VIEW:SIMPLE VIEWS/COMPLEX VIEWS

**Simple views:**

**Problem statement**: The HR department needs access to employee contact information, such as name, email, and phone number, but should not see sensitive information like salary or personal identification numbers.

**Query:** Create a simple view that includes only the fields HR staff needs, hiding sensitive columns.

Solution

|  |
| --- |
| CREATE VIEW HR\_EmployeeContact AS  SELECT  employee\_id,  name,  email,  phone  FROM  Employees;  SELECT \* FROM HR\_EmployeeContact; |

**scenario 2: Creating a Custom View for a Specific Department**

**Problem statement**: The Sales department only needs access to information about customers, specifically their IDs, names, and cities. The rest of the data (like customer credit scores or billing addresses) is irrelevant for their daily tasks.

Problem statement: Create a view with only the necessary columns, focusing on the data that the Sales team needs.

|  |
| --- |
| CREATE VIEW Sales\_CustomerInfo AS  SELECT  Customer\_id, name,city  FROM  Customers;  SELECT \* FROM Sales\_CustomerInfo; |

**Scenario 3: Standardized View of Product Information**

**Problem statement**:The Product Management team needs a consistent view of product details to analyze offerings. They only need the product name, description, and category, but not inventory or supplier details.

Query :Create a simple view that presents product details in a standardized format.

|  |
| --- |
| CREATE VIEW Product\_Details AS  SELECT  product\_id,  product\_name,  description,  category  FROM  Products;  SELECT \* FROM Product\_Details; |

**Scenario 4: Presenting Simplified Customer Orders for Customer Support**

**Problem statement** :Customer Support agents need to see the latest orders placed by customers to assist with inquiries. However, they should only have access to customer names, order dates, and status—not financial details like order amounts or payment info.

Query :Create a view that shows only the essential fields for customer support tasks.

|  |
| --- |
| CREATE VIEW CustomerSupport\_Orders AS  SELECT  customer\_id,  name,  order\_date,  status  FROM  Orders;  SELECT \* FROM CustomerSupport\_Orders; |

**Scenario 5: Creating a View for Recent Activity**

**Problem statement**: A team wants a view of recent user activity from a `UserActivity` table, showing only the user ID and last login date for users who logged in within the last 30 days.

Query : Create a view that filters data to show only recent activity.

|  |
| --- |
| CREATE VIEW RecentUserLogins AS  SELECT  user\_id,  last\_login  FROM  UserActivity  WHERE  last\_login >= CURRENT\_DATE - INTERVAL '30 days';  SELECT \* FROM RecentUserLogins; |

**Scenario 6: View for Product Pricing Information**

**Problem statement** :A marketing analyst needs access to product pricing information, specifically the product name and price, but should not see inventory levels or internal cost details.

|  |
| --- |
| Query : Create a view that provides a simple view of product names and prices.  CREATE VIEW Marketing\_ProductPricing AS  SELECT  product\_name,  price  FROM  Products;  SELECT \* FROM Marketing\_ProductPricing; |

**Scenario 7: Simplified Customer Profile View**

**Problem statement** :The Customer Relations team needs quick access to customer profiles, focusing only on fields like `customer\_id`, `name`, and `membership\_level` to manage loyalty programs.

Query Approach: Create a view to show only essential profile fields.

Solution:

|  |
| --- |
| CREATE VIEW CustomerProfiles AS  SELECT  customer\_id,  name,  membership\_level  FROM  Customers;  SELECT \* FROM CustomerProfiles; |

**Scenario 8: Summarized Employee Directory**

Scenario: A general employee directory is needed to display employee names and departments to all users without showing sensitive personal details.

Query : Create a view with only the name and department fields from the employee data.

Solution:

|  |
| --- |
| CREATE VIEW EmployeeDirectory AS  SELECT  name,  department  FROM  Employees;  SELECT \* FROM EmployeeDirectory; |

**Complex views:**

**Scenario 1: Frequently Used Boats**

Scenario: The marina management wants to see which boats are most frequently used, including the total number of reservations per boat**, the Frequently Used Boats contains the tables: Sailors**, **Boats**, **Reservations**:

**Query: Create a view that joins Boats and Reservations, counting reservations grouped by boat\_id.**

**solution**

|  |
| --- |
| CREATE VIEW FrequentlyUsedBoats  AS SELECT b.boat\_id, b.boat\_name, b.color,  COUNT(r.reservation\_id) AS total\_reservations  FROM Boats b LEFT JOIN Reservations r ON b.boat\_id = r.boat\_id  GROUP BY b.boat\_id, b.boat\_name, b.color  HAVING COUNT(r.reservation\_id) > 10;  SELECT \* FROM FrequentlyUsedBoats; |

**Scenario: E-commerce Database**

You are a database administrator for an e-commerce platform. The database consists of the following tables

**1. Products**

**2.Categories**

**3. Orders**

**4. Order Details**

**5. Customers**

**Question 1:**

**Create a view `ProductSalesSummary` to show the total sales revenue for each product, including product name, total quantity sold, and total revenue.**

solution

**CREATE VIEW ProductSalesSummary AS**

**SELECT**

**P.ProductID,**

**P.ProductName,**

**SUM(OD.Quantity) AS TotalQuantitySold,**

**SUM(OD.Quantity \* P.Price) AS TotalRevenue**

**FROM**

**Products P**

**JOIN**

**OrderDetails OD ON P.ProductID = OD.ProductID**

**GROUP BY**

**P.ProductID, P.ProductName;**

**Question 2:**

**Write a query to fetch the top 3 products with the highest sales revenue from the `ProductSalesSummary` view.**

**Solution:**

**SELECT**

**ProductName,**

**TotalRevenue**

**FROM**

**ProductSalesSummary**

**ORDER BY**

**TotalRevenue DESC**

**LIMIT 3;**

**Question 3:**

**You notice that the `Price` of a specific product (`ProductID = 101`) is incorrect. Update the price in the `Products` table via a view.**

**Solution:**

**CREATE VIEW ProductDetails AS**

**SELECT**

**ProductID,**

**ProductName,**

**Price**

**FROM**

**Products;**

**-- Updating via view**

**UPDATE ProductDetails**

**SET Price = 25.99**

**WHERE ProductID = 101;**

**Question 4:**

**Create a view `CustomerOrderSummary` to display each customer's name, their country, and the total amount spent across all orders.**

**Solution:**

**CREATE VIEW CustomerOrderSummary AS**

**SELECT**

**C.CustomerID,**

**C.CustomerName,**

**C.Country,**

**SUM(OD.Quantity \* P.Price) AS TotalSpent**

**FROM**

**Customers C**

**JOIN**

**Orders O ON C.CustomerID = O.CustomerID**

**JOIN**

**OrderDetails OD ON O.OrderID = OD.OrderID**

**JOIN**

**Products P ON OD.ProductID = P.ProductID**

**GROUP BY**

**C.CustomerID, C.CustomerName, C.Country;**

**Question 5:**

**Create a view `LowStockProducts` to display products with stock below 50, including product name, category, and stock level.**

**Solution:**

**CREATE VIEW LowStockProducts AS**

**SELECT**

**P.ProductName,**

**C.CategoryName,**

**P.Stock**

**FROM**

**Products P**

**JOIN**

**Categories C ON P.CategoryID = C.CategoryID**

**WHERE**

**P.Stock < 50;**

**Question 6:**

**Dealing with Non-Updatable Views**

**Why might the view `CustomerOrderSummary` not be updatable? Provide a solution to handle updates.?**

**Solution:**

**- The `CustomerOrderSummary` view is not updatable because it involves multiple tables, aggregate functions (`SUM`), and a `GROUP BY` clause.**

**- These elements break the direct one-to-one relationship between the view and underlying tables.**

**Solution:**

**Instead of attempting to update such a view, update the base tables directly:**

**UPDATE Customers**

**SET Country = 'USA'**

**WHERE CustomerID = 1;**