Pedal Works — Service and Sales Analysis

Filtering, joins and aggregation across a bicycle retail and repair chain, generated end to end from a business brief.

Instructions

Answer each of the 8 questions below using a single SQL SELECT statement. Every table reference must be schema-qualified using the 'bikeshop' prefix (e.g., bikeshop.customers), and every column must be table-qualified (e.g., customers.firstName) — do not use aliases or bare column names. Do not use table aliases at any point in your queries. Compile all 8 queries into a single .sql file, with each query preceded by a comment indicating its question number (e.g., -- Question 1). Submit your .sql file to the course portal before the deadline.

Scenario

The BikeShop network operates multiple retail and service locations across the Midwest and beyond, tracking everything from customer bike registrations to detailed repair order histories. Management wants to better understand shop performance, customer value, and inventory trends. As a junior data analyst, you have been tasked with writing SQL queries to answer key business questions using the bikeShop relational database.

Database Structure

Pedal Works runs a small chain of neighbourhood bicycle shops. Each store sells new and refurbished bikes and runs a service counter where customers drop bikes off for repair. Technicians log the services they perform and the parts they consume on every job.

Questions

1. Retrieve the first name, last name, and email of all customers who live in the city of 'Green Bay'. Order the results by last name ascending. *(8 pts)*

Solution:

```
SELECT customers.firstName, customers.lastName, customers.email FROM bikeshop.customers WHERE c
```

2. List the bike brand, model, bike type, and purchase price for all bikes of type 'Mountain', ordered by purchase price from highest to lowest. *(8 pts)*

Solution:

```
SELECT bikes.brand, bikes.model, bikes.bikeType, bikes.purchasePrice FROM bikeshop.bikes WHERE
```

3. Retrieve the part name, part number, category, and unit cost for all parts whose part name contains the word 'bundle' (case-insensitive match is acceptable). Order results by unit cost ascending. *(8 pts)*

Solution:

```
SELECT parts.partName, parts.partNumber, parts.category, parts.unitCost FROM bikeshop.parts WHE
```

4. List the employee first name, last name, role, and hourly rate for all employees whose role is 'Technician'. Order the results by hourly rate descending. *(8 pts)*

Solution:

```
SELECT employees.firstName, employees.lastName, employees.role, employees.hourlyRate FROM bikes
```

5. List the repair order ID, drop-off date, status, and the first and last name of the employee who handled each repair order for all repair orders currently in 'In Progress' status. Order results by drop-off date ascending. *(12 pts)*

> *Hint: Join repairOrders to employees using the foreign key relationship.*

Solution:

```
SELECT repairOrders.repairOrderId, repairOrders.dropOffDate, repairOrders.status, employees.fir
```

6. For each store, show the store name, city, and the total number of repair orders that have been assigned to that store. Include all stores, even those with zero repair orders. Order results by total repair orders descending. *(12 pts)*

> *Hint: Use a LEFT JOIN so that stores with no repair orders still appear. GROUP BY on the store identifier.*

Solution:

```
SELECT stores.storeName, stores.city, COUNT(repairOrders.repairOrderId) AS totalRepairOrders FR
```

7. Find each bike brand along with the total number of bikes registered and the average purchase price for that brand. Only include brands that have more than 30 bikes registered in the system. Order results by average purchase price descending. *(12 pts)*

> *Hint: Use HAVING to filter groups after aggregation.*

Solution:

```
SELECT bikes.brand, COUNT(bikes.bikeId) AS totalBikes, AVG(bikes.purchasePrice) AS avgPurchaseP
```

8. For each customer, show their full name (first and last), their city, the total number of repair orders across all their bikes, and the total line total revenue generated from repair items linked to those repair orders. Only include customers who have generated more than $500 in total line total revenue. Order the results by total revenue descending. *(32 pts)*

> *Hint: Chain joins from customers → bikes → repairOrders → repairItems. Use COUNT(DISTINCT ...) for repair orders to avoid inflation from multiple line items.*

Solution:

```
SELECT customers.firstName, customers.lastName, customers.city, COUNT(DISTINCT repairOrders.rep
```

Answer Key

Q1. (8 pts) — *select, where*

Retrieve the first name, last name, and email of all customers who live in the city of 'Green Bay'. Order the results by last name ascending.

```
SELECT customers.firstName, customers.lastName, customers.email FROM bikeshop.customers WHERE c
```

Tables: bikeshop.customers
Fields: customers.firstName, customers.lastName, customers.email, customers.city

Q2. (8 pts) — *select, where, order_by*

List the bike brand, model, bike type, and purchase price for all bikes of type 'Mountain', ordered by purchase price from highest to lowest.

```
SELECT bikes.brand, bikes.model, bikes.bikeType, bikes.purchasePrice FROM bikeshop.bikes WHERE
```

Tables: bikeshop.bikes
Fields: bikes.brand, bikes.model, bikes.bikeType, bikes.purchasePrice

Q3. (8 pts) — *select, where, like*

Retrieve the part name, part number, category, and unit cost for all parts whose part name contains the word 'bundle' (case-insensitive match is acceptable). Order results by unit cost ascending.

```
SELECT parts.partName, parts.partNumber, parts.category, parts.unitCost FROM bikeshop.parts WHE
```

Tables: bikeshop.parts
Fields: parts.partName, parts.partNumber, parts.category, parts.unitCost

Q4. (8 pts) — *select, where*

List the employee first name, last name, role, and hourly rate for all employees whose role is 'Technician'. Order the results by hourly rate descending.

```
SELECT employees.firstName, employees.lastName, employees.role, employees.hourlyRate FROM bikes
```

Tables: bikeshop.employees
Fields: employees.firstName, employees.lastName, employees.role, employees.hourlyRate

Q5. (12 pts) — *join, where, order_by*

List the repair order ID, drop-off date, status, and the first and last name of the employee who handled each repair order for all repair orders currently in 'In Progress' status. Order results by drop-off date ascending.

```
SELECT repairOrders.repairOrderId, repairOrders.dropOffDate, repairOrders.status, employees.fir
```

Tables: bikeshop.repairOrders, bikeshop.employees

Fields: repairOrders.repairOrderId, repairOrders.dropOffDate, repairOrders.status, employees.firstName, employees.lastName

Q6. (12 pts) — *join, group_by, aggregate*

For each store, show the store name, city, and the total number of repair orders that have been assigned to that store. Include all stores, even those with zero repair orders. Order results by total repair orders descending.

```
SELECT stores.storeName, stores.city, COUNT(repairOrders.repairOrderId) AS totalRepairOrders FR
```

Tables: bikeshop.stores, bikeshop.repairOrders

Fields: stores.storeName, stores.city, repairOrders.repairOrderId

Q7. (12 pts) — *join, group_by, aggregate, having*

Find each bike brand along with the total number of bikes registered and the average purchase price for that brand. Only include brands that have more than 30 bikes registered in the system. Order results by average purchase price descending.

```
SELECT bikes.brand, COUNT(bikes.bikeId) AS totalBikes, AVG(bikes.purchasePrice) AS avgPurchaseP
```

Tables: bikeshop.bikes

Fields: bikes.brand, bikes.bikeId, bikes.purchasePrice

Q8. (32 pts) — *join, group_by, aggregate, having, order_by*

For each customer, show their full name (first and last), their city, the total number of repair orders across all their bikes, and the total line total revenue generated from repair items linked to those repair orders. Only include customers who have generated more than $500 in total line total revenue. Order the results by total revenue descending.

```
SELECT customers.firstName, customers.lastName, customers.city, COUNT(DISTINCT repairOrders.rep
```

Tables: bikeshop.customers, bikeshop.bikes, bikeshop.repairOrders, bikeshop.repairItems

Fields: customers.firstName, customers.lastName, customers.city, repairOrders.repairOrderId, repairItems.lineTotal