SQL Aggregation Functions, GROUP BY & HAVING

Practice Workbook — Library Management System

Beginner Level | Read every explanation carefully before writing any query

Before You Start — Read This First

Everything in this workbook is based on the Library Management System database you already created and filled with data. Before you write any query, take a moment to remind yourself of the tables and what they contain:

Table	What it stores
Library	LibraryID, Name, Location, ContactNumber, EstablishedYear
Book	BookID, ISBN, Title, Genre, Price, IsAvailable, ShelfLocation, LibraryID
Member	MemberID, FullName, Email, PhoneNumber, MembershipStartDate
Staff	StaffID, FullName, Position, ContactNumber, LibraryID
Loan	LoanID, MemberID, BookID, LoanDate, DueDate, ReturnDate, Status
Payment	PaymentID, LoanID, PaymentDate, Amount, PaymentMethod
Review	ReviewID, BookID, MemberID, Rating, Comments, ReviewDate

Tip: If you are ever unsure which columns a table has, run SELECT * FROM TableName; first to see all the data.

Section 1 — Aggregation Functions

What Is an Aggregation Function?

A normal SELECT query returns one row for every row in the table. An aggregation function is different — it reads many rows and returns a single summary number. Think of it like this: instead of listing every book, you ask "how many books are there in total?" and get back one number.

SQL has five main aggregation functions:

Function	What it does
`COUNT(*)`	Counts every row in the table, including rows where some columns are NULL
`COUNT(column)`	Counts only the rows where that specific column is NOT NULL — skips NULLs automatically
`SUM(column)`	Adds up all the numbers in a numeric column
`AVG(column)`	Divides the total by the count to give the average
`MIN(column)`	Finds the single smallest value in a column
`MAX(column)`	Finds the single largest value in a column

Syntax — One Aggregation, Whole Table

When you use an aggregation function without GROUP BY, it looks at every row and gives you one result row back.

```
SELECT COUNT(*) AS TotalRows
FROM   TableName;

-- The AS keyword gives the result column a readable name (alias)
-- Always use an alias with aggregation functions
```

Example: SELECT COUNT(*) AS TotalStaff FROM Staff; — counts all staff members

Section 1 Tasks

Note: Every task in this section returns exactly ONE row. You do not need GROUP BY yet.

Task 1.1: Count the total number of books in the Book table.

Columns to use: *No specific column needed — use COUNT(*)*

Expected result: *1 row — a single number*

Write your query here:

```
SELECT COUNT(*) AS TotalBooks
FROM Book;
```

Task 1.2: Count how many members are registered in the system.

Table: *Member*

Write your query here:

```
SELECT COUNT(*) AS TotalMembers
FROM Member;
```

Task 1.3: Find the total sum of all book prices combined.

Column to sum: *Price in the Book table*

Function to use: *SUM*

Write your query here:

```
SELECT SUM(Price) AS TotalBookPrice
FROM Book;
```

Task 1.4: Calculate the average price of all books.

Function to use: *AVG*

Expected result: *1 row — one decimal number*

Write your query here:

```
SELECT AVG(Price) AS AverageBookPrice
FROM Book;
```

Task 1.5: Find the cheapest book price AND the most expensive book price — both in the same single query.

Functions to use: *MIN and MAX together*

Expected result: *1 row — two columns side by side*

Hint: *You can write multiple aggregation functions in one SELECT: SELECT MIN(col) AS ..., MAX(col) AS ...*

Write your query here:

```
SELECT
    MIN(Price) AS CheapestPrice,
    MAX(Price) AS MostExpensivePrice
FROM Book;
```

Task 1.6: Count how many loans currently have Status = 'Overdue'.

Hint: *Add a WHERE clause to filter only the rows where Status = 'Overdue', then count them*

Note: *WHERE still works with COUNT — filter first, then count*

Write your query here:

```
SELECT COUNT(*) AS TotalOverdueLoans
FROM Loan
WHERE Status = 'Overdue';
```

Task 1.7: Find the highest rating ever given in the Review table.

Column to use: *Rating*

Expected result: *Should be 5*

Write your query here:

```
SELECT MAX(Rating) AS HighestRating
FROM Review;
```

Task 1.8: Find the lowest rating ever given in the Review table.

Write your query here:

```
SELECT MIN(Rating) AS LowestRating
FROM Review;
```

Task 1.9: Calculate the total amount of all fines collected — the sum of every payment ever made.

Table: *Payment*

Column: *Amount*

Write your query here:

```
SELECT SUM(Amount) AS TotalFinesCollected
FROM Payment;
```

Task 1.10: Count how many loans have already been returned (meaning ReturnDate is NOT NULL).

Key concept: *COUNT(column) skips NULL values. If you write COUNT(ReturnDate), rows where ReturnDate is NULL will not be counted*

Hint: *Do not use WHERE here — use COUNT(ReturnDate) directly and let it skip the NULLs for you*

Write your query here:

```
SELECT COUNT(ReturnDate) AS TotalReturnedLoans
FROM Loan;
```

Section 2 — GROUP BY

What Is GROUP BY?

So far, aggregation functions have looked at the entire table and returned one number. GROUP BY lets you break the table into groups and get one number per group.

Without GROUP BY: "How many books are there in total?" → one row, one number
With GROUP BY Genre: "How many books are there in each genre?" → one row per genre

The database first separates the rows into groups based on the column you give GROUP BY, then runs the aggregation function inside each group.

Syntax

```
SELECT   column_to_group_by,  AGG_FUNCTION(another_column) AS alias
FROM     TableName
GROUP BY column_to_group_by;
```

Golden Rule: Every column you put in SELECT that is NOT inside an aggregation function MUST also appear in GROUP BY. If you forget this, SQL will give you an error.

Worked Example — Step by Step

Question: "How many books belong to each genre?"

Step 1: Decide what your groups are → Genre

Step 2: Decide what you want to calculate per group → COUNT(*)

Step 3: Put the group column in both SELECT and GROUP BY

```
SELECT   Genre,  COUNT(*) AS BookCount
FROM     Book
GROUP BY Genre;

-- Result: one row per genre, showing how many books are in each
```

Part 2-A — Simple GROUP BY Tasks

Task 2.1: Count how many books belong to each Genre.

Expected result: *Four rows — one for Fiction, Non-fiction, Reference, Children*

Columns in SELECT: *Genre and COUNT(*)*

Write your query here:

```
SELECT Genre,
COUNT(*) AS TotalBooks
FROM Book
GROUP BY Genre;
```

Task 2.2: Count how many staff members work in each Library. Group by LibraryID.

Table: *Staff*

Expected result: *One row per library — showing the LibraryID and the staff count*

Write your query here:

```
SELECT LibararyID,
COUNT(*) AS TotalStaff
FROM Staff
GROUP BY LibararyID;
```

Task 2.3: Count how many loans exist for each loan Status (Issued, Returned, Overdue).

Table: *Loan*

Expected result: *Three rows — one per status*

Write your query here:

```
SELECT Status,
COUNT(*) AS TotalLoans
FROM Loan
GROUP BY Status;
```

Task 2.4: Calculate the average book price for each Genre.

Function to use: *AVG(Price)*

Expected result: *One row per genre showing average price*

Write your query here:

```
SELECT Genre,
AVG(Price) AS AveragePrice
FROM Book
GROUP BY Genre;
```

Task 2.5: Find the total price (SUM) of all books in each Genre.

Write your query here:

```
SELECT Genre,
SUM(Price) AS TotalPrice
FROM Book
GROUP BY Genre;
```

Task 2.6: Find the most expensive book price (MAX) within each Genre.

Write your query here:

```
SELECT Genre,
MAX(Price) AS MostExpensivePrice
FROM Book
GROUP BY Genre;
```

Task 2.7: Count how many reviews each rating value has received.

Table: *Review*

Group by: *Rating*

Expected result: *One row per rating value (1, 2, 3, 4, 5) — showing how many times each was given*

Write your query here:

```
SELECT Rating,
COUNT(*) AS TotalReviews
FROM Review
GROUP BY Rating
ORDER BY Rating;
```

Task 2.8: Count how many books each Library owns. Group by LibraryID.

Table: *Book*

Expected result: *One row per library*

Write your query here:

```
SELECT LibraryID,
COUNT(*) AS TotalBooks
FROM Book
GROUP BY LibraryID;
```

Task 2.9: Count how many loans each Member has made. Group by MemberID.

Table: *Loan*

Expected result: *One row per member who has at least one loan*

Write your query here:

```
SELECT MemberID,
COUNT(*) AS TotalLoans
FROM Loan
GROUP BY MemberID;
```

Task 2.10: Find the cheapest book (MIN Price) in each Genre.

Write your query here:

```
SELECT Genre,
MIN(Price) AS CheapestPrice
FROM Book
GROUP BY Genre;
```

Part 2-B — GROUP BY with ORDER BY

You can sort the results of a GROUP BY query using ORDER BY. You can even order by the aggregated column using its alias.

```
SELECT   Genre,  COUNT(*) AS BookCount
FROM     Book
GROUP BY Genre
ORDER BY BookCount DESC;   -- most books first
```

Task 2.11: Count books per Genre, ordered from the genre with the most books down to the fewest.

Hint: *Add ORDER BY after GROUP BY and reference your alias*

Write your query here:

```
SELECT Genre,
COUNT(*) AS TotalBooks
FROM Book
GROUP BY Genre
ORDER BY TotalBooks DESC;
```

Task 2.12: Show the average price per Genre, ordered from cheapest average to most expensive.

Write your query here:

```
SELECT Genre,
AVG(Price) AS AveragePrice
FROM Book
GROUP BY Genre
ORDER BY AveragePrice ASC;
```

Task 2.13: Count loans per Status and order the results alphabetically by Status name.

Write your query here:

```
SELECT Status,
COUNT(*) AS TotalLoans
FROM Loan
GROUP BY Status
ORDER BY Status ASC;
```

Task 2.14: Show the total payment amount collected per PaymentMethod, ordered from the highest total to the lowest.

Table: *Payment*

Write your query here:

```
SELECT Method AS PaymentMethod,
SUM(Amount) AS TotalCollected
FROM Payment
GROUP BY Method
ORDER BY TotalCollected DESC;
```

Task 2.15: Count how many reviews each book has received (group by BookID), ordered so the most-reviewed book appears first.

Table: *Review*

Write your query here:

```
SELECT BookID,
COUNT(*) AS TotalReviews
FROM Review
GROUP BY BookID
ORDER BY TotalReviews DESC;
```

Section 3 — HAVING

What Is HAVING?

After GROUP BY creates groups and calculates aggregation values, you sometimes want to keep only certain groups — for example, only genres that have more than 3 books, or only members who have borrowed more than 1 book.

This is what HAVING is for. It filters groups after they have been created, in the same way WHERE filters individual rows before grouping.

Clause	When it runs / what it filters
WHERE	Runs BEFORE grouping — filters individual rows — CANNOT use aggregation functions like COUNT or SUM here
HAVING	Runs AFTER grouping — filters whole groups — CAN and MUST use aggregation functions here

The Most Important Rule

Wrong — this will cause an error: WHERE COUNT(*) > 3

Right — this is correct: HAVING COUNT(*) > 3

Syntax

```
SELECT   column,  AGG_FUNCTION(col) AS alias
FROM     TableName
GROUP BY column
HAVING   AGG_FUNCTION(col)  condition;

-- Example:
SELECT   Genre,  COUNT(*) AS BookCount
FROM     Book
GROUP BY Genre
HAVING   COUNT(*) > 3;   -- only show genres with more than 3 books
```

Part 3-A — Simple HAVING Tasks

Task 3.1: Show only the genres that have MORE THAN 3 books.

Hint: *GROUP BY Genre, then HAVING COUNT(*) > 3*

Write your query here:

```
SELECT Genre,
COUNT(*) AS TotalBooks
FROM Book
GROUP BY Genre
HAVING COUNT(*) > 3;
```

Task 3.2: Show only the libraries that have AT LEAST 2 staff members.

Table: *Staff — group by LibraryID*

Write your query here:

```
SELECT LibararyID,
COUNT(*) AS TotalStaff
FROM Staff
GROUP BY LibararyID
HAVING COUNT(*) >= 2;
```

Task 3.3: Show only the members (by MemberID) who have borrowed MORE THAN 1 book.

Table: *Loan — group by MemberID*

Write your query here:

```
SELECT MemberID,
COUNT(*) AS TotalLoans
FROM Loan
GROUP BY MemberID
HAVING COUNT(*) > 1;
```

Task 3.4: Show only the genres where the average book price is MORE THAN 30.

Hint: *HAVING AVG(Price) > 30*

Write your query here:

```
SELECT Genre,
AVG(Price) AS AveragePrice
FROM Book
GROUP BY Genre
HAVING AVG(Price) > 30;
```

Task 3.5: Find books (by BookID) that have received AT LEAST 2 reviews.

Table: *Review — group by BookID*

Write your query here:

```
SELECT BookID,
COUNT(*) AS TotalReviews
FROM Review
GROUP BY BookID
HAVING COUNT(*) >= 2;
```

Task 3.6: Show genres where the total sum of all book prices is MORE THAN 50.

Hint: *HAVING SUM(Price) > 50*

Write your query here:

```
SELECT Genre,
SUM(Price) AS TotalPrice
FROM Book
GROUP BY Genre
HAVING SUM(Price) > 50;
```

Task 3.7: Find payment methods where the total collected amount is MORE THAN 15.

Table: *Payment — group by PaymentMethod*

Write your query here:

```
SELECT Method AS PaymentMethod,
SUM(Amount) AS TotalCollected
FROM Payment
GROUP BY Method
HAVING SUM(Amount) > 15;
```

Task 3.8: Show loan statuses that appear MORE THAN 3 times in the Loan table.

Write your query here:

```
SELECT Status,
COUNT(*) AS TotalCount
FROM Loan
GROUP BY Status
HAVING COUNT(*) > 3;
```

Task 3.9: Show members (by MemberID) who have written AT LEAST 2 reviews.

Table: *Review — group by MemberID*

Write your query here:

```
SELECT MemberID,
COUNT(*) AS TotalReviews
FROM Review
GROUP BY MemberID
HAVING COUNT(*) >= 2;
```

Task 3.10: Find libraries (by LibraryID) that own MORE THAN 2 books.

Table: *Book — group by LibraryID*

Write your query here:

```
SELECT LibraryID,
COUNT(*) AS TotalBooks
FROM Book
GROUP BY LibraryID
HAVING COUNT(*) > 2;
```

Part 3-B — WHERE and HAVING Together

You can use WHERE and HAVING in the same query. Remember: WHERE runs first and removes individual rows, then GROUP BY creates groups from what is left, then HAVING filters those groups.

```
SELECT   Genre,  COUNT(*) AS AvailableCount
FROM     Book
WHERE    IsAvailable = 1          -- Step 1: keep only available books
GROUP BY Genre                    -- Step 2: group what is left
HAVING   COUNT(*) > 1;           -- Step 3: keep only groups with more
than 1
```

Task 3.11: Count available books per genre (available means IsAvailable = 1). Show only genres that have more than 1 available book.

Structure: *WHERE filters rows → GROUP BY groups → HAVING filters groups*

Write your query here:

```
SELECT Genre,
COUNT(*) AS AvailableBooks
FROM Book
WHERE IsAvailable = 1
GROUP BY Genre
HAVING COUNT(*) > 1;
```

Task 3.12: Among Fiction and Children books only, show genres where the average price is above 15.

Hint: *Use WHERE Genre IN ('Fiction', 'Children') then GROUP BY then HAVING AVG(Price) > 15*

Write your query here:

```
SELECT Genre,
AVG(Price) AS AveragePrice
FROM Book
WHERE Genre IN ('Fiction', 'Children')
GROUP BY Genre
HAVING AVG(Price) > 15;
```

Task 3.13: Among Overdue and Issued loans only, count per member and show members with more than 1 active loan.

Hint: *WHERE Status IN ('Overdue', 'Issued')*

Write your query here:

```
SELECT MemberID,
COUNT(*) AS ActiveLoans
FROM Loan
WHERE Status IN ('Overdue', 'Issued')
GROUP BY MemberID
HAVING COUNT(*) > 1;
```

Task 3.14: Look only at reviews with Rating >= 3. Group by book and show books that have at least 2 good reviews.

Write your query here:

```
SELECT BookID,
COUNT(*) AS GoodReviews
FROM Review
WHERE Rating >= 3
GROUP BY BookID
HAVING COUNT(*) >= 2;
```

Task 3.15: Find genres that have more than 1 book priced below 20.

Hint: *WHERE Price < 20 first, then group, then HAVING*

Write your query here:

```
SELECT Genre,
COUNT(*) AS BooksBelow20
FROM Book
WHERE Price < 20
GROUP BY Genre
HAVING COUNT(*) > 1;
```

Section 4 — Aggregation with JOIN

Combining Aggregation and JOIN

So far all your aggregation tasks used a single table. In real life, the column you want to group by is often in a different table from the column you want to count or sum. The solution is to JOIN the tables first, then use GROUP BY.

The pattern is always the same:

```
-- 1. JOIN the tables to get the columns you need in one place
-- 2. GROUP BY the column you want to group on
-- 3. Apply the aggregation function

SELECT   T1.ColumnName,  AGG(T2.ColumnName) AS alias
FROM     Table1 T1
JOIN     Table2 T2  ON  T1.ID = T2.ID
GROUP BY T1.ColumnName;
```

Important: When you GROUP BY a column that came from a JOIN, always include the primary key of that table in GROUP BY alongside the display column — for example GROUP BY LIB.LibraryID, LIB.Name. This avoids ambiguity.

Part 4-A — Aggregation + JOIN Tasks

Task 4.1: Show each library name alongside the number of books it owns.

Tables needed: *Library and Book*

Join condition: *Library.LibraryID = Book.LibraryID*

Hint: *JOIN first, then GROUP BY LIB.LibraryID, LIB.Name, then COUNT(B.BookID)*

Write your query here:

```
SELECT LI.LibararyID, LI.Name AS LibraryName,
COUNT(B.BookID) AS TotalBooks
FROM Libarary LI
LEFT JOIN Book B
   ON LI.LibararyID = B.LibraryID
GROUP BY LI.LibararyID, LI.Name;
```

Task 4.2: Show each member's full name alongside their total number of loans.

Tables needed: *Member and Loan*

Join condition: *Member.MemberID = Loan.MemberID*

Write your query here:

```
SELECT M.MemberID, M.FullName,
COUNT(L.LoanID) AS TotalLoans
FROM Member M LEFT JOIN Loan L
ON M.MemberID = L.MemberID
GROUP BY  M.MemberID, M.FullName;
```

Task 4.3: Show each book's title alongside the number of times it has been borrowed.

Tables needed: *Book and Loan*

Join condition: *Book.BookID = Loan.BookID*

Write your query here:

```
SELECT B.BookID, B.Title,
COUNT(L.LoanID) AS TimesBorrowed
FROM Book B LEFT JOIN Loan L
ON B.BookID = L.BookID
GROUP BY  B.BookID, B.Title;
```

Task 4.4: Show each book's title alongside its average review rating.

Tables needed: *Book and Review*

Function: *AVG(R.Rating)*

Hint: *Cast the rating for a cleaner decimal: AVG(CAST(R.Rating AS DECIMAL(3,2)))*

Write your query here:

```
SELECT B.BookID, B.Title,
AVG(CAST(R.Rating AS DECIMAL(3,2))) AS AverageRating
FROM Book B LEFT JOIN Review R
ON B.BookID = R.BookID
GROUP BY  B.BookID, B.Title;
```

Task 4.5: Show each library name alongside the total value (SUM of Price) of all the books it owns.

Tables needed: *Library and Book*

Write your query here:

```
SELECT LI.LibararyID, LI.Name AS LibraryName,
SUM(B.Price) AS TotalBookValue
FROM Libarary LI LEFT JOIN Book B
ON LI.LibararyID = B.LibraryID
GROUP BY  LI.LibararyID, LI.Name;
```

Task 4.6: Show each library name alongside the number of staff who work there.

Tables needed: *Library and Staff*

Write your query here:

```
SELECT LI.LibararyID, LI.Name AS LibraryName,
COUNT(S.StaffID) AS TotalStaff
FROM Libarary LI LEFT JOIN Staff S
ON LI.LibararyID = S.LibararyID
GROUP BY  LI.LibararyID,  LI.Name;
```

Task 4.7: Show each member's full name alongside the total fine amount they have paid.

Tables needed: *Member, Loan, and Payment — three tables*

Hint: *JOIN Member to Loan, then JOIN Loan to Payment, then GROUP BY member, then SUM(P.Amount)*

Write your query here:

```
SELECT M.MemberID, M.FullName,
SUM(P.Amount) AS TotalFinePaid
FROM Member M LEFT JOIN Loan L
ON M.MemberID = L.MemberID
LEFT JOIN Payment P
ON L.LoanID = P.LoanID
GROUP BY  M.MemberID, M.FullName;
```

Task 4.8: Show each genre and the number of DISTINCT members who have borrowed books in that genre.

Tables needed: *Book and Loan*

Hint: *COUNT(DISTINCT L.MemberID) — this avoids counting the same member twice if they borrowed multiple books in the same genre*

Write your query here:

```
SELECT B.Genre,
COUNT(DISTINCT L.MemberID) AS DistinctBorrowers
FROM Book B INNER JOIN Loan L
ON B.BookID = L.BookID
GROUP BY B.Genre;
```

Task 4.9: Show each book's title, how many times it was borrowed, and its average rating — all in one query.

Tables needed: *Book, Loan, and Review*

Hint: *Use LEFT JOIN so that books with no loans and books with no reviews still appear in the result*

Two aggregations: *COUNT(DISTINCT L.LoanID) for borrows, AVG(CAST(R.Rating AS DECIMAL(3,2))) for rating*

Write your query here:

```
SELECT B.BookID, B.Title,
COUNT(DISTINCT L.LoanID) AS TimesBorrowed,
AVG(CAST(R.Rating AS DECIMAL(3,2))) AS AverageRating
FROM Book B LEFT JOIN Loan L
ON B.BookID = L.BookID
LEFT JOIN Review R
ON B.BookID = R.BookID
GROUP BY  B.BookID, B.Title;
```

Task 4.10: Show each genre with three pieces of information: total number of books, number of available books, and average price.

Table: *Book only — no JOIN needed for this one*

Hint for available count: *Use a CASE expression inside SUM: SUM(CASE WHEN IsAvailable = 1 THEN 1 ELSE 0 END)*

Write your query here:

```
SELECT Genre,
COUNT(*) AS TotalBooks,
SUM(CASE WHEN IsAvailable = 1 THEN 1 ELSE 0 END) AS AvailableBooks,
AVG(Price) AS AveragePrice
FROM Book
GROUP BY Genre;
```

Part 4-B — Aggregation + JOIN + HAVING

Now add HAVING on top of your JOIN queries to filter the groups.

Task 4.11: Show only the library names that own MORE THAN 2 books.

Tables: *Library and Book*

Write your query here:

```
SELECT LI.LibararyID, LI.Name AS LibraryName,
COUNT(B.BookID) AS TotalBooks
FROM Libarary LI LEFT JOIN Book B
ON LI.LibararyID = B.LibraryID
GROUP BY  LI.LibararyID, LI.Name
HAVING COUNT(B.BookID) > 2;
```

Task 4.12: Show only the members' full names who have borrowed MORE THAN 1 book.

Tables: *Member and Loan*

Write your query here:

```
SELECT  M.MemberID, M.FullName,
COUNT(L.LoanID) AS TotalLoans
FROM Member M INNER JOIN Loan L
ON M.MemberID = L.MemberID
GROUP BY  M.MemberID, M.FullName
HAVING COUNT(L.LoanID) > 1;
```

Task 4.13: Show only the book titles that have an average rating ABOVE 4.

Tables: *Book and Review*

Write your query here:

```
SELECT  B.BookID, B.Title,
AVG(CAST(R.Rating AS DECIMAL(3,2))) AS AverageRating
FROM Book B INNER JOIN Review R
ON B.BookID = R.BookID
GROUP BY B.BookID, B.Title
HAVING AVG(CAST(R.Rating AS DECIMAL(3,2))) > 4;
```

Task 4.14: Show only the genres where more than 3 loans have been made in total.

Tables: *Book and Loan*

Write your query here:

```
SELECT B.Genre,
COUNT(L.LoanID) AS TotalLoans
FROM Book B INNER JOIN Loan L
ON B.BookID = L.BookID
GROUP BY  B.Genre
HAVING COUNT(L.LoanID) > 3;
```

Task 4.15: Show only the libraries where the total fines collected from their books are MORE THAN 10.

Tables: *Library, Book, Loan, and Payment — four tables*

Hint: *Join all four in a chain: Library → Book → Loan → Payment*

Write your query here:

```
SELECT LI.LibararyID, LI.Name AS LibraryName,
SUM(P.Amount) AS TotalFinesCollected
FROM Libarary LI INNER JOIN Book B
ON B.LibraryID = LI.LibararyID
INNER JOIN Loan L
ON L.BookID = B.BookID
INNER JOIN Payment P
ON P.LoanID = L.LoanID
GROUP BY LI.LibararyID, LI.Name
HAVING SUM(P.Amount) > 10;
```

Section 5 — Putting It All Together

SQL Execution Order

SQL does not run your clauses in the order you write them. Understanding the real execution order will help you avoid errors and reason about your queries correctly.

Order	Clause — what it does
1st	FROM — load all rows from the table(s)
2nd	JOIN — combine tables
3rd	WHERE — filter individual rows (before grouping)
4th	GROUP BY — group the remaining rows
5th	HAVING — filter whole groups (after grouping)
6th	SELECT — calculate the output columns and aliases
7th	ORDER BY — sort the final results

These five challenge tasks require you to use all the clauses together in the correct order. Read each task carefully before you write anything.

Task 5.1: Count available books (WHERE IsAvailable = 1) per genre. Show only genres with more than 1 available book. Order the results from most available books to fewest.

Clauses needed: *FROM → WHERE → GROUP BY → HAVING → ORDER BY*

Write your query here:

```
SELECT Genre,
COUNT(*) AS AvailableBooks
FROM Book
WHERE IsAvailable = 1
GROUP BY Genre
HAVING COUNT(*) > 1
ORDER BY AvailableBooks DESC;
```

Task 5.2: For each library, show the library name and the total number of loans ever made from its books. Show only libraries that have had more than 2 loans. Order by loan count from highest to lowest.

Tables: *Library, Book, Loan*

Clauses needed: *FROM → JOIN → JOIN → GROUP BY → HAVING → ORDER BY*

Write your query here:

```
SELECT LI.LibararyID, LI.Name AS LibraryName,
COUNT(L.LoanID) AS TotalLoans
FROM Libarary LI INNER JOIN Book B
ON B.LibraryID = LI.LibararyID
INNER JOIN Loan L
ON L.BookID = B.BookID
GROUP BY LI.LibararyID, LI.Name
HAVING COUNT(L.LoanID) > 2
ORDER BY TotalLoans DESC;
```

Task 5.3: Show each member's name and how many overdue loans they have (Status = 'Overdue'). Show only members who have at least 1 overdue loan. Order alphabetically by name.

Tables: *Member and Loan*

Hint: *WHERE filters to Overdue rows first, then you group and count*

Write your query here:

```
SELECT M.MemberID, M.FullName,
COUNT(L.LoanID) AS OverdueLoans
FROM Member M INNER JOIN Loan L
ON M.MemberID = L.MemberID
WHERE L.Status = 'Overdue'
GROUP BY M.MemberID, M.FullName
HAVING COUNT(L.LoanID) >= 1
ORDER BY M.FullName ASC;
```

Task 5.4: For each book, show the title, total number of borrows, and average review rating. Show only books that have been borrowed at least once AND have an average rating above 3. Order by average rating from highest to lowest.

Tables: *Book, Loan, Review*

Hint: *Use LEFT JOIN so books with no reviews still appear — then HAVING filters them out*

Two conditions in HAVING: *HAVING COUNT(...) >= 1 AND AVG(...) > 3*

Write your query here:

```
SELECT B.BookID, B.Title,
COUNT(DISTINCT L.LoanID) AS TotalBorrows,
AVG(CAST(R.Rating AS DECIMAL(3,2))) AS AverageRating
FROM Book B LEFT JOIN Loan L
ON B.BookID = L.BookID
LEFT JOIN Review R
ON B.BookID = R.BookID
GROUP BY B.BookID, B.Title
HAVING COUNT(DISTINCT L.LoanID) >= 1
AND AVG(CAST(R.Rating AS DECIMAL(3,2))) > 3
ORDER BY AverageRating DESC;
```

Task 5.5: For each genre, show the total number of books, the average price, and the cheapest price. Show only genres where the average price is between 15 and 50. Order by average price from lowest to highest.

Hint: *HAVING AVG(Price) BETWEEN 15 AND 50*

Write your query here:

```
SELECT Genre,
COUNT(*) AS TotalBooks,
AVG(Price) AS AveragePrice,
MIN(Price) AS CheapestPrice
FROM Book
GROUP BY Genre
HAVING AVG(Price) BETWEEN 15 AND 50
ORDER BY AveragePrice ASC;
```

Task 5.6 — Challenge: Build a complete library summary report. For each library show: library name, city (Location), total number of books, total number of staff, total number of loans ever made, and total fines collected. Show only libraries that have had at least one loan. Order by total loans from highest to lowest.

Tables needed: *Library, Book, Staff, Loan, Payment — five tables*

Hint: *Use LEFT JOIN for all joins so libraries with missing data still appear*

Hint: *Use COUNT(DISTINCT ...) for books, staff, and loans to avoid inflated counts from multiple joins*

Note: *This is the hardest query in the workbook — think through each join carefully before writing*

Write your query here:

```
SELECT LI.LibararyID, LI.Name AS LibraryName,
LI.Location AS City,
COUNT(DISTINCT B.BookID)  AS TotalBooks,
COUNT(DISTINCT S.StaffID) AS TotalStaff,
COUNT(DISTINCT L.LoanID)  AS TotalLoans,
SUM(P.Amount) AS TotalFinesCollected
FROM Libarary LI LEFT JOIN Book B
ON B.LibraryID = LI.LibararyID
LEFT JOIN Staff S
ON S.LibararyID = LI.LibararyID
LEFT JOIN Loan L
ON L.BookID = B.BookID
LEFT JOIN Payment P
ON P.LoanID = L.LoanID
GROUP BY LI.LibararyID, LI.Name, LI.Location
HAVING COUNT(DISTINCT L.LoanID) >= 1
ORDER BY TotalLoans DESC;
```

Evaluation Criteria

Section / Criteria	Points
Section 1 — Aggregation Functions (10 tasks)	15 pts
Section 2 — GROUP BY (15 tasks)	20 pts
Section 3 — HAVING (15 tasks)	25 pts
Section 4 — Aggregation with JOIN (15 tasks)	25 pts
Section 5 — Full challenge queries (6 tasks)	10 pts
Code quality — comments, aliases, formatting	5 pts
TOTAL	**100 pts**

Submission Guidelines

1. **SQL Script (.sql file):** One file with all your queries, organised by section. Add a comment above every query explaining in plain words what it does.

2. **Screenshots:** At least one screenshot per section showing the query and its result.

3. **Reflection (half a page):** Explain the difference between WHERE and HAVING in your own words, using one example of each from your own work today.

Quick Reference Card

```
-- COUNT all rows (including NULLs)
SELECT COUNT(*) AS alias FROM table;

-- COUNT non-NULL values only
SELECT COUNT(column) AS alias FROM table;

-- SUM, AVG, MIN, MAX
SELECT SUM(col), AVG(col), MIN(col), MAX(col) FROM table;

-- GROUP BY — one result row per group
SELECT col, COUNT(*) AS alias FROM table GROUP BY col;

-- HAVING — filter groups (never use WHERE for this)
SELECT col, COUNT(*) AS alias FROM table GROUP BY col HAVING COUNT(*) >
5;
```

```
-- Full correct order
SELECT   col, AGG(col2) AS alias
FROM     table
JOIN     other ON ...
WHERE    condition_on_individual_rows
GROUP BY col
HAVING   condition_on_groups
ORDER BY alias DESC;
```