SQL

1. LastName, Department, and Salary.

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Department VARCHAR(50),

Salary DECIMAL(10, 2) );

1. Write an SQL query to insert a new employee with the following details into the Employees table:
   * EmployeeID: 101
   * FirstName: 'John'
   * LastName: 'Doe'
   * Department: 'HR'
   * Salary: 50000.00

INSERT INTO Employees (EmployeeID, FirstName, LastName, Department, Salary)

VALUES (101, 'John', 'Doe', 'HR', 50000.00);

1. Write a SQL query to update the salary of the employee with EmployeeID 101 to 55000.00.

UPDATE Employees SET Salary = 55000.00 WHERE EmployeeID = 101;

1. Create a query to retrieve the FirstName and LastName of employees who work in the 'Sales' department.

SELECT FirstName, LastName FROM Employees WHERE Department = 'Sales';

1. Write an SQL query to retrieve the details of employees whose salary is greater than 60000.00.

SELECT \* FROM Employees WHERE Salary > 60000.00;

1. Create a query to retrieve the FirstName, LastName, and Department of employees who earn more than 70000.00 and work in the 'Marketing' department.

SELECT FirstName, LastName, Department

FROM Employees

WHERE Salary > 70000.00 AND Department = 'Marketing';

1. Write a SQL query to calculate the average salary of all employees.

SELECT AVG(Salary) AS AverageSalary FROM Employees;

1. Create a query to count the total number of employees.

SELECT COUNT(\*) AS TotalEmployees FROM Employees;

1. Write an SQL query to delete the employee with EmployeeID 101 from the Employees table.

DELETE FROM Employees WHERE EmployeeID = 101;

1. Create a query to retrieve the highest salary among all employees.

SELECT MAX(Salary) AS HighestSalary FROM Employees;

1. Create a Departments table:

CREATE TABLE Departments (DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(50));

1. Insert some department data:

INSERT INTO Departments (DepartmentID, DepartmentName)

VALUES (1, 'HR'),

(2, 'Sales'),

(3, 'Marketing');

1. Modify the Employees table to include a DepartmentID column:

ALTER TABLE Employees ADD COLUMN DepartmentID INT;

1. Update the existing employee records to include DepartmentID:

UPDATE Employees SET DepartmentID = 1 WHERE EmployeeID = 101;

1. Write a query to retrieve the FirstName, LastName, and DepartmentName of employees by joining the Employees and Departments tables:

SELECT e.FirstName, e.LastName, d.DepartmentName

FROM Employees e

JOIN Departments d

ON e.DepartmentID =d.DepartmentID;

1. Create a Projects table:

CREATE TABLE Projects(ProjectID INT PRIMARY KEY,

ProjectName VARCHAR(100),

DepartmentID INT,

FOREIGN KEY (DepartmentID)

REFERENCES Departments(DepartmentID));

1. Insert some project data:

INSERT INTO Projects (ProjectID, ProjectName, DepartmentID)

VALUES (1, 'Project A', 1), -- HR Department

(2, 'Project X', 2), -- Sales Department

(3, 'Project M', 3); -- Marketing Department

1. Modify the Employees table to include a ProjectID column:

ALTER TABLE Employees ADD COLUMN ProjectID INT;

1. Update the existing employee records to include ProjectID:

UPDATE Employees SET ProjectID = 1 WHERE EmployeeID = 101; -- John Doe working on Project A

1. Write a query to retrieve the FirstName, LastName, DepartmentName, and ProjectName of employees by joining all three tables:

SELECT e.FirstName, e.LastName, d.DepartmentName, p.ProjectName

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID

JOIN Projects p ON e.ProjectID = p.ProjectID;

Quiz 2:

* TABLE **City** with attributes: ID, Name,  CountryCode, DistrictName, Population
* TABLE **Country** with attributes: CountryCode, Name, Continent, Region, SurfaceArea, IndepYear, Population, LifeExpectancy, GNP, GNPOld, LocalName, GovernmentForm, HeadOfState, Capital
* TABLE **CountryLanguage** with attributes: CountryCode, Language, IsOfficial, Percentage

1. Identify Candidate Keys for given tables (City, Country, CountryLanguage)

* City Table: ID assuming each city has a unique city id.
* Country Table: CountryCode assuming each country has a unique country code.
* CountryLanguage Table: CountryCode does not work for CountryLanguage by itself. There are potentially other candidate keys for Country (like Name) and City (like Name, District, CountryCode)

2. For the three tables (City, Country, CountryLanguage), discuss whether each is best represented as an entity or a relationship.

* **City Table** makes the most sense as an **entity**. Each row corresponds to a distinct city that is unlikely to ever change. A city is a "thing."
* **Country Table** also makes sense as an **entity** for similar reasons to the City Table. Again, each row in the Country table represents a distinct country that has a concrete identity meaning it is unlikely to ever change. A country is a "thing."
* **CountryLanguage Table** makes a bit more sense to represent as a many-to-many **relationship** between countries and languages as they seem more like entities on their own (Country and Language) but in this table, they are being combined to show a correlation.

3. Describe the query

SELECT ci.Name, co.Name, ci.population

FROM City ci, Country co

WHERE ci.CountryCode = co.CountryCode;

* Retrieves data from two tables, City and Country, by joining them based on their CountryCode attributes. It selects the city name, country name, and city population. This query is used to obtain a summary of cities along with their respective countries and populations.

Quiz 3:

SELECT c.name, COUNT(cl.Language) AS n

FROM country c, countrylanguage cl

WHERE c.code = cl.CountryCode

AND cl.isOfficial = 'T'

AND EXISTS (

SELECT \*

FROM CountryLanguage cl

WHERE c.code = cl.CountryCode

AND Language = 'English')

GROUP BY c.name

HAVING n > 2;

1. Does it have a correlated or uncorrelated nested query and why?

* The condition c.code=cl.CountryCode in the subquery references the c.code column from the outer loop. This means the subquery needs the specific country code for each iteration to evaluate whether "English" is an official language for that particular country. This creates a dependency where the subquery's results are determined by the outer loop's row so it can be considered correlated.

***FlightID        Date                 Origin         Destination        NoOfPassengers***

*U111          1/1/2013          Chicago           Denver                      100*

*U111          1/2/2013          Chicago           Denver                      108*

*U222          1/1/2013          Denver            Seattle                       115*

*U333          1/1/2013          Chicago           Orlando                    160*

2a. Identify two anomalies (insert, modify, or delete) that can occur with the current setup of the table (for each of the two you identify, be sure to describe which kind of anomaly it is and why).

* Right now it seems that the Flight ID is correlated with the Origin and Destination but not necessarily the date. So it is possible for there to be entries with the same data and FlightID (e.g. multiple flights going from Chicago to Denver on the same day).
* If you UPDATE the FlightID U111 whose current data is 1/1/2013, to 1/2/2013 because it got delayed by a day for example it would lead to a duplicate primary key violation. This change would not be allowed and can be considered an anomaly because you should be able to dynamically change the dates of Flights without complication.
* If you INSERT a flight with the same ID (because it has the same destination and origin) and date that already exists in the table it would create an anomaly and would not be added to the table. For example in this case, if we wanted to add another Flight from Chicago to Denver on 1/1/2013 it would not be allowed because it "already exists" in the table even though this may just be its own unique flight.

2b. Describe (define) the tables that will result when normalizing table FLIGHT to 3NF.

We could then break up the dependency between FlightID and (Origin, Destination)

I believe if normalizing to 3NF there should be 2 tables to think about:

* Table with attributes: FlightID (Primary Key), Date, NoOfPassengers
  + This table will contain information about each flight, where each flight is uniquely identified by FlightID.
* Table with attributes: FlightID (Primary Key & Foreign Key), Origin, Destination
  + This table adds the ScheduleID attribute to uniquely identify when a flight should be leaving and can compare/match it with the FlightID.

These tables should solve the error that occurs with having two flights with the same origin and destination on the same day.