All questions can be viewed easily on my GitHub here:

<https://github.com/jacket430/ABM-College/tree/main/SQL/Final>

Each question block can be fired in their entirety.

Question 1

CREATE TABLE Students(

    StudentID INT PRIMARY KEY IDENTITY(1,1) NOT NULL,

    Name NVARCHAR(100) NOT NULL,

    Email NVARCHAR(100) NOT NULL,

    EnrollmentDate DATE NOT NULL DEFAULT GETDATE() -- Enrollment date sets to today's date by default, can be changed.

);

CREATE TABLE Exams(

    ExamID INT PRIMARY KEY IDENTITY(1,1) NOT NULL,

    Title NVARCHAR(100) NOT NULL,

    Date DATE NOT NULL

);

CREATE TABLE Questions(

    QuestionID INT PRIMARY KEY IDENTITY(1,1) NOT NULL,

    ExamID INT NOT NULL,

    Text NVARCHAR(MAX) NOT NULL,

    CONSTRAINT FK\_Questions\_Exams FOREIGN KEY (ExamID) REFERENCES Exams(ExamID) -- Constrained foreign key linking ExamID to Exams table.

);

CREATE TABLE Results(

    ResultID INT PRIMARY KEY IDENTITY(1,1) NOT NULL,

    StudentID INT NOT NULL,

    ExamID INT NOT NULL,

    QuestionID INT NOT NULL,

    Answer NVARCHAR(MAX) NOT NULL,

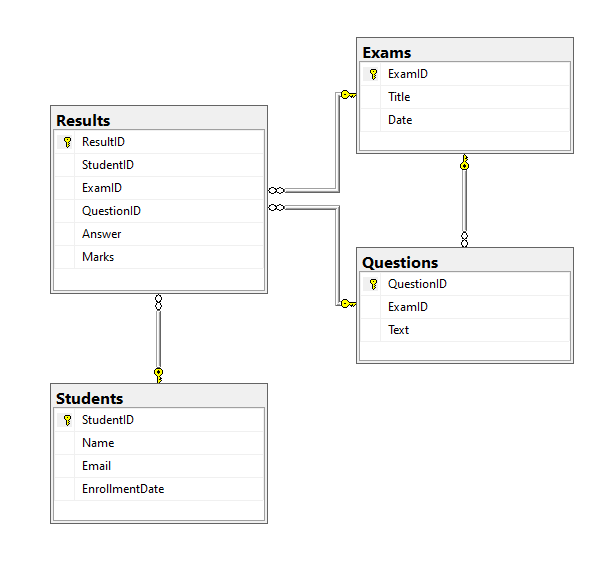
    Marks INT NOT NULL,

    CONSTRAINT FK\_Results\_Students FOREIGN KEY (StudentID) REFERENCES Students(StudentID), -- Constrained foreign key linking StudentID to Students.

    CONSTRAINT FK\_Results\_Exams FOREIGN KEY (ExamID) REFERENCES Exams(ExamID), -- Constrained foreign key linking ExamID to Exams.

    CONSTRAINT FK\_Results\_Questions FOREIGN KEY (QuestionID) REFERENCES Questions(QuestionID), -- Constrained foreign key linking QuestionID to Questions.

);



Question 2

CREATE TABLE Students(

    StudentID INT PRIMARY KEY IDENTITY(1,1) NOT NULL,

    Name NVARCHAR(100) NOT NULL,

    Email NVARCHAR(100) NOT NULL,

    EnrollmentDate DATE NOT NULL DEFAULT GETDATE() -- Enrollment date sets to today's date by default, can be changed.

);

CREATE TABLE Instructors(

    InstructorID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

    Name NVARCHAR(100) NOT NULL,

    Email NVARCHAR(100) NOT NULL,

    HireDate DATE NOT NULL DEFAULT GETDATE() -- Same deal as above

);

CREATE TABLE Courses (

    CourseID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

    Title NVARCHAR(100) NOT NULL,

    InstructorID INT NOT NULL,

    CONSTRAINT FK\_Courses\_Instructors FOREIGN KEY (InstructorID) REFERENCES Instructors(InstructorID)

);

CREATE TABLE Enrollments (

    EnrollmentID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

    StudentID INT NOT NULL,

    CourseID INT NOT NULL,

    EnrollmentDate DATE NOT NULL,

    CONSTRAINT FK\_Enrollments\_Students FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

    CONSTRAINT FK\_Enrollments\_Courses FOREIGN KEY (CourseID) REFERENCES Courses(CourseID),

    CONSTRAINT UQ\_Enrollments\_StudentCourse UNIQUE (StudentID, CourseID) -- One student per CourseID

);

CREATE TABLE Assignments (

    AssignmentID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

    CourseID INT NOT NULL,

    Title NVARCHAR(100) NOT NULL,

    DueDate DATE NOT NULL,

    MaxScore DECIMAL(3, 2) NOT NULL,

    CONSTRAINT FK\_Assignments\_Courses FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);

CREATE TABLE Submissions (

    SubmissionID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

    StudentID INT NOT NULL,

    AssignmentID INT NOT NULL,

    SubmissionDate DATE NOT NULL,

    Score DECIMAL(3, 2) NOT NULL,

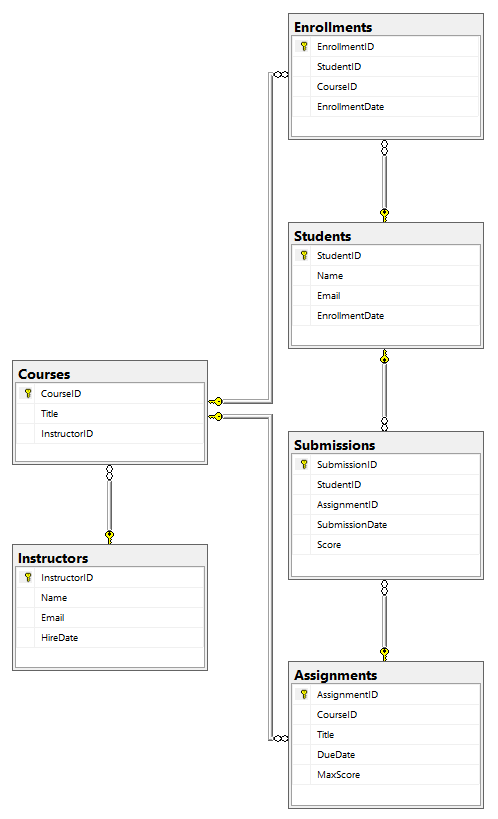
    CONSTRAINT FK\_Submissions\_Students FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

    CONSTRAINT FK\_Submissions\_Assignments FOREIGN KEY (AssignmentID) REFERENCES Assignments(AssignmentID),

    CONSTRAINT UQ\_Submissions\_StudentAssignment UNIQUE (StudentID, AssignmentID),

    CONSTRAINT CK\_Submissions\_Score\_Range CHECK (Score >= 0) -- Must contain a mark

);



Question 3

-- Query 1: Retrieve the books and authors borrowed by a member named 'John Doe'

SELECT b.title AS book, a.name AS author

FROM borrowing br

JOIN member m ON br.member\_id = m.member\_id

JOIN book b ON br.book\_id = b.book\_id

JOIN book\_author ba ON b.book\_id = ba.book\_id

JOIN author a ON ba.author\_id = a.author\_id

WHERE m.name = 'John Doe';

-- Query 2: Retrieve pairs of members who have borrowed at least one book in common

SELECT m1.name AS member1, m2.name AS member2

FROM member m1

JOIN member m2 ON m1.member\_id < m2.member\_id

WHERE NOT EXISTS (

  SELECT 1

  FROM borrowing b1

  JOIN book bo ON b1.book\_id = bo.book\_id

  WHERE b1.member\_id = m1.member\_id

  AND NOT EXISTS (

    SELECT 1

    FROM borrowing b2

    WHERE b2.member\_id = m2.member\_id

    AND b2.book\_id = bo.book\_id

  )

);

-- Query 3: Retrieve members who have not borrowed any books written by 'Mark Twain'

SELECT m.name

FROM member m

WHERE NOT EXISTS (

  SELECT 1

  FROM borrowing b

  JOIN book bo ON b.book\_id = bo.book\_id

  JOIN book\_author ba ON bo.book\_id = ba.book\_id

  JOIN author a ON ba.author\_id = a.author\_id

  WHERE b.member\_id = m.member\_id

  AND a.name = 'Mark Twain'

);

-- Query 4: Retrieve the books, the count of distinct members who borrowed them, and the average fine amount for each book, sorted by the count in descending order and book title in ascending order

SELECT bo.title AS book, COUNT(DISTINCT b.member\_id) AS cnt, AVG(r.fine) AS avg

FROM book bo

LEFT JOIN borrowing b ON bo.book\_id = b.book\_id

LEFT JOIN [return] r ON b.borrowing\_id = r.borrowing\_id

GROUP BY bo.title

ORDER BY cnt DESC, bo.title ASC;

-- Query 5: Retrieve the member name and the most borrowed book title for each member, sorted by member name

WITH borrowing\_count AS (

  SELECT

    m.member\_id,

    b.title,

    COUNT(\*) AS count,

    ROW\_NUMBER() OVER (PARTITION BY m.member\_id ORDER BY COUNT(\*) DESC) AS row\_num

  FROM member m

  LEFT JOIN borrowing bo ON m.member\_id = bo.member\_id

  LEFT JOIN book b ON bo.book\_id = b.book\_id

  GROUP BY m.member\_id, b.title

)

SELECT

  m.name AS member,

  bc.title AS book

FROM member m

LEFT JOIN borrowing\_count bc ON m.member\_id = bc.member\_id AND bc.row\_num = 1

ORDER BY m.name;

-- Query 6: Retrieve the author name and the average number of books borrowed per member for each author, sorted by the average in descending order and author name in ascending order

SELECT a.name, AVG(num\_borrowed\_per\_member) AS avg\_borrowed

FROM (

  SELECT ba.author\_id, COUNT(DISTINCT b.book\_id) AS num\_borrowed\_per\_member

  FROM borrowing b

  JOIN book\_author ba ON b.book\_id = ba.book\_id

  GROUP BY ba.author\_id, b.member\_id

) sub

JOIN author a ON sub.author\_id = a.author\_id

GROUP BY a.name

ORDER BY avg\_borrowed DESC, a.name ASC;

-- Query 7: Retrieve the names of members who have borrowed books from all available genres

WITH genre\_count AS (

  SELECT COUNT(DISTINCT genre) AS total\_genres

  FROM book

), member\_borrowed\_genres AS (

  SELECT m.name, COUNT(DISTINCT b.genre) AS borrowed\_genres

  FROM member m

  LEFT JOIN borrowing bo ON m.member\_id = bo.member\_id

  LEFT JOIN book b ON bo.book\_id = b.book\_id

  GROUP BY m.name

)

SELECT name

FROM member\_borrowed\_genres

WHERE borrowed\_genres = (SELECT total\_genres FROM genre\_count);

-- Query 8: Retrieve the author name and the average number of books borrowed per member for each author, sorted by the average in descending order and author name in ascending order

SELECT sub.author, AVG(sub.books\_borrowed\_per\_member) AS avg\_borrowed\_per\_member

FROM (

  SELECT a.name AS author, m.member\_id, COUNT(DISTINCT b.book\_id) AS books\_borrowed\_per\_member

  FROM borrowing b

  INNER JOIN book bk ON b.book\_id = bk.book\_id

  INNER JOIN book\_author ba ON bk.book\_id = ba.book\_id

  INNER JOIN author a ON ba.author\_id = a.author\_id

  INNER JOIN member m ON b.member\_id = m.member\_id

  GROUP BY a.name, m.member\_id

) AS sub

GROUP BY sub.author

ORDER BY avg\_borrowed\_per\_member DESC, sub.author ASC;

-- Query 9: Retrieve the names of members who have not been fined or have a fine amount of 0

SELECT m.name

FROM member m

LEFT JOIN borrowing b ON m.member\_id = b.member\_id

LEFT JOIN [return] r ON b.borrowing\_id = r.borrowing\_id

WHERE r.fine IS NULL OR r.fine = 0;

-- Query 10: Retrieve the genre, the count of distinct books borrowed, and the total fine amount for each genre, sorted by the count in descending order and genre in ascending order

SELECT bk.genre, COUNT(DISTINCT b.book\_id) AS total\_borrowed,

       COALESCE(SUM(r.fine), 0) AS total\_fine

FROM borrowing b

INNER JOIN book bk ON b.book\_id = bk.book\_id

LEFT JOIN "return" r ON b.borrowing\_id = r.borrowing\_id

GROUP BY bk.genre

ORDER BY total\_borrowed DESC, bk.genre ASC;

-- Query 11: Duplicate of Query 1. Refer to Query 1 for the answer.

Question 4

-- Query 1: Retrieve movies and actors for a specific customer

SELECT m.title AS movie, a.name AS actor

FROM rental r

JOIN customer c ON r.customer\_id = c.customer\_id

JOIN movie m ON r.movie\_id = m.movie\_id

JOIN movie\_actor ma ON m.movie\_id = ma.movie\_id

JOIN actor a ON ma.actor\_id = a.actor\_id

WHERE c.name = 'Alice Johnson';

-- Query 2: Retrieve pairs of customers who have rented the same movies

WITH CustomerMovies AS (

    SELECT customer\_id, movie\_id

    FROM rental

), CustomerPairs AS (

    SELECT DISTINCT c1.customer\_id AS customer1, c2.customer\_id AS customer2

    FROM CustomerMovies c1

    CROSS JOIN CustomerMovies c2

    WHERE c1.customer\_id <> c2.customer\_id

)

SELECT c1.name AS customer1, c2.name AS customer2

FROM CustomerPairs cp

JOIN customer c1 ON cp.customer1 = c1.customer\_id

JOIN customer c2 ON cp.customer2 = c2.customer\_id

WHERE NOT EXISTS (

    SELECT 1

    FROM CustomerMovies cm

    WHERE cm.customer\_id = cp.customer1

    AND NOT EXISTS (

        SELECT 1

        FROM CustomerMovies cm2

        WHERE cm2.customer\_id = cp.customer2

        AND cm2.movie\_id = cm.movie\_id

    )

);

-- Query 3: Retrieve customers who have rented movies with a specific actor

SELECT DISTINCT c.name

FROM customer c

JOIN rental r ON c.customer\_id = r.customer\_id

JOIN movie\_actor ma ON r.movie\_id = ma.movie\_id

JOIN actor a ON ma.actor\_id = a.actor\_id

WHERE a.name = 'Robert Downey Jr.'

AND NOT EXISTS (

    SELECT 1

    FROM rental r2

    JOIN movie\_actor ma2 ON r2.movie\_id = ma2.movie\_id

    JOIN actor a2 ON ma2.actor\_id = a2.actor\_id

    WHERE r2.customer\_id = c.customer\_id

    AND a2.name <> 'Robert Downey Jr.'

);

-- Query 4: Retrieve movies with the count of customers who rented them and the average fine

SELECT m.title AS movie, COUNT(DISTINCT r.customer\_id) AS cnt, AVG(rt.fine) AS avg

FROM movie m

LEFT JOIN rental r ON m.movie\_id = r.movie\_id

LEFT JOIN [return] rt ON r.rental\_id = rt.rental\_id

GROUP BY m.title

ORDER BY cnt DESC, m.title ASC;

-- Query 5: Retrieve customers who rented the most movies, along with their most rented movie

WITH CustomerMovieRentalCount AS (

    SELECT c.name AS customer, m.title AS movie, COUNT(\*) AS rental\_count

    FROM rental r

    JOIN customer c ON r.customer\_id = c.customer\_id

    JOIN movie m ON r.movie\_id = m.movie\_id

    GROUP BY c.name, m.title

), MaxRentals AS (

    SELECT customer, MAX(rental\_count) AS max\_rental\_count

    FROM CustomerMovieRentalCount

    GROUP BY customer

)

SELECT cmrc.customer, cmrc.movie

FROM CustomerMovieRentalCount cmrc

JOIN MaxRentals mr ON cmrc.customer = mr.customer AND cmrc.rental\_count = mr.max\_rental\_count

UNION

SELECT c.name AS customer, NULL AS movie

FROM customer c

LEFT JOIN rental r ON c.customer\_id = r.customer\_id

WHERE r.customer\_id IS NULL;

-- Query 6: Retrieve customers who have rented movies from all genres they have rented before

SELECT c.name

FROM customer c

WHERE NOT EXISTS (

    SELECT DISTINCT genre

    FROM movie

    WHERE genre NOT IN (

        SELECT DISTINCT m.genre

        FROM rental r

        JOIN movie m ON r.movie\_id = m.movie\_id

        WHERE r.customer\_id = c.customer\_id

    )

)

ORDER BY c.name;

-- Query 7: Retrieve actors with the average number of movies rented per customer

SELECT a.name AS actor,

       AVG(CAST(movie\_count AS FLOAT)) AS avg\_rented\_per\_customer

FROM actor a

JOIN movie\_actor ma ON a.actor\_id = ma.actor\_id

JOIN movie m ON ma.movie\_id = m.movie\_id

JOIN (

    SELECT r.movie\_id, COUNT(\*) AS movie\_count

    FROM rental r

    GROUP BY r.movie\_id

) rc ON m.movie\_id = rc.movie\_id

GROUP BY a.actor\_id, a.name

ORDER BY avg\_rented\_per\_customer DESC, a.name;

-- Query 8: Retrieve customers who have not returned any movies with fines

SELECT DISTINCT c.name

FROM customer c

WHERE NOT EXISTS (

    SELECT 1

    FROM rental r

    JOIN [return] ret ON r.rental\_id = ret.rental\_id

    WHERE r.customer\_id = c.customer\_id

    AND ret.return\_date > r.due\_date

    AND ret.fine > 0

)

AND EXISTS (

    SELECT 1

    FROM rental r

    WHERE r.customer\_id = c.customer\_id

)

ORDER BY c.name;

-- Query 9: Retrieve genres with the total number of rentals and total fine amount

SELECT m.genre,

       COUNT(\*) AS total\_rented,

       COALESCE(SUM(ret.fine), 0) AS total\_fine

FROM movie m

JOIN rental r ON m.movie\_id = r.movie\_id

LEFT JOIN [return] ret ON r.rental\_id = ret.rental\_id

GROUP BY m.genre

ORDER BY total\_rented DESC, m.genre;

-- Query 10: Retrieve pairs of actors who have acted in the same movies

WITH actor\_movies AS (

    SELECT DISTINCT a.actor\_id, a.name, m.movie\_id

    FROM actor a

    JOIN movie\_actor ma ON a.actor\_id = ma.actor\_id

    JOIN movie m ON ma.movie\_id = m.movie\_id

)

SELECT DISTINCT am1.name AS actor1, am2.name AS actor2

FROM actor\_movies am1

JOIN actor\_movies am2 ON am1.actor\_id < am2.actor\_id

WHERE NOT EXISTS (

    SELECT 1

    FROM actor\_movies am3

    WHERE am3.actor\_id = am1.actor\_id

    AND NOT EXISTS (

        SELECT 1

        FROM actor\_movies am4

        WHERE am4.actor\_id = am2.actor\_id

        AND am3.movie\_id = am4.movie\_id

    )

)

ORDER BY am1.name, am2.name;

Question 5

-- Query 1: Get courses and students taught by Dr. Smith

SELECT c.title AS course, s.name AS student

FROM course c

JOIN professor p ON c.professor\_id = p.professor\_id

JOIN enrollment e ON c.course\_id = e.course\_id

JOIN student s ON e.student\_id = s.student\_id

WHERE p.name = 'Dr. Smith'

ORDER BY c.title, s.name;

-- Query 2: Get pairs of students who share at least one course

SELECT s1.name AS student1, s2.name AS student2

FROM student s1

JOIN student s2 ON s1.student\_id < s2.student\_id

WHERE EXISTS (

    SELECT 1

    FROM enrollment e1

    JOIN enrollment e2 ON e1.course\_id = e2.course\_id

    WHERE e1.student\_id = s1.student\_id

    AND e2.student\_id = s2.student\_id

)

ORDER BY s1.name, s2.name;

-- Query 3: Get students taught only by Dr. Johnson

SELECT s.name

FROM student s

WHERE EXISTS (

    SELECT 1

    FROM enrollment e

    JOIN course c ON e.course\_id = c.course\_id

    JOIN professor p ON c.professor\_id = p.professor\_id

    WHERE e.student\_id = s.student\_id

    AND p.name = 'Dr. Johnson'

)

AND NOT EXISTS (

    SELECT 1

    FROM enrollment e

    JOIN course c ON e.course\_id = c.course\_id

    JOIN professor p ON c.professor\_id = p.professor\_id

    WHERE e.student\_id = s.student\_id

    AND p.name != 'Dr. Johnson'

)

ORDER BY s.name;

-- Query 4: Get course enrollment count and average grade

SELECT c.title AS course,

       COUNT(DISTINCT e.student\_id) AS enrollment\_count,

       AVG(CAST(g.grade AS FLOAT)) AS average\_grade

FROM course c

LEFT JOIN enrollment e ON c.course\_id = e.course\_id

LEFT JOIN grade g ON e.enrollment\_id = g.enrollment\_id

GROUP BY c.course\_id, c.title

ORDER BY enrollment\_count DESC, c.title ASC;

-- Query 5: Get top grade for each student

WITH GradesCalculation AS (

    SELECT s.name AS student, c.title AS course, g.grade,

           ROW\_NUMBER() OVER (PARTITION BY s.student\_id ORDER BY g.grade DESC) AS rn

    FROM student s

    LEFT JOIN enrollment e ON s.student\_id = e.student\_id

    LEFT JOIN course c ON e.course\_id = c.course\_id

    LEFT JOIN grade g ON e.enrollment\_id = g.enrollment\_id

)

SELECT student, course

FROM GradesCalculation

WHERE rn = 1

ORDER BY student;

-- Query 6: Get students enrolled in all departments

SELECT s.name

FROM student s

WHERE NOT EXISTS (

    SELECT DISTINCT c.department

    FROM course c

    EXCEPT

    SELECT DISTINCT c2.department

    FROM enrollment e

    JOIN course c2 ON e.course\_id = c2.course\_id

    WHERE e.student\_id = s.student\_id

)

ORDER BY s.name;

-- Query 7: Get average number of students per course for each professor

SELECT p.name AS professor, AVG(student\_count) AS avg\_students\_per\_course

FROM professor p

LEFT JOIN (

    SELECT c.professor\_id, c.course\_id, COUNT(e.student\_id) AS student\_count

    FROM course c

    LEFT JOIN enrollment e ON c.course\_id = e.course\_id

    GROUP BY c.professor\_id, c.course\_id

) AS course\_counts ON p.professor\_id = course\_counts.professor\_id

GROUP BY p.professor\_id, p.name

ORDER BY avg\_students\_per\_course DESC, p.name ASC;

-- Query 8: Get students with grades above or equal to 50

SELECT DISTINCT s.name

FROM student s

WHERE NOT EXISTS (

    SELECT 1

    FROM enrollment e

    JOIN grade g ON e.enrollment\_id = g.enrollment\_id

    WHERE e.student\_id = s.student\_id AND g.grade < 50

)

ORDER BY s.name;

-- Query 9: Get total students and courses count per department

SELECT c.department, COUNT(DISTINCT e.student\_id) AS total\_students, COUNT(DISTINCT c.course\_id) AS total\_courses

FROM course c

LEFT JOIN enrollment e ON c.course\_id = e.course\_id

GROUP BY c.department

ORDER BY total\_students DESC, c.department ASC;

-- Query 10: Get pairs of professors who teach the same courses

SELECT DISTINCT p1.name AS professor1, p2.name AS professor2

FROM professor p1

JOIN professor p2 ON p1.professor\_id < p2.professor\_id

WHERE NOT EXISTS (

    SELECT c1.course\_id

    FROM course c1

    WHERE c1.professor\_id = p1.professor\_id

    AND NOT EXISTS (

        SELECT c2.course\_id

        FROM course c2

        WHERE c2.professor\_id = p2.professor\_id

        AND c1.title = c2.title

    )

)

ORDER BY p1.name, p2.name;

Question 6

-- Query 1: Retrieve products and their categories for a specific customer

SELECT p.name AS product, c.name AS category

FROM customer cu

JOIN [order] o ON cu.customer\_id = o.customer\_id

JOIN order\_item oi ON o.order\_id = oi.order\_id

JOIN product p ON oi.product\_id = p.product\_id

JOIN category c ON p.category\_id = c.category\_id

WHERE cu.name = 'Alice Johnson'

ORDER BY p.name, c.name;

-- Query 2: Retrieve pairs of customers who have ordered the same products

WITH customer\_products AS (

    SELECT cu.customer\_id, cu.name, p.product\_id

    FROM customer cu

    JOIN [order] o ON cu.customer\_id = o.customer\_id

    JOIN order\_item oi ON o.order\_id = oi.order\_id

    JOIN product p ON oi.product\_id = p.product\_id

    GROUP BY cu.customer\_id, cu.name, p.product\_id

)

SELECT cp1.name AS customer1, cp2.name AS customer2

FROM customer\_products cp1

JOIN customer\_products cp2 ON cp1.customer\_id < cp2.customer\_id

WHERE NOT EXISTS (

    SELECT 1

    FROM customer\_products cp3

    WHERE cp3.customer\_id = cp1.customer\_id

    AND NOT EXISTS (

        SELECT 1

        FROM customer\_products cp4

        WHERE cp4.customer\_id = cp2.customer\_id

        AND cp3.product\_id = cp4.product\_id

    )

)

ORDER BY cp1.name, cp2.name;

-- Query 3: Retrieve customers who have only ordered products in the 'Electronics' category

SELECT cu.name

FROM customer cu

WHERE EXISTS (

    SELECT 1

    FROM [order] o

    JOIN order\_item oi ON o.order\_id = oi.order\_id

    JOIN product p ON oi.product\_id = p.product\_id

    JOIN category c ON p.category\_id = c.category\_id

    WHERE o.customer\_id = cu.customer\_id

    AND c.name = 'Electronics'

)

AND NOT EXISTS (

    SELECT 1

    FROM [order] o

    JOIN order\_item oi ON o.order\_id = oi.order\_id

    JOIN product p ON oi.product\_id = p.product\_id

    JOIN category c ON p.category\_id = c.category\_id

    WHERE o.customer\_id = cu.customer\_id

    AND c.name != 'Electronics'

)

ORDER BY cu.name;

-- Query 4: Retrieve products with the count of unique customers who ordered them and the average rating

SELECT p.name AS product,

       COUNT(DISTINCT o.customer\_id) AS cnt,

       AVG(CAST(r.rating AS FLOAT)) AS avg

FROM product p

LEFT JOIN order\_item oi ON p.product\_id = oi.product\_id

LEFT JOIN [order] o ON oi.order\_id = o.order\_id

LEFT JOIN review r ON p.product\_id = r.product\_id

GROUP BY p.product\_id, p.name

ORDER BY cnt DESC, p.name;

-- Query 5: Retrieve the product with the maximum quantity ordered for each customer

WITH customer\_max\_quantity AS (

    SELECT cu.customer\_id, cu.name AS customer, MAX(oi.quantity) AS max\_quantity

    FROM customer cu

    LEFT JOIN [order] o ON cu.customer\_id = o.customer\_id

    LEFT JOIN order\_item oi ON o.order\_id = oi.order\_id

    GROUP BY cu.customer\_id, cu.name

),

customer\_product AS (

    SELECT cmq.customer\_id, cmq.customer, cmq.max\_quantity, p.name AS product

    FROM customer\_max\_quantity cmq

    LEFT JOIN [order] o ON cmq.customer\_id = o.customer\_id

    LEFT JOIN order\_item oi ON o.order\_id = oi.order\_id AND oi.quantity = cmq.max\_quantity

    LEFT JOIN product p ON oi.product\_id = p.product\_id

)

SELECT customer,

       MAX(product) AS product

FROM customer\_product

GROUP BY customer\_id, customer

ORDER BY customer;

-- Query 6: Retrieve customers who have ordered products from all categories

SELECT cu.name

FROM customer cu

WHERE NOT EXISTS (

    SELECT c.category\_id

    FROM category c

    WHERE NOT EXISTS (

        SELECT 1

        FROM [order] o

        JOIN order\_item oi ON o.order\_id = oi.order\_id

        JOIN product p ON oi.product\_id = p.product\_id

        WHERE o.customer\_id = cu.customer\_id

        AND p.category\_id = c.category\_id

    )

)

ORDER BY cu.name;

-- Query 7: Retrieve categories with the average price and average rating of products in each category

SELECT c.name AS category,

       AVG(p.price) AS avg\_price,

       AVG(CAST(r.rating AS FLOAT)) AS avg\_rating

FROM category c

JOIN product p ON c.category\_id = p.category\_id

LEFT JOIN review r ON p.product\_id = r.product\_id

GROUP BY c.category\_id, c.name

ORDER BY avg\_price DESC, c.name;

-- Query 8: Retrieve customers who have given reviews with a rating of 4 or higher

SELECT cu.name

FROM customer cu

WHERE EXISTS (

    SELECT 1

    FROM review r

    WHERE r.customer\_id = cu.customer\_id

)

AND NOT EXISTS (

    SELECT 1

    FROM review r

    WHERE r.customer\_id = cu.customer\_id

    AND r.rating < 4

)

ORDER BY cu.name;

-- Query 9: Retrieve categories with the total quantity of products ordered and the total number of orders

SELECT c.name AS category,

       SUM(oi.quantity) AS total\_products\_ordered,

       COUNT(DISTINCT o.order\_id) AS total\_orders

FROM category c

JOIN product p ON c.category\_id = p.category\_id

JOIN order\_item oi ON p.product\_id = oi.product\_id

JOIN [order] o ON oi.order\_id = o.order\_id

GROUP BY c.category\_id, c.name

ORDER BY total\_products\_ordered DESC, c.name;

-- Query 10: Retrieve pairs of products that have been ordered by the same customers

WITH product\_customers AS (

    SELECT p.product\_id, p.name, o.customer\_id

    FROM product p

    JOIN order\_item oi ON p.product\_id = oi.product\_id

    JOIN [order] o ON oi.order\_id = o.order\_id

    GROUP BY p.product\_id, p.name, o.customer\_id

)

SELECT pc1.name AS product1, pc2.name AS product2

FROM product\_customers pc1

JOIN product\_customers pc2 ON pc1.product\_id < pc2.product\_id

WHERE NOT EXISTS (

    SELECT 1

    FROM product\_customers pc3

    WHERE pc3.product\_id = pc1.product\_id

    AND NOT EXISTS (

        SELECT 1

        FROM product\_customers pc4

        WHERE pc4.product\_id = pc2.product\_id

        AND pc3.customer\_id = pc4.customer\_id

    )

)

ORDER BY pc1.name, pc2.name;

Question 7

-- Query 1: Retrieve books and authors for a customer named 'John Doe'

SELECT DISTINCT b.title AS book, a.name AS author

FROM purchase p

JOIN book b ON p.book\_id = b.book\_id

JOIN author a ON b.author\_id = a.author\_id

JOIN customer c ON p.customer\_id = c.customer\_id

WHERE c.name = 'John Doe';

-- Query 2: Retrieve pairs of customers who have purchased the same books

SELECT DISTINCT c1.name AS customer1, c2.name AS customer2

FROM customer c1

JOIN customer c2 ON c1.customer\_id < c2.customer\_id

WHERE NOT EXISTS (

    SELECT p1.book\_id

    FROM purchase p1

    WHERE p1.customer\_id = c1.customer\_id

    AND NOT EXISTS (

        SELECT 1

        FROM purchase p2

        WHERE p2.customer\_id = c2.customer\_id

        AND p2.book\_id = p1.book\_id

    )

);

-- Query 3: Retrieve customers who have purchased books only by 'Jane Austen'

SELECT c.name

FROM customer c

WHERE NOT EXISTS (

    SELECT 1

    FROM purchase p

    JOIN book b ON p.book\_id = b.book\_id

    JOIN author a ON b.author\_id = a.author\_id

    WHERE p.customer\_id = c.customer\_id

    AND a.name != 'Jane Austen'

)

AND EXISTS (

    SELECT 1

    FROM purchase p

    JOIN book b ON p.book\_id = b.book\_id

    JOIN author a ON b.author\_id = a.author\_id

    WHERE p.customer\_id = c.customer\_id

    AND a.name = 'Jane Austen'

);

-- Query 4: Retrieve books with the highest number of customers who purchased them, along with the average rating

SELECT b.title AS book,

       COUNT(DISTINCT p.customer\_id) AS cnt,

       AVG(CAST(r.rating AS FLOAT)) AS avg

FROM book b

LEFT JOIN purchase p ON b.book\_id = p.book\_id

LEFT JOIN review r ON b.book\_id = r.book\_id

GROUP BY b.book\_id, b.title

ORDER BY cnt DESC, b.title ASC;

-- Query 5: Retrieve customers who have given the highest rating to a book, along with the book title

WITH CustomerMaxRating AS (

    SELECT c.customer\_id, c.name, MAX(r.rating) AS max\_rating

    FROM customer c

    LEFT JOIN review r ON c.customer\_id = r.customer\_id

    GROUP BY c.customer\_id, c.name

)

SELECT cmr.name AS customer,

       COALESCE(b.title, 'NULL') AS book

FROM CustomerMaxRating cmr

LEFT JOIN review r ON cmr.customer\_id = r.customer\_id AND cmr.max\_rating = r.rating

LEFT JOIN book b ON r.book\_id = b.book\_id

ORDER BY cmr.name, b.title;

-- Query 6: Retrieve customers who have purchased all books in a genre

SELECT c.name

FROM customer c

WHERE NOT EXISTS (

    SELECT DISTINCT b.genre

    FROM book b

    WHERE NOT EXISTS (

        SELECT 1

        FROM purchase p

        WHERE p.customer\_id = c.customer\_id

        AND p.book\_id = b.book\_id

    )

);

-- Query 7: Retrieve authors along with the average price of their books, sorted by average price in descending order

SELECT a.name AS author, AVG(b.price) AS avg\_price

FROM author a

JOIN book b ON a.author\_id = b.author\_id

GROUP BY a.author\_id, a.name

ORDER BY avg\_price DESC, a.name ASC;

-- Query 8: Retrieve customers who have given a minimum rating of 3 or have not given any ratings

SELECT DISTINCT c.name

FROM customer c

LEFT JOIN review r ON c.customer\_id = r.customer\_id

GROUP BY c.customer\_id, c.name

HAVING MIN(r.rating) >= 3 OR MIN(r.rating) IS NULL;

-- Query 9: Retrieve genres along with the total number of books purchased and total number of purchases, sorted by total books purchased in descending order

SELECT b.genre,

       COUNT(DISTINCT p.book\_id) AS total\_books\_purchased,

       COUNT(p.purchase\_id) AS total\_purchases

FROM book b

LEFT JOIN purchase p ON b.book\_id = p.book\_id

GROUP BY b.genre

ORDER BY total\_books\_purchased DESC, b.genre ASC;

-- Query 10: Retrieve pairs of authors who have customers in common

SELECT DISTINCT a1.name AS author1, a2.name AS author2

FROM author a1

JOIN author a2 ON a1.author\_id < a2.author\_id

WHERE NOT EXISTS (

    SELECT DISTINCT p1.customer\_id

    FROM purchase p1

    JOIN book b1 ON p1.book\_id = b1.book\_id

    WHERE b1.author\_id = a1.author\_id

    AND NOT EXISTS (

        SELECT 1

        FROM purchase p2

        JOIN book b2 ON p2.book\_id = b2.book\_id

        WHERE b2.author\_id = a2.author\_id

        AND p2.customer\_id = p1.customer\_id

    )

);

Question 8

-- Query 1: Retrieve top 3 books with the highest number of purchases and their average rating

SELECT TOP 3

    b.title AS book,

    COUNT(p.purchase\_id) AS num\_purchases,

    AVG(CAST(r.rating AS FLOAT)) AS avg\_rating

FROM

    book b

    LEFT JOIN purchase p ON b.book\_id = p.book\_id

    LEFT JOIN review r ON b.book\_id = r.book\_id

GROUP BY

    b.book\_id, b.title

ORDER BY

    num\_purchases DESC,

    b.title ASC;

-- Query 2: Retrieve pairs of books that have been purchased by the same customer

SELECT

    b1.title AS book1,

    b2.title AS book2

FROM

    book b1

    CROSS JOIN book b2

WHERE

    b1.book\_id <> b2.book\_id

    AND NOT EXISTS (

        SELECT 1

        FROM purchase p1

        WHERE p1.book\_id = b1.book\_id

        AND NOT EXISTS (

            SELECT 1

            FROM purchase p2

            WHERE p2.book\_id = b2.book\_id

            AND p1.customer\_id = p2.customer\_id

        )

    );

-- Query 3: Retrieve names of customers who have given a rating of 5 and a rating of 1

SELECT DISTINCT c.name

FROM customer c

JOIN review r ON c.customer\_id = r.customer\_id

WHERE r.rating = 5

AND EXISTS (

    SELECT 1

    FROM review r2

    WHERE r2.customer\_id = c.customer\_id

    AND r2.rating = 1

);

-- Query 4: Retrieve authors with their total revenue and average rating of their books

SELECT

    a.name AS author,

    SUM(b.price \* p.num\_purchases) AS total\_revenue,

    AVG(CAST(r.rating AS FLOAT)) AS avg\_rating

FROM

    author a

    JOIN book b ON a.author\_id = b.author\_id

    LEFT JOIN (

        SELECT book\_id, COUNT(\*) AS num\_purchases

        FROM purchase

        GROUP BY book\_id

    ) p ON b.book\_id = p.book\_id

    LEFT JOIN review r ON b.book\_id = r.book\_id

GROUP BY

    a.author\_id, a.name

ORDER BY

    total\_revenue DESC,

    a.name ASC;

-- Query 5: Retrieve titles of books that have not been purchased by customers who have purchased a fiction book

SELECT DISTINCT b.title AS book

FROM book b

JOIN purchase p ON b.book\_id = p.book\_id

WHERE NOT EXISTS (

    SELECT 1

    FROM customer c

    WHERE EXISTS (

        SELECT 1

        FROM purchase p2

        JOIN book b2 ON p2.book\_id = b2.book\_id

        WHERE p2.customer\_id = c.customer\_id

        AND b2.category = 'Fiction'

    )

    AND NOT EXISTS (

        SELECT 1

        FROM purchase p3

        WHERE p3.customer\_id = c.customer\_id

        AND p3.book\_id = b.book\_id

    )

);

-- Query 6: Retrieve customers, their favorite book, and its rating

SELECT

    c.name AS customer,

    b.title AS book,

    r.rating

FROM

    customer c

    LEFT JOIN (

        SELECT

            customer\_id,

            book\_id,

            rating,

            ROW\_NUMBER() OVER (PARTITION BY customer\_id ORDER BY rating DESC) AS rn

        FROM review

    ) r ON c.customer\_id = r.customer\_id AND r.rn = 1

    LEFT JOIN book b ON r.book\_id = b.book\_id;

-- Query 7: Retrieve books with their price, number of purchases, where the price is higher than the average price of their category

WITH CategoryAvg AS (

    SELECT

        category,

        AVG(price) AS avg\_price

    FROM book

    GROUP BY category

)

SELECT

    b.title AS book,

    b.price,

    COUNT(p.purchase\_id) AS num\_purchases

FROM

    book b

    JOIN CategoryAvg ca ON b.category = ca.category

    LEFT JOIN purchase p ON b.book\_id = p.book\_id

WHERE

    b.price > ca.avg\_price

GROUP BY

    b.book\_id, b.title, b.price;

-- Query 8: Retrieve pairs of authors where the average price of the first author's books is lower than the average price of the second author's books

WITH AuthorAvgPrice AS (

    SELECT

        author\_id,

        AVG(price) AS avg\_price

    FROM book

    GROUP BY author\_id

)

SELECT

    a1.name AS author1,

    a2.name AS author2

FROM

    AuthorAvgPrice ap1

    JOIN author a1 ON ap1.author\_id = a1.author\_id

    CROSS JOIN AuthorAvgPrice ap2

    JOIN author a2 ON ap2.author\_id = a2.author\_id

WHERE

    ap1.author\_id <> ap2.author\_id

    AND ap1.avg\_price < ap2.avg\_price;

-- Query 9: Retrieve names of customers who have reviewed books from all categories

SELECT c.name

FROM customer c

WHERE NOT EXISTS (

    SELECT DISTINCT category

    FROM book

    EXCEPT

    SELECT DISTINCT b.category

    FROM review r

    JOIN book b ON r.book\_id = b.book\_id

    WHERE r.customer\_id = c.customer\_id

);

-- Query 10: Retrieve categories of books with their total number of reviews and average rating

SELECT

    b.category,

    COUNT(r.review\_id) AS total\_reviews,

    AVG(CAST(r.rating AS FLOAT)) AS avg\_rating

FROM

    book b

    LEFT JOIN review r ON b.book\_id = r.book\_id

GROUP BY

    b.category

ORDER BY

    total\_reviews DESC,

    b.category ASC;

Question 9

-- Query 1: Retrieve the title and price of all books

SELECT title, price

FROM book;

-- Query 2: Retrieve the distinct names of authors whose books have a price greater than 10

SELECT DISTINCT a.name

FROM author a

JOIN book b ON a.author\_id = b.author\_id

WHERE b.price > 10;

-- Query 3: Retrieve the distinct names of customers who made a purchase

SELECT DISTINCT c.name

FROM customer c

JOIN purchase p ON c.customer\_id = p.customer\_id;

-- Query 4: Retrieve the title of each book and the number of reviews it has received

SELECT b.title, COUNT(r.review\_id) AS num\_reviews

FROM book b

LEFT JOIN review r ON b.book\_id = r.book\_id

GROUP BY b.book\_id, b.title;

-- Query 5: Retrieve the title of the book with the highest rating and its maximum rating

SELECT TOP 1 b.title, MAX(r.rating) AS rating

FROM book b

JOIN review r ON b.book\_id = r.book\_id

GROUP BY b.book\_id, b.title

ORDER BY rating DESC;

-- Query 6: Retrieve the title and price of books with a price less than 20

SELECT title, price

FROM book

WHERE price < 20;

-- Query 7: Retrieve the distinct names of customers who have given a rating of 5 in their reviews

SELECT DISTINCT c.name

FROM customer c

JOIN review r ON c.customer\_id = r.customer\_id

WHERE r.rating = 5;

-- Query 8: Retrieve the titles of books written by the author 'Mark Twain'

SELECT b.title

FROM book b

JOIN author a ON b.author\_id = a.author\_id

WHERE a.name = 'Mark Twain';

-- Query 9: Retrieve the average price of all books

SELECT AVG(price) AS avg\_price

FROM book;

-- Query 10: Retrieve the title of books purchased by the customer 'Alice Johnson' along with the purchase date

SELECT b.title, p.purchase\_date

FROM purchase p

JOIN book b ON p.book\_id = b.book\_id

JOIN customer c ON p.customer\_id = c.customer\_id

WHERE c.name = 'Alice Johnson';