**Database Management System-ICT330**

Name

Institutional Affiliation

**Database Management System-ICT330**

**Question 1**

Answer

**Customer** (customer\_ID(PK), email\_address, contact\_number, phone, IsCorporate)

**Corporate** (registration\_number(PK) (FK), year\_established

**Individual** (customer\_ID(PK), customer\_name

**Rental Request** (rental\_code(PK), collection\_date\_time, return\_date\_time, collection\_location, car\_model)

**Account** (account\_type(PK), credit\_limit, ourstanding\_balance)

**Quotation** (quotation\_number(PK), date, validity\_period)

**Question 2**

Answer

1. **Functional and Multi-Valued Dependencies**

ItemOffer (ItemNo, ItemDescription, OfferID, OfferStartDate, OfferEndDate, NonOfferPrice, OfferPrice, ItemStockLevel, ReceiptNo, purchaseDatetime, QtyPurchased)

Assume that a purchase is made by different customers and have different offer price depending on the item number, purchase date time, and quantity purchased. Assume that customers can have different `

**ItemNo OfferPrice**

**ItemNo NonOfferPrice**

**ItemNo purchasedDatetime**

**ItemNo QtyPurchased**

In the above case ItemNo is unique, but OfferPrice, purchasedDateTime, and QtyPurchased is treated be changing with time depending on the ItemNo

1. **State the candidate key(s) for the table ItemOffer**

**Answer**

(ItemNo, OfferPrice, NonOfferPrice, purchasedDatetime, QtyPurchased.

1. **Normalize the table to BCNF and 4NF**

Consider MVDs

ITEMOFFER\_2 (ItemNo, ItemDescription, OfferID, OfferStartDate, OfferEndDate NonOfferPrice, OfferPrice)

ITEMOFFERSTOCK (ItemNo, purchaseDate, ItemStockLevel, QtyPurchased)

ITEMOFFERPRICE (ItemNo, OfferID, OfferPrice, NonOfferPrice)

ITEMOFFER\_2 (ItemNo, ItemDescription, OfferID, OfferStartDate, OfferEndDate NonOfferPrice, OfferPrice)

Based on the above scenario, OfferID is not considered as NOT in BCNF

Hence, move OfferID OfferStartDate OfferStartDate OfferEndDate NonOfferPrice into another table, making OfferID to be the foreign key in the original table.

ITEMOFFERSTOCK (ItemNo, purchaseDate, ItemStockLevel, QtyPurchased)

In the above table, ItemNo is considered to be in BCNF no field will be moved into another table

ITEMOFFERPRICE (ItemNo, OfferID, OfferPrice, NonOfferPrice)

In the above table, ItemNo is in BCF, hence nothing needs to be done in this table.

**ALL THE ABOVE TABLES ARE NOW IN BCNF AND 4NF**

**Question 3**

1. **Logical Data Model**

-- Database: `equipment loans`

-- Table structure for table `customers`

CREATE TABLE `customers` (

 `id` int(11) NOT NULL,

 `name` varchar(30) NOT NULL,

 `address` varchar(100) NOT NULL,

 `contact` varchar(8) NOT NULL,

 `dateOfBirth` datetime NOT NULL,

 `occupation` varchar(30) NOT NULL

)

-- Table structure for table `damagereport`

CREATE TABLE `damagereport` (

 `reportID` int(11) NOT NULL,

 `damageType` varchar(16) NOT NULL

);

-- Table structure for table `equipment`

CREATE TABLE `equipment` (

 `equipmentCode` varchar(5) NOT NULL,

 `name` varchar(50) NOT NULL,

 `description` varchar(255) NOT NULL,

 `RentalRatePerDay` int(50) NOT NULL

)

-- Table structure for table `loan`

CREATE TABLE `loan` (

 `id` varchar (9) NOT NULL,

 `customer\_id` int(9) NOT NULL,

 `equipmentCode` varchar(5) NOT NULL,

 `startDate` datetime NOT NULL,

 `returnDate` datetime NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

ALTER TABLE `customers`

 ADD PRIMARY KEY (`id`);

-- Indexes for table `damagereport`

ALTER TABLE `damagereport`

 ADD PRIMARY KEY (`reportID`);

-- Indexes for table `equipment`

ALTER TABLE `equipment`

 ADD PRIMARY KEY (`equipmentCode`);

-- Indexes for table `loan`

--

ALTER TABLE `loan`

 ADD PRIMARY KEY (`id`),

 ADD UNIQUE KEY `equipmentCode` (`equipmentCode`);

COMMIT;

1. **SQL Statements to populate Data to Tables**

INSERT INTO `equipment` (`equipmentCode`, `name`, `description`, `RentalRatePerDay`) VALUES ('DIV09', 'Well Suit', 'Full Suit', '20.00');

INSERT INTO `equipment` (`equipmentCode`, `name`, `description`, `RentalRatePerDay`) VALUES ('CAM01', 'Tent 2 Persons', '2 Person Tent', '15');

INSERT INTO `equipment` (`equipmentCode`, `name`, `description`, `RentalRatePerDay`) VALUES ('CAM03', 'Tent 6 Persons', '6 Person Tent', '22.00');

INSERT INTO `equipment` (`equipmentCode`, `name`, `description`, `RentalRatePerDay`) VALUES ('CAM04', 'Camp Stove', '2 Burner Camping Stove', '8.00');

INSERT INTO `equipment` (`equipmentCode`, `name`, `description`, `RentalRatePerDay`) VALUES ('DIV04', 'Mask', 'Diving Mask', '4.00');

INSERT INTO `equipment` (`equipmentCode`, `name`, `description`, `RentalRatePerDay`) VALUES ('DIV05', 'Dive Torch', 'Diving Torch Requires 4 D size battery', '10.00');

INSERT INTO `customers` (`id`, `name`, `address`, `contact`, `dateOfBirth`, `occupation`) VALUES ('001', 'Well Suit', '23 Haig Road', '92142331', '1976-03-15 ', 'Teacher');

INSERT INTO `customers` (`id`, `name`, `address`, `contact`, `dateOfBirth`, `occupation`) VALUES ('002', 'Camp Stove', '5 Still Road', 88128833, 1985-07-05 ', 'Teacher');

INSERT INTO `customers` (`id`, `name`, `address`, `contact`, `dateOfBirth`, `occupation`) VALUES ('001', 'Well Suit', 'Haig Road', '92142331', '1976-03-15 ', 'Teacher');

INSERT INTO `customers` (`id`, `name`, `address`, `contact`, `dateOfBirth`, `occupation`) VALUES ('003', 'Damien Law', '10 Bed Rock Road', '82138213', '1988-01-23', 'Nurse');

**(b) Write SQL Statements for Each Task**

**(i).**

*SELECT*

 *equipment.equipmentCode,*

 *equipment.`name`,*

 *equipment.description,*

 *equipment.RentalRatePerDay,*

 *customers.dateOfBirth*

*FROM*

 *customers*

 *INNER JOIN*

 *loan*

 *ON*

 *customers.id = loan.customer\_id*

 *INNER JOIN*

 *loan*

 *ON*

 *loan.customer\_id = loan.customer\_id*

 *INNER JOIN*

 *equipment*

 *ON*

 *loan.equipmentCode = equipment.equipmentCode*

*WHERE*

 *equipment.RentalRatePerDay >10; AND*

 *customers.dateOfBirth > "1988--08-24";*



 (ii) Query to list first 3 letters …..

SELECT DISTINCT

 equipment.equipmentCode AS `Start Code`,

 COUNT( equipment.equipmentCode ) AS `Number of Equipment for June`

FROM

 customers,

 equipment

 INNER JOIN loan ON equipment.equipmentCode = loan.equipmentCode

WHERE

 loan.startDate = LIKE '%[2021-06]%'

 AND LEN ( equipment.equipmentCode )= 3;

 **(iii) Create a view named CustomerSummary**

CREATE VIEW CustomerSummary AS SELECT

customers.id,

customers.`name`,

customers.address,

customers.contact,

customers.dateOfBirth,

customers.occupation,

equipment.equipmentCode AS `Number of Equipment Rented`,

equipment.RentalRatePerDay AS `Total Rent`,

damagereport.reportID AS NumDamage

FROM

 customers

 INNER JOIN loan ON customers.id = loan.customer\_id

 INNER JOIN equipment ON loan.equipmentCode = loan.equipmentCode `

 INNER JOIN

 damagereport;

 **(iv) Average Rental Per Loan**

SELECT

 customers.`name`,

 customers.contact,

 COUNT( equipment.equipmentCode ) AS `Number of Equipment Rented`,

 equipment.RentalRatePerDay AS `Total Rent`,

 AVG( equipment.RentalRatePerDay ) AS `Average Rental`,

 MAX( equipment.RentalRatePerDay ) AS `Rental for the biggest Customer`,

 damagereport.reportID AS NumDamage

FROM

 customers,

 equipment,

 damagereport,

 loan

GROUP BY

 equipment.equipmentCode

**Question 4**

1. **Two transactions using Joint Accounts**

**Lost Update**

Based on the case of two transactions using joint account, the issue of the lost update would occur. The first transaction reads this record, does some processing then updates this record and finally commits its work. The second transaction reads the record then updates it immediately and commits. Both transactions do not update this record to the same value. This leads to a loss for the update statement performed by second transaction.

**Deadlock**

A deadlock occurs if each of two transactions needs exclusive use of some resource. In this case deadlock would occur since transaction 1 holds a resource that is needed by transaction 2. This means that the second transaction (transferring money to b2) must wait for the resource being held by the second transaction (transferring money to b1). Both transactions are in a deadlock state and the only way of resolving this is by to cancelling one of the transactions, thus releasing its resources

1. **Identify a Role for a Database for Banking Application**

Database for banking application would have many roles such as loan management, management for customers’ deposits and withdrawals, among others. For customers’ withdrawals the role that should be granted is update which can be associated with read and write permission. However, the role that should not be granted is alter which can be associated with a write permission.

**(c). Explain the Database Recovery Procedure**

 i. The recovery of database starts with Transaction T1

 ii. At that point, the transaction T2 and T3 are put into UNDO queue

 III. Moving forward the examined transactions with reference to time moves Forward and reaches the BEGIN of T4. Then the transaction T4 is put into the UNDO queue

Rollback is the explicit command in most cases, that is used to for starting the UNDO process of the named transactions T1 to T4. The Database Management System receives the Rollback command and UNDO the modifications made in all the transactions (T1 to T4)