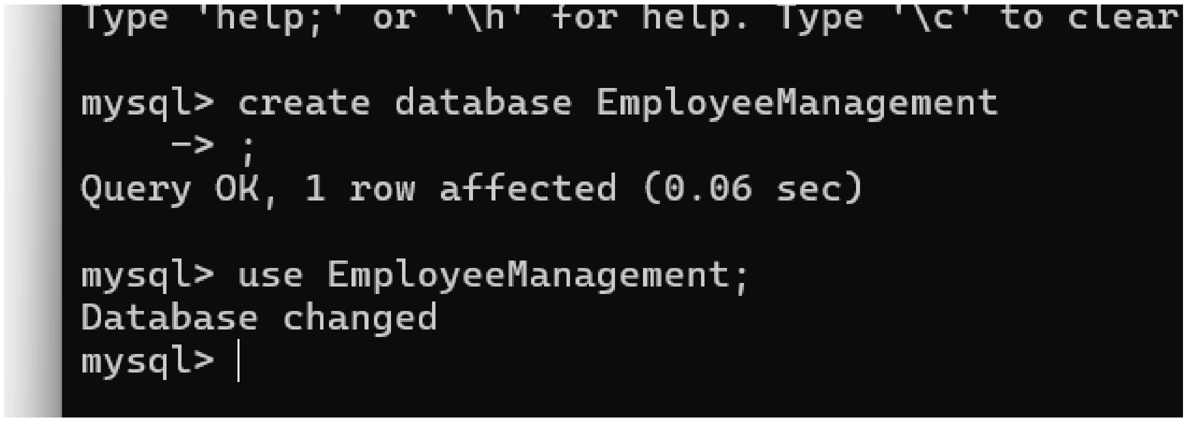
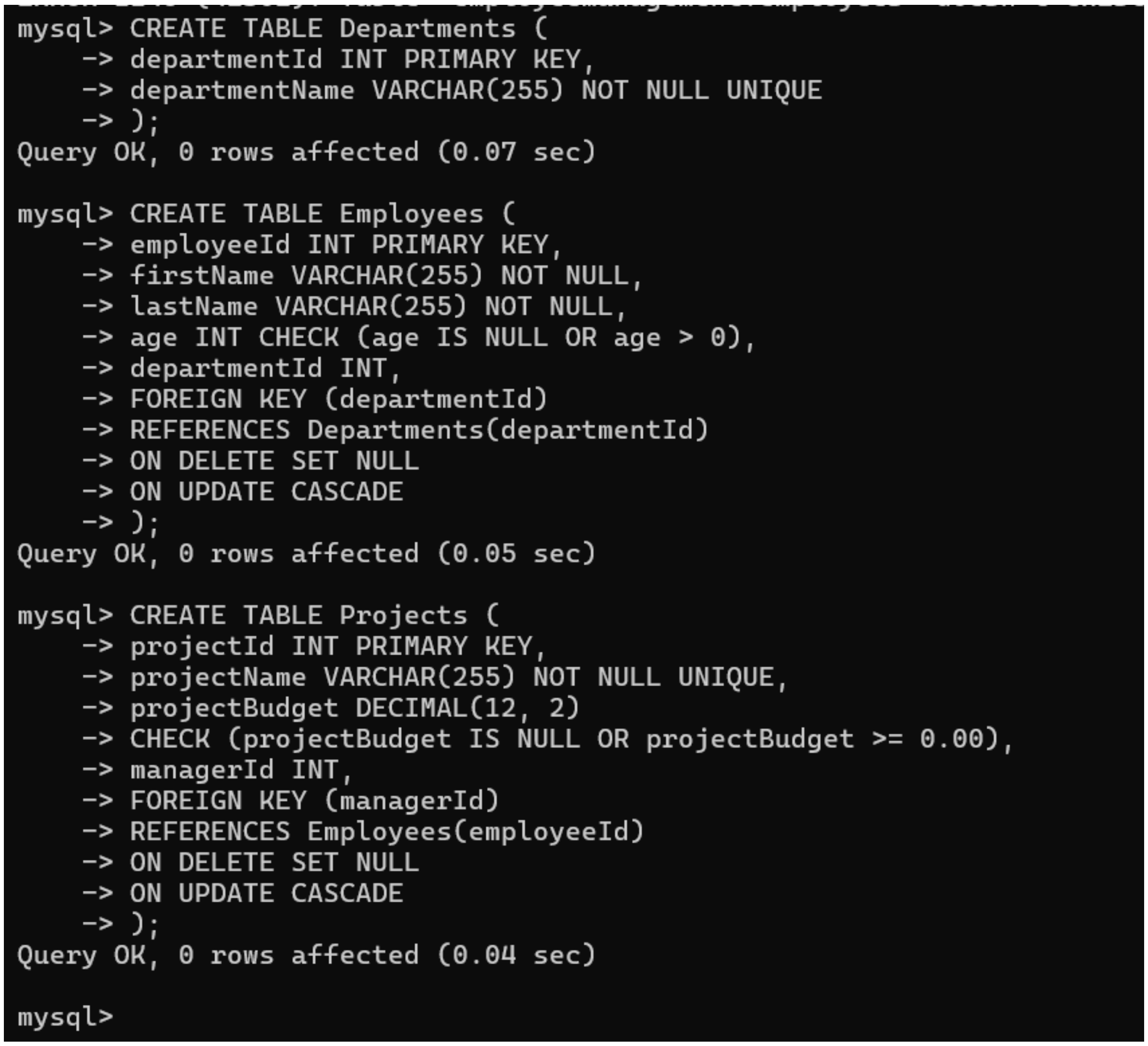
Step 1: Database Design



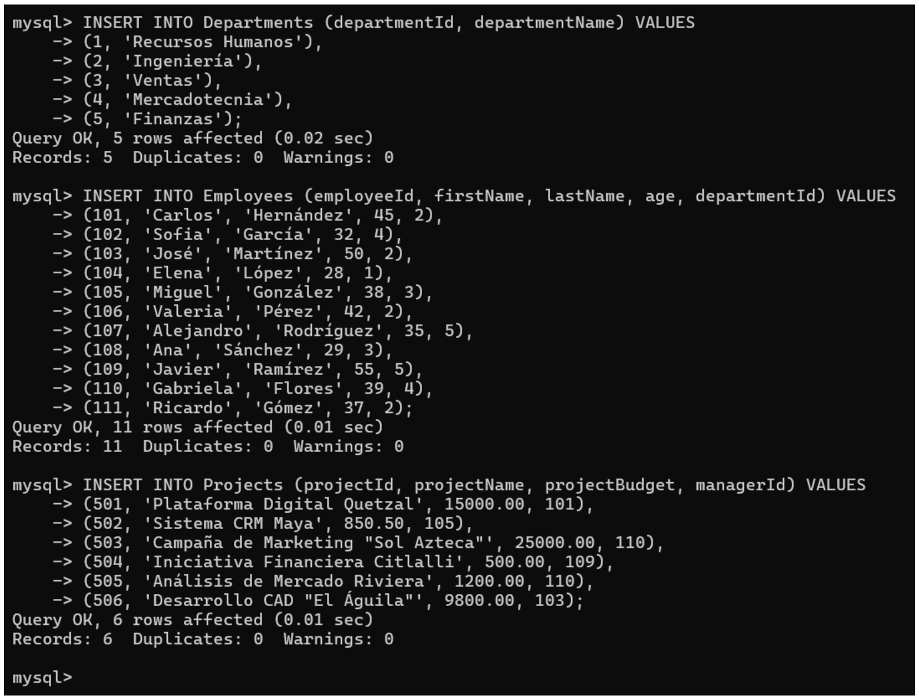
```
Type 'help;' or '\h' for help. Type '\c' to clear

mysql> create database EmployeeManagement
    -> ;
Query OK, 1 row affected (0.06 sec)

mysql> use EmployeeManagement;
Database changed
mysql>
```



```
mysql> CREATE TABLE Departments (
    -> departmentId INT PRIMARY KEY,
    -> departmentName VARCHAR(255) NOT NULL UNIQUE
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> CREATE TABLE Employees (
    -> employeeId INT PRIMARY KEY,
    -> firstName VARCHAR(255) NOT NULL,
    -> lastName VARCHAR(255) NOT NULL,
    -> age INT CHECK (age IS NULL OR age > 0),
    -> departmentId INT,
    -> FOREIGN KEY (departmentId)
    -> REFERENCES Departments(departmentId)
    -> ON DELETE SET NULL
    -> ON UPDATE CASCADE
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE Projects (
    -> projectId INT PRIMARY KEY,
    -> projectName VARCHAR(255) NOT NULL UNIQUE,
    -> projectBudget DECIMAL(12, 2)
    -> CHECK (projectBudget IS NULL OR projectBudget >= 0.00),
    -> managerId INT,
    -> FOREIGN KEY (managerId)
    -> REFERENCES Employees(employeeId)
    -> ON DELETE SET NULL
    -> ON UPDATE CASCADE
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
```

Step 2: Data Manipulation and Retrieval

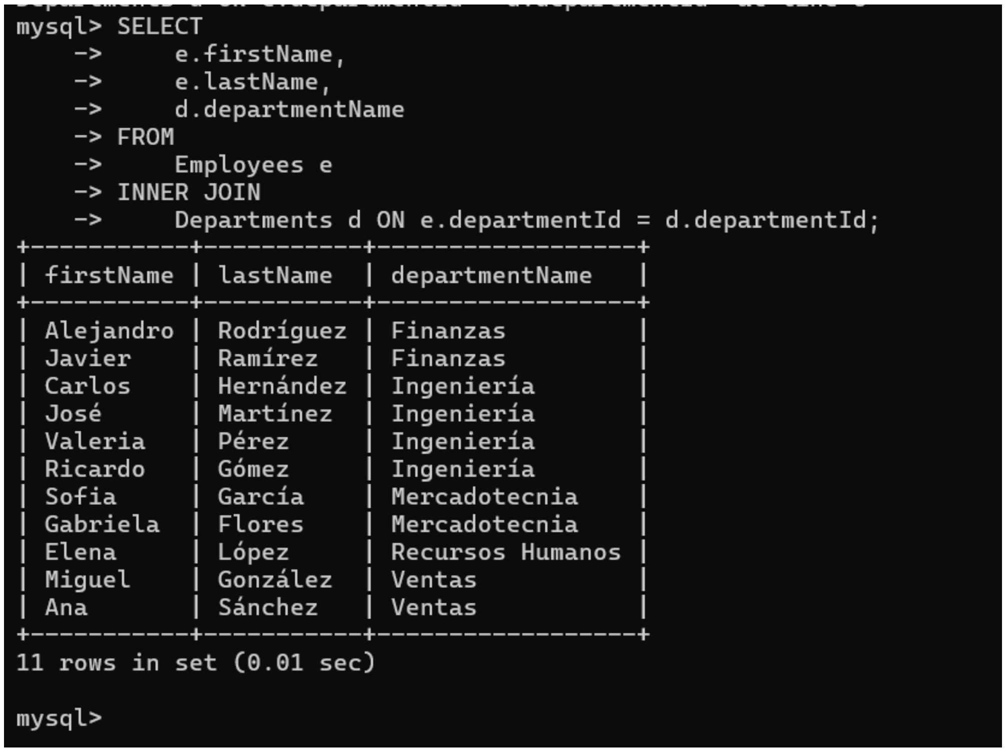
Insert Sample Data



```
mysql> INSERT INTO Departments (departmentId, departmentName) VALUES
    -> (1, 'Recursos Humanos'),
    -> (2, 'Ingeniería'),
    -> (3, 'Ventas'),
    -> (4, 'Mercadotecnia'),
    -> (5, 'Finanzas');
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> INSERT INTO Employees (employeeId, firstName, lastName, age, departmentId) VALUES
    -> (101, 'Carlos', 'Hernández', 45, 2),
    -> (102, 'Sofia', 'García', 32, 4),
    -> (103, 'José', 'Martínez', 50, 2),
    -> (104, 'Elena', 'López', 28, 1),
    -> (105, 'Miguel', 'González', 38, 3),
    -> (106, 'Valeria', 'Pérez', 42, 2),
    -> (107, 'Alejandro', 'Rodríguez', 35, 5),
    -> (108, 'Ana', 'Sánchez', 29, 3),
    -> (109, 'Javier', 'Ramírez', 55, 5),
    -> (110, 'Gabriela', 'Flores', 39, 4),
    -> (111, 'Ricardo', 'Gómez', 37, 2);
Query OK, 11 rows affected (0.01 sec)
Records: 11  Duplicates: 0  Warnings: 0

mysql> INSERT INTO Projects (projectId, projectName, projectBudget, managerId) VALUES
    -> (501, 'Plataforma Digital Quetzal', 15000.00, 101),
    -> (502, 'Sistema CRM Maya', 850.50, 105),
    -> (503, 'Campaña de Marketing "Sol Azteca"', 25000.00, 110),
    -> (504, 'Iniciativa Financiera Citlalli', 500.00, 109),
    -> (505, 'Análisis de Mercado Riviera', 1200.00, 110),
    -> (506, 'Desarrollo CAD "El Águila"', 9800.00, 103);
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql>
```

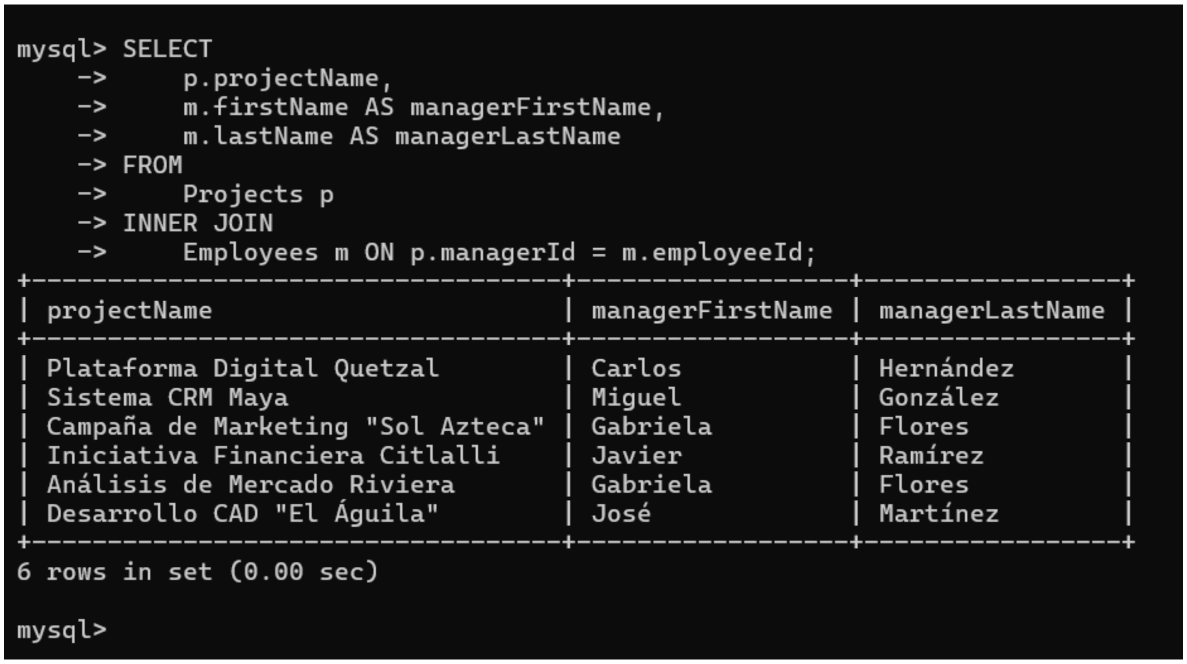
Write SQL Queries Using Joins

Display the list of all employees along with their department names.



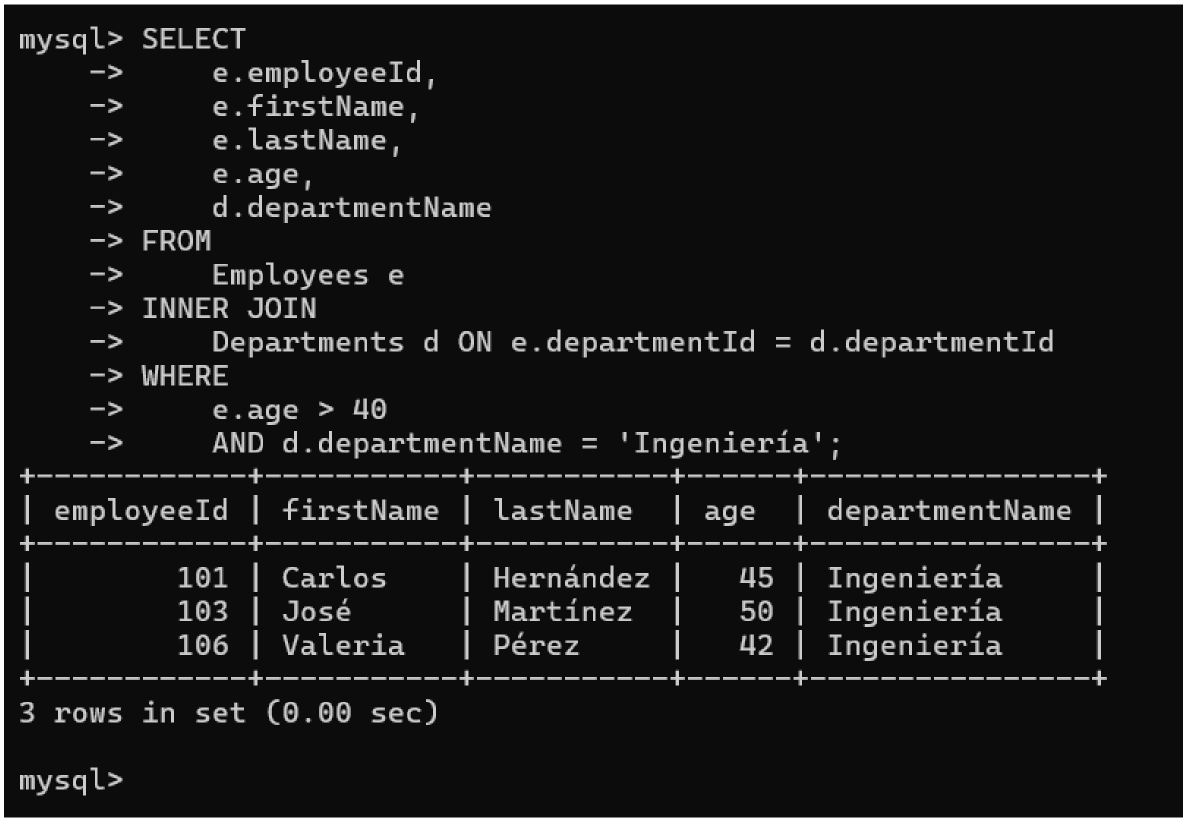
```
mysql> SELECT
    ->     e.firstName,
    ->     e.lastName,
    ->     d.departmentName
    -> FROM
    ->     Employees e
    -> INNER JOIN
    ->     Departments d ON e.departmentId = d.departmentId;
+-----------+-----------+------------------+
| firstName | lastName  | departmentName   |
+-----------+-----------+------------------+
| Alejandro | Rodríguez | Finanzas         |
| Javier    | Ramírez   | Finanzas         |
| Carlos    | Hernández | Ingeniería       |
| José      | Martínez  | Ingeniería       |
| Valeria   | Pérez     | Ingeniería       |
| Ricardo   | Gómez     | Ingeniería       |
| Sofia     | García    | Mercadotecnia    |
| Gabriela  | Flores    | Mercadotecnia    |
| Elena     | López     | Recursos Humanos |
| Miguel    | González  | Ventas           |
| Ana       | Sánchez   | Ventas           |
+-----------+-----------+------------------+
11 rows in set (0.01 sec)

mysql>
```

Show all projects along with the names of the managers assigned to them.



```
mysql> SELECT
    ->     p.projectName,
    ->     m.firstName AS managerFirstName,
    ->     m.lastName AS managerLastName
    -> FROM
    ->     Projects p
    -> INNER JOIN
    ->     Employees m ON p.managerId = m.employeeId;
+------------------------------------+------------------+-----------------+
| projectName                        | managerFirstName | managerLastName |
+------------------------------------+------------------+-----------------+
| Plataforma Digital Quetzal         | Carlos           | Hernández       |
| Sistema CRM Maya                   | Miguel           | González        |
| Campaña de Marketing "Sol Azteca"  | Gabriela         | Flores          |
| Iniciativa Financiera Citlalli     | Javier           | Ramírez         |
| Análisis de Mercado Riviera        | Gabriela         | Flores          |
| Desarrollo CAD "El Águila"         | José             | Martínez        |
+------------------------------------+------------------+-----------------+
6 rows in set (0.00 sec)

mysql>
```

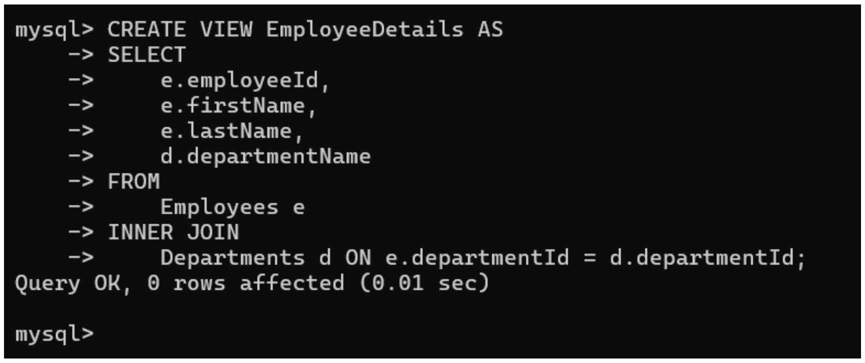
Retrieve a list of employees over the age of 40 who work in the Engineering department



```
mysql> SELECT
    ->     e.employeeId,
    ->     e.firstName,
    ->     e.lastName,
    ->     e.age,
    ->     d.departmentName
    -> FROM
    ->     Employees e
    -> INNER JOIN
    ->     Departments d ON e.departmentId = d.departmentId
    -> WHERE
    ->     e.age > 40
    ->     AND d.departmentName = 'Ingeniería';
+------------+-----------+-----------+------+----------------+
| employeeId | firstName | lastName  | age  | departmentName |
+------------+-----------+-----------+------+----------------+
|        101 | Carlos    | Hernández |   45 | Ingeniería     |
|        103 | José      | Martínez  |   50 | Ingeniería     |
|        106 | Valeria   | Pérez     |   42 | Ingeniería     |
+------------+-----------+-----------+------+----------------+
3 rows in set (0.00 sec)

mysql>
```

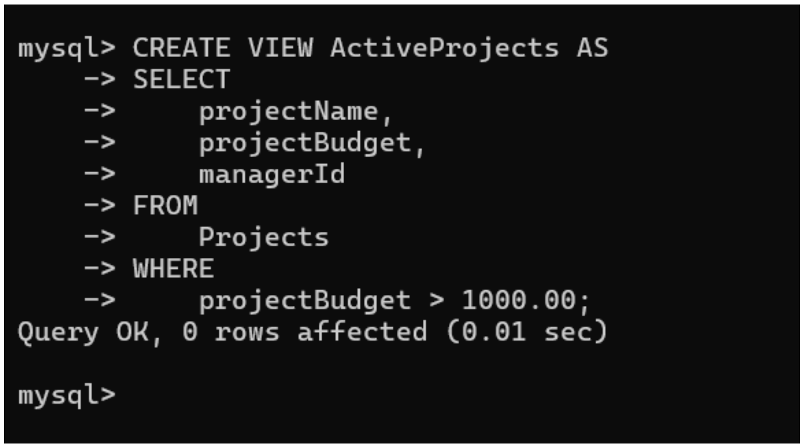
Create Views to Simplify Data Access.

Create a view named EmployeeDetails that shows employeeId, firstName, lastName, and departmentName.



```
mysql> CREATE VIEW EmployeeDetails AS
    -> SELECT
    ->     e.employeeId,
    ->     e.firstName,
    ->     e.lastName,
    ->     d.departmentName
    -> FROM
    ->     Employees e
    -> INNER JOIN
    ->     Departments d ON e.departmentId = d.departmentId;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

Create a view named ActiveProjects that shows projectName, projectBudget, and managerId for all projects with a budget over $1,000.



```
mysql> CREATE VIEW ActiveProjects AS
    -> SELECT
    ->     projectName,
    ->     projectBudget,
    ->     managerId
    -> FROM
    ->     Projects
    -> WHERE
    ->     projectBudget > 1000.00;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

Approach to Database Design and Choosing Constraints:

The project involved designing a relational database to manage information about employees, departments, and projects. A normalized approach was adopted, resulting in three primary tables: Departments, Employees, and Projects. This structure minimizes data redundancy (e.g., department names are stored only once) and enhances data integrity.

Key constraints were meticulously chosen and implemented to maintain data accuracy and enforce business rules:

Primary Keys (PK): departmentId, employeeId, and projectId were defined as PKs to ensure each record in their respective tables is uniquely identifiable.

Foreign Keys (FK): Relationships between tables were established using FKs: Employees.departmentId references Departments(departmentId). Projects.managerId references Employees(employeeId). These enforce referential integrity. We utilized ON

DELETE SET NULL to prevent accidental data deletion cascades (e.g., deleting a department sets the employees' department to NULL rather than deleting the employees) and ON UPDATE CASCADE to ensure consistency if a referenced key were ever updated.

- NOT NULL**:** Applied to essential fields like firstName, lastName, departmentName, and projectName to guarantee completeness of core data.
- UNIQUE**:** Enforced on departmentName and projectName to prevent duplicate entries, ensuring distinct identifiers beyond the primary key.
- CHECK**:** Implemented to validate data values upon insertion or update, specifically ensuring Employees.age is positive if specified (CHECK (age IS NULL OR age > 0)) and Projects.projectBudget is non-negative (CHECK (projectBudget IS NULL OR projectBudget >= 0.00)).

How Joins Were Used to Retrieve Data Across Tables:

Joins are essential for combining related information stored across different tables. In this project, INNER JOIN was primarily used:

Employee-Department Linking: To display a list of all employees along with their respective department names, an INNER JOIN was performed between the Employees table (e) and the Departments table (d) using the condition e.departmentId = d.departmentId. This effectively combined rows where a matching department ID existed in both tables.

Project-Manager Linking: To show project names along with the first and last names of their assigned managers, an INNER JOIN connected the Projects table (p) with the Employees table (m) based on p.managerId = m.employeeId.

Filtered Retrieval: Joins were also crucial for queries involving conditions across tables. For instance, retrieving employees over 40 in the 'Ingeniería' department required joining Employees and Departments first, then applying the WHERE clause (e.age > 40 AND d.departmentName = 'Ingeniería') to the combined result set.

How Views Were Used to Simplify Data Access and Why:

Views are stored SQL queries that act as virtual tables, created to abstract complexity and streamline data access. Two views were implemented:

EmployeeDetails **View:** This view encapsulates the INNER JOIN between Employees and Departments.

Why: It provides a simplified way for users or applications to retrieve employee information including their department name without needing to write or understand the underlying join logic repeatedly. This enhances code readability, reduces potential errors, and ensures consistency.

ActiveProjects **View:** This view selects projects specifically where the projectBudget is greater than $1,000.

Why: It offers a convenient shortcut to access a frequently needed subset of project data (high-budget projects). Users can query this view directly, simplifying their queries and ensuring the filtering logic (WHERE projectBudget > 1000) is applied consistently.

Views, in general, help simplify complex queries, improve data security (by potentially restricting access to certain columns/rows, though not heavily utilized here), and make the database easier to interact with.