records.

USER INTERFACE CREATION

For creating a user interface for inserting data, PHP will be used via a text editor or Visual Studio Code. PHP and HTML can be used together to implement user interface and retrieve data from a database. Below is the foundation of an PHP HTML code.

```
<html>

<head>
    <title>My PHP Page</title>
</head>

<body>
    <?php
    // This is a PHP comment
    echo "Hello, world!";
    ?>
</body>

</html>
```

ADDING CUSTOMER DETAILS

INDEX CODE:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>CUSTOMER DETAILS</title>
</head>

<body>
    <center>
        <h1>CUSTOMER DETAILS</h1>
        <p>Enter all necessary customer details below.</p>
        <p>[U+200E]</p>
        <h2>Personal Information</h2>

        <form action="insert_customer.php" method="post">

            <p>
                <label for="FullName">Full Name:</label>
                <input type="text" name="customer_name"
id="c_name">
            </p>

            <p>
                <label for="Email">Email Address:</label>
                <input type="text" name="email_address"
id="email_ad">
            </p>

            <p>
                <label for="ContactNumber">Contact Number
(+971-XX-XXXXXXX):</label>
                <input type="text" name="contact_number"
id="contact_num">
            </p>

            <p>[U+200E]</p>
            <h2>Address Information</h2>

            <p>
                <label for="Emirate">Emirate:</label>
                <select name="emirate_id">
                    <option value=1>Sharjah</option>
                    <option value=2>Dubai</option>
```

```
                    <option value=3>Ras Al Khaimah</option>
                    <option value=4>Abu Dhabi</option>
                    <option value=5>Ajman</option>
                    <option value=6>Umm Al Quwain</option>
                    <option value=7>Fujairah</option>
                </select>
            </p>

            <p>
                <label for="Area">Area:</label>
                <input type="text" name="area" id="ad_area">
            </p>

            <p>
                <label for="Street">Street:</label>
                <input type="text" name="street"
id="ad_street">
            </p>

            <p>[U+200E]</p>

            <input type="submit" value="INSERT DETAILS">
        </form>
    </center>
</body>

</html>
```

OUTPUT:

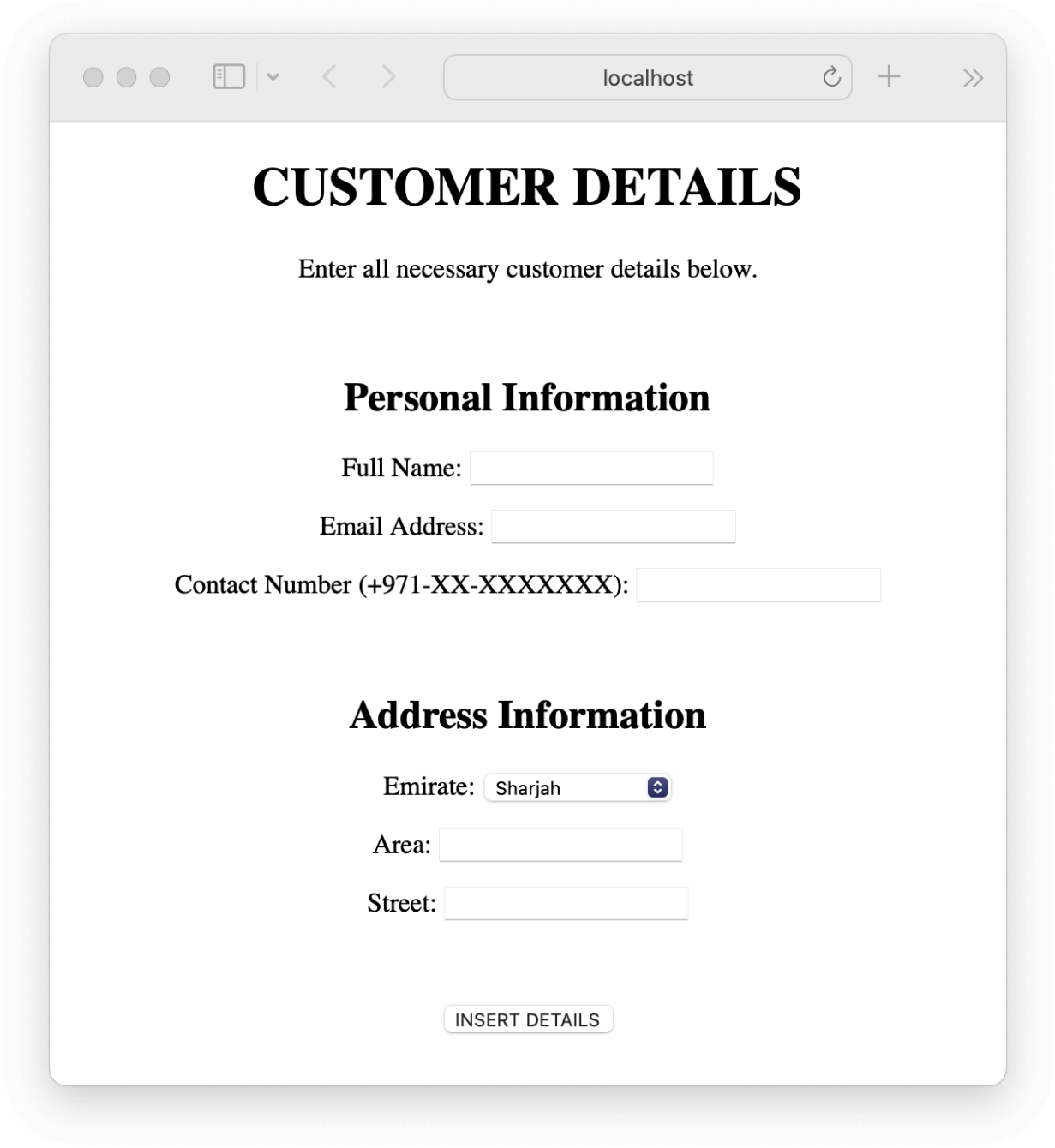


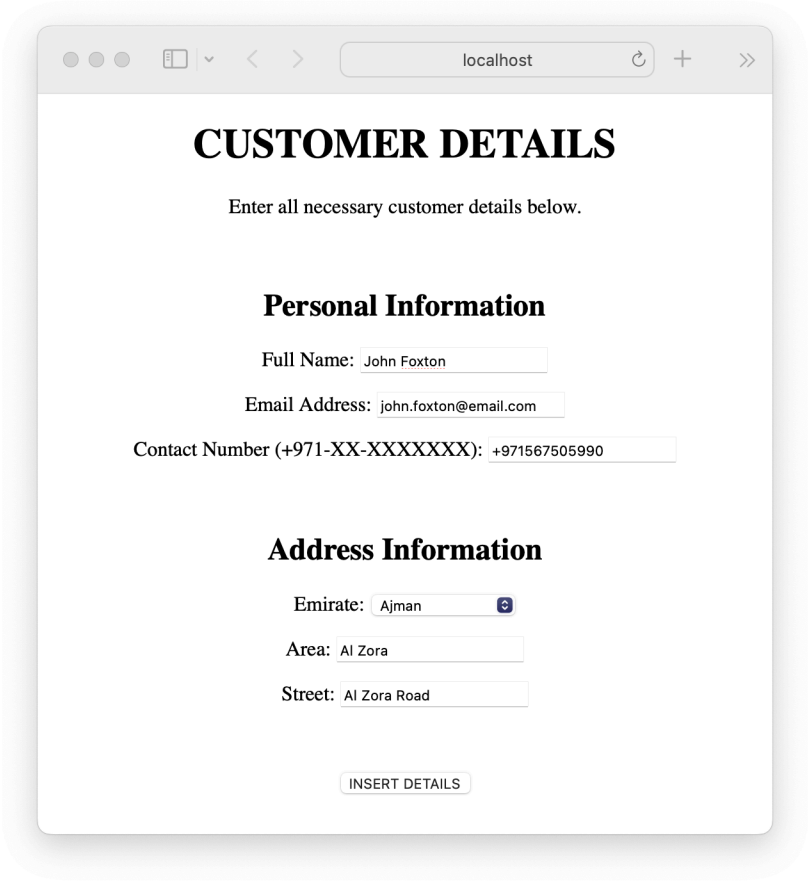
Figure 41: Output of customer_index.php when executed

INSERT CODE:

```
<!DOCTYPE html>
<html>

<head>
    <title>CUSTOMER PLUG-INS</title>
</head>

<body>
    <center>
        <?php
        $conn = mysqli_connect("localhost", "root", "", "plug-
ins");

        if ($conn === false) {
            die("ERROR: Could not connect. "
                . mysqli_connect_error());
        }

        $customer_name = $_POST['customer_name'];
        $email_address = $_POST['email_address'];
        $contact_number = $_POST['contact_number'];
        $emirate_id = $_POST['emirate_id'];
        $area = $_POST['area'];
        $street = $_POST['street'];

        // Insert data into the customer table
        $c_sql = "INSERT INTO customer (customer_name) VALUES
('$customer_name')";
        if (mysqli_query($conn, $c_sql)) {
            $customer_id = mysqli_insert_id($conn);

            // Insert data into address, email, contact number
            $a_sql = "INSERT INTO address (customer_id,
emirate_id, area, street) VALUES ($customer_id, $emirate_id,
'$area', '$street')";
            $e_sql = "INSERT INTO email (customer_id,
email_address) VALUES ($customer_id, '$email_address')";
            $n_sql = "INSERT INTO contact_number (customer_id,
contact_number) VALUES ($customer_id, '$contact_number')";

            mysqli_query($conn, $a_sql) && mysqli_query($conn,
$e_sql) && mysqli_query($conn, $n_sql);

            echo "<h3>CUSTOMER ADDED SUCCESSFULLY</h3>";
            echo nl2br("\n Full Name: $customer_name\n Email
```

```
Address: $email_address\n Contact Number: $contact_number\n
Emirate: $emirate_id\n Area: $area\n Street: $street");
        } else {
            echo "ERROR. "
                . mysqli_error($conn);
        }

        // Close connection
        mysqli_close($conn);
        ?>
    </center>
</body>

</html>
```

OUTPUT:



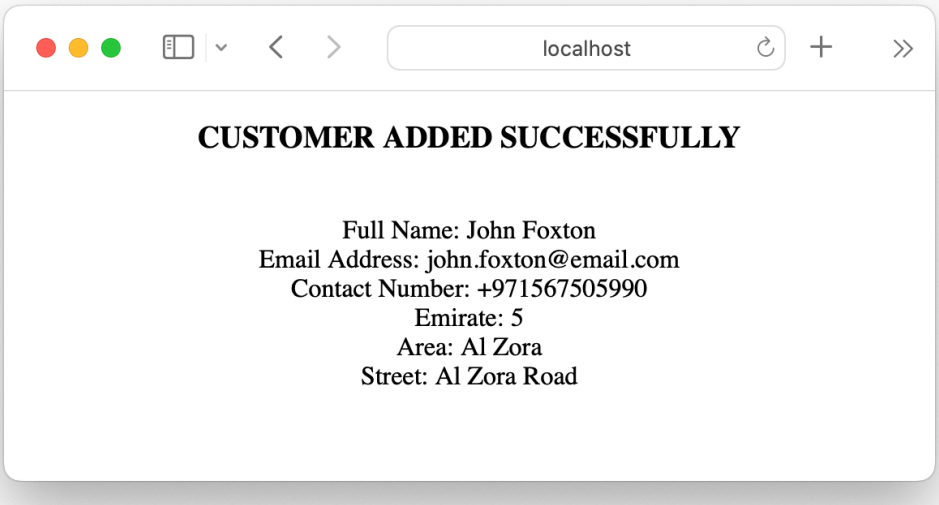


Figure 42: Outputs of insert_customer.php when executed

customer_id ▼ 1	customer_name
1005	John Foxton

contact_number_id ▼ 1	customer_id	staff_id	supplier_id	contact_number
21	1005	*NULL*	*NULL*	+971567505990

customer_id	staff_id	supplier_id	emirate_id	area	street
1005	*NULL*	*NULL*	5	Al Zora	Al Zora Road

email_id ▼ 1	customer_id	staff_id	supplier_id	email_address
21	1005	*NULL*	*NULL*	john.foxton@email.com

Figure 43: Outputs of database tables when insert_customer.php was executed

PLACING ORDER

INDEX CODE:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>ORDER DETAILS</title>
</head>

<body>
    <center>
        <h1>ORDER DETAILS</h1>

        <p>Enter all necessary order details below.</p>

        <p>[U+200E]</p>

        <p>
        <form action="insert_order.php" method="post">

            <label for="customer">Customer:</label>
            <select name="customer_id">
                <?php
                $conn = mysqli_connect("localhost", "root", "",
"plug-ins");
                $stmt = mysqli_prepare($conn, "SELECT
customer_id, customer_name FROM customer");
                mysqli_stmt_execute($stmt);
                mysqli_stmt_bind_result($stmt, $customer_id,
$customer_name);

                while (mysqli_stmt_fetch($stmt)) {
                    echo "<option
value=\"$customer_id\">$customer_name</option>";
                }

                mysqli_stmt_close($stmt);
                mysqli_close($conn);
                ?>
            </select>

            </p>

            <p>
                <label for="staff">Staff:</label>
```

```
                <select name="staff_id">
                    <?php
                    $conn = mysqli_connect("localhost", "root",
"", "plug-ins");
                    $stmt = mysqli_prepare($conn, "SELECT
staff_id, staff_name FROM staff");
                    mysqli_stmt_execute($stmt);
                    mysqli_stmt_bind_result($stmt, $staff_id,
$staff_name);

                    while (mysqli_stmt_fetch($stmt)) {
                        echo "<option
value=\"$staff_id\">$staff_name</option>";
                    }

                    mysqli_stmt_close($stmt);
                    mysqli_close($conn);
                    ?>
                </select>
            </p>

            <p>
                <label for="product">Product:</label>
                <select name="product_id">
                    <?php
                    $conn = mysqli_connect("localhost", "root",
"", "plug-ins");
                    $stmt = mysqli_prepare($conn, "SELECT
product_id, product_name, product_price FROM product");
                    mysqli_stmt_execute($stmt);
                    mysqli_stmt_bind_result($stmt, $product_id,
$product_name, $product_price);

                    while (mysqli_stmt_fetch($stmt)) {
                        echo "<option
value=\"$product_id\">$product_name - $product_price</option>";
                    }

                    mysqli_stmt_close($stmt);
                    mysqli_close($conn);
                    ?>
                </select>
            </p>

            <p>
                <label for="Quantity">Product Quantity:</label>
                <input type="text"
```

```
name="order_product_quantity" id="o_quantity">
            </p>

            <p>
                <label for="TotalAmount">Order Total
Price:</label>
                <input type="text" name="order_total_price"
id="o_total">
            </p>

            <p>
                <label for="PaymentMethod">Payment
Method:</label>
                <select name="order_payment_method">
                    <option value='Cash'>Cash</option>
                    <option value='Debit Card'>Debit
Card</option>
                    <option value='Credit Card'>Credit
Card</option>
                </select>
            </p>

            <p>
                <label for="Date">Date (YYYY-MM-DD):</label>
                <input type="text" name="order_date"
id="o_date">
            </p>

            <p>[U+200E]</p>

            <input type="submit" value="PLACE ORDER">
        </form>
    </center>
</body>

</html>
```

In the code above, $stmt selects the specific columns of a table and binds those values to variables in the code, via mysqli_stmt_bind_result(). It will then output the result in an HTML format select option or dropdown list via echo. This will then loop, via while loop with mysqli_stmt_fetch(), until there are no more values to be outputted.

OUTPUT:

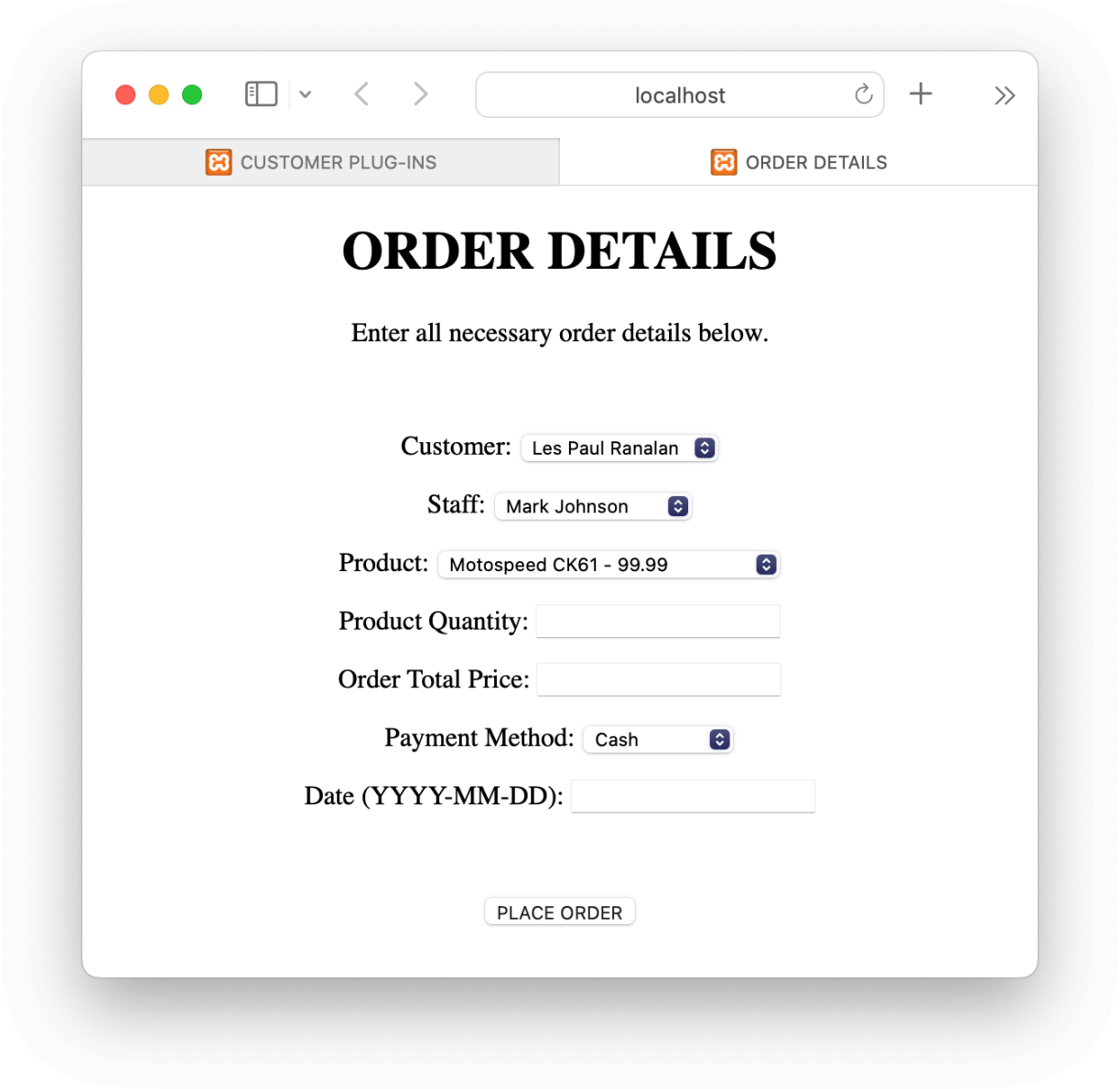


Figure 44: Output of index_order.php when executed

INSERT CODE:

```
<!DOCTYPE html>
<html>

<head>
    <title>ORDER PLUG-INS</title>
</head>

<body>
    <center>
        <?php
        $conn = mysqli_connect("localhost", "root", "", "plug-
ins");

        if ($conn === false) {
            die("ERROR: Could not connect. "
                . mysqli_connect_error());
        }

        $customer_id = $_POST['customer_id'];
        $staff_id = $_POST['staff_id'];
        $product_id = $_POST['product_id'];
        $order_product_quantity =
$_POST['order_product_quantity'];
        $order_total_price = $_POST['order_total_price'];
        $order_payment_method = $_POST['order_payment_method'];
        $order_date = $_POST['order_date'];

        // Insert data into the order table
        $o_sql = "INSERT INTO order_ (customer_id, staff_id,
order_total_price, order_payment_method, order_date) VALUES
('$customer_id', '$staff_id', '$order_total_price',
'$order_payment_method', '$order_date')";
        if (mysqli_query($conn, $o_sql)) {
            $order_id = mysqli_insert_id($conn);

            // Insert data into order_items table
            $oi_sql = "INSERT INTO order_items (order_id,
product_id, order_product_quantity) VALUES ('$order_id',
'$product_id', '$order_product_quantity')";

            mysqli_query($conn, $oi_sql);

            echo "<h3>ORDER PLACED SUCCESSFULLY</h3>";
            echo nl2br("\n Customer ID: $customer_id\n Staff
ID: $staff_id\n Product ID: $product_id\n Product Quantity:
```

```
$order_product_quantity\n Order Total: $order_total_price\n
Payment Method: $order_payment_method\n Order Date:
$order_date");
        } else {
            echo "ERROR. "
                . mysqli_error($conn);
        }

        // Close connection
        mysqli_close($conn);
        ?>
    </center>
</body>

</html>
```

OUTPUT:

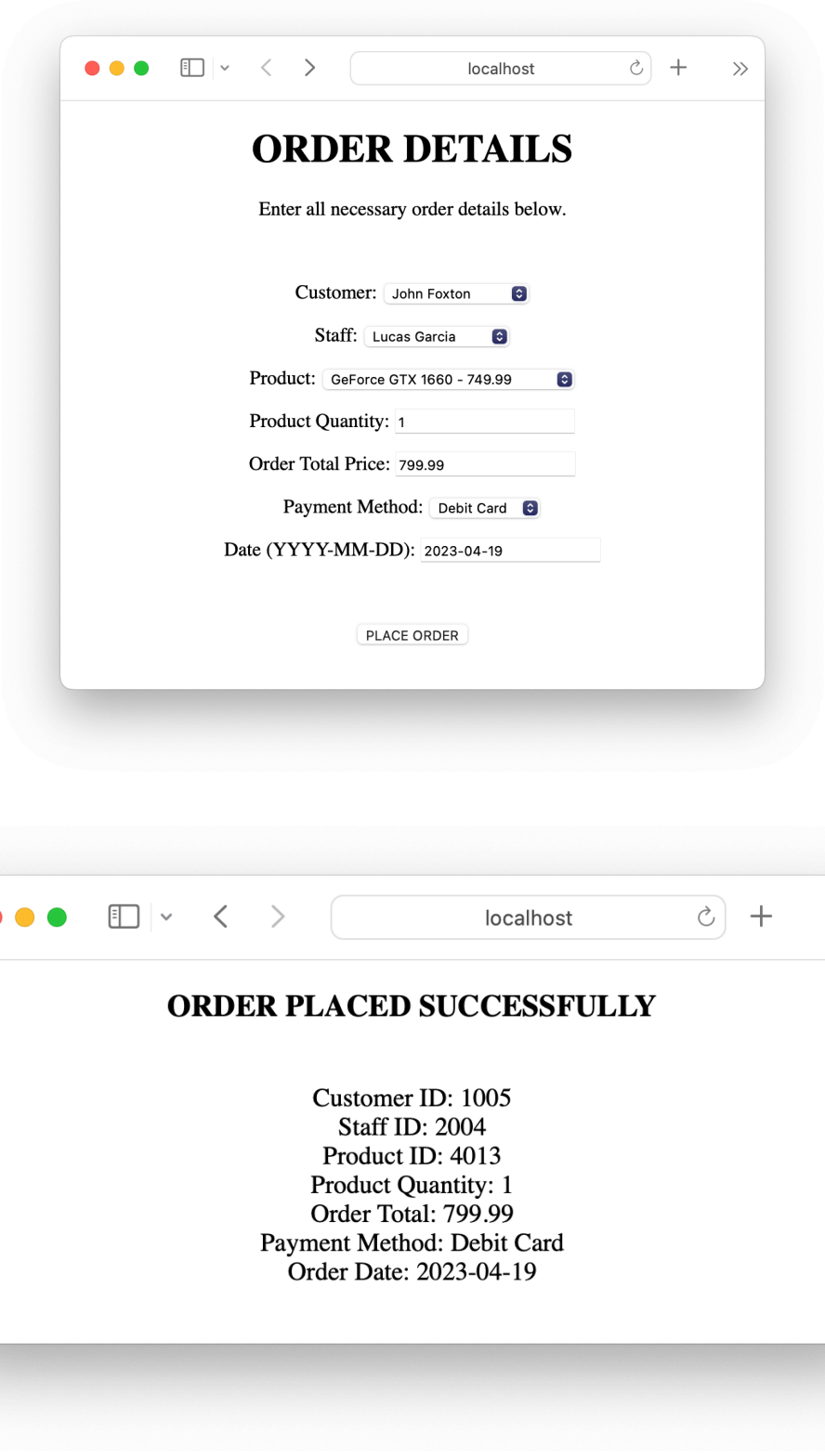


Figure 45: Outputs of insert_order.php when executed

order_id ▼ 1	customer_id	staff_id	order_total_price	order_payment_method	order_date
10018	1005	2004	799.99	Debit Card	2023-04-19

serial_id ▼ 1	order_id	product_id	order_product_quantity
18	10018	4013	1

Figure 46: Output of database tables when insert_order.php was executed

SEARCH FOR PRODUCT VIA NAME OR CATEGORY

SEARCH CODE:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>PRODUCT PLUG-INS</title>
</head>

<body>
    <center>
        <h1>SEARCH PRODUCT</h1>
        <p>Select a search method, and enter keyword to search
for products.</p>
        <p>[U+200E]</p>

        <form action="display_product.php" method="post">

            <p>
                <label for="SearchVia">Search via:</label>
                <select name="search_id">
                    <option value=1>Product Name</option>
                    <option value=2>Product Category</option>
                </select>
            </p>

            <p>
                <label for="name">Keyword:</label>
                <input type="text" name="name" id="p_name">
            </p>

            <input type="submit" value="SEARCH">
        </form>
</body>

</html>
```

OUTPUT:

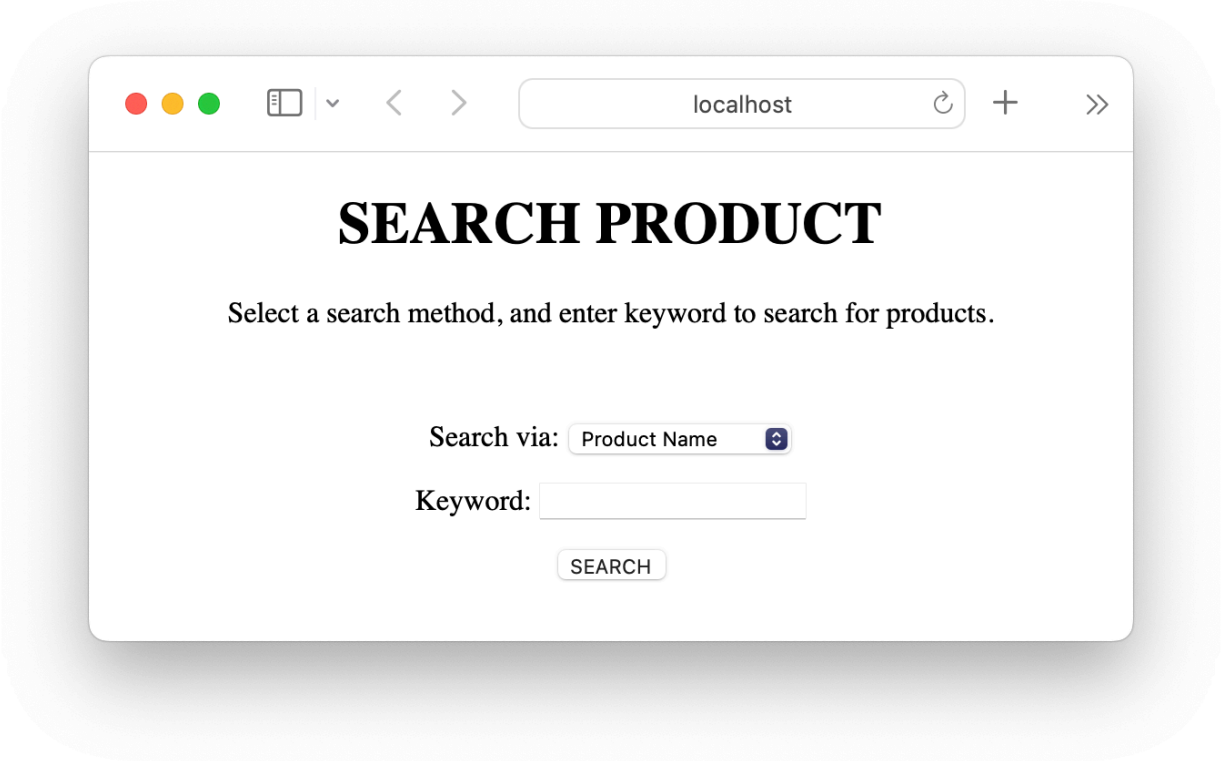


Figure 47: Output of search_product.php when executed

DISPLAY CODE:

```
<table border="2">
  <tr>
    <td>product_id</td>
    <td>product_name</td>
    <td>product_category</td>
    <td>product_price</td>
    <td>product_stock_quantity</td>
    <td>product_availability</td>
  </tr>

  <?php
  $host = "localhost";
  $user = "root";
  $pass = "";
  $db = "plug-ins";

  $conn = mysqli_connect($host, $user, $pass, $db);
  if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
  }

  $search_id = $_POST['search_id'];
  $name = $_POST['name'];

  if ($search_id == 1) {
    $n_sql = "select * from product where product_name LIKE
'%$name%'";
    $n_data = mysqli_query($conn, $n_sql);
    while ($n_info = mysqli_fetch_array($n_data)) {
  ?>
      <tr>
        <td> <?php echo  $n_info['product_id']; ?> </td>
        <td> <?php echo  $n_info['product_name'];   ?> </td>
        <td> <?php echo  $n_info['product_category'];    ?>
</td>
        <td> <?php echo  $n_info['product_price'];   ?> </td>
        <td> <?php echo $n_info['product_stock_quantity'];
?> </td>
        <td> <?php echo $n_info['product_availability'];    ?>
</td>
      </tr>
    <?php
    }
  }
```

```
  else {
    $c_sql = "select * from product where product_category LIKE
'%$name%'";
    $c_data = mysqli_query($conn, $c_sql);
    while ($c_info = mysqli_fetch_array($c_data)) {
    ?>
      <tr>
        <td> <?php echo  $c_info['product_id']; ?> </td>
        <td> <?php echo  $c_info['product_name'];   ?> </td>
        <td> <?php echo  $c_info['product_category'];    ?>
</td>
        <td> <?php echo  $c_info['product_price'];   ?> </td>
        <td> <?php echo $c_info['product_stock_quantity'];
?> </td>
        <td> <?php echo $c_info['product_availability'];    ?>
</td>
      </tr>
  <?php
    }
  }

  ?>
</table>
```

OUTPUT:

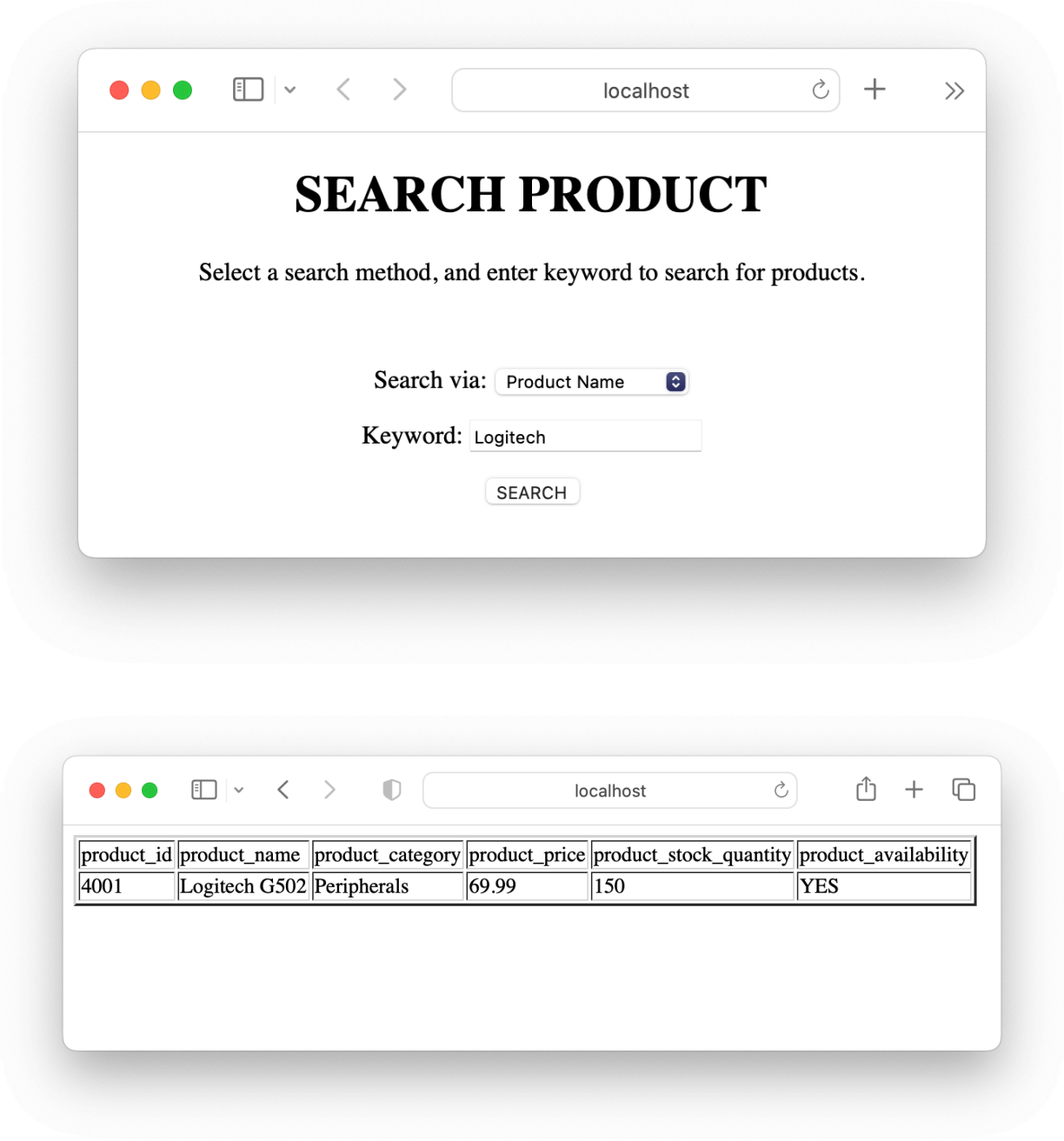
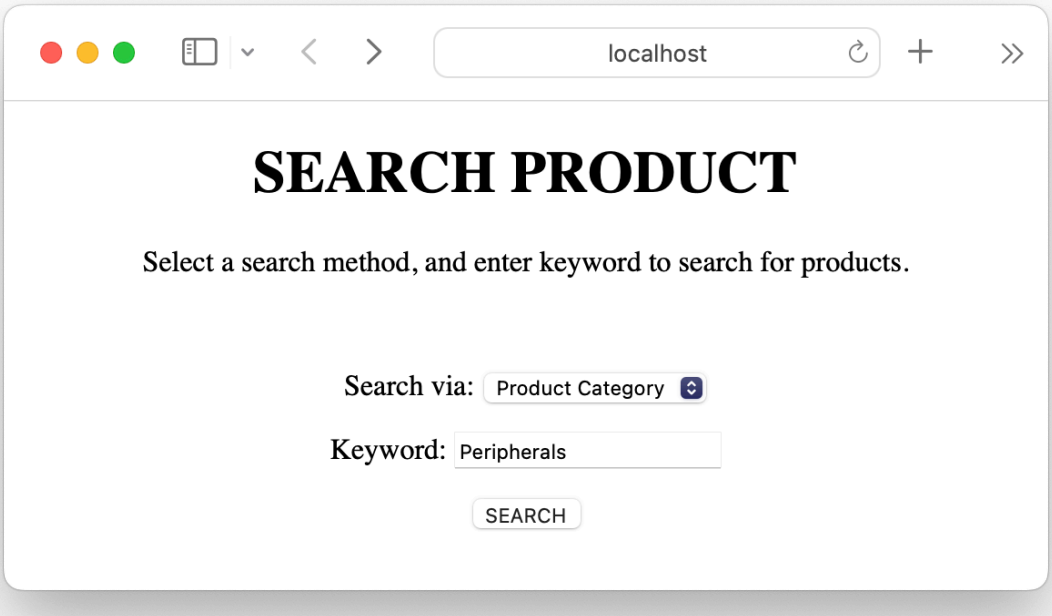
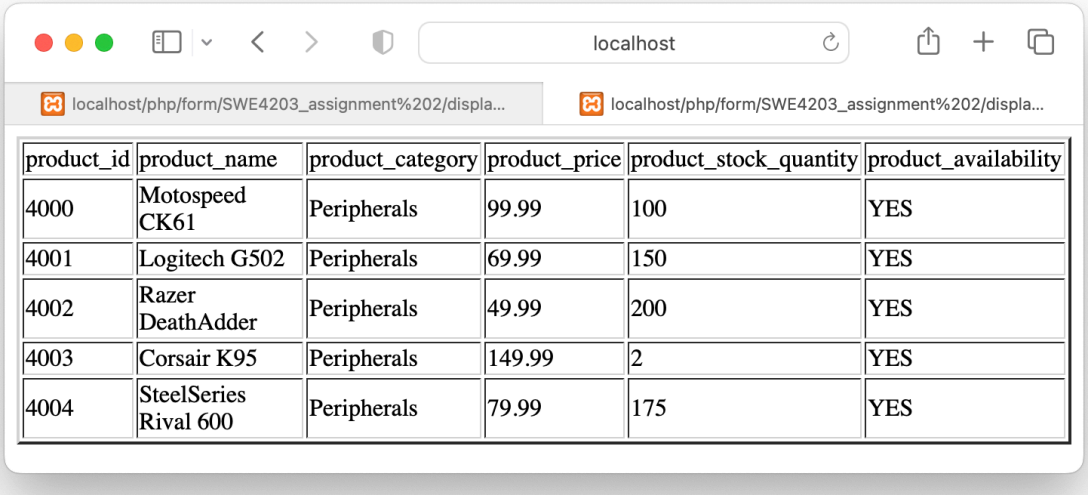


Figure 48: Output of display_product.php when executed via searching product name, Logitech





product_id	product_name	product_category	product_price	product_stock_quantity	product_availability
4000	Motospeed CK61	Peripherals	99.99	100	YES
4001	Logitech G502	Peripherals	69.99	150	YES
4002	Razer DeathAdder	Peripherals	49.99	200	YES
4003	Corsair K95	Peripherals	149.99	2	YES
4004	SteelSeries Rival 600	Peripherals	79.99	175	YES

Figure 49: Output of display_product.php when executed via searching product category, Peripherals

GENERATE DAILY ORDERS PLACED REPORT FOR SPECIFIC DAY

SEARCH CODE:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>ORDER PLUG-INS</title>
</head>

<body>
    <center>
        <h1>ORDER DAILY REPORT</h1>
        <p>Enter date to generate report of orders placed on
that specific date.</p>
        <p>[U+200E]</p>

        <form action="display_order.php" method="post">
            <p>
                <label for="Date">Date (YYYY-MM-DD):</label>
                <input type="text" name="order_date"
id="o_date">
            </p>
            <input type="submit" value="GENERATE">
        </form>
</body>

</html>
```

OUTPUT:

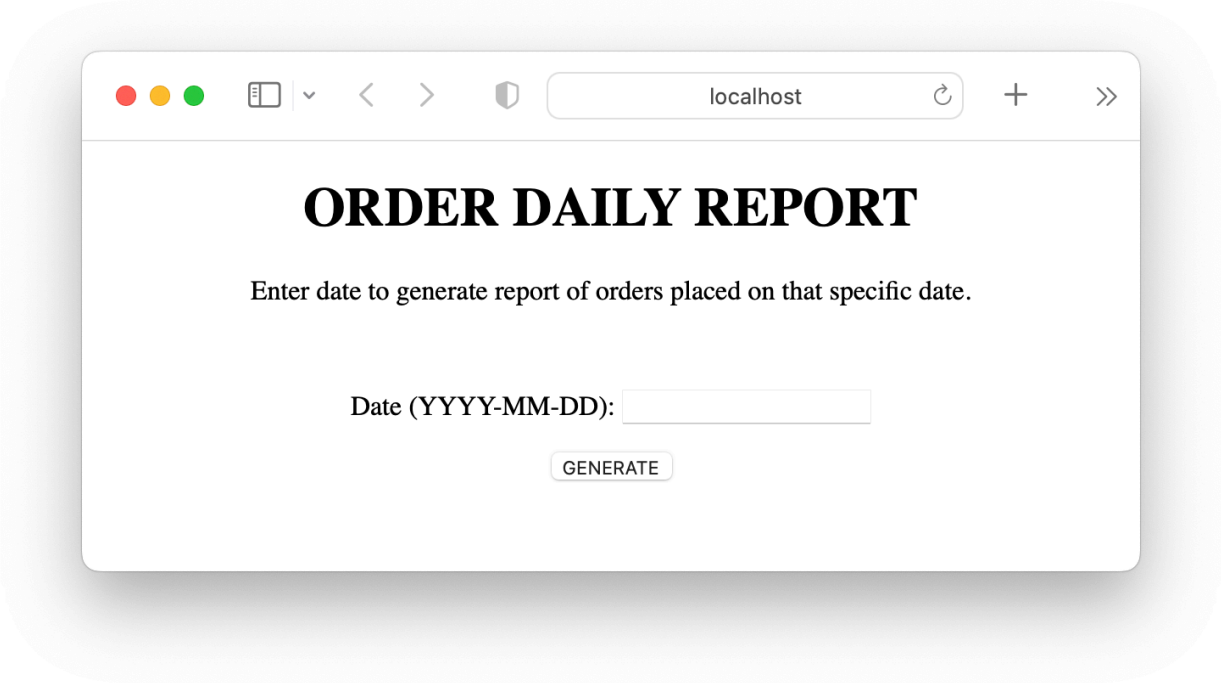


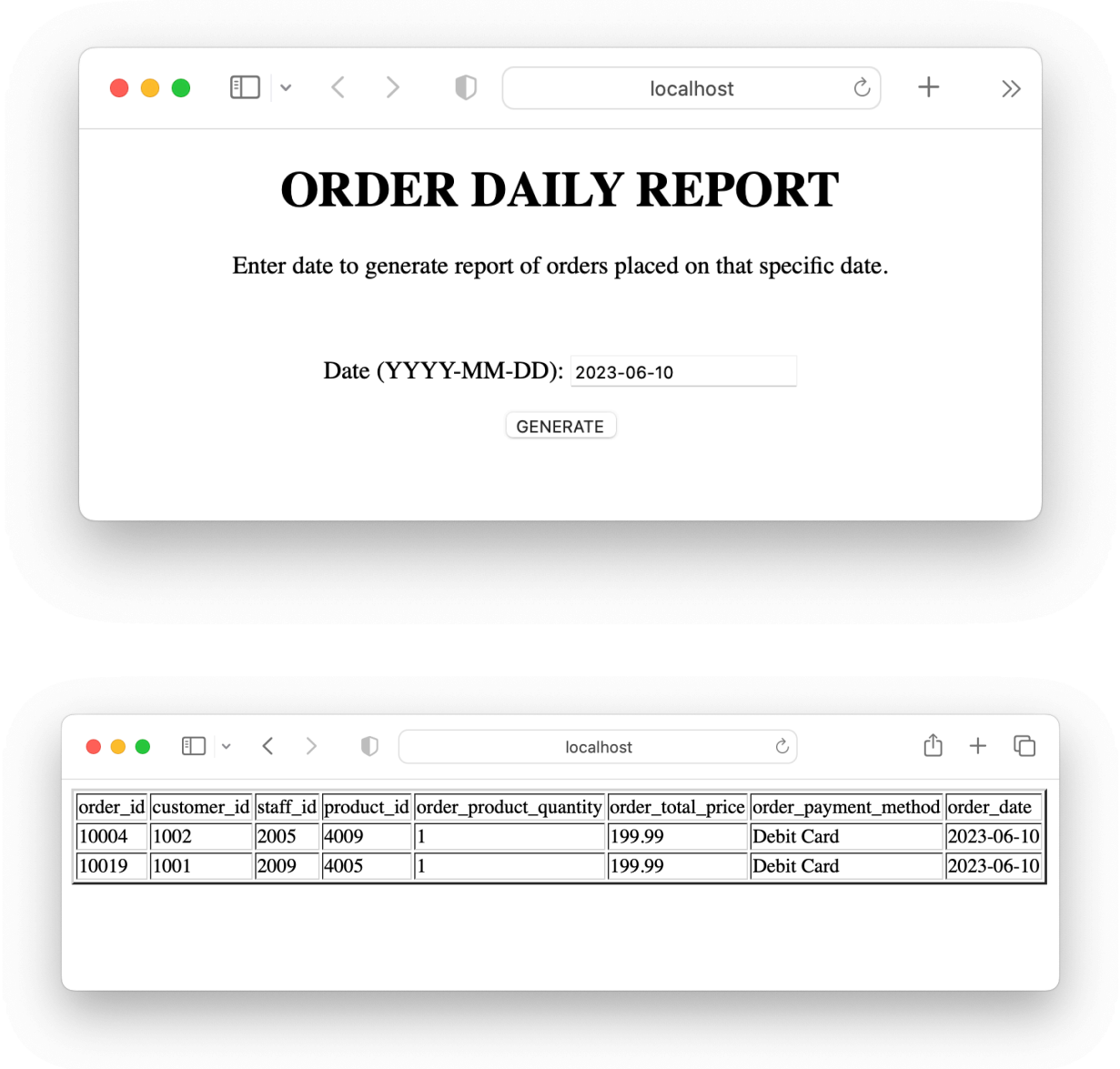
Figure 50: Output of search_order.php when executed

DISPLAY CODE:

```
<table border="2">
  <tr>
    <td>order_id</td>
    <td>customer_id</td>
    <td>staff_id</td>
    <td>product_id</td>
    <td>order_product_quantity</td>
    <td>order_total_price</td>
    <td>order_payment_method</td>
    <td>order_date</td>
  </tr>
  <?php
  $host = "localhost";
  $user = "root";
  $pass = "";
  $db = "plug-ins";

  $conn = mysqli_connect($host, $user, $pass, $db);
  if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
  }

  $order_date = $_POST['order_date'];

  $sql = "select order_.order_id, customer_id, staff_id,
product_id, order_product_quantity, order_total_price,
order_payment_method, order_date from order_ JOIN order_items
ON order_items.order_id = order_.order_id WHERE order_date =
'$order_date'";
  $data = mysqli_query($conn, $sql);
  while ($info = mysqli_fetch_array($data)) {
  ?>
    <tr>
      <td> <?php echo  $info['order_id']; ?> </td>
      <td> <?php echo  $info['customer_id'];   ?> </td>
      <td> <?php echo  $info['staff_id'];    ?> </td>
      <td> <?php echo  $info['product_id'];   ?> </td>
      <td> <?php echo $info['order_product_quantity'];    ?>
</td>
      <td> <?php echo $info['order_total_price'];    ?> </td>
      <td> <?php echo $info['order_payment_method'];    ?>
</td>
      <td> <?php echo $info['order_date'];    ?> </td>
    </tr>
  <?php
```

```
    }

    ?>
</table>
```

OUTPUT:



order_id	customer_id	staff_id	product_id	order_product_quantity	order_total_price	order_payment_method	order_date
10004	1002	2005	4009	1	199.99	Debit Card	2023-06-10
10019	1001	2009	4005	1	199.99	Debit Card	2023-06-10

Figure 51: Output of display_order.php when executed via searching for specific date

DOCUMENTATION

It refers to the process of recording and organizing information about the database design, function, structure, and maintenance. This includes constraints, security, normalization, and database relationship diagrams.

DESIGNER VIEW

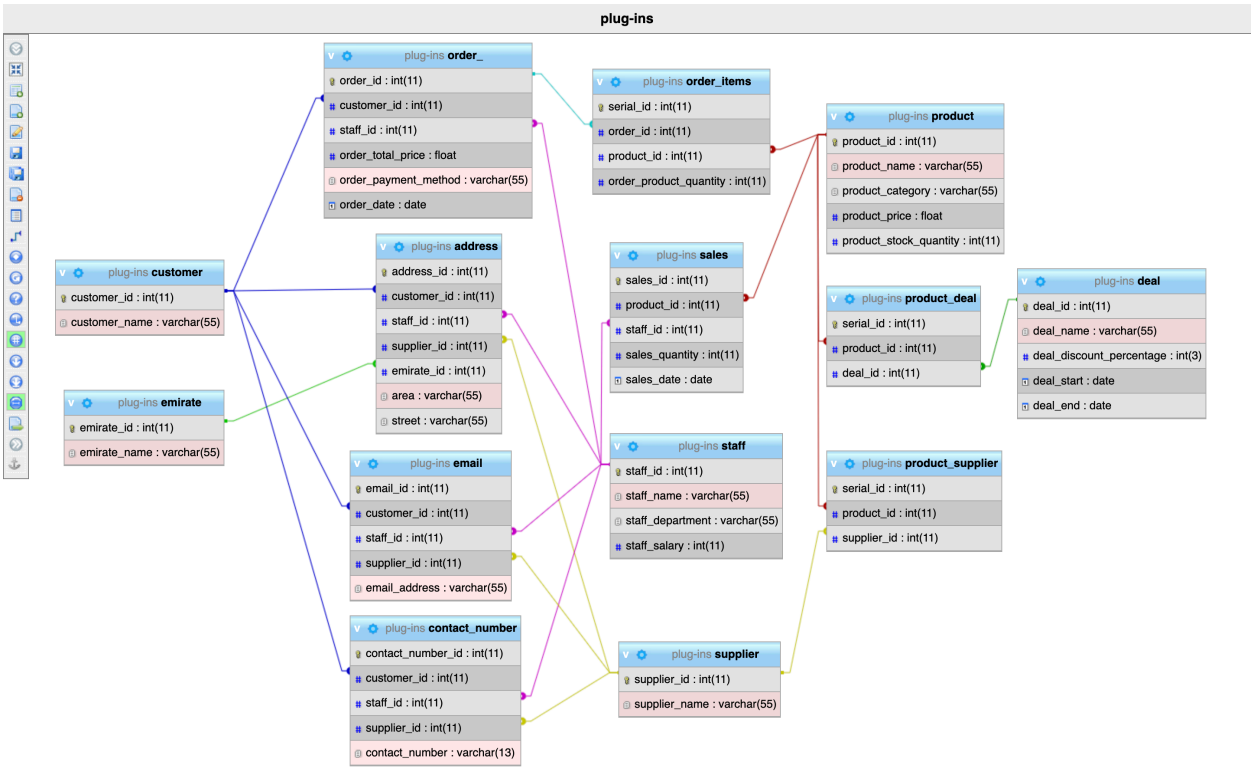


Figure 52: Designer view of plug-ins database

As seen in the diagram above, the designer view provides a convenient way to visually design and modify the database schemas, without the use of SQL code. The above figure shows all the attributes of each table that are present in the database, an attribute with a key icon represents a primary key while the rest of the attributes represent fields that are

to be entered. The lines are color-coordinated and represents the relationships of each tables utilizing the foreign key. Example: customer ID of customer table on the left has relationships in order, address, contact number, and email tables, all of which corresponds to customer ID of each table. In this case, the relationships are displayed via blue lines. Same applies to every other table that have relationships which are displayed in different colors.

INTEGRITY CONSTRAINTS

The documentation of the integrity constraints of database are already mentioned and explained in-depth in the previous chapters, this also includes screenshots of the syntax. To summarize: Primary Key, Foreign Key, Check, NOT NULL, and AUTO_INCREMENT are the constraints that were used in the database. Below are the constraints being tested.

PRIMARY KEY:

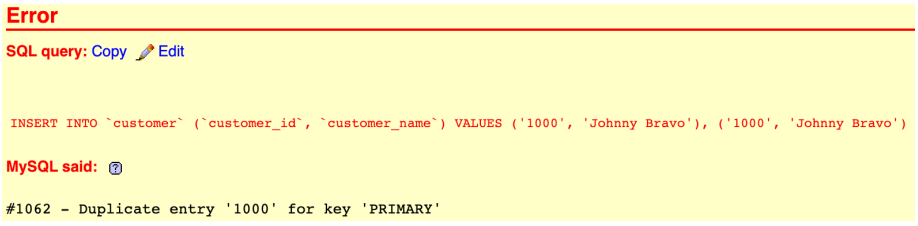


Figure 53: Error message of Primary Key constraint

Error occurs when trying to insert a duplicate entry in the table. In this case, it is customer ID, 1000.

FOREIGN KEY:

Error

SQL query: Copy

```
INSERT INTO product_deal (product_id, deal_id) VALUES (4014, 50123);
```

MySQL said:

```
#1452 - Cannot add or update a child row: a foreign key constraint fails (`plug-ins`.`product_deal`, CONSTRAINT `product_deal_ibfk_1` FOREIGN KEY (`product_id`) REFERENCES `product` (`product_id`) ON DELETE CASCADE ON UPDATE CASCADE)
```

Figure 54: Error message of Foreign Key constraint

Error occurs when inserting IDs (4014 and 50123) that does not exist in parent tables, therefore resulting in Foreign Key constraint error.

CHECK:

Column	Type	Function	Null	Value
contact_number_id	int(11)			
customer_id	int(11)		☑	
staff_id	int(11)		☐	Emma Davis - 2001
supplier_id	int(11)		☐	XYZ Inc. - 3001
contact_number	varchar(13)			+97156750599(

Figure 55: INSERT tab of contact number table

Error

SQL query: Copy Edit

```
INSERT INTO `contact_number` (`contact_
```

MySQL said:

```
#4025 - CONSTRAINT `CONSTRAINT_1` failed for `plug-ins`.`contact_number`
```

Figure 56: Error of CHECK constraint

Error occurs when entering two or more IDs as one row into contact number table. This is because of CHECK constraint wherein it only allows one ID to be entered per row.

NOT NULL:

Error

SQL query: Copy

```
INSERT INTO supplier VALUES (3006, NULL);
```

MySQL said:

```
#1048 - Column 'supplier_name' cannot be null
```

Error occurs when inserting a NULL value in supplier name which has the NOT NULL constraint.

Figure 57: Error message of NOT NULL constraint

AUTO INCREMENT:

Column	Type	Function	Null	Value
customer_id	int(11)			
customer_name	varchar(55)			Flery Sampang

Figure 58: INSERT tab of customer table

1 row inserted.
Inserted row id: 1009

Figure 59: Success message with AUTO INCREMENT

With AUTO INCREMENT of the Primary Key, there is no need to specify the customer ID value as it is automatically assigned by the database. This makes the database easier to insert new data without the need to remember which IDs are already taken.

NORMALIZATION

It is the process of organizing data in a database so that it is consistent, efficient, and easier to manage. It is a technique used to eliminate redundant data, this is done by breaking large tables into smaller tables, making it more manageable.

In this database, each table must be normalized up to third normal form (3NF) for easy querying of data. The first step would be to decompose tables into 1NF, secondly 2NF, and finally 3NF. Below is all the table structure of each normal form.

CUSTOMER

0NF

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**customer_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**customer_contact_number**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
5	**customer_address**	varchar(100)	utf8mb4_general_ci		Yes	*NULL*		

Figure 60: 0NF of customer table

Unnormalized form of customer table.

1NF

Customer table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**customer_emirate**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**customer_area**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
5	**customer_street**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Contact number table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_contact_number**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Email table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 61: 1NF of customer table

Customer address was split into 3 columns, namely: emirate, area, and street.

Customer email and contact numbers were moved to separate tables. This is done because every cell should be atomic, containing only one value.

2NF

Customer table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Contact number table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_contact_number**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Email table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Address table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_street**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**customer_area**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**customer_emirate**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 62: 2NF of customer table

Customer address details were separated into a new table. No composite key is present in the table, thus there is no partial dependency.

3NF

Customer table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Contact number table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_contact_number**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Email table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Address table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**customer_id**	int(11)			Yes	*NULL*		
2	**customer_street**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**customer_area**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**emirate_id**	int(11)			Yes	*NULL*		

Figure 63: 3NF of customer table

Customer emirate was changed to emirate ID and acts as a foreign key to the emirate table. This is done because of a transitive dependency between area and street.

STAFF

0NF

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**staff_contact_number**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**staff_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
5	**staff_department**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
6	**staff_salary**	float			Yes	*NULL*		
7	**staff_address**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 64: 0NF of staff table

Unnormalized form of staff table.

1NF

Staff table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**staff_department**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**staff_salary**	float			Yes	*NULL*		
5	**staff_emirate**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
6	**staff_area**	varchar(55)	utf8mb4_general_ci		No	*None*		
7	**staff_street**	varchar(55)	utf8mb4_general_ci		No	*None*		

Email table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Contact number table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_contact_number**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 65: 1NF of staff table

Staff email and contact number details were moved to separate tables.

Staff address was split into 3 columns. This is to make sure every cell is atomic.

2NF

Staff table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**staff_department**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**staff_salary**	float			Yes	*NULL*		

Email table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Contact number table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_contact_number**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Address table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_emirate**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**staff_area**	varchar(55)	utf8mb4_general_ci		No	*None*		
4	**staff_street**	varchar(55)	utf8mb4_general_ci		No	*None*		

Figure 66: 2NF of staff table

Staff address details were moved to the address table. There is no composite key present, thus there is no partial dependency.

3NF

Staff table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**staff_department**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**staff_salary**	float			Yes	*NULL*		

Email table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Contact number table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**staff_contact_number**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Address table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**staff_id**	int(11)			Yes	*NULL*		
2	**emirate_id**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**staff_area**	varchar(55)	utf8mb4_general_ci		No	*None*		
4	**staff_street**	varchar(55)	utf8mb4_general_ci		No	*None*		

Figure 67: 3NF of staff table

Staff emirate was changed to emirate ID as a foreign key referencing to the emirate table. This is done because of transitive dependency between area, street, and emirate.

SUPPLIER

0NF

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**supplier_id**	int(11)			Yes	*NULL*		
2	**supplier_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**supplier_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**supplier_contact_number**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
5	**supplier_address**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 68: 0NF of supplier table

Unnormalized form of supplier table

1NF

Supplier table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**supplier_id**	int(11)			Yes	*NULL*		
2	**supplier_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**supplier_emirate**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**supplier_area**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
5	**supplier_street**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Email table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**supplier_id**	int(11)			Yes	*NULL*		
2	**supplier_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Contact number table

Name	Type	Collation	Attributes	Null	Default	Comments
supplier_id	int(11)			Yes	*NULL*	
supplier_contact_number	varchar(55)	utf8mb4_general_ci		Yes	*NULL*	

Figure 69: 1NF of supplier table

Supplier contact number and email was separated into contact number and email table. Supplier address details was split into 3 columns. This is done because every cell should be atomic.

2NF

Supplier table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**supplier_id**	int(11)			Yes	*NULL*		
2	**supplier_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Email table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**supplier_id**	int(11)			Yes	*NULL*		
2	**supplier_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Contact number table

Name	Type	Collation	Attributes	Null	Default	Comments
supplier_id	int(11)			Yes	*NULL*	
supplier_contact_number	varchar(55)	utf8mb4_general_ci		Yes	*NULL*	

Address table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**supplier_id**	int(11)			Yes	*NULL*		
2	**supplier_emirate**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**supplier_area**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**supplier_street**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 70: 2NF of supplier table

Staff address details was separated to address table. No composite key is present, thus partial dependency is non-existent.

3NF

Supplier table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**supplier_id**	int(11)			Yes	*NULL*		
2	**supplier_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Email table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**supplier_id**	int(11)			Yes	*NULL*		
2	**supplier_email**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Contact number table

Name	Type	Collation	Attributes	Null	Default	Comments
supplier_id	int(11)			Yes	*NULL*	
supplier_contact_number	varchar(55)	utf8mb4_general_ci		Yes	*NULL*	

Address table

Name	Type	Collation	Attributes	Null	Default	Comments	Extra
supplier_id	int(11)			Yes	*NULL*		
emirate_id	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
supplier_area	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
supplier_street	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 71: 3NF of supplier table

Supplier emirate was changed to emirate ID, acting as a foreign key to reference to emirate table. This is done because there is a transitive dependency between street, area, and emirate.

PRODUCT

0NF

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**product_id**	int(11)			Yes	*NULL*		
2	**supplier_id**	int(11)			Yes	*NULL*		
3	**deal_id**	int(11)			Yes	*NULL*		
4	**product_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
5	**product_category**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
6	**product_price**	int(11)			Yes	*NULL*		
7	**product_stock_quantity**	int(11)			Yes	*NULL*		
8	**product_availability**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 72: 0NF of product table

Unnormalized form of product table

1NF

Product table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**product_id**	int(11)			Yes	*NULL*		
2	**product_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**product_category**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**product_price**	int(11)			Yes	*NULL*		
5	**product_stock_quantity**	int(11)			Yes	*NULL*		
6	**product_availability**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Product supplier table

Name	Type	Collation	Attributes	Null	Default	Comments	Extra
product_id	int(11)			Yes	*NULL*		
supplier_id	int(11)			Yes	*NULL*		

Product deal table

Name	Type	Collation	Attributes	Null	Default	Comments	Extra
product_id	int(11)			Yes	*NULL*		
deal_id	int(11)			Yes	*NULL*		

Figure 73: 1NF of product table

Foreign keys deal ID and supplier ID were separated into new tables. This is done to reduce data redundancy and that every cell should be atomic.

2NF

Product table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**product_id**	int(11)			Yes	*NULL*		
2	**product_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**product_category**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**product_price**	int(11)			Yes	*NULL*		
5	**product_stock_quantity**	int(11)			Yes	*NULL*		
6	**product_availability**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Product supplier table

Name	Type	Collation	Attributes	Null	Default	Comments	Extra
product_id	int(11)			Yes	*NULL*		
supplier_id	int(11)			Yes	*NULL*		

Product deal table

Name	Type	Collation	Attributes	Null	Default	Comments	Extra
product_id	int(11)			Yes	*NULL*		
deal_id	int(11)			Yes	*NULL*		

Figure 74: 2NF of product table

Based on observation, there is no apparent partial dependency between the columns.

No changes were made to the tables.

3NF

Product table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**product_id**	int(11)			Yes	*NULL*		
2	**product_name**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
3	**product_category**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
4	**product_price**	int(11)			Yes	*NULL*		
5	**product_stock_quantity**	int(11)			Yes	*NULL*		
6	**product_availability**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Product supplier table

Name	Type	Collation	Attributes	Null	Default	Comments	Extra
product_id	int(11)			Yes	*NULL*		
supplier_id	int(11)			Yes	*NULL*		

Product deal table

Name	Type	Collation	Attributes	Null	Default	Comments	Extra
product_id	int(11)			Yes	*NULL*		
deal_id	int(11)			Yes	*NULL*		

Figure 75: 3NF of product table

Based on observation, there appear to be no transitive dependencies between the columns.

No changes are made to the tables.

ORDER

0NF

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**order_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**product_id**	int(11)			Yes	*NULL*		
5	**order_product_quantity**	int(11)			Yes	*NULL*		
6	**order_total_price**	float			Yes	*NULL*		
7	**order_payment_method**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
8	**order_date**	date			Yes	*NULL*		

Figure 76: 0NF of order table

Unnormalized form of order table

1NF

Order table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**order_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**order_total_price**	float			Yes	*NULL*		
5	**order_payment_method**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
6	**order_date**	date			Yes	*NULL*		

Order items table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**order_id**	int(11)			Yes	*NULL*		
2	**product_id**	int(11)			Yes	*NULL*		
3	**order_product_quantity**	int(11)			Yes	*NULL*		

Figure 77: 1NF of order table

Order product ID and product quantity details were separated to a new table as to reduce data redundancy and to ensure that every cell is atomic.

2NF

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**order_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**order_total_price**	float			Yes	*NULL*		
5	**order_payment_method**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
6	**order_date**	date			Yes	*NULL*		

Order items table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**order_id**	int(11)			Yes	*NULL*		
2	**product_id**	int(11)			Yes	*NULL*		
3	**order_product_quantity**	int(11)			Yes	*NULL*		

Figure 78: 2NF of order table

Based on observation, there seems to be partial dependency between the columns.

No changes were made to the table.

3NF

Order table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**order_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**order_total_price**	float			Yes	*NULL*		
5	**order_payment_method**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
6	**order_date**	date			Yes	*NULL*		

Order items table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**order_id**	int(11)			Yes	*NULL*		
2	**product_id**	int(11)			Yes	*NULL*		
3	**order_product_quantity**	int(11)			Yes	*NULL*		

Figure 79: 3NF of order table

Based on observation, there seem to be no transitive dependency between the columns.

No changes were made to the table.

ADDRESS, CONTACT NUMBER, EMAIL

With all the three entities having similar tables of address, email, and contact number, it is possible to merge all of them into one single table, resulting in lesser table quantities and being more efficient. Address, contact number, and email table are created to reference all three entities together in one single table, displaying all details in one table.

Below is the table structure of each with combined columns.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**address_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**supplier_id**	int(11)			Yes	*NULL*		
5	**emirate_id**	int(11)			Yes	*NULL*		
6	**area**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		
7	**street**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 80: Table structure of address table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**email_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**supplier_id**	int(11)			Yes	*NULL*		
5	**email_address**	varchar(55)	utf8mb4_general_ci		Yes	*NULL*		

Figure 81: Table structure of email table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**contact_number_id**	int(11)			Yes	*NULL*		
2	**customer_id**	int(11)			Yes	*NULL*		
3	**staff_id**	int(11)			Yes	*NULL*		
4	**supplier_id**	int(11)			Yes	*NULL*		
5	**contact_number**	varchar(13)	utf8mb4_general_ci		Yes	*NULL*		

Figure 82: Table structure of contact number table

SECURITY

It refers to the protection of sensitive data stored in a database from unauthorized access, deletion, or modification. This is where implementing security measures comes in to ensure that the data is secure and protected from any unwanted threats, hackers, malware, etc.

There are many ways to secure data in a database, such as:

ROLE-BASED ACCESS CONTROL (RBAC)

It is the verification of an identity of a user before allowing access to the database. Afterwards, a user is then granted access specific privileges to specific data such as tables and columns. Below is the syntax for creating new user, including the GRANT and REVOKE syntax.

```
CREATE USER 'user'@'%' IDENTIFIED BY 'password';
```

```
GRANT privilege1, privilege2, ... ON database_name.table_name
TO 'username'@'hostname';
```

```
REVOKE privilege1, privilege2, ... ON database_name.table_name
FROM 'username'@'hostname';
```

In phpMyAdmin, it is possible to create a user by selecting the “Privileges tab” and clicking “Add user account” under “New.”

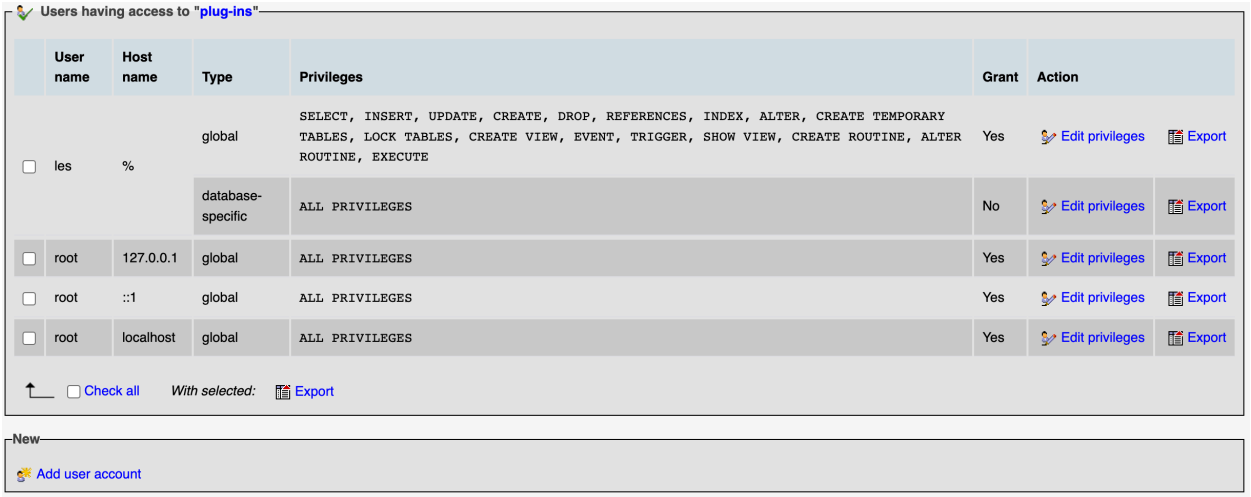


Figure 83: Privileges tab in phpMyAdmin

After clicking “Add user account”, enter the necessary details such as usernames, permissions, etc.

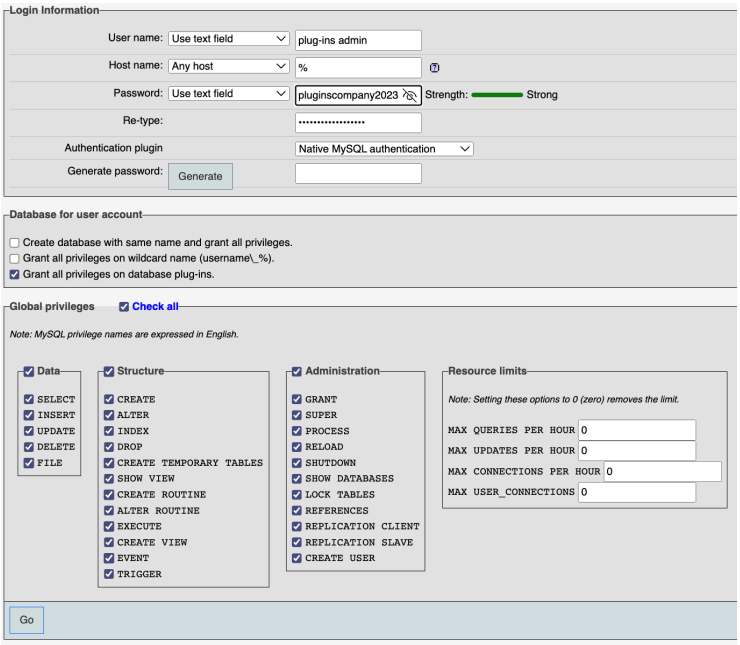


Figure 84: Adding admin user account

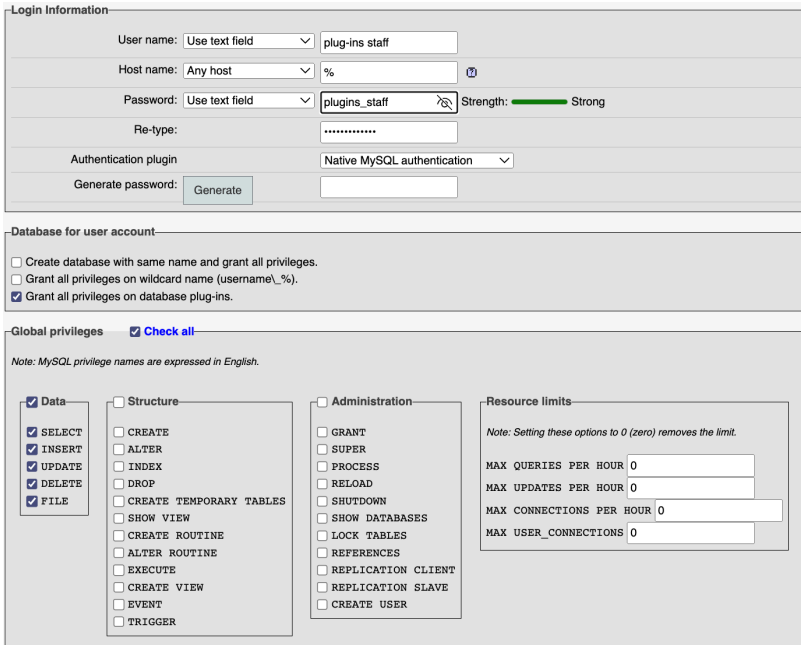


Figure 85: Creating staff user account

	User name	Host name	Password	Global privileges	User group	Grant
☐	Any	%	No	USAGE		No
☐	Any	localhost	No	USAGE		No
☐	les	%	Yes	SELECT, INSERT, UPDATE, CREATE, DROP, RELOAD, SHUTDOWN, PROCESS, FILE, REFERENCES, INDEX, ALTER, SHOW DATABASES, SUPER, CREATE TEMPORARY TABLES, LOCK TABLES, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, EVENT, TRIGGER, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EXECUTE		Yes
☐	plug-ins admin	%	Yes	ALL PRIVILEGES		Yes
☐	plug-ins staff	%	Yes	SELECT, INSERT, UPDATE, DELETE, FILE		No
☐	pma	localhost	No	USAGE		No
☐	root	127.0.0.1	No	ALL PRIVILEGES		Yes
☐	root	::1	No	ALL PRIVILEGES		Yes
☐	root	localhost	No	ALL PRIVILEGES		Yes

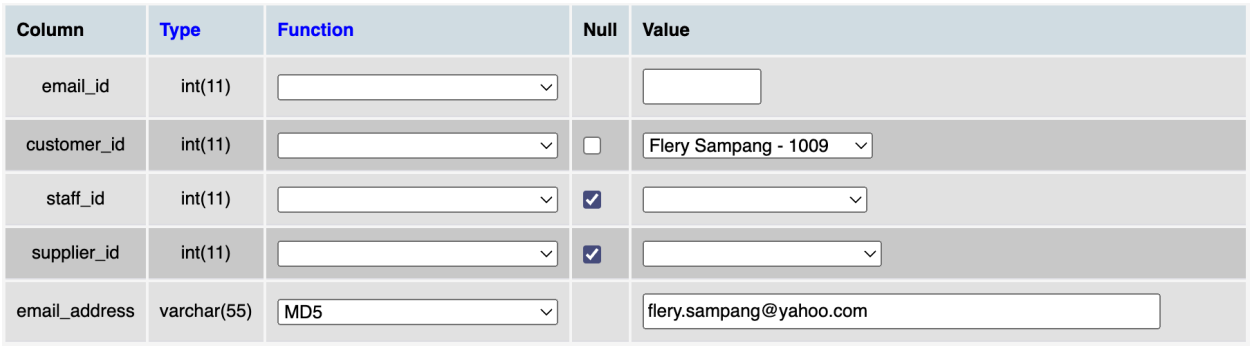
Figure 86: Plug-ins staff and admin user accounts added successfully

ENCRYPTION

It involves the conversion of sensitive data into an unreadable format which can only be decrypted with a password or a decryption key. Through this, the data remains confidential. Below is the syntax for encryption.

```
INSERT INTO table name
VALUES
      (value1, encryption method(value2), ... );
```

In phpMyAdmin, when trying to insert data, it is possible to encrypt data in a specific column via selecting the encryption method under “Function”. In this case, MD5 was used to encrypt email address of customer ID, 1009.



Column	Type	Function	Null	Value
email_id	int(11)			
customer_id	int(11)			Flery Sampang - 1009
staff_id	int(11)		☑	
supplier_id	int(11)		☑	
email_address	varchar(55)	MD5		flery.sampang@yahoo.com

Figure 87: Insert tab of email table, specifying MD5 encryption method

email_id ▼ 1	customer_id	staff_id	supplier_id	email_address
23	1009	*NULL*	*NULL*	8cd81d8f7ef2be1c0f78557300c25236

Figure 88: Updated email table with MD5 encryption

Note that it is possible to use different encryption methods such as AES, Blowfish, RSA, etc. by using the same steps above.

AUDITING

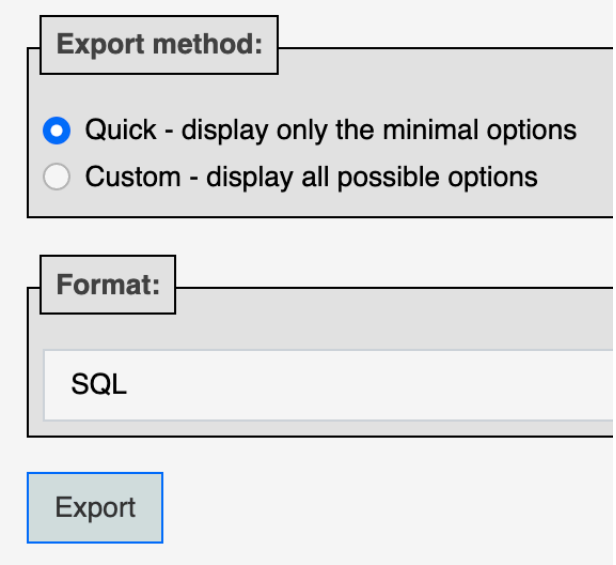
It involves the recording and monitoring of all activities that have occurred in the database. This includes login attempts, data modification, and data access. Through this, auditing helps detect suspicious activities that may help prevent unwanted access.

In phpMyAdmin, it is possible to have a query log via built-in plugin, but it is present only on older versions. Manually creating a query log as a table in a database tend to be more complex but it is possible by referencing different sources on the Internet.

BACKUP & RECOVERY

It involves the regular backups of the database in case of data loss or data breach. It is a feature of every database management software (DBMS), and it is essential when working with database.

In phpMyAdmin, backup and recovery can be done via the use of the built-in Export and Import feature.



Select SQL as the format and click “Export”, this will download a .sql file onto the local directory and it serves as a backup of the database.

Figure 89: Export tab of plug-ins database

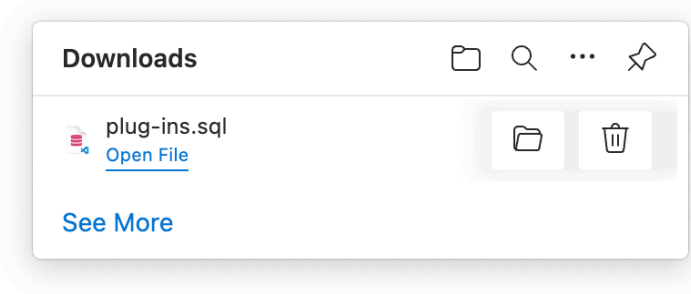


Figure 90: Plug-ins database exported

When recovering a database, this is then done through the Import tab. Select the .sql file to be imported via “Choose File” and then click on “Import”.

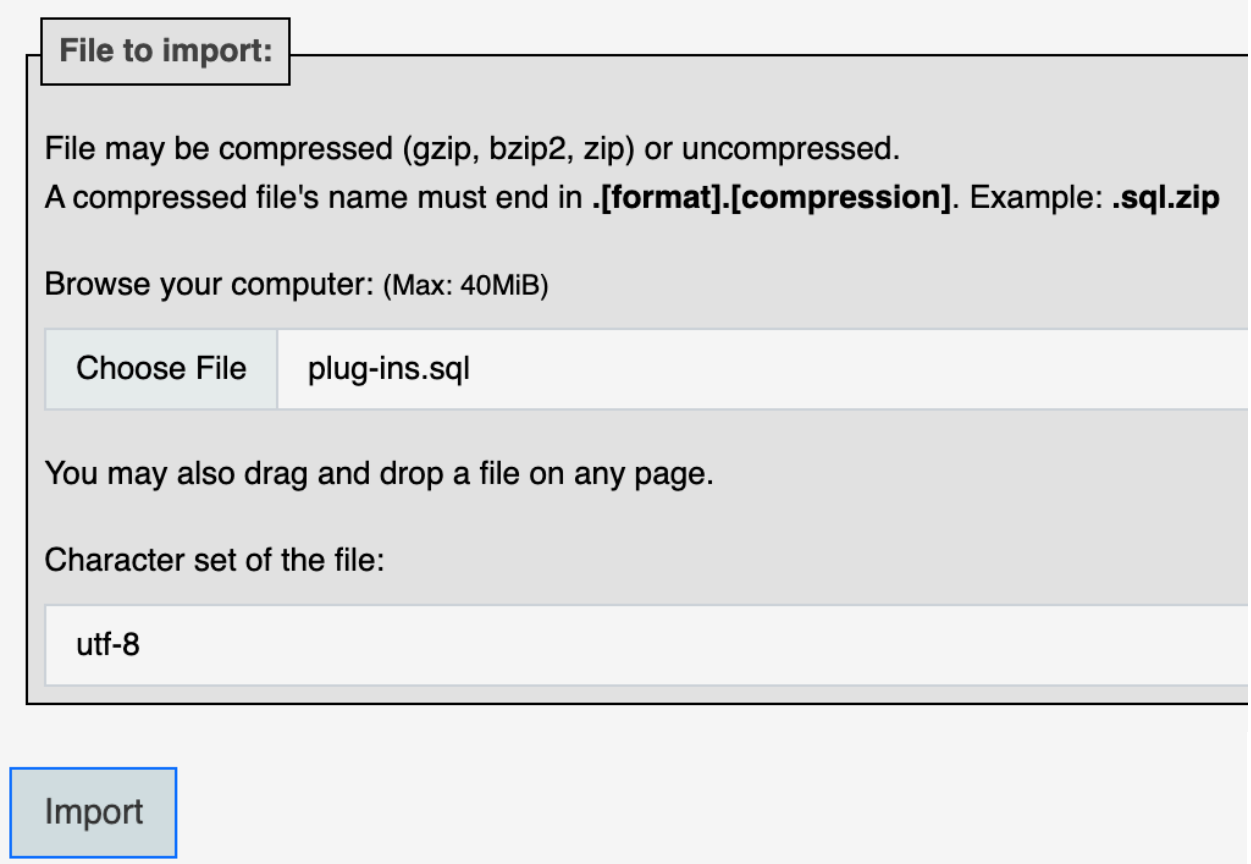


Figure 91: Import tab of plug-ins database

CONCLUSION

To conclude this report, the database of Plug-Ins Company was completed. Along with that, it is proven to be working and functional, together with the user interface that was created through PHP; it can insert and query through data in the database. Testing was also conducted on the database design including the tests of integrity constraints and the foreign key links.

Additionally, all components of the database were explained thoroughly in their corresponding chapters. Screenshots and syntaxes are also provided for easier understanding.