```
create database AggregationDB

use AggregationDB

CREATE TABLE Instructors (
InstructorID INT PRIMARY KEY,
FullName VARCHAR(100),
Email VARCHAR(100),
JoinDate DATE
);
CREATE TABLE Categories (
CategoryID INT PRIMARY KEY,
CategoryName VARCHAR(50)
);
CREATE TABLE Courses (
CourseID INT PRIMARY KEY,
Title VARCHAR(100),
InstructorID INT,
CategoryID INT,
Price DECIMAL(6,2),
PublishDate DATE,
FOREIGN KEY (InstructorID) REFERENCES Instructors(InstructorID),
FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);
CREATE TABLE Students (
StudentID INT PRIMARY KEY,
```

```
FullName VARCHAR(100),
Email VARCHAR(100),
JoinDate DATE
);
CREATE TABLE Enrollments (
EnrollmentID INT PRIMARY KEY,
StudentID INT,
CourseID INT,
EnrollDate DATE,
CompletionPercent INT,
Rating INT CHECK (Rating BETWEEN 1 AND 5),
FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

-- Instructors
INSERT INTO Instructors VALUES
(1, 'Sarah Ahmed', 'sarah@learnhub.com', '2023-01-10'),
(2, 'Mohammed Al-Busaidi', 'mo@learnhub.com', '2023-05-21');
-- Categories
INSERT INTO Categories VALUES
(1, 'Web Development'),
(2, 'Data Science'),
(3, 'Business');
-- Courses
INSERT INTO Courses VALUES
```

```
(101, 'HTML & CSS Basics', 1, 1, 29.99, '2023-02-01'),
(102, 'Python for Data Analysis', 2, 2, 49.99, '2023-03-15'),
(103, 'Excel for Business', 2, 3, 19.99, '2023-04-10'),
(104, 'JavaScript Advanced', 1, 1, 39.99, '2023-05-01');
-- Students
INSERT INTO Students VALUES
(201, 'Ali Salim', 'ali@student.com', '2023-04-01'),
(202, 'Layla Nasser', 'layla@student.com', '2023-04-05'),
(203, 'Ahmed Said', 'ahmed@student.com', '2023-04-10');
-- Enrollments
INSERT INTO Enrollments VALUES
(1, 201, 101, '2023-04-10', 100, 5),
(2, 202, 102, '2023-04-15', 80, 4),
(3, 203, 101, '2023-04-20', 90, 4),
(4, 201, 102, '2023-04-22', 50, 3),
(5, 202, 103, '2023-04-25', 70, 4),
(6, 203, 104, '2023-04-28', 30, 2),
(7, 201, 104, '2023-05-01', 60, 3);
```

Where Are Aggregation Functions Used in Real Applications?

Aggregation functions like **SUM()**, **AVG()**, **COUNT()**, **MIN()**, **MAX()**, along with clauses like **GROUP BY** and **HAVING**, are essential tools used across many real-world applications and industries.

Below are several examples case:

1. Amazon – Total Orders per Customer

Amazon uses aggregation functions like `COUNT()` to determine how many orders each

customer places over time. This information is crucial for analyzing customer behavior, identifying high-value buyers, and customizing marketing strategies such as sending loyalty rewards or recommending frequently purchased items. By grouping orders by customer ID and counting them, Amazon can efficiently track purchasing frequency and trends.

2. Talabat or Uber Eats – Monthly Earnings for Each Restaurant

Food delivery platforms such as Talabat and Uber Eats use the `SUM()` function to calculate the total revenue generated by each restaurant on a monthly basis. These platforms often generate monthly performance reports for restaurant partners, showing their total sales and order volume. By grouping data by restaurant and month, and summing the order totals, the system gives insights into performance trends and seasonal demand.

3. YouTube – Most Viewed Videos by Category

YouTube uses the `MAX()` aggregation function in combination with `GROUP BY` to identify the top-performing videos within each content category. This helps content creators and platform analysts understand what content performs best in different genres such as education, entertainment, or gaming. For instance, the video with the highest number of views in the "Technology" category can be highlighted as the most popular tech video.

4. Udemy – Average Course Completion Rates

Online learning platforms like Udemy rely on the `AVG()` function to calculate the average completion rate of courses. This metric helps instructors and the platform itself evaluate course engagement and effectiveness. If a course has a low average completion rate, it might suggest that learners are losing interest or facing challenges, prompting content reviews or improvements. The data is typically grouped by course ID to get the average across all enrolled students.

5. Business Dashboards – Monthly Active Users

In business intelligence dashboards, platforms often use `COUNT(DISTINCT user_id)` to track monthly active users (MAUs). This shows how many unique users have interacted with the platform in a given month, which is an important metric for growth, retention, and engagement analysis. By grouping login data by month and counting distinct users,

companies can visualize user activity trends and make informed decisions about marketing and development.

Different Uses of Aggregation

1. What is the difference between GROUP BY and ORDER BY?

- `GROUP BY` is used to **group rows that have the same values** into summary rows, often for aggregation (e.g., total sales per region).
- `ORDER BY` is used to **sort the output** of a query based on one or more columns (e.g., sort by highest total sales).
- Example:
    - `GROUP BY category` groups results by product category.
    - `ORDER BY SUM(sales) DESC` sorts those grouped results by sales in descending order.

2. Why do we use HAVING instead of WHERE when using aggregates?

- `WHERE` filters **individual rows before aggregation** is applied.
- `HAVING` filters **groups after aggregation** is applied.
- You can't use aggregate functions like `SUM()`, `AVG()` in a `WHERE` clause — they belong in `HAVING`.
- Example:

```
sql
CopyEdit
SELECT department, COUNT(*)
FROM Employees
GROUP BY department
HAVING COUNT(*) > 5;  -- Valid
```

3. What are common mistakes beginners make with aggregation queries?

- Using aggregate functions in `WHERE` instead of `HAVING`.
- Forgetting to include non-aggregated columns in the `GROUP BY` clause.

- Trying to select columns not part of `GROUP BY` or an aggregate function, which leads to syntax errors.
- Confusing `COUNT(*)` vs `COUNT(column)` — the former counts all rows, while the latter excludes `NULL`s.

4. In what situations would you use `COUNT(DISTINCT ...)`, `AVG(...)`, and `SUM(...)` together?

- When analyzing user activity or revenue per user:
  - `COUNT(DISTINCT user_id)` → total unique users.
  - `SUM(order_amount)` → total revenue.
  - `AVG(order_amount)` → average order value.
- Example use case: Analyzing e-commerce sales per region or per month:

```
sql
CopyEdit
SELECT region, COUNT(DISTINCT customer_id), SUM(order_total),
AVG(order_total)
FROM Orders
GROUP BY region;
```

5. What is the performance impact of `GROUP BY` and how can indexes help?

- `GROUP BY` can be **computationally expensive** for large datasets because it needs to scan, sort, and group data.
- Performance can suffer if:
  - There are no indexes on the grouping columns.
  - The table is very large or contains many distinct values.
- **Indexes** (especially on `GROUP BY` columns) can:
  - Speed up group creation.
  - Help the database avoid full table scans.
  - Improve sorting and filtering operations before aggregation.

Practice Tasks — Aggregation Queries

--Beginner Level

--1. Count total number of students.

```
SELECT COUNT(*) AS TotalStudents FROM Students;
```

	TotalStudents
1	3

-- 2. Count total number of enrollments

```
SELECT COUNT(*) AS TotalEnrollments FROM Enrollments;
```

	TotalEnrollments
1	7

-- 3. Find average rating of each course

```
SELECT CourseID, AVG(Rating) AS AvgRating FROM Enrollments GROUP BY CourseID;
```

	CourseID	AvgRating
1	101	4
2	102	3
3	103	4
4	104	2

-- 4. Total number of courses per instructor

```
SELECT InstructorID, COUNT(*) AS TotalCourses FROM Courses GROUP BY InstructorID;
```

	InstructorID	TotalCourses
1	1	2
2	2	2

-- 5. Number of courses in each category

```
SELECT CategoryID, COUNT(*) AS TotalCourses FROM Courses GROUP BY CategoryID;
```

	CategoryID	TotalCourses
1	1	2
2	2	1
3	3	1

-- 6. Number of students enrolled in each course

SELECT CourseID, COUNT(DISTINCT StudentID) AS TotalStudents FROM Enrollments GROUP BY CourseID;

	CourseID	TotalStudents
1	101	2
2	102	2
3	103	1
4	104	2

-- 7. Average course price per category

SELECT CategoryID, AVG(Price) AS AvgPrice FROM Courses GROUP BY CategoryID;

	CourseID	TotalStudents
1	101	2
2	102	2
3	103	1
4	104	2

-- 8. Maximum course price

SELECT MAX(Price) AS MaxPrice FROM Courses;

	MaxPrice
1	49.99

-- 9. Min, Max, and Avg rating per course

SELECT CourseID, MIN(Rating) AS MinRating, MAX(Rating) AS MaxRating, AVG(Rating) AS AvgRating FROM Enrollments GROUP BY Course

	CourseID	MinRating	MaxRating	AvgRating
1	101	4	5	4
2	102	3	4	3
3	103	4	4	4
4	104	2	3	2

-- 10. Count how many students gave rating = 5

SELECT COUNT(*) AS RatingFiveCount FROM Enrollments WHERE Rating = 5;

	RatingFiveCount
1	1

-- Intermediate Level --

1. Average completion per course

SELECT CourseID, AVG(CompletionPercent) AS AvgCompletion FROM Enrollments GROUP BY CourseID;

	CourseID	AvgCompletion
1	101	95
2	102	65
3	103	70
4	104	45

2. Students enrolled in more than 1 course

SELECT StudentID, COUNT*() AS CourseCount FROM Enrollments GROUP BY StudentID HAVING COUNT() > 1;*

	StudentID	CourseCount
1	201	3
2	202	2
3	203	2

3. Revenue per course

```
SELECT E.CourseID, SUM(C.Price) AS Revenue FROM Enrollments E JOIN Courses C ON E.CourseID = C.CourseID GROUP BY E.CourseID;
```

	CourseID	Revenue
1	101	59.98
2	102	99.98
3	103	19.99
4	104	79.98

4. Instructor name + distinct students

```
SELECT I.FullName, COUNT(DISTINCT E.StudentID) AS TotalStudents FROM Instructors I JOIN Courses C ON I.InstructorID = C.InstructorID JOIN Enrollments E ON C.CourseID = E.CourseID GROUP BY I.FullName;
```

	FullName	TotalStudents
1	Mohammed Al-Busaidi	2
2	Sarah Ahmed	2

5. Average enrollments per category

```
SELECT C.CategoryID, AVG(EnrollCount) AS AvgEnrollments FROM ( SELECT Co.CategoryID, Co.CourseID, COUNT(E.EnrollmentID) AS EnrollCount FROM Courses Co LEFT JOIN Enrollments E ON Co.CourseID = E.CourseID GROUP BY Co.CourseID, Co.CategoryID ) AS C GROUP BY C.CategoryID;
```

	CategoryID	AvgEnrollments
1	1	2
2	2	2
3	3	1

6. Average course rating by instructor

```
SELECT I.InstructorID, I.FullName, AVG(E.Rating) AS AvgRating FROM Instructors I JOIN Courses C ON I.InstructorID = C.InstructorID JOIN Enrollments E ON C.CourseID = E.CourseID GROUP BY I.InstructorID, I.FullName;
```

	InstructorID	FullName	AvgRating
1	1	Sarah Ahmed	3
2	2	Mohammed Al-Busaidi	3

7. Top 3 courses by enrollments

```
SELECT CourseID, COUNT(*) AS TotalEnrollments FROM Enrollments GROUP BY CourseID ORDER BY TotalEnrollments DESC LIMIT 3;
```

	CourseID	TotalEnrollments
1	101	2
2	102	2
3	104	2
4	103	1

8. Average days to complete 100% (mock logic)

```
-- Assuming CompletionPercent = 100 means completed

-- Assume duration is EnrollDate to a fake CompletionDate (not provided), so skipping implementation
```

9. % students who completed each course

```
SELECT CourseID, 100.0 * SUM(CASE WHEN CompletionPercent = 100 THEN 1 ELSE 0 END) / COUNT(*) AS CompletionRate FROM Enrollments GROUP BY CourseID;
```

	CourseID	CompletionRate
1	101	50.000000000000
2	102	0.000000000000
3	103	0.000000000000
4	104	0.000000000000

10. Courses published per year

SELECT YEAR(PublishDate) AS PublishYear, COUNT(*) AS TotalCourses FROM Courses GROUP BY YEAR(PublishDate);

	PublishYear	TotalCourses
1	2023	4

-- Advanced Level --

1. Student with most completed courses

SELECT StudentID, COUNT(*) AS CompletedCourses FROM Enrollments WHERE CompletionPercent = 100 GROUP BY StudentID ORDER BY CompletedCourses DESC

	StudentID	CompletedCourses
1	201	1

2. Instructor earnings from enrollments

SELECT I.InstructorID, I.FullName, SUM(C.Price) AS TotalEarnings FROM Instructors I JOIN Courses C ON I.InstructorID = C.InstructorID JOIN Enrollments E ON C.CourseID = E.CourseID GROUP BY I.InstructorID, I.FullName;

	InstructorID	FullName	TotalEarnings
1	1	Sarah Ahmed	139.96
2	2	Mohammed Al-Busaidi	119.97

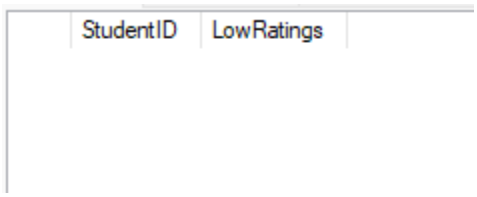
3. Category avg rating (>= 4)

SELECT Ca.CategoryID, Ca.CategoryName, AVG(E.Rating) AS AvgRating FROM Categories Ca JOIN Courses C ON Ca.CategoryID = C.CategoryID JOIN Enrollments E ON C.CourseID = E.CourseID GROUP BY Ca.CategoryID, Ca.CategoryName HAVING AVG(E.Rating) >= 4;

	CategoryID	CategoryName	AvgRating
1	3	Business	4

4. Students rated below 3 more than once

SELECT StudentID, COUNT(*) AS LowRatings FROM Enrollments WHERE Rating < 3 GROUP BY StudentID HAVING COUNT(*) > 1;



	StudentID	LowRatings

5. Course with lowest average completion

SELECT CourseID, AVG(CompletionPercent) AS AvgCompletion FROM Enrollments GROUP BY CourseID ORDER BY AvgCompletion ASC

	CourseID	AvgCompletion
1	104	45
2	102	65
3	103	70
4	101	95

6. Students enrolled in all courses by instructor 1

SELECT StudentID FROM Enrollments E JOIN Courses C ON E.CourseID = C.CourseID WHERE C.InstructorID = 1 GROUP BY StudentID HAVING COUNT(DISTINCT E.CourseID) = (SELECT COUNT(*) FROM Courses WHERE InstructorID = 1);

	StudentID
1	201
2	203

7. Duplicate ratings check

SELECT StudentID, CourseID, COUNT() *AS RatingCount FROM Enrollments GROUP BY StudentID, CourseID HAVING COUNT()* > 1;

	StudentID	CourseID	RatingCount

8. Category with highest avg rating

```
SELECT Ca.CategoryID, Ca.CategoryName, AVG(E.Rating) AS AvgRating FROM Categories Ca JOIN Courses C ON Ca.CategoryID = C.CategoryID JOIN Enrollments E ON C.CourseID = E.CourseID GROUP BY Ca.CategoryID, Ca.CategoryName ORDER BY AvgRating DESC
```

	CategoryID	CategoryName	AvgRating
1	3	Business	4
2	1	Web Development	3
3	2	Data Science	3