Problem context:

# Task 1: Functional Dependencies

* *Identify the non-trivial FDs on the relation Abnormal\_Rel. Supplement your description with diagram(s).*

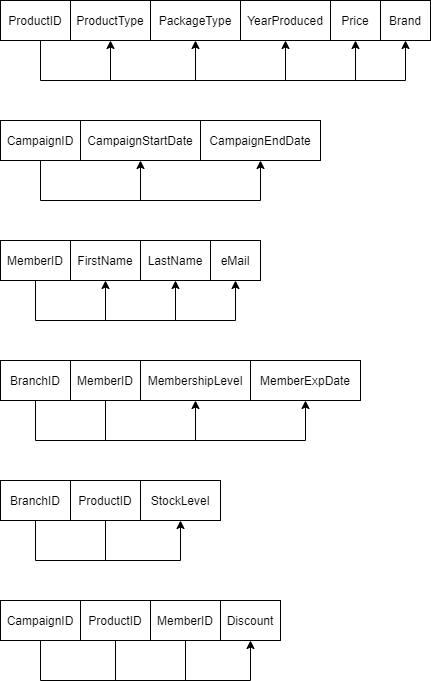
ProductID **->** ProductType, PackageType, YearProduced, Price, Brand CampaignID **->** CampaignStartDate, CampaignEndDate

MemberID **->** FirstName, LastName, eMail

BrandID, MemberID **->** MembershipLevel, MemberExpDate BranchID, ProductID **->** StockLevel

CampaignID, ProductID, MemberID **->** Discount

*Functional Dependencies diagram:*



* *Identify the Candidate key(s) of Abnormal\_Rel.*

The candidate keys of Abnormal\_Rel are {ProductID}, {CampaignID}, {MemberID},

{BranchID, MemberID}, {BranchID, ProductID}, {CampaignID, ProductID, MemberID} and

{ProductID, BranchID, eMail}.

# Task 2: Anomalies

* + *Insertion anomaly:*
    - The relation Abnormal\_Rel is highly susceptible to insertion anomaly because it exists a situation when we enter new data into the relation, but other referencing attributes are currently absent in the relation. This

therefore leads to data redundancy due to the data omission.

* + - Example: we assume that the ‘ BranchID ’ attribute must not be null, if a new product is added into the databased but not specified for any branch then that product cannot be entered as it will cause database inconsistencies.
  + *Deletion anomaly:*
    - Deletion can also happen in the relation Abnormal\_Rel as important data can be lost unintentionally when we delete other data without awareness of their referencing attributes. A considerable decrease in data accuracy and integrity can result from deletion anomaly.
    - Example: when a specific branch stops operating and gets removed from the databases, its current products and stock level information is deleted at the same time. This points out grouping all attributes which are not relation to others into one relation can causes intended loss of data.
  + *Update anomaly:*
    - Update anomaly is a kind of data inconsistency when a partial update is carried out in the Abnormal\_Rel relation. As there exist attributes that are dependent on other one or more attribute; therefore, we need to search for all referencing data before updating any data element to avoid data redundancy.
    - Example: if we want to change the Price of a product but we do not update the modified price in all branches that distribute the mentioned product, this would cause update anomaly as it is not synchronous throughout the whole databases of all branches.

# Task 3: Normalization

* + *What is the highest NF that the relation Abnormal\_Rel satisfies? Explain why.*

The Abnormal\_Rel relation can achieve BCNF as the highest normal form because based on all the functional dependencies, the determinant in each relation that can be derived is a candidate key so that it will satisfy BCNF.

* + *Normalize/decompose Abnormal\_Rel until you get relations that are in 3NF. Use appropriate illustration to aid the understanding of your work.*

We assume that the relation Abnormal\_Rel is currently in the 1NF and (*ProductID*,

*BrandID*) is the primary key.

**Abnormal\_Rel** (*ProductID*, *BranchID*, MemberID, CampaignID, ProductType, PackageType, YearProduced, Price, Brand, StockLevel, CampaignStartDate, CampaignEndDate, FirstName, LastName, eMail, MembershipLevel, MemberExpDate, Discount)

**2NF:** remove all partial dependencies.

**Product** (*ProductID*, ProductType, PackageType, YearProduced, Price, Brand)

**Product\_Stock** (*ProductID*, *BrandID*, StockLevel) **Product\_Distribution1** (*ProductID*, *BranchID*, MemberID, CampaignID, CampaignStartDate, CampaignEndDate, FirstName, LastName, eMail, MembershipLevel, MemberExpDate, Discount)

**3NF:** remove all transitive dependencies.

**Product** (*ProductID*, ProductType, PackageType, YearProduced, Price, Brand)

**Product\_Stock** (*ProductID*, *BrandID*, StockLevel)

**Campaign** (*CampaignID*, CampaignStartDate, CampaignEndDate) **Member** (*MemberID*, FirstName, LastName, eMail) **Product\_Distribution2** (*ProductID*, *BranchID*, *CampaignID*, *MemberID*, MembershipLevel, MemberExpDate, Discount)

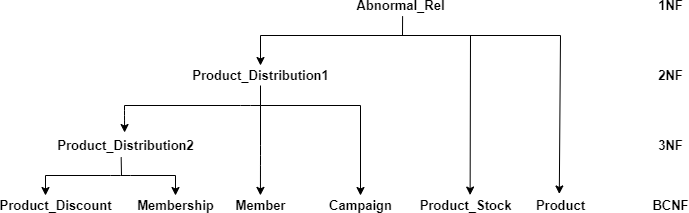
* + *Check if the resultant relations are in BCNF. If not, decompose them as necessary until you get all of them in BCNF.*

Relations **Product**, **Product\_Stock**, **Campaign**, **Member** are in BCNF but **Product\_Distribution2** is currently not in BCNF. Based on two functional dependencies:

BrandID, MemberID **->** MembershipLevel, MemberExpDate CampaignID, ProductID, MemberID **->** Discount

We can derive the last relation into two relations that achieve BCNF.

**Membership** (*BrandID*, *MemberID*, MembershipLevel, MemberExpDate)

**Product\_Discount** (*CampaignID*, *ProductID*, *MemberID*, Discount) Review of Normalization process (UNF to BCNF):

# Task 4: Table Creation and Population

* + *Copy and paste your DDL code for creating each table/relation in BCNF obtained in Task 3.*

CREATE TABLE `product` (

`ProductID` varchar(50) NOT NULL,

`ProductType` varchar(50) NOT NULL,

`PackageType` varchar(50) NOT NULL,

`YearProduced` int(4) NOT NULL,

`Price` decimal(10,2) NOT NULL,

`Brand` varchar(50) NOT NULL, PRIMARY KEY (`ProductID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `product\_stock` (

`BrandID` varchar(50) NOT NULL,

`ProductID` varchar(50) NOT NULL,

`StockLevel` int(11) NOT NULL, PRIMARY KEY (`BrandID`),

CONSTRAINT `product\_stock\_ibfk\_1` FOREIGN KEY (`ProductID`) REFERENCES

`product` (`ProductID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `campaign` (

`CampaignID` varchar(50) NOT NULL,

`CampaignStartDate` DATE NOT NULL,

`CampaignEndDate` DATE NOT NULL, PRIMARY KEY (`CampaignID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `member` (

`MemberID` varchar(60) NOT NULL,

`FirstName` varchar(50) NOT NULL,

`LastName` varchar(50) NOT NULL,

`eMail` varchar(100) NOT NULL, PRIMARY KEY (`MemberID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `membership` (

`BrandID` varchar(50) NOT NULL,

`MemberID` varchar(60) NOT NULL,

`MembershipLevel` varchar(20) NOT NULL,

`MemberExpDate` date NOT NULL, PRIMARY KEY (`BrandID`,`MemberID`),

CONSTRAINT `membership\_ibfk\_1` FOREIGN KEY (`MemberID`) REFERENCES

`member` (`MemberID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `product\_discount` (

`CampaignID` varchar(50) NOT NULL,

`ProductID` varchar(50) NOT NULL,

`MemberID` varchar(20) NOT NULL,

`Discount` decimal(5,2) NOT NULL,

PRIMARY KEY (`ProductID`,`CampaignID`,`MemberID`),

CONSTRAINT `product\_discount\_ibfk\_1` FOREIGN KEY (`CampaignID`) REFERENCES

`campaign` (`CampaignID`),

CONSTRAINT `product\_discount\_ibfk\_2` FOREIGN KEY (`ProductID`) REFERENCES

`product` (`ProductID`),

CONSTRAINT `product\_discount\_ibfk\_3` FOREIGN KEY (`MemberID`) REFERENCES

`member` (`MemberID`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

* + *Copy and paste your SQL code for inserting at least five rows of data into each of these table.*

insert into `product`(`ProductID`, `ProductType`, `PackageType`, `YearProduced`,

`Price`, `Brand`) values ('1','wine','bottle','2010','899.00','Penfolds Grange 2010'),

('2','wine','bottle','2016','848.99','Penfolds Grange 2016'), ('3','beer','can','2020','4.39','Brick Lane Sidewiner Hazy Pale Low Alc 355mL'), ('4','spirit','bottle','2011','164.99','Corowa Mad Dog Morgan Whisky 500mL'), ('5','beer','bottle','2020','5.09','Crown Lager 375mL');

insert into `product\_stock` (`BrandID`, `ProductID`, `StockLevel`) values ('1','1','100'),

('5','1','55'),

('2','3','200'),

('4','2','15'),

('6','5','255');

insert into `campaign` (`CampaignID`, `CampaignStartDate`, `CampaignEndDate`) values

('1','2021-12-21','2021-12-27'),

('2','2021-12-20','2021-12-26'),

('3','2021-9-20','2021-9-27'),

('4','2021-4-15','2021-4-19'),

('5','2020-12-28','2021-1-3');

insert into `member` (`MemberID`,`FirstName`,`LastName`,`eMail`) values ('1','Jonathan','Vuong','jonaking@gmail.com'),

('2','Simone','Singh','ssingh12@gmail.com'),

('3','Banh','Ngoo','ngobakha1st@gmail.com'),

('4','Bruno','Pogba','manuplayer@gmail.com'),

('5','Mixi','Phung','mixi1989@gmail.com');

insert into `membership` (`BrandID`,`MemberID`,`MembershipLevel`,`MemberExpDate`) values ('5','1','Gold','2021-10-15'),

('1','4','Platinum','2021-7-29'),

('2','2','Platinum','2021-11-22'),

('6','2','Platinum','2021-11-22'),

('4','3','Silver','2021-11-1');

insert into `product\_discount` (`CampaignID`,`ProductID`,`MemberID`,`Discount`) values

('2','1','1','15.00'),

('2','3','2','20.00'),

('2','5','2','20.00'),

('3','1','4','20.00'),

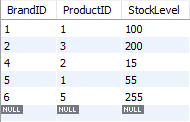
('5','2','3','10.00');

* + *Copy and paste the SELECT \* query to display the content of each table above, and screenshot of the content as displayed.*

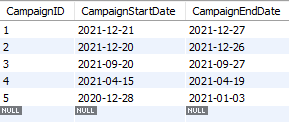
select \* from product;



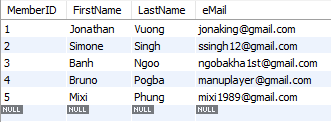
select \* from product\_stock;



select \* from campaign;



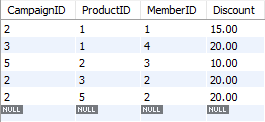
select \* from member;



select \* from membership;



select \* from product\_discount;



# Task 5: SQL Queries

Copy and paste the SQL queries followed by their results (screenshot) for each of the following query

**[Query 1]** *List the branches (ID) of MA that have in stock at least 5 bottles of Penfold Grange 2010.*

select pt.BrandID

from product p, product\_stock pt where p.ProductID = pt.ProductID and p.Brand = 'Penfolds Grange 2010' and pt.StockLevel >= 5;

Graphical user interface, application, Teams  Description automatically generated

**[Query 2]** *List details of each beer that Simone Singh will be entitled to get 20% discount on.*

select \*

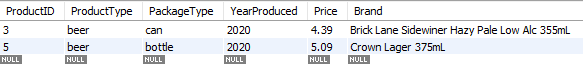
from product p1 where p1.ProductID in (select p2.ProductID

from product p2, member m, membership ms, product\_discount pd, campaign c where p2.ProductID = pd.ProductID

and m.MemberID = ms.MemberID and m.MemberID = pd.MemberID and c.CampaignID = pd.CampaignID and m.FirstName = 'Simone'

and m.LastName = 'Singh' and p2.ProductType = 'beer' and pd.Discount = '20.00'

and '2021-12-24' >= c.CampaignStartDate and '2021-12-24' <= c.CampaignEndDate);



**[Query 3]** *Generate a list of all email addresses of members whose card will expire in the month after the coming month, ordered appropriately.*

select distinct m.eMail

from member m, membership ms where m.MemberID = ms.MemberID

and ms.MemberExpDate >= '2021-11-01' and ms.MemberExpDate <='2021-11-30'

order by ms.BrandID asc, ms.MemberExpDate asc, m.eMail asc;

Text  Description automatically generated

**[Query 4]** *Determine how many times Penfold Grange 2010 has gone on sale since Covid- 19 related lockdown started (assume it to be March 01, 2020).*

select p.Brand, count(pd.ProductID) as 'Number of Sales' from product p, campaign c, product\_discount pd

where pd.ProductID = p.ProductID and pd.CampaignID = c.CampaignID and p.Brand = 'Penfolds Grange 2010';

Table  Description automatically generated