Database Design & Manipulation (Includes SQL)

This project aims to get the proper skills to design and manipulate the Database using normalisation, transformation, SQL.

Task 1: Functional Dependencies

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

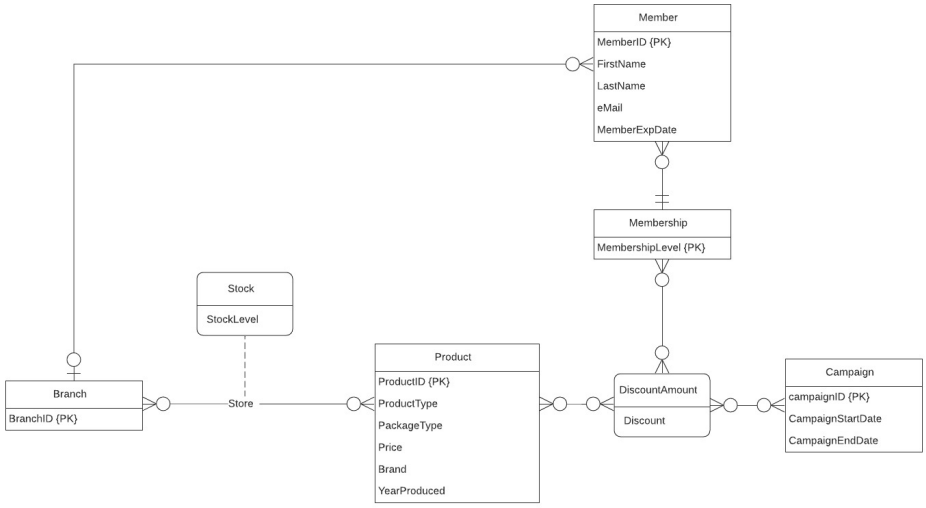
ProductID -> ProductType, PackageType, YearProduced, Price, Brand

campaignID-> CampaignStartDate, CampaignEndDate

MemberID -> FirstName, LastName, eMail, MembershipLevel, MemberExpDate, BranchID

ProductID, BranchID -> StockLevel

ProductID, campaignID, membershipLevel -> Discount



Descriptions based on the diagram:

ProductID determines ProductType, PackageType, YearProduced, Price, Brand.

campaignID determines CampaignStartDate, CampaignEndDate.

MemberID determines FirstName, LastName, eMail, MembershipLevel, MemberExpDate, BranchID.

ProductID, BranchID determine the stock level of each branch since every branch has a different amount of products' needs.

ProductID, campaignID, membershipLevel determine the discount rates since each products' discount rate is different by the membership level and campaign.

- *Identify the Candidate key(s) of Abnormal_Rel.*

ProductID, BranchID, campaignID, MemberID

Task 2: Anomalies

ProductID	BranchID	campaignID	MemberID	ProductType	PackageType	YearProduced	Price	Brand	Stock Level	Campaign StartDate	CampaignEndDate	First Name	Last Name	eMail	MembershipLevel	MemberExpDate	Discount
P101	B400	C111	M02	wine	box	2021	35	Yellow Tail	50	2021-09-11	2021-12-12	Ariana	Grande	AriBest@gmail.com	Sliver	2021-10-10	5
P102	B400	C111	M03	beer	box	2020	40	VB	85	2021-09-11	2021-12-12	Jessie	Lee	Jessie12@google.com	VIP	2022-12-20	30
P102	B440	C111	M99	beer	box	2020	40	VB	70	2021-09-11	2021-12-12	Jason	Kim	JasonTheMan@google.com	Gold	2023-09-09	25
P103	B440	C124	M10	wine	box	2021	30	Penfold Grange	60	2021-08-08	2022-01-22	Sam	Smith	SmithSam98@google.com	Sliver	2021-12-11	12
P104	B478	C124	M540	beer	bottle	2021	20	Heineken	20	2021-08-08	2022-01-22	Maren	Morris	MarenMaren@google.com	Gold	2022-01-03	15

The above table is "Abnormal_Rel" with random data to show and explain anomalies.

Discount means discount rates. The unit of discount rates is %.

The unit of price is $.

The datetime data is saved in the format of 'yyyy-mm-dd'.

- *Modification anomalies*
    - *Not susceptible.*
    - *The anomaly could occur when changing the PackageType of the P102 product from box to bottle.*
    - *Example: Changing PackageType in only the 2nd row of the table without changing the 3rd row makes it impossible to judge whether p102's PackageType is box or bottle.*
- *Deletion anomalies*
    - *Not susceptible.*
    - *The anomaly could occur when deleting the 5th row of the table.*
    - *Example: If the 5th row is deleted from the table, information about the bottle type is also lost because there is only one tuple whose PackageType is bottle.*
- *Insertion anomalies*
    - *Not susceptible.*
    - *The anomaly could occur when adding new row with some empty attributes.*
    - *Example: If a new campaign C125 is started and there are no members who have benefited from the campaign yet, this campaign cannot be entered into the table because the memberID field has not been filled.*

These anomalies are caused by trying to express multiple dependencies that exist between attributes as a single relation. Therefore, need to make different tables through normalization to avoid these anomalies.

Task 3: Normalization

- *What is the highest NF that the relation Abnormal_Rel satisfies? Explain why.*

The highest NF of Abnormal_Rel is 1NF.

Reasons:

1. Each row has only one column value(atomic value). No multi-value attributes. Therefore, Abnormal_Rel is more advanced than UNF.

2. In Task 1, when checked the functional dependencies, Abnormal_Rel had several partial dependencies. Every non-primary-key attribute in Abnormal_Rel is not fully functionally dependent on the primary key. Therefore it is not 2NF.

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

Before

Before

Abnormal_Rel

(**ProductID, BranchID, campaignID, MemberID**) -> ProductType, PackageType, YearProduced, Price, Brand, StockLevel, CampaignStartDate, CampaignEndDate, FirstName, LastName, eMail, MembershipLevel, MemberExpDate, Discount

3NF

Product

ProductID -> ProductType, PackageType, YearProduced, Price, Brand

ProductID	ProductType	PackageType	YearProduced	Price	Brand

Campaign

CampaignID -> CampaignStartDate, CampaignEndDate

CampaignID	CampaignStartDate	CampaignEndDate

Branch

BranchID

BranchID

Member

MemberID -> FirstName, LastName, eMail, *MembershipLevel* ,*BranchID* , MemberExpDate

MemberID	FirstName	LastName	eMail	*MemberShipLevel*	*BranchID*	MemberExpDate

Membership

MembershipLevel

MembershipLevel

Stock

(***ProductID, BranchID***) -> StockLevel

ProductID	***BranchID***	StockLevel

Red, Italic: FK

Bold, Underlined: PK

Discount

(***ProductID, campaignID, MembershipLevel***) -> Discount

ProductID	***campaignID***	***MembershipLevel***	Discount

Bold, Underlined, Italic, Red: FK & PK

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

The Stock table has two candidate keys, but the non-prime attribute “StockLevel” is not a determinant of other prime attributes.
The Discount table has three candidate keys, but the non-prime attribute “Discount” is not a determinant of other prime attributes.
Therefore, normalized tables before are already BCNF.

Task 4: Table Creation and Population

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

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;

SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;

SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_Z
ERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema Project

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `Project` DEFAULT CHARACTER SET utf8 ;

USE `Project` ;



-- -----------------------------------------------------

-- Table `Project`.`Branch`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Project`.`Branch` (

  `BranchID` VARCHAR(4) NOT NULL,

  PRIMARY KEY (`BranchID`))

ENGINE = InnoDB;



-- -----------------------------------------------------

-- Table `Project`.`Product`

-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `Project`.`Product` (
  `ProductID` VARCHAR(4) NOT NULL,
  `ProductType` VARCHAR(20) NULL,
  `PackageType` VARCHAR(20) NULL,
  `YearProduced` INT NULL,
  `Price` DECIMAL(5,2) NULL,
  `Brand` VARCHAR(20) NULL,
  PRIMARY KEY (`ProductID`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Project`.`Campaign`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Project`.`Campaign` (
  `CampaignID` VARCHAR(4) NOT NULL,
  `CampaignStartDate` DATETIME NULL,
  `CampaignEndDate` DATETIME NULL,
  PRIMARY KEY (`CampaignID`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Project`.`Member`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Project`.`Member` (
  `MemberID` VARCHAR(4) NOT NULL,
  `FirstName` VARCHAR(10) NULL,
  `LastName` VARCHAR(10) NULL,
```

```
  `eMail` VARCHAR(35) NULL,

  `MemberExpDate` DATETIME NULL,

  `MembershipLevel` VARCHAR(10) NOT NULL,

  `BranchID` VARCHAR(4) NOT NULL,

  PRIMARY KEY (`MemberID`),

  Constraint Membership_fk FOREIGN KEY (MembershipLevel) references Membership
(MembershipLevel),

  Constraint BranchID_m_fk FOREIGN KEY (BranchID) references Branch (BranchID)

  )

ENGINE = InnoDB;


-- -----------------------------------------------------

-- Table `Project`.`Membership`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Project`.`Membership` (

  `MembershipLevel` VARCHAR(10) NOT NULL,

  PRIMARY KEY (`MembershipLevel`))

ENGINE = InnoDB;


-- -----------------------------------------------------

-- Table `Project`.`Stock`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Project`.`Stock` (

  `ProductID` VARCHAR(4) NOT NULL,

  `BranchID` VARCHAR(4) NOT NULL,

  `StockLevel` INT NOT NULL,

   PRIMARY KEY (`ProductID`,`BranchID`),
```

```
  Constraint Product_fk FOREIGN KEY (ProductID) references Product (ProductID),
  Constraint Branch_fk FOREIGN KEY (BranchID) references Branch (BranchID)
 )
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Project`.`Discount`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Project`.`Discount` (
 `ProductID` VARCHAR(4) NOT NULL,
 `CampaignID` VARCHAR(4) NOT NULL,
 `MembershipLevel` VARCHAR(10) NOT NULL,
 `Discount` INT NOT NULL,
  PRIMARY KEY (`ProductID`,`CampaignID`,`MembershipLevel`),
  Constraint Product_D_fk FOREIGN KEY (ProductID) references Product (ProductID),
  Constraint Campaign_D_fk FOREIGN KEY (CampaignID) references Campaign
(CampaignID),
  Constraint Membership_D_fk FOREIGN KEY (MembershipLevel) references
Membership (MembershipLevel)
 )
ENGINE = InnoDB;
```

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

```
-- insert values for membership

insert into Membership Values ("VIP");

insert into Membership Values ("Platinum");

insert into Membership Values ("Gold");

insert into Membership Values ("Sliver");

insert into Membership Values ("Bronze");


-- insert values for branch

insert into Branch Values ("B101");

insert into Branch Values ("B102");

insert into Branch Values ("B103");

insert into Branch Values ("B104");

insert into Branch Values ("B105");


-- insert values for member

insert into Member Values ("M357","Simone","Singh","simone123@gmail.com","2022-02-24","Gold","B101");

insert into Member Values ("M203","Jane","Doe","janejane4ever@gmail.com","2021-11-24","VIP","B102");

insert into Member Values ("M11","Ariana","Grande","ari468@gmail.com","2021-12-12","Sliver","B103");

insert into Member Values ("M580","Justin","Biber","justin96@gmail.com","2022-01-30","Bronze","B104");

insert into Member Values ("M721","Jason","Lee","jasonlee1996@gmail.com","2050-02-04","VIP","B105");
```

```
-- insert values for product
insert into Product Values ("P399","wine","bottle",2015,15.5,"Yellow Tail");
insert into Product Values ("P400","wine","bottle",2010,12,"Penfold Grange");
insert into Product Values ("P570","beer","box",2020,35,"Heineken");
insert into Product Values ("P300","beer","bottle",2020,5.5,"VB");
insert into Product Values ("P203","beer","bottle",2021,3.2,"Pure Blonde");

-- insert values for Campaign
insert into Campaign Values ("C432","2020-07-12","2022-01-12");
insert into Campaign Values ("C590","2021-02-12","2022-03-12");
insert into Campaign Values ("C300","2015-10-12","2019-02-25");
insert into Campaign Values ("C672","2021-09-23","2022-05-21");
insert into Campaign Values ("C401","2019-11-17","2020-02-24");

-- insert values for Stock
insert into Stock Values ("P400","B101",10);
insert into Stock Values ("P400","B102",20);
insert into Stock Values ("P400","B103",4);
insert into Stock Values ("P300","B104",20);
insert into Stock Values ("P203","B105",500);

-- insert values for Discount
insert into Discount Values ("P203","C432","Gold",30);
insert into Discount Values ("P300","C590","Gold",20);
insert into Discount Values ("P400","C300","Gold",70);
insert into Discount Values ("P400","C672","Gold",10);
insert into Discount Values ("P570","C401","Gold",20);
```

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

-- display Membership

select * from Membership;

Result Grid Filter Rows: Search Edi

MembershipLevel
Bronze
Gold
Platinum
Sliver
VIP
NULL

-- display Branch

select * from Branch;

Result Grid Filter Rows: Search

BranchID
B101
B102
B103
B104
B105
NULL

-- display Member

select * from Member;

Result Grid Filter Rows: Search Edit: Export/Import:

MemberID	FirstName	LastName	eMail	MemberExpDate	MembershipLevel	BranchID
M11	Ariana	Grande	ari468@gmail.com	2021-12-12 00:00:00	Sliver	B103
M203	Jane	Doe	janejane4ever@gmail.com	2021-11-24 00:00:00	VIP	B102
M357	Simone	Singh	simone123@gmail.com	2022-02-24 00:00:00	Gold	B101
M580	Justin	Biber	justin96@gmail.com	2022-01-30 00:00:00	Bronze	B104
M721	Jason	Lee	jasonlee1996@gmail.com	2050-02-04 00:00:00	VIP	B105
NULL	NULL	NULL	NULL	NULL	NULL	NULL

-- display Product

select * from Product;

Result Grid Filter Rows: Search Edit: Export/Import:

ProductID	ProductType	PackageType	YearProduced	Price	Brand
P203	beer	bottle	2021	3.20	Pure Blonde
P300	beer	bottle	2020	5.50	VB
P399	wine	bottle	2015	15.50	Yellow Tail
P400	wine	bottle	2010	12.00	Penfold Grange
P570	beer	box	2020	35.00	Heineken
NULL	NULL	NULL	NULL	NULL	NULL

-- display Campaign

select * from Campaign;

Result Grid Filter Rows: Search Edit:

CampaignID	CampaignStartDate	CampaignEndDate
C300	2015-10-12 00:00:00	2019-02-25 00:00:00
C401	2019-11-17 00:00:00	2020-02-24 00:00:00
C432	2020-07-12 00:00:00	2022-01-12 00:00:00
C590	2021-02-12 00:00:00	2022-03-12 00:00:00
C672	2021-09-23 00:00:00	2022-05-21 00:00:00
NULL	NULL	NULL

-- display Stock

select * from Stock;

Result Grid Filter Rows: Search

ProductID	BranchID	StockLevel
P203	B105	500
P300	B104	20
P400	B101	10
P400	B102	20
P400	B103	4
NULL	NULL	NULL

-- display Discount

select * from Discount;

Result Grid Filter Rows: Search Ec

ProductID	CampaignID	MembershipLevel	Discount
P203	C432	Gold	30
P300	C590	Gold	20
P400	C300	Gold	70
P400	C672	Gold	10
P570	C401	Gold	20
NULL	NULL	NULL	NULL

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.

-- Query 1

select b.BranchID from Branch b, Product p, Stock s where s.BranchID=b.BranchID and s.ProductID=p.ProductID

and p.Brand="Penfold Grange" and p.YearProduced=2010

and s.Stocklevel>=5;

Result Grid Filter Rows: Search Export:

BranchID
B101
B102

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

```
-- Query 2

select p.ProductID, p.ProductType, p.PackageType, p.YearProduced,p.Brand,

p.Price OriginalPrice, format(Price*(1-d.Discount/100),2) DiscountedPrice

from Product p, Campaign c, Member m, Membership ms, Discount d

where p.ProductID=d.ProductID and c.CampaignID=d.CampaignID

and ms.Membershiplevel=d.Membershiplevel and
ms.Membershiplevel=d.Membershiplevel

and d.Discount=20 and m.FirstName="Simone" and m.Lastname="Singh"

and "2021-12-24" between c.CampaignStartDate and c.CampaignEndDate

and m.MemberExpDate >="2021-12-24";
```

Result Grid Filter Rows: Search Export:

ProductID	ProductType	PackageType	YearProduced	Brand	OriginalPrice	DiscountedPrice
P300	beer	bottle	2020	VB	5.50	4.40

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

```
-- Query 3

select b.BranchID, m.MemberExpDate, m.eMail, now() QueryRunDate

from Branch b, Member m
```

where b.BranchID=m.BranchID and

year(DATE_ADD(now(), INTERVAL 2 MONTH))=year(m.MemberExpDate) and

month(DATE_ADD(now(), INTERVAL 2 MONTH))=month(m.MemberExpDate)

order by b.BranchID asc, m.MemberExpDate asc, m.eMail asc;

Result Grid Filter Rows: Search Export:

BranchID	MemberExpDate	eMail	QueryRunDate
B102	2021-11-24 00:00:00	janejane4ever@gmail.com	2021-09-27 21:41:59

[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).

-- Query 4

select count(*) SaleCountAfterCovid

from Product p, Campaign c, Discount d

where p.ProductID=d.ProductID and c.CampaignID=d.CampaignID and

p.Brand="Penfold Grange" and p.YearProduced=2010 and

c.CampaignStartDate>="2020-03-01"

Result Grid Filter Rows: Search Export:

SaleCountAfterCovid
1