# UoL logoUniversity of London

# Assessment Coversheet

Complete this coversheet and read the instructions below carefully.

**Candidate Number**:

KJ1050

**Degree Title**:

BSc Computer Science

**Course/Module Title**:

Databases, Networks and the Web

**Course/Module Code:**

CM2040

**Enter the numbers, and sub-sections, of the questions in the order in which you have attempted them:**

**Question 2: a, b, c**

**Question 3: a, b, c, d**

**Date**: 15.03.2022

**Instructions to Candidates**

1. Complete this coversheet and begin typing your answers on the page below, or, submit the coversheet with your handwritten answers (where handwritten answers are permitted or required as part of your online timed assessment).
2. Clearly state the question number, and any sub-sections, at the beginning of each answer and also note them in the space provided above.
3. For typed answers, use a plain font such as Arial or Calibri and font size 11 or larger.
4. Where permission has been given in advance, handwritten answers (including diagrams or mathematical formulae) must be done on light coloured paper using blue or black ink.
5. Reference your diagrams in your typed answers. Label diagrams clearly.

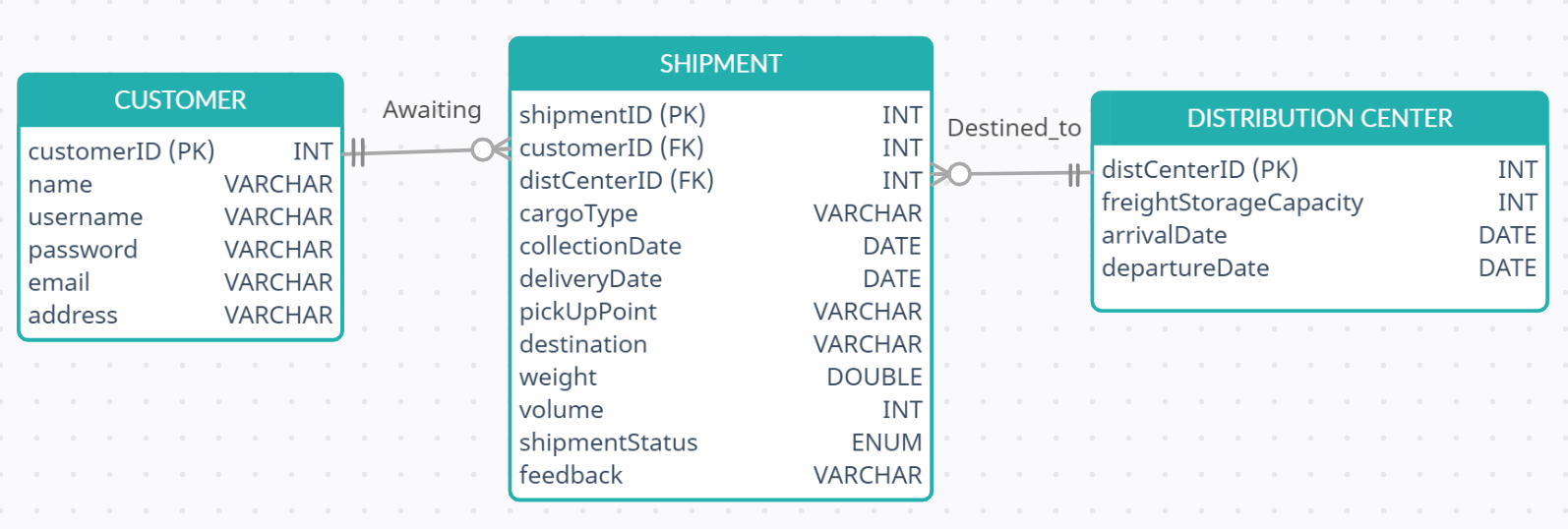
**The Examiners will attach great importance to legibility, accuracy and clarity of expression.**

**SECTION B**

**Question 2**

1. For the ship2europe database, create an Entity Relationship Diagram (ERD). Entities, relationships between entities, and appropriate association types should all be included in your diagram. Use only three tables in the diagram.

Entity relationship diagram is presented in the figure 1.

Figure 1 – Entity Relationship Diagram for the “ship2europe” database

**Entities:**

1. Customer, 2.Shipment, 3.Distribution Center

**Relationships and associations:**

1. “Awaiting” – Customer awaiting shipment – one-to-many relation (Customer can have several shipments, but every shipment must have customer).

2. “Destined to” – Shipment is destined to distribution center – one-to-many relation (Each shipment is destined exactly to one distribution center, and every distribution center can hold several shipments).

1. For each table of the database you designed in phase 1(a), list primary and foreign keys.

**Table 1** – Customer: **Primary key – customerID** (INT). We can choose to use username as primary key (because username is also unique parameter that defines each entry), but, primary key should not change, so I added ID field as primary key.

**Table 2** – Shipment: **Primary key – shipmentID** (INT). Same as before, it allows us to identify each shipment. **Foreign keys: customerID** (INT), **distCenterID** (INT). Foreign keys allow us to connect entities among themselves. I am also wasn’t sure about “destination” attribute and it connection to the data center, so I assumed that distribution center is not final destination, so we need another field to bridge “Shipment” and “Distribution center” tables. Using ID-s as foreign keys we can clearly determine which customer awaits current shipment, and to which distribution center should it go.

**Table 3** – Distribution Center: Primary key – **distCenterID** (INT). Same as tables 1 and 2.

1. Create two of the tables you mentioned in your answer to part 1(a) with the appropriate SQL code.

**Table 1 – Customer** (ID auto increment, all values are required, username unique):

CREATE TABLE customer (

customerID INT(10) NOT NULL AUTO\_INCREMENT,

name VARCHAR(16) NOT NULL,

username VARCHAR(16) NOT NULL,

password VARCHAR(16) NOT NULL,

email VARCHAR(16) NOT NULL,

address VARCHAR(50) NOT NULL,

PRIMARY KEY (customerID),

UNIQUE KEY 'username' (username));

**Table 2 – Shipment** (ID auto increment, all values are required, 2 foreign keys, collection and delivery dates are dates, weight is double, status should be one from selected list of values, so I decided to use ENUM type):

CREATE TABLE shipment (

shipmentID INT(10) NOT NULL AUTO\_INCREMENT,

customerID INT(10) NOT NULL,

distCenterID INT(10) NOT NULL,

cargoType VARCHAR(16) NOT NULL,

collectionDate DATE NOT NULL,

deliveryDate DATE NOT NULL,

pickUpPoint VARCHAR(50) NOT NULL,

destination VARCHAR(50) NOT NULL,

weight DOUBLE(6,2) NOT NULL,

volume INT(10) NOT NULL,

shipmentStatus ENUM('0','1','2','3','4','5') NOT NULL,

feedback VARCHAR(225) NOT NULL,

PRIMARY KEY (shipmentID),

FOREIGN KEY (customerID) REFERENCES customer(customerID),

FOREIGN KEY (distCenterID) REFERENCES dist\_center(distCenterID));

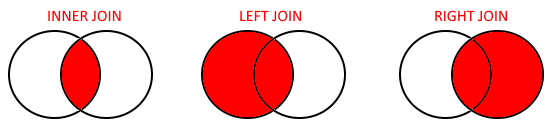
**Question 3**

1. Describe what an SQL INNER JOIN statement is. Similarly define the RIGHT and LEFT JOIN operations. Give set theory visual examples of the 3 operations.

SQL **INNER JOIN** statement is the command to **select entries that have matching values in both tables**. For example, command - ‘SELECT name FROM user INNER JOIN orders ON name.userID = orders.userID;’ - will return the list of names of users who have active orders. From the set theory perspective INNER JOIN is intersection of two data sets (A∩B). Visual examples are provided on the figure 2.

SQL **LEFT JOIN** statement is the command to **select ALL entries from the initial table (left one) and entries that have matching values from the ‘joined’ table (right one)**. For example, command - ‘SELECT name, orderID FROM user LEFT JOIN orders ON name.userID = orders.userID;’ - will return the list of all user names with order identification numbers (if user have active order) or ‘NULL’ value if user have no active orders. From the set theory perspective LEFT JOIN is the first data set OR the union of the first set and the intersection of the two sets (A U [A∩B]).

SQL **RIGHT JOIN** statement is the same as left join, but in reverse. It **selects ALL entries from the ‘joined’ table and entries that have matching values from the initial table**. From the set theory perspective RIGHT JOIN is the second data set OR the union of the second set and the intersection of the two sets (B U [A∩B]).

Figure 2 – SQL JOIN statements (set theory visualization)

1. Write a SQL JOIN statement that finds total damage costs of incidents where the customer named ‘Bill Crook’ was involved.

**SELECT SUM(damage\_cost) FROM Filled INNER JOIN Customer ON Filled.customer\_id = Customer.customer\_id WHERE Customer.name = 'Bill Crook';**

To calculate total damage we use SUM function to calculate total value from the list of all returned results of damage costs. We use INNER JOIN to select only matching data entries.

1. Write a SQL statement to find all types of incidents where a customer named ‘Bill Crook’ was involved.

**SELECT DISTINCT type FROM Incident WHERE case\_number IN (SELECT case\_number FROM Filled INNER JOIN Customer ON Filled.customer\_id = Customer.customer\_id WHERE Customer.name = 'Bill Crook');**

First of all, we prepare the list of all cases where a customer named ‘Bill Crook’ was involved (Select statement in brackets, where we join data from 2 tables as in (a)). And then we select all incident types (because we count several entries of the same type as one type of the incident I choose to use DISTINCT command) that has case number equal to the data from the list using IN operator command.

1. Write a SQL statement that finds the total number of incidents between 2010 and 2020 inclusive.

**SELECT COUNT(case\_number) FROM Incident WHERE year BETWEEN 2010 AND 2020;**

To count number of incidents we use COUNT function to count number of values returned by the SELECT command. To filter entries by the value range we can use [ BETWEEN val1 AND val2 ] operator construct to select year values from the range including both edge values.