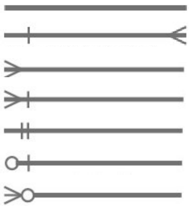
Q1) What SQL statement would you use to create SQL database named *Test*?

a) CREATE DBT Test;
b) CREATE Test;
c) **CREATE DATABASE Test;**
d) MAKE DATABASE Test;

Q2) What SQL statement would you use to remove SQL database named *Test*?

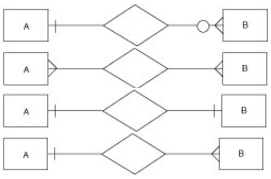
a) **DROP DATABASE Test;**
b) REMOVE Test;
c) DROP DBT Test;
d) REMOVE DATABASE Test;

Q3) Assign individual relationships that one can find in ERD (Entity Relationship Diagram) to their appropriate descriptions.



a) 1) zero or many (optional) = **g)**
b) 2) one or more (mandatory) = **d)**
c) 3) one to one = **a)**
d) 4) one and only one (mandatory) = **e)**
e) 5) one to many (mandatory) = **b)**
f) 6) many = **c)**
g) 7) zero or one (optional) = **f)**

Q4) Which kind of relationship best describes relationship between *Products* (A) and *OrderDetails* (B) tables?



a)
b)
c)
d)

Q5) Fulfill missing parts in SQL statement below to create *Products* table.

CREATE TABLE Products (
ProductID INT **PRIMARY KEY**,

ProductName VARCHAR(255),
SupplierID INT,
CategoryID INT,

Unit **VARCHAR(255)**,
Price **INT**
);

Q6) Primary Key is a combination of two types of constraints. Find the correct one in the list below. a) UNIQUE & DEFAULT

b) CHECK & UNIQUE
c) **UNIQUE & NOT NULL**
d) CHECK & AUTO INCREMENT

Q7) Which field in the *Orders* table has a role of FOREIGN KEY in relation to *Customers* table? **a)**

CustomerID

b) OrderID
c) OrderDate
d) CustomerName

Q8) What SQL statement would you use to put a new record into the *Orders* table?

a) INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES (200, 125, '2018-02-05');
b) PUT INTO Orders (OrderID, CustomerID, OrderDate) VALUES (200, 125, '2018-02-05');
c) INSERT VALUES (200, 125, '2018-02-05') INTO Orders (OrderID, CustomerID, OrderDate);
d) PUT VALUES (200, 125, '2018-02-05') INTO Orders (OrderID, CustomerID, OrderDate);

Q9) What SQL statement would you use to modify the existing record of *CustomerID* to 10 for *OrderID* #10308 in the *Orders* table?

a) CHANGE SET Orders CustomerID = 10 WHERE OrderID = 10308;
b) UPDATE SET Orders CustomerID = 10 WHERE OrderID = 10308;
c) CHANGE Orders SET CustomerID = 10 WHERE OrderID = 10308;
d) **UPDATE Orders SET CustomerID = 10 WHERE OrderID = 10308;**

Q10) What SQL statement would you use to delete order with *OrderID* #10308 from the *Orders* table?

a) DELETE FROM Orders WHERE OrderID = 10308;
b) CUT FROM Orders WHERE OrderID = 10308;
c) MOVE Orders WHERE OrderID = 10308;
d) REMOVE FROM Orders WHERE OrderID = 10308;

Q11) What SQL statement would you use to add an *Age* field to the *Customers* table?

a) UPDATE TABLE Customers ADD Age INT;
b) **ALTER TABLE Customers ADD Age INT;**
c) UPDATE TABLE ADD Age INT Customers;
d) ALTER TABLE ADD Age INT Customers;

Q12) What SQL statement would you use to remove *Customers* table from *Test* database?

a) REMOVE TABLE Customers;
b) REMOVE Customers;
c) DROP Customers;
d) **DROP TABLE Customers;**

Q13) Fulfill missing parts in SQL statement below to extract all records from *Customers* table.

SELECT *
FROM Customers;

Q14) Fulfill missing parts in SQL statement below to extract *CustomerName* and *Address* from *Customers* table.

SELECT CustomerName, Address
FROM Customers;

Q15) Fulfill missing parts in SQL statement below to extract all distinct countries from *Customers* table.

```
SELECT DISTINCT Country
 FROM Customers;
```

Q16) Fulfill missing parts in SQL statement below to extract all records from *Products* table that will include only products with price higher than 20 EUR.

```
SELECT *
 FROM Products

 WHERE Price > 20;
```

Q17) Fulfill missing parts in SQL statement below to extract all records from *Customers* table that will include only those customers who have NULL values in *Address* field.

```
SELECT *
 FROM Customers

 WHERE  Address IS NULL;
```

Q18) Fulfill missing parts in SQL statement below to extract all records from *Customers* table that will include only those customers who are from Germany or UK.

```
SELECT *
 FROM Customers

 WHERE Country = “Germany” OR Country = “UK”  ;
```

Q19) Fulfill missing parts in SQL statement below to extract all records from *Customers* table that will include only those customers who are not from USA.

```
SELECT *
 FROM Customers

 WHERE NOT Country = “USA”
```

Q20) Fulfill missing parts in SQL statement below to extract all records from *Products* table that will include only those products that are supplied by supplier with *SupplierID* #1 and that belong to *CategoryID* #2.

SELECT *
FROM Products

WHERE SupplierID = 1 AND CategoryID = 2;

Q21) Fulfill missing parts in SQL statement below to arrange records in *Products* table according to *Price* in descending order.

SELECT *
FROM Products

ORDER BY Price DESC;

Q22) Fulfill missing parts in SQL statement below to extract the first 50 records from *Customers* table.

SELECT TOP 50 *
FROM Customers;

Q23) Fulfill missing parts in SQL statement below to find maximum *Price* for products listed in *Products* table.

SELECT MAX(Price)
FROM Products;

Q24) What statement will you use to count number of records within *Customers* table?

a) SELECT ALL FROM Customers;
b) SELECT N FROM Customers;
c) **SELECT COUNT(*) FROM Customers;**
d) SELECT NROW FROM Customers

Q25) Fulfill missing parts in SQL statement below to find average *Price* for products listed in *Products* table.

SELECT AVG(Price)
FROM Products;

Q26) Fulfill missing parts in SQL statement below to find overal number of ordered products using *Quantity* field in *OrderDetails* table.

SELECT SUM(Quantity)
FROM OrderDetails;

Q27) Fulfill missing parts in SQL statement below to find all customers listed in the *Customers* table whose name starts with letter „b".

SELECT *
FROM Customers

WHERE CustomerName **LIKE "b%"**;

Q28) Fulfill missing parts in SQL statement below to find all customers listed in the *Customers* table whose name starts with letter „b“ and ends with letter „o“.

```
SELECT *
FROM Customers

WHERE CustomerName LIKE “b%o”;
```

Q29) Fulfill missing parts in SQL statement below to find all customers listed in the *Customers* table whose name has letter „b“ in the second position.

```
SELECT *
FROM Customers

WHERE CustomerName LIKE “_b%”;
```

Q30) Fulfill missing parts in SQL statement below to find all customers listed in the *Customers* table who live in Germany, UK, and USA.

```
SELECT *
FROM Customers

WHERE Country IN (“Germany“, “UK“, “USA“);
```

Q31) Fulfill missing parts in SQL statement below to find all products listed in the *Products* table whose price belongs to range from 5 to 25 EUR, including the begin and end values.

```
SELECT *
FROM Products

WHERE Price BETWEEN 5 AND 20;
```

Q32) What statement would you use to change temporarily name of the *CustomerName* field to *Customer* within *Customer* table?

a) **SELECT CustomerName AS Customer FROM Customers;**
b) SELECT CustomerName ALIAS Customer FROM Customers;
c) SELECT CustomerName LIKE Customer FROM Customers;
d) SELECT CustomerName TO Customer FROM Customers;

Q33) Fulfill missing parts in SQL statement below to select all orders with existing customer information.

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders

INNER JOIN Customers ON Orders.CustomerID = Customers. CustomerID;
```

Q34) Fulfill missing parts in SQL statement below to select all customers and any orders they might have.

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers

LEFT JOIN Orders ON  Customers.CustomerID = Orders.CustomerID;
```

Q35) Fulfill missing parts in SQL statement below to select all customers and any orders they might have.

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Orders

RIGHT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Q36) Fulfill missing parts in SQL statement below to select all customers and all orders.

SELECT Customers.CustomerName, Orders.OrderID
FROM **Customers**

FULL OUTER JOIN Orders **ON** Customers.CustomerID = **Orders**.CustomerID;

Q37) What operator would you use to merge selects from two different tables with the same number of columns in the same order and with similar data types?

a) JOIN
b) MERGE
c) UNITE
d) **UNION**

Q38) Fulfill missing parts in SQL statement below to calculate overal *Quantity* for each *ProductID* and arrange the resulting list in descending order according to this new metric.

SELECT **ProductID, SUM(Quantity)** AS Overall_Quantity
FROM OrderDetails

GROUP BY ProductID **ORDER BY Overall_Quantity DESC**;

Q39) Fulfill missing parts in SQL statement below to filter products whose overal *Quantity* is higher than 100 and arrange the resulting list in descending order according to the overal *Quantity*.

SELECT **ProductID, SUM(Quantity)** AS Overall_Quantity
FROM OrderDetails

GROUP BY ProductID **HAVING Overall_Quantity > 100 ORDER BY Overall_Quantity DESC**;

Q40) Fulfill missing parts in SQL statement below to create new field *Price_Level* that will classify products listed in the *Products* table as “Cheap‟ when their *Price* will be lower than 10 EUR or as “Expensive‟ otherwise.

SELECT ProductID, Price,
CASE

WHEN Price < 10 **THEN** “Cheap”
ELSE “Expensive”
END AS Price_Level

FROM Products