

MOTOR INSURANCE PROJECT

TABLE CREATION:-

1.CUSTOMERS TABLE:

QUERY WITH OUTPUT:

```
CREATE TABLE customers (
```

```
    customer_id INT PRIMARY KEY AUTO_INCREMENT,  
    full_name VARCHAR(100) NOT NULL,  
    email VARCHAR(100),  
    phone_number VARCHAR(20),  
    address TEXT  
);
```

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:** Shows the database structure with Schemas, Tables, Views, and Stored Procedures.
- SQL Editor:** Displays the SQL code for creating the 'customers' table:

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    full_name VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    phone_number VARCHAR(20),
    address TEXT
);
```
- Result Grid:** Shows the data inserted into the 'customers' table. The columns are customer_id, full_name, email, phone_number, and address. The data includes 14 rows of sample data.
- Information Panel:** Shows details for the 'policy_types' table, including its columns: policy_type_id (int AI PK), type_name (varchar(50)), base_rate (decimal(11,2)), and description (text).
- Output Panel:** Shows the output of the query, indicating 1 row affected.

2. VEHICLES TABLE:

QUERY WITH OUTPUT:

```
CREATE TABLE vehicles (
```

```
    vehicle_id INT PRIMARY KEY AUTO_INCREMENT,  
    customer_id INT NOT NULL,  
    registration_number VARCHAR(20) UNIQUE NOT NULL,  
    make VARCHAR(50),  
    model VARCHAR(50),  
    year INT,  
    engine_capacity INT,  
    vehicle_type VARCHAR(50),
```

```
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
```

```
);
```

The screenshot shows the MySQL Workbench interface with the 'motor vehicles' tab selected. The SQL editor pane contains the CREATE TABLE statement for the 'vehicles' table. The results pane below shows the data inserted into the table.

```
CREATE TABLE vehicles (
    vehicle_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT NOT NULL,
    registration_number VARCHAR(20) UNIQUE NOT NULL,
    make VARCHAR(50),
    model VARCHAR(50),
    year INT,
    engine_capacity INT,
    vehicle_type VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

vehicle_id	customer_id	registration_number	make	model	year	engine_capacity	vehicle_type
1	1	AB123CD	Toyota	Corolla	2018	1800	Sedan
2	2	XY456ZT	Honda	Civic	2020	2000	pickup truck
3	3	MN789QR	Ford	Focus	2019	1600	Hatchback
4	4	GH521KL	Chevrolet	Cruze	2017	1800	Sedan
5	5	JK654LM	Hyundai	Elantra	2021	2000	Bus
6	6	ZK987YU	Nissan	Altima	2022	2500	Sedan
7	7	CV123BN	BMW	320i	2020	2000	Minivan
8	8	WE456RF	Mercedes-Benz	C300	2019	2200	Sedan

3.INSURENCE TABLE:

QUERY WITH OUTPUT:

```
CREATE TABLE insurence (
```

```
    insurence_id INT PRIMARY KEY AUTO_INCREMENT,  
    insurence_name VARCHAR(100) NOT NULL,  
    contact_email VARCHAR(100),  
    contact_phone VARCHAR(20)
```

```
);
```

The screenshot shows the MySQL Workbench interface. In the top-left pane, the 'Schemas' tree is visible with nodes for 'insurance', 'policies', 'policy_type', 'premium_calculation', and 'vehicles'. The central workspace displays the SQL editor with the following code:

```
CREATE TABLE insurence (  
    insurence_id INT PRIMARY KEY AUTO_INCREMENT,  
    insurence_name VARCHAR(100) NOT NULL,  
    contact_email VARCHAR(100),  
    contact_phone VARCHAR(20)  
);
```

To the right of the SQL editor is a message box stating: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help." Below the SQL editor is the 'Result Grid' pane, which contains the following data:

insurance_id	insurance_name	contact_email	contact_phone
1	SafeDrive Insurance	contact@safedrive.com	1234567890
2	AutoShield Co.	support@autosield.com	2345678901
3	PrimeCover Inc.	info@primecover.com	3456789012
4	DriveSafe Insure	hello@drivesafe.com	4567890123
5	SecureAuto Group	help@secureauto.com	5678901234
6	ProtectSure Ltd.	service@protectsure.com	6789012345
7	CarGuard Insurance	support@carguard.com	7890123456
8	AutoPlus Insurance	contact@autoplus.com	8901234567
9	ReliableCover Co.	info@reliablecover.com	9012345678
10	NextGen Insurance	support@nextgeninsure.com	0123456789
11	SureDrive Solutions	hello@suredrive.com	1239876543
12	TrustAuto Insurance	contact@trustauto.com	2340875432
13	CityDrive Insurance	support@citydrive.com	3457654321
14	UrbanShield Insure	info@urbanshield.com	4566543210
15	Nationwide Vehicle ...	service@nationwidecover.com	5675432109

4.POLICY_TYPE TABLE:

QUERY WITH OUTPUT:

```
CREATE TABLE policy_types (
```

```
    policy_type_id INT PRIMARY KEY AUTO_INCREMENT,  
    type_name VARCHAR(50) NOT NULL,  
    base_rate DECIMAL(10,2) NOT NULL,  
    description TEXT  
);
```

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the database is set to 'Local instance MySQL80'. The main area displays the creation of the 'policy_types' table:

```

CREATE TABLE policy_types (
    policy_type_id INT PRIMARY KEY AUTO_INCREMENT,
    type_name VARCHAR(50) NOT NULL,
    base_rate DECIMAL(10,2) NOT NULL,
    description TEXT
)

```

Below the creation statement, the table structure is shown:

Table: policy_types

Columns:

Column Name	Type	Properties
policy_type_id	int AI PK	
type_name	varchar(50)	
base_rate	decimal(10,2)	
description	text	

The data grid below contains 15 rows of policy types:

policy_type_id	type_name	base_rate	description
1	Comprehensive	1200.00	Full coverage including damage to own vehicle a...
2	Third-Party Only	500.00	Covers liability to third parties only.
3	Third-Party, Fire and Theft	750.00	Covers third-party liability, fire damage, and th...
4	Collision Coverage	900.00	Covers damage to your own vehicle after a coll...
5	Personal Injury Protection	600.00	Covers medical expenses for you and passengers.
6	Uninsured Motorist	550.00	Protects against drivers without insurance.
7	Underinsured Motorist	580.00	Covers losses when other driver has insufficient...
8	Pay-As-You-Drive	400.00	Variable rate based on mileage driven.
9	Usage-Based Insurance	450.00	Monitors driving behavior to determine premiums.
10	No-Fault Insurance	700.00	Covers your injuries regardless of fault.
11	Gap Insurance	300.00	Covers the difference between car value and lo...
12	Rental Reimbursement	200.00	Pays for rental vehicle if your car is being repai...
13	Roadside Assistance	100.00	Covers towing, fuel delivery, and minor repairs.
14	Custom Parts Coverage	350.00	Covers aftermarket accessories and modifications.
15	Mechanical Breakdown Ins...	950.00	Covers repairs not typically under warranty.

5.PREMIUM_CALCULATION TABLE:

QUERY WITH OUTPUT:

CREATE TABLE premium_calculations (

```

calculation_id INT PRIMARY KEY AUTO_INCREMENT,
vehicle_id INT NOT NULL,
policy_type_id INT NOT NULL,
base_premium DECIMAL(10,2),
risk_factor DECIMAL(5,2),
discount DECIMAL(5,2),
final_premium DECIMAL(10,2),
calculation_date DATETIME DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id),
FOREIGN KEY (policy_type_id) REFERENCES
policy_types(policy_type_id)
);

```

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** Local instance MySQL80, File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Schemas:** insurance, policies, policy_types, premium_calculations, vehicles.
- Table: policy_types** (selected):

Columns:	policy_type_id	int AI PK
type_name	varchar(50)	
base_rate	decimal(10,2)	
description	text	
- Table: premium_calculations** (selected):


```

CREATE TABLE premium_calculations (
    calculation_id INT PRIMARY KEY AUTO_INCREMENT,
    vehicle_id INT NOT NULL,
    policy_type_id INT NOT NULL,
    base_premium DECIMAL(10,2),
    risk_factor DECIMAL(5,2),
    discount DECIMAL(5,2),
    final_premium DECIMAL(10,2),
    calculation_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id),
    FOREIGN KEY (policy_type_id) REFERENCES policy_types(policy_type_id)
);
      
```
- Result Grid:** Shows data for premium_calculations:

calculation_id	vehicle_id	policy_type_id	base_premium	risk_factor	discount	final_premium	calculation_date
11	11	11	300.00	1.30	0.00	390.00	2025-06-20 12:52:23
12	12	12	200.00	1.00	0.00	200.00	2025-06-20 12:52:23
13	13	13	100.00	1.20	0.05	114.00	2025-06-20 12:52:23
14	14	14	350.00	1.10	0.10	346.50	2025-06-20 12:52:23
15	15	15	950.00	1.25	0.05	1128.13	2025-06-20 12:52:23
16	16	16	1100.00	1.30	0.00	1430.00	2025-06-20 12:52:23
17	17	17	1600.00	1.50	0.10	2160.00	2025-06-20 12:52:23
18	18	18	1200.00	1.40	0.05	1440.00	2025-06-20 12:52:23

6.POLICIES TABLE:

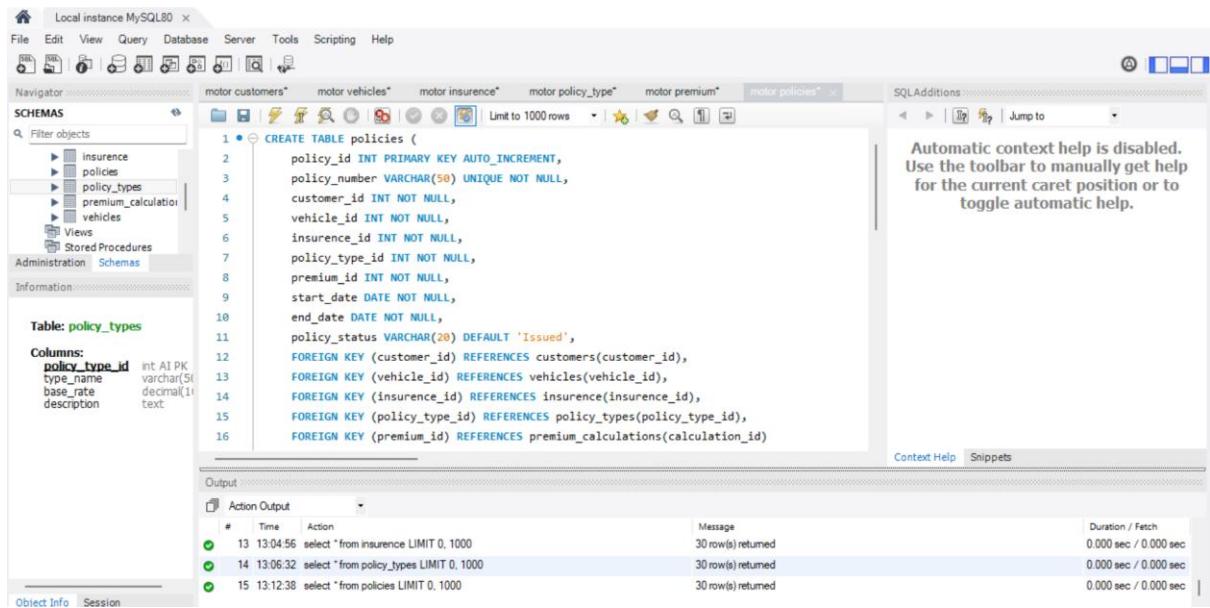
QUERY WITH OUTPUT:

CREATE TABLE policies (

```

policy_id INT PRIMARY KEY AUTO_INCREMENT,
policy_number VARCHAR(50) UNIQUE NOT NULL,
customer_id INT NOT NULL,
vehicle_id INT NOT NULL,
insurance_id INT NOT NULL,
policy_type_id INT NOT NULL,
premium_id INT NOT NULL,
start_date DATE NOT NULL,
end_date DATE NOT NULL,
policy_status VARCHAR(20) DEFAULT 'Issued',
FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
      
```

FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id),
 FOREIGN KEY (insurance_id) REFERENCES insurance(insurance_id),
 FOREIGN KEY (policy_type_id) REFERENCES policy_types(policy_type_id),
 FOREIGN KEY (premium_id) REFERENCES premium_calculations(calculation_id)
);



The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** Local instance MySQL80, File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Toolbar:** Standard MySQL Workbench icons.
- Navigator:** Schemas section shows the database structure with tables: insurance, policies, policy_types, premium_calculations, vehicles.
- SQL Editor:** A CREATE TABLE statement for the 'policies' table is displayed. The table has columns: policy_id (PK), policy_number, customer_id, vehicle_id, insurance_id, policy_type_id, premium_id, start_date, end_date, and policy_status. It includes foreign key constraints linking to other tables: customer_id to customers, vehicle_id to vehicles, insurance_id to insurance, policy_type_id to policy_types, and premium_id to premium_calculations.
- Output Panel:** Shows the results of three SELECT queries:

#	Time	Action	Message	Duration / Fetch
13	13:04:56	select * from insurance LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
14	13:06:32	select * from policy_types LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
15	13:12:38	select * from policies LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec

ALTER TABLE:

QUERY WITH OUTPUT:

ADD,MODIFY COLUMN:

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator Schemas Administration Schemas Information

Table: policy_types

Columns:

policy_type_id	int AI PK
type_name	varchar(50)
base_rate	decimal(10,2)
description	text

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid | Form Editor | Read Only | Context Help | Snippets

Action Output

#	Time	Action	Message	Duration / Fetch
12	13:03:14	select * from vehicles LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
13	13:04:56	select * from insurance LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
14	13:06:32	select * from policy_types LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
15	13:12:38	select * from policies LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
16	13:14:52	select * from policies LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
17	13:23:23	ALTER TABLE customers ADD date_of_birth DATE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.079 sec
18	13:25:37	ALTER TABLE customers MODIFY email VARCHAR(150)	0 rows(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.047 sec

Object Info Session

CHANGE COLUMN:

File Edit View Query Database Server Tools Scripting Help

Navigator Schemas Administration Schemas Information

Table: policy_types

Columns:

policy_type_id	int AI PK
type_name	varchar(50)
base_rate	decimal(10,2)
description	text

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid | Form Editor | Read Only | Context Help | Snippets

Action Output

#	Time	Action	Message	Duration / Fetch
16	13:14:52	select * from policies LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
17	13:23:23	ALTER TABLE customers ADD date_of_birth DATE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.079 sec
18	13:25:37	ALTER TABLE customers MODIFY email VARCHAR(150)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.047 sec
19	13:25:51	desc customers	6 row(s) returned	0.016 sec / 0.000 sec
20	13:28:21	ALTER TABLE customers CHANGE phone_number contact_number VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.063 sec
21	13:29:30	DESC CUSTOMERS	6 row(s) returned	0.000 sec / 0.000 sec

DROP COLUMN

Local Instance MySQL5.7

File Edit View Query Database Server Tools Scripting Help

Navigator: motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*

SCHEMAS: insurance policies policy_types premium_calculation vehicles

Table: policy_types

Columns:

policy_type_id	int AI PK
type_name	varchar(50)
base_rate	decimal(11,2)
description	text

Result Grid: Filter Rows: Export: Wrap Cell Content: Result Grid Form Editor

Field Type Null Key Default Extra

customer_id int NO PRI auto_increment
 full_name varchar(100) NO
 email varchar(150) YES
 contact_number varchar(20) YES
 address text YES

Action Output:

#	Time	Action	Message	Duration / Fetch
18	13:25:37	ALTER TABLE customers MODIFY email VARCHAR(150)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.047 sec
19	13:25:51	desc customers	6 row(s) returned	0.016 sec / 0.000 sec
20	13:28:21	ALTER TABLE customers CHANGE phone_number contact_number VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.063 sec
21	13:29:30	DESC CUSTOMERS	6 row(s) returned	0.000 sec / 0.000 sec
22	13:31:18	ALTER TABLE customers DROP COLUMN date_of_birth	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.063 sec
23	13:31:23	DESC CUSTOMERS	5 row(s) returned	0.000 sec / 0.000 sec

Output: Action Output

SQLAdditions: Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

AGGREGATE FUNCTION:

1.COUNT TOTAL NUMBER OF VEHICLES:

File Edit View Query Database Server Tools Scripting Help

Navigator: motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*

SCHEMAS: insurance policies policy_types premium_calculation vehicles

Table: policy_types

Columns:

policy_type_id	int AI PK
type_name	varchar(50)
base_rate	decimal(11,2)
description	text

Result Grid: Filter Rows: Export: Wrap Cell Content: Result Grid Form Editor

total_vehicles

Action Output:

#	Time	Action	Message	Duration / Fetch
21	13:29:30	DESC CUSTOMERS	6 row(s) returned	0.000 sec / 0.000 sec
22	13:31:18	ALTER TABLE customers DROP COLUMN date_of_birth	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.063 sec
23	13:31:23	DESC CUSTOMERS	5 row(s) returned	0.000 sec / 0.000 sec
24	13:34:48	select * from customers LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
25	13:34:57	select * from vehicles LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
26	13:38:08	SELECT COUNT(*) AS total_vehicles FROM vehicles LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Output: Action Output

SQLAdditions: Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

2.AVG ENGINE CAPACITY:

The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL code:

```
motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*
44 • (29, 'GH654N81', 'Toyota', 'Camry', 2019, 2500, 'Sedan'),
45 • (30, 'JK987PO', 'Honda', 'Accord', 2018, 2400, 'Two_wheeler');
46 • select * from vehicles;
47 • SELECT COUNT(*) AS total_vehicles FROM vehicles;
48 • SELECT AVG(engine_capacity) AS average_engine_capacity FROM vehicles;
49
```

The result grid shows the output of the last query:

average_engine_capacity
2110.7143

The status bar at the bottom right indicates "Read Only".

3.TOTAL NUMBER OF VEHICLES PER VEHICLE TYPE:

The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL code:

```
motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*
47 • SELECT COUNT(*) AS total_vehicles FROM vehicles;
48 • SELECT AVG(engine_capacity) AS average_engine_capacity FROM vehicles;
49
50 • SELECT vehicle_type, COUNT(*) AS count_per_type
51   FROM vehicles
52   GROUP BY vehicle_type;
```

The result grid shows the output of the last query:

vehicle_type	count_per_type
Sedan	7
pickup truck	2
Hatchback	1
Bus	1
Minivan	1
Three_wheeler	1
Hatchback	3
Tow truck	1
Electric	1
SUV	8
Van	1
Four_wheeler	1
Electric SUV	1
Two_wheeler	1

4.TOTAL ENGINE CAPACITY PER VEHICLES TYPE:

File Edit View Query Database Server Tools Scripting Help

Navigator: motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies* SQLAdditions

```

SELECT vehicle_type, COUNT(*) AS count_per_type
FROM vehicles
GROUP BY vehicle_type;

SELECT vehicle_type, SUM(engine_capacity) AS total_engine_capacity

```

Table: policy_types

Columns:

policy_type_id	int AI PK
type_name	varchar(50)
base_rate	decimal(10,2)
description	text

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid

vehicle_type	total_engine_capacity
Sedan	15200
pickup truck	4200
Hatchback	1600
Bus	2000
Minivan	2000
Three_wheeler	2000
Hatchback	6100
Tow truck	2500
Electric	NULL
SUV	17300
Van	2000
Four_wheeler	1800
Electric SUV	NULL
Two_wheeler	2400

Form Editor Field Types Query Stats

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

5. MAXIMUM AND MINIMUM ENGINE CAPACITY:

File Edit View Query Database Server Tools Scripting Help

Navigator: motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies* SQLAdditions

```

FROM vehicles
GROUP BY vehicle_type;

SELECT
    MAX(engine_capacity) AS max_engine_capacity,
    MIN(engine_capacity) AS min_engine_capacity

```

Table: policy_types

Columns:

policy_type_id	int AI PK
type_name	varchar(50)
base_rate	decimal(10,2)
description	text

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid

max_engine_capacity	min_engine_capacity
2500	1500

Form Editor Field Types Query Stats

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result 8 x Read Only Context Help Snippets

6. COUNT VEHICLES PER MAKE

File Edit View Query Database Server Tools Scripting Help

Navigator: motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*

```

62
63 •   SELECT make, COUNT(*) AS count_per_make
64   FROM vehicles
65   GROUP BY make
66   ORDER BY count_per_make DESC;
67

```

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid

make	count_per_make
Toyota	3
Honda	3
Ford	2
Chevrolet	2
Hyundai	2
Nissan	2
BMW	2
Mercedes-Benz	2
Audi	2
Volkswagen	2
Kia	2
Mazda	2
Subaru	2
Tesla	2

Result 9 x Read Only Context Help Snippets

Output: Action Output

Information: Schemas

Table: policy_types

Columns:

- policy_type_id int AI PK
- type_name varchar(51)
- base_rate decimal(11)
- description text

7. FIND VEHICLES TYPE WITH MORE THAN 2 ENTIRES

File Edit View Query Database Server Tools Scripting Help

Navigator: motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*

```

67
68 •   SELECT vehicle_type, COUNT(*) AS count
69   FROM vehicles
70   GROUP BY vehicle_type
71   HAVING COUNT(*) > 2;
72

```

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid

vehicle_type	count
Sedan	7
Hatchback	3
SUV	8

Result 11 x Read Only Context Help Snippets

Information: Schemas

Table: policy_types

Columns:

- policy_type_id int AI PK
- type_name varchar(51)
- base_rate decimal(11)
- description text

8. AVERAGE ENGINE CAPACITY PER YEAR

The screenshot shows the MySQL Workbench application. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, Help, and a toolbar with various icons. The left sidebar has sections for Navigator, Schemas, Administration, and Schemas. The main area shows a query editor with the following SQL code:

```
72
73 •   SELECT year, AVG(engine_capacity) AS avg_engine_capacity
74   FROM vehicles
75   GROUP BY year
76   ORDER BY year;
77
```

The result grid below displays the output of the query:

year	avg_engine_capacity
2017	2033.3333
2018	1825.0000
2019	2100.0000
2020	2200.0000
2021	2250.0000
2022	2125.0000
2023	NULL

A context help message on the right says: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

USING OPERATORS:

The screenshot shows the SSMS interface with the following details:

- File Edit View Query Database Server Tools Scripting Help**
- Navigator** pane on the left lists **motor customers***, **motor vehicles*** (selected), **motor insurance***, **motor policy_type***, **motor premium***, and **motor policie**.
- SCHEMAS** pane shows objects under **motor vehicles**: **insurance**, **policies**, **policy_types** (selected), **premium_calculation**, **vehicles**, **Views**, and **Stored Procedures**.
- Administration Schemas** tab is selected.
- Information** tab is selected.
- Table: policy_types** is listed.
- Columns:**

policy_type_id	int AI PK
type_name	varchar(5)
base_rate	decimal(1)
description	text
- Query Editor:**

```
75 GROUP BY year
76 ORDER BY year;
77
78 • SELECT COUNT(*) AS count_high_capacity_recent
79 FROM vehicles
80 WHERE engine_capacity > 2000 AND year > 2020;
```
- Result Grid:**

count_high_capacity_recent
4

The screenshot shows the SSMS interface with the following details:

- File Edit View Query Database Server Tools Scripting Help**
- Schemas** node expanded, showing objects like insurance, policies, policy_types, premium_calculation, vehicles, Views, and Stored Procedures.
- Table: policy_types** is selected in the bottom left.
- motor customers***, **motor vehicles*** (highlighted), **motor insurance***, **motor policy_type***, **motor premium***, and **motor policies*** are listed in the top right.
- Query Editor:**
 - Code:

```
79    FROM vehicles
80    WHERE engine_capacity > 2000 AND year > 2020;
81
82 •  SELECT SUM(engine_capacity) AS sum_capacity_sedan_or_suv
83    FROM vehicles
84    WHERE vehicle_type = 'Sedan' OR vehicle_type = 'SUV';
```
 - Result Grid:

sum_capacity_sedan_or_suv
32500

File Edit View Query Database Server Tools Scripting Help

Navigator motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*

SCHEMAS

Filter objects

- insurance
- policies
- policy_types**
- premium_calculation
- vehicles
- Views
- Stored Procedures

Administration Schemas

Information Table: **policy_types**

Columns:

policy_type_id	int AI PK
-----------------------	-----------

```

83   FROM vehicles
84   WHERE vehicle_type = 'Sedan' OR vehicle_type = 'SUV';
85
86 •   SELECT AVG(engine_capacity) AS avg_capacity_excluding_hatchbacks
87   FROM vehicles
88   WHERE NOT (vehicle_type = 'Hatchback' OR vehicle_type = 'Hatchback ');

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

avg_capacity_excluding_hatchbacks
2141.6667

File Edit View Query Database Server Tools Scripting Help

Navigator motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*

SCHEMAS

Filter objects

- insurance
- policies
- policy_types**
- premium_calculation
- vehicles
- Views
- Stored Procedures

Administration Schemas

Information Table: **policy_types**

```

87   FROM vehicles
88   WHERE NOT (vehicle_type = 'Hatchback' OR vehicle_type = 'Hatchback ');
89
90 •   SELECT COUNT(*) AS count_efficient_or_tesla
91   FROM vehicles
92   WHERE (engine_capacity >= 2000 AND year >= 2020) OR make = 'Tesla';

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

count_efficient_or_tesla
18

File Edit View Query Database Server Tools Scripting Help

Navigator motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*

SCHEMAS

Filter objects

- insurance
- policies
- policy_types**
- premium_calculation
- vehicles
- Views
- Stored Procedures

Administration Schemas

Information Table: **policy_types**

```

92   WHERE (engine_capacity >= 2000 AND year >= 2020) OR make = 'Tesla';
93
94 •   SELECT COUNT(*) AS count_non_electric_with_capacity
95   FROM vehicles
96   WHERE engine_capacity IS NOT NULL AND vehicle_type NOT LIKE '%Electric%';
97

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

count_non_electric_with_capacity
28

Schemas

```

96 WHERE engine_capacity IS NOT NULL AND vehicle_type NOT LIKE '%Elect
97
98 • SELECT vehicle_type, COUNT(*) AS count_above_2000cc
99 FROM vehicles
100 WHERE engine_capacity > 2000
101 GROUP BY vehicle_type;

```

Table: policy_types

Columns:

policy_type_id	int AI PK
type_name	varchar(50)
base_rate	decimal(10,2)
description	text

Result Grid

vehicle_type	count_above_2000cc
Sedan	4
Tow truck	1
SUV	4
Backup truck	1
Hatchback	1
Two_wheeler	1

JOINS

File Edit View Query Database Server Tools Scripting Help

Navigator Schemas

```

54 • select * from policies;
55
56
57 • SELECT *
58 FROM policies p
59 JOIN customers c ON p.customer_id = c.customer_id
60 LIMIT 0, 1000;
61

```

Table: policy_types

Columns:

policy_type_id	int AI PK
type_name	varchar(50)
base_rate	decimal(10,2)
description	text

Result Grid

policy_id	policy_number	customer_id	vehicle_id	insurance_id	policy_type_id	premium_id	start_date	end_date
1	POL100001	1	1	1	1	1	2024-06-01	2025-05-31
2	POL100002	2	2	2	2	2	2024-06-15	2025-05-31
3	POL100003	3	3	3	3	3	2024-07-01	2025-06-30
4	POL100004	4	4	4	4	4	2024-07-10	2025-06-30
5	POL100005	5	5	5	5	5	2024-07-20	2025-06-30
6	POL100006	6	6	6	6	6	2024-08-01	2025-07-31
7	POL100007	7	7	7	7	7	2024-08-10	2025-07-31
8	POL100008	8	8	8	8	8	2024-08-15	2025-07-31
9	POL100009	9	9	9	9	9	2024-09-01	2025-08-31

Output

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

INNER JOINS:

Screenshot of MySQL Workbench showing an INNER JOIN query:

```

48 • select * from premium_calculations;
49
50 • SELECT *
51   FROM premium_calculations pc
52   INNER JOIN vehicles v ON pc.vehicle_id = v.vehicle_id;
53
    
```

Table: policy_types

Columns:

- policy_type_id** int AI PK
- type_name varchar(50)
- base_rate decimal(11,2)
- description text

Result Grid:

calculation_id	vehicle_id	policy_type_id	base_premium	risk_factor	discount	final_premium	calculation_date
1	1	1	1200.00	1.10	0.05	1254.00	2025-06-20 12:52:2
2	2	2	500.00	1.20	0.00	600.00	2025-06-20 12:52:2
3	3	3	750.00	1.15	0.10	776.25	2025-06-20 12:52:2
4	4	4	900.00	1.00	0.05	855.00	2025-06-20 12:52:2
5	5	5	600.00	1.30	0.10	702.00	2025-06-20 12:52:2
6	6	6	550.00	1.10	0.00	605.00	2025-06-20 12:52:2
7	7	7	580.00	1.25	0.05	686.25	2025-06-20 12:52:2
8	8	8	400.00	1.20	0.00	480.00	2025-06-20 12:52:2
9	9	9	450.00	1.15	0.05	491.63	2025-06-20 12:52:2
10	10	10	700.00	1.10	0.10	693.00	2025-06-20 12:52:2
11	11	11	300.00	1.30	0.00	390.00	2025-06-20 12:52:2
12	12	12	200.00	1.00	0.00	200.00	2025-06-20 12:52:2

Screenshot of MySQL Workbench showing an INNER JOIN query:

```

66 •
67   FROM
68     policies p
69   JOIN
70     customers c ON p.customer_id = c.customer_id
71   JOIN
72     insurance i ON p.insurance_id = i.insurance_id
73   LIMIT 0, 1000;
    
```

Table: policy_types

Columns:

- policy_type_id** int AI PK
- type_name varchar(50)
- base_rate decimal(11,2)
- description text

Result Grid:

customer_id	policy_id	insurance_id
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
R	R	R

Screenshot of MySQL Workbench showing a LEFT JOIN query:

```

73
74 • SELECT
75   p.policy_type_id,
76   c.customer_id
77   FROM
78     policies p
79   LEFT JOIN
80     customers c ON p.customer_id = c.customer_id;
    
```

Table: policy_types

Columns:

- policy_type_id** int AI PK
- type_name varchar(50)
- base_rate decimal(11,2)
- description text

Result Grid:

policy_type_id	customer_id
1	1
2	2
3	3
4	4
5	5
6	6
7	7

The screenshot shows the SSMS interface with the following details:

- File, Edit, View, Query, Database, Server, Tools, Scripting, Help** menu bar.
- Navigator** pane on the left showing **Schemas** (insurance, policies, policy_types, premium_calculations, vehicles) and **Views, Stored Procedures**.
- Information** pane below the Navigator.
- Table: policy_types** information in the center-left.
- Columns:**
 - policy_type_id**: int AI PK
 - type_name**: varchar(50)
 - base_rate**: decimal(10, 2)
 - description**: text
- Query Editor** pane with the following T-SQL code:

```
SELECT
    p.policy_type_id,
    c.customer_id,
    v.vehicle_id,
    i.insurance_id
FROM
    policies p
JOIN
    vehicles v ON p.vehicle_id = v.vehicle_id
JOIN
    customers c ON p.customer_id = c.customer_id
JOIN
    insurance i ON p.insurance_id = i.insurance_id
```
- Result Grid** pane at the bottom showing the following data:

policy_type_id	customer_id	vehicle_id	insurance_id
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5

Right-hand sidebar: **SQLAdditions** and **Automati** (partially visible).

SUBQUERY:

1.SUBQUERY IN WHERE CLAUSE:

The screenshot shows the SSMS interface with the following details:

- Toolbar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Schemas:** insurance, policies, policy_types, premium_calculations, vehicles.
- Query Editor:** Contains a T-SQL script for calculating premiums based on vehicle IDs.

```
motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* x motor policies*
46    (30, 30, 400.00, 1.10, 0.05, 418.00);
47
48 • select * from premium_calculations;
49
50 • SELECT *
51   FROM premium_calculations
52   WHERE vehicle_id IN (
53     SELECT vehicle_id
54     FROM vehicles)
```
- Result Grid:** Shows the output of the query with one row of data:

calculation_id	vehicle_id	policy_type_id	base_premium	risk_factor	discount	final_premium	calculation_date
1	1	1	1200.00	1.10	0.05	1254.00	2025-06-20 12:52:23
*	HULL	HULL	HULL	HULL	HULL	HULL	HULL
- Buttons:** Result Grid, Form, Wrap Cell Content, Export/Import, Edit, Filter Rows, and a toolbar with various icons.

2. SUBQUERY IN SELECT CLAUSE:

The screenshot shows a SQL query window in SSMS. The code is as follows:

```

File Edit View Query Database Server Tools Scripting Help
Navigator motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*
schemas
Table: policy_types
Columns:
policy_type_id int AI PK
type_name varchar(50)
base_rate decimal(10,2)
description text
SELECT
    calculation_id,
    vehicle_id,
    (SELECT registration_number
     FROM vehicles
     WHERE vehicles.vehicle_id = pc.vehicle_id) AS registration_number,
    final_premium
FROM premium_calculations pc;

```

The result grid shows the following data:

	calculation_id	vehicle_id	registration_number	final_premium
1	1	AB123CD	1254.00	
2	2	XY456ZT	600.00	
3	3	MN789QR	776.25	
4	4	GH321KL	855.00	
5	5	JK654LM	702.00	
6	6	ZX987YU	605.00	
7	7	CV123BN	686.25	
8	8	WE456RF	480.00	

3. SUBQUERY IN FROM CLAUSE (INLINE VIEW)

The screenshot shows a SQL query window in SSMS. The code is as follows:

```

File Edit View Query Database Server Tools Scripting Help
Navigator motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies*
schemas
Table: policy_types
Columns:
policy_type_id int AI PK
type_name varchar(50)
base_rate decimal(10,2)
description text
SELECT
    pt.type_name,
    avg_data.avg_final_premium
FROM (
    SELECT
        policy_type_id,
        AVG(final_premium) AS avg_final_premium
    FROM premium_calculations
    GROUP BY policy_type_id
) AS avg_data
JOIN policy_types pt ON pt.policy_type_id = avg_data.policy_type_id;

```

The result grid shows the following data:

	type_name	avg_final_premium
1	Comprehensive	1254.000000
2	Third-Party Only	600.000000
3	Third-Party, Fire and Theft	776.250000
4	Collision Coverage	855.000000
5	Personal Injury Protection	702.000000
6	Uninsured Motorist	605.000000

STORED PROCEDURE:

QUERY:

The screenshot shows the SSMS interface with the 'Schemas' node selected in the left pane. A table named 'policy_types' is currently selected. In the main code editor area, a stored procedure is being created:

```
48
49  DELIMITER //
50
51 • CREATE PROCEDURE AddPremiumCalculation (
52     IN in_vehicle_id INT,
53     IN in_policy_type_id INT,
54     IN in_base_premium DECIMAL(10,2),
55     IN in_risk_factor DECIMAL(5,2),
56     IN in_discount DECIMAL(5,2)
57 )
58 BEGIN
59     DECLARE calc_final_premium DECIMAL(10,2);
60
61     -- Validate base_premium
62     IF in_base_premium <= 0 THEN
63         SIGNAL SQLSTATE '45000'
64         SET MESSAGE_TEXT = 'Base premium must be greater than 0';
65     END IF;
66
67     -- Calculate final premium
```

The screenshot continues from the previous one, showing the completion of the stored procedure:

```
66
67     -- Calculate final premium
68     SET calc_final_premium = in_base_premium + (in_base_premium * in_risk_factor / 100) - in_discount;
69     -- Insert into the table
70     INSERT INTO premium_calculations (
71         vehicle_id,
72         policy_type_id,
73         base_premium,
74         risk_factor,
75         discount,
76         final_premium
77     ) VALUES (
78         in_vehicle_id,
79         in_policy_type_id,
80         in_base_premium,
81         in_risk_factor,
82         in_discount,
83         calc_final_premium
84     );
85 END //
```

OUTPUT

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the tabs 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'Help' are visible. Below the toolbar, there are tabs for 'motor customers*', 'motor vehicles*', 'motor insurance*', 'motor policy_type*', 'motor premium*', and 'motor policies*'. The 'Schemas' tab is selected in the left sidebar, showing tables like 'insurance', 'policies', 'policy_types', 'premium_calculations', 'vehicles', 'Views', and 'Stored Procedures'. The main area displays a query window with the following code:

```
84 END //
85 DELIMITER ;
86
87 • CALL AddPremiumCalculation(1, 2, 5000.00, 10.00, 300.00);
88
89
90 • SELECT *
91 FROM premium_calculations
```

Below the code, a 'Result Grid' shows the output of the 'SELECT' statement:

calculation_id	vehicle_id	policy_type_id	base_premium	risk_factor	discount	final_premium	calculation_date
31	1	2	5000.00	10.00	300.00	5200.00	2025-06-20 15:51:50
HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

TRIGGERS

SQL TRIGGERS FOR THE INSURANCE TABLE:

AFTER INSERT TRIGGERS:

The screenshot shows the MySQL Workbench interface with the 'Schemas' tab selected in the left sidebar. The main area displays the creation of an 'insurance_log' table and an 'after_insurance_insert' trigger:

```
40
41 • CREATE TABLE insurance_log (
42     log_id INT PRIMARY KEY AUTO_INCREMENT,
43     insurance_id INT,
44     action_type VARCHAR(20),
45     log_time DATETIME DEFAULT CURRENT_TIMESTAMP
46 );
47 DELIMITER //
48
49 • CREATE TRIGGER after_insurance_insert
50     AFTER INSERT ON insurance
51     FOR EACH ROW
52     BEGIN
53         INSERT INTO insurance_log (insurance_id, action_type)
54             VALUES (NEW.insurance_id, 'INSERT');
55     END //
```

In the bottom right corner, a message box states: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

The 'Output' pane at the bottom shows the execution log:

#	Time	Action	Message	Duration / Fetch
82	15:55:51	CREATE TABLE insurance_log (log_id INT PRIMARY KEY AUTO_INCREMENT,	0 row(s) affected	0.016 sec
83	15:56:22	CREATE TRIGGER after_insurance_insert AFTER INSERT ON insurance FOR EA...	0 row(s) affected	0.015 sec
84	15:56:51	CREATE TRIGGER before_insurance_update BEFORE UPDATE ON insurance FO...	0 row(s) affected	0.016 sec

BEFORE UPDATE TRIGGER

The screenshot shows the MySQL Workbench interface. In the left sidebar, under 'Schemas', the 'motor' schema is selected. In the main pane, a code editor displays the following SQL code:

```
DELIMITER //
CREATE TRIGGER before_insurance_update
BEFORE UPDATE ON insurance
FOR EACH ROW
BEGIN
    IF NEW.contact_phone IS NOT NULL AND LEFT(NEW.contact_phone, 1) NOT IN ('+', '0') THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid contact phone number format.';
    END IF;
END //

DELIMITER ;
```

The code defines a trigger named 'before_insurance_update' that runs before an update operation on the 'insurance' table. It checks if the 'contact_phone' field is not null and if its first character is not a '+' or '0'. If either condition is true, it signals an error with the message 'Invalid contact phone number format.'.

In the bottom right corner of the code editor, there is a note: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Below the code editor, the 'Output' tab shows the results of the execution:

#	Time	Action	Message	Duration / Fetch
82	15:55:51	CREATE TABLE insurance_log (log_id INT PRIMARY KEY AUTO_INCREMENT)	0 row(s) affected	0.016 sec
83	15:56:22	CREATE TRIGGER after_insurance_insert AFTER INSERT ON insurance FOR EACH ROW	0 row(s) affected	0.015 sec
84	15:56:51	CREATE TRIGGER before_insurance_update BEFORE UPDATE ON insurance FOR EACH ROW	0 row(s) affected	0.016 sec

AFTER DELETE TRIGGERS:

DELIMITER //

CREATE TRIGGER after_insurance_delete

AFTER DELETE ON insurance

FOR EACH ROW

BEGIN

INSERT INTO insurance_log (insurance_id, action_type)

VALUES (OLD.insurance_id, 'DELETE');

END //

DELIMITER ;

File Edit View Query Database Server Tools Scripting Help

Navigator: motor customers* motor vehicles* motor insurance* motor policy_type* motor premium* motor policies* SQLAdditions... Limit to 1000 rows Jump to

Schemas

- insurance
- policies
- policy_types**
- premium_calculation
- vehicles
- Views
- Stored Procedures

Administration Schemas

Table: policy_types

Columns:

policy_type_id	int AI PK
type_name	varchar(5)
base_rate	decimal(1)
description	text

67 END IF;

68 END //

69

70 DELIMITER ;

71 DELIMITER //

72

73 • CREATE TRIGGER after_insurance_delete
AFTER DELETE ON insurance
FOR EACH ROW
BEGIN

77 INSERT INTO insurance_log (insurance_id, action_type)
VALUES (OLD.insurance_id, 'DELETE');

78 END //

80

81 DELIMITER ;

82

Output:

#	Time	Action	Message	Du
83	15:56:22	CREATE TRIGGER after_insurance_insert AFTER INSERT ON insurance FOR EA...	0 row(s) affected	0.0
84	15:56:51	CREATE TRIGGER before_insurance_update BEFORE UPDATE ON insurance FO...	0 row(s) affected	0.0
85	16:00:25	CREATE TRIGGER after_insurance_delete AFTER DELETE ON insurance FOR E...	0 row(s) affected	0.0

Context Help Snippets