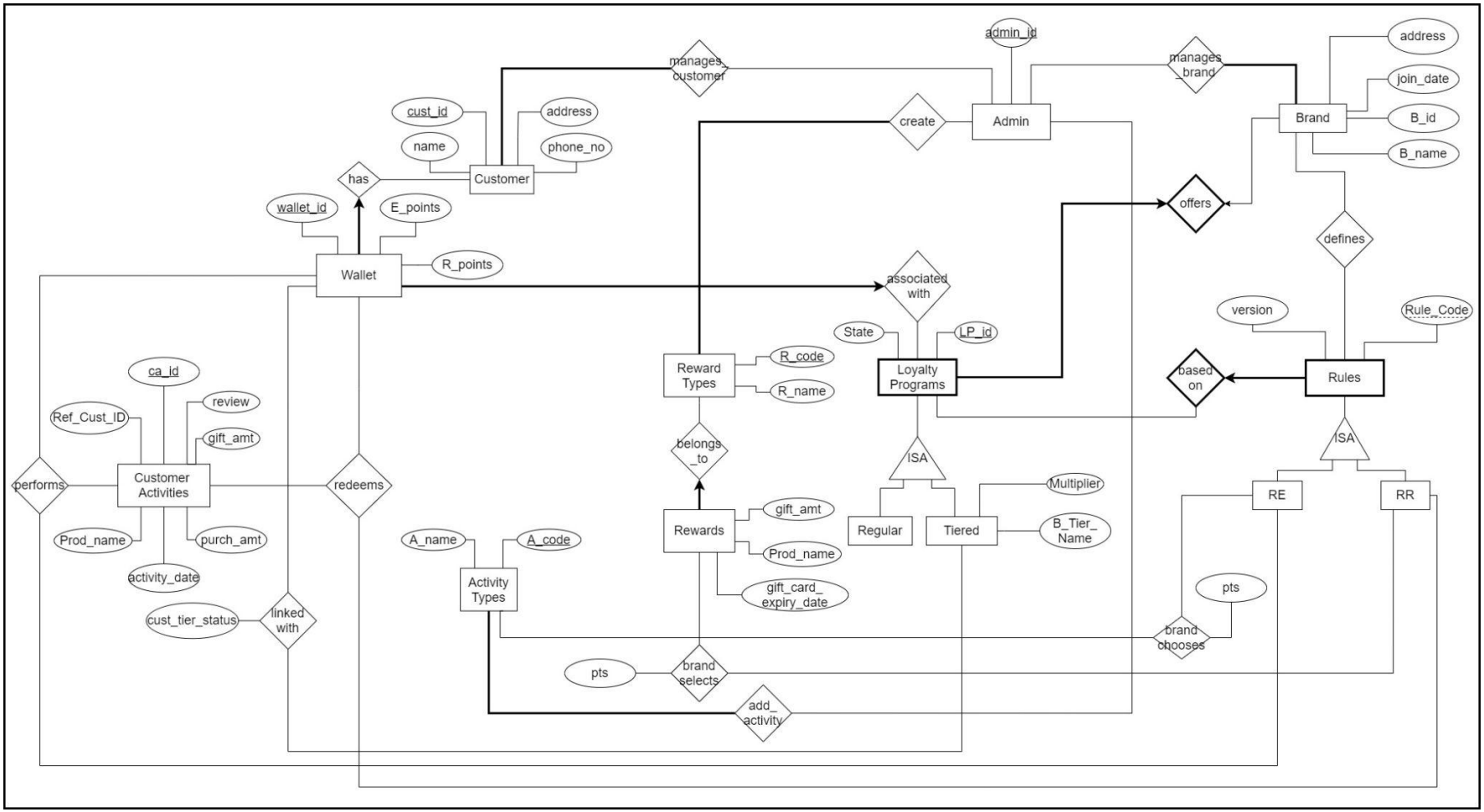
CUSTOMER MARKETPLACE LOYALTY APPLICATION
CSC 540
PROJECT TEAM #27

Manasi Sanjay Ghosalkar	**mghosal**
Nishitha Yza	**nyza**
Sudharsan Janardhanan	**sjanard**
Yugalee Vijay Patil	**ypatil**

TABLE OF CONTENTS

ER DIAGRAM
ASSUMPTIONS
RELATIONAL SCHEMA, FUNCTIONAL DEPENDENCIES AND CREATE STATEMENTS
DESCRIPTION OF CONSTRAINTS
SHOW QUERIES
TECHNOLOGY STACK

ER DIAGRAM


admin_id
manages_customer
manages_brand
address
join_date
B_id
B_name
cust_id
address
name
phone_no
create
Admin
Brand
Customer
has
wallet_id
E_points
offers
defines
Wallet
R_points
associated with
version
Rule_Code
State
LP_id
Reward Types
R_code
R_name
Loyalty Programs
based on
Rules
ca_id
review
Ref_Cust_ID
gift_amt
belongs_to
ISA
ISA
performs
Customer Activities
redeems
Multiplier
RE
RR
Rewards
gift_amt
Regular
Tiered
B_Tier_Name
Prod_name
purch_amt
A_name
A_code
Prod_name
activity_date
Activity Types
gift_card_expiry_date
pts
cust_tier_status
linked with
brand chooses
pts
pts
brand selects
add_activity

ASSUMPTIONS

1. Administrators add each entity to the database, i.e they authorize new brands and customers. They are also in charge of adding new activity types and reward types. Besides this, new customers and brands can add themselves to the marketplace application using the signup functionality.

2. Each customer can participate in a loyalty program with different brands. This means that although a customer has a globally unique WALLET_ID, the number of points they earn and redeem with respect to each Loyalty Program will differ. So the system needs to keep track of the number of points earned, the customer tier status, the number of points redeemed and the threshold required for each Loyalty Program in a single wallet.

3. The Loyalty Program's tiers can be identified by a unique name for each brand. Multiple brands can have overlapping names for the tiers

4. The marketplace application can accommodate three activity types namely *purchase*, *leave a review* and *refer a friend*. Other activity types can be added later as well.

5. The marketplace application can accommodate two reward types namely *Gift Card* and *Free Product*. Other activity types can be added later as well.

6. Gift cards have an attribute called gift card amount which is the actual dollar value of the gift card. This value can be subtracted from the purchase amount if a customer chooses to make a purchase by using a gift card. When a customer chooses to redeem points by buying a gift card, the gift card gets stored in customer activities corresponding to the customer and the loyalty program. Later, when making a purchase, the customer has an option to use this gift card. If the customer chooses to do so, the gift_card amount gets subtracted from the purchase amount and the customer has to pay only the difference.

7. A brand can enroll in the marketplace application but have an inactive loyalty program, a program that hasn't been released to the application. Brands can also choose to not have a loyalty program at all.

8. If a customer earns enough points to be promoted to a tier, they remain at that tier and cannot be demoted even if they have insufficient points for that tier. In short, the customer tier status is immutable.

9. It is also assumed that every purchase a user performs earns the user a fixed number of points according to the respective RE Rule, as long as the amount of purchase is positive.

DESCRIPTION OF CONSTRAINTS

1. Brand sequence

```
CREATE SEQUENCE brand_id_seq START WITH 1 INCREMENT BY 1;
/
CREATE OR REPLACE TRIGGER brand_insert
BEFORE INSERT ON BRAND
FOR EACH ROW

BEGIN
SELECT concat('Brand0',brand_id_seq.NEXTVAL)
INTO   :new.B_ID
FROM   dual;
END;
/
```

The sequence mentioned above automatically creates a Brand ID attribute for each entry upon an insert in the BRAND table. The Brand ID attribute is expected to follow the format 'BrandXX' where XX denotes the ID number. An example entry is for the first brand **Brand01**.

2. Customer ID sequence

```
CREATE SEQUENCE customer_id_seq START WITH 1
INCREMENT BY 1;
/
CREATE OR REPLACE TRIGGER customer_insert
BEFORE INSERT ON CUSTOMER
FOR EACH ROW

BEGIN
  SELECT concat('C000',customer_id_seq.NEXTVAL)
  INTO   :new.CUST_ID
  FROM   dual;
END;
/
```

The sequence mentioned above automatically creates a Customer ID attribute for each entry upon an insert in the Customer table. The Customer ID attribute is expected to follow the format 'C000XX' where XX denotes the ID number. An example entry is for the first customer **C0001**.

3. Wallet ID sequence

```
CREATE SEQUENCE wallet_id_seq START WITH 1
INCREMENT BY 1;
```

```
/
CREATE OR REPLACE TRIGGER wallet_insert
BEFORE INSERT ON CUSTOMER
FOR EACH ROW

BEGIN
  SELECT concat('W000',wallet_id_seq.NEXTVAL)
  INTO   :new.WALLET_ID
  FROM   dual;
END;
/
```

The sequence mentioned above automatically creates a Wallet ID attribute for each entry upon an insert in the WALLET table. The Brand ID attribute is expected to follow the format 'W000XX' where XX denotes the ID number. An example entry for the first customer is W0001.

4. Customer Activities ID sequence

```
CREATE SEQUENCE ca_id_seq START WITH 1
INCREMENT BY 1;
/
CREATE OR REPLACE TRIGGER ca_insert
BEFORE INSERT ON CUSTOMER_ACTIVITIES
FOR EACH ROW

BEGIN
  SELECT ca_id_seq.NEXTVAL
  INTO   :new.CA_ID
  FROM   dual;
END;
/
```

The sequence mentioned above automatically creates an entry in the CUSTOMER_ACTIVITIES table that keeps a log of all the transactions that takes place in the marketplace upon an insert in the CUSTOMER_ACTIVITIES table. It follows an integer format. Eg. The first entry will have C_ID = 1.

5. Reward ID sequence

```
CREATE SEQUENCE reward_id_seq START WITH 1
INCREMENT BY 1;
/
CREATE OR REPLACE TRIGGER reward_insert
BEFORE INSERT ON REWARD
```

```
FOR EACH ROW

BEGIN
  SELECT reward_id_seq.NEXTVAL
  INTO   :new.REWARD_ID
  FROM   dual;
END;
/
```

The sequence mentioned above creates an entry that increments the REWARD_ID attribute before each insert or update. The REWARD_ID is unique for each reward so it cannot remain the same for any two rewards. Thus, this sequence is utilized to automate the task. Eg. The first entry will have REWARD_ID = 1.

6. AddLoyaltyProgram sequence

```
CREATE SEQUENCE lp_regular_id_seq START WITH 1
INCREMENT BY 1;
/
CREATE SEQUENCE lp_tier_id_seq START WITH 1
INCREMENT BY 1;
/
CREATE PROCEDURE addLoyaltyProgram(bid VARCHAR, LPName VARCHAR, tier
VARCHAR) AS
BEGIN
    INSERT INTO LOYALTY_PROGRAM
  SELECT B_ID,
  case when tier='Regular' then concat('RLP0',lp_regular_id_seq.NEXTVAL) else
concat('TLP0',lp_tier_id_seq.NEXTVAL) end as LP_ID,
  LPName as LP_Name,
  'Invalid' as STATE
  FROM BRAND where B_ID=bid;
END;
/
```

The sequence mentioned above creates the LP_ID attribute for the LOYALTY_PROGRAM table before each insert. For a Brand that follows a Regular Loyalty program, the LP_ID follows the pattern RLPX, where X denotes the ID number. Eg. RLP1 for the first Regular Brand. For a tiered brand, it follows the pattern TLPX. For eg. TLP1 for the first Tiered Brand.

7. UpdateRERule sequence

```
CREATE PROCEDURE updateRERule(reRuleCode VARCHAR, activityType VARCHAR, points
NUMBER, bid VARCHAR) AS
BEGIN
INSERT INTO RE_RULE
SELECT B_ID,
    LP_ID,
   RE_RULE_CODE,
   RE_VERSION+1,
   points as POINTS,
   A_CODE
   FROM RE_RULE
   WHERE B_ID=bid and A_CODE=(select A_CODE from ACTIVITY_TYPE where
A_NAME=activityType) and RE_RULE_CODE=reRuleCode and
   RE_VERSION=(select max(RE_VERSION)from RE_RULE where B_ID=bid and
A_CODE=(select A_CODE from ACTIVITY_TYPE where A_NAME=activityType) and
RE_RULE_CODE=reRuleCode group by B_ID,A_CODE);
END;
/
```

The procedure adds a new record of an updated RE_RULE_CODE. If a RE_RULE has version 1, when we want to change the number of required points for a tier for a new version, this procedure allows you to do that. For example, ‘Brand X’ has a three-tier system with 0,100,200 as the required amount of points to enter the next tier with version 1. If we want to update version 2 to have a points requirement of 10,150,300, we can do so with this procedure.

8. UpdateRRRule sequence

```
CREATE PROCEDURE updateRRRule(rrRuleCode VARCHAR, rewardType VARCHAR, points
NUMBER, bid VARCHAR) AS
BEGIN
INSERT INTO RR_RULE
SELECT B_ID,
    LP_ID,
   RR_RULE_CODE,
   RR_VERSION+1,
   points as POINTS
   FROM RR_RULE
   WHERE B_ID=bid and RR_RULE_CODE=rrRuleCode and
   RR_VERSION=(select max(RR_VERSION)from RR_RULE where B_ID=bid and
RR_RULE_CODE=rrRuleCode group by B_ID,RR_RULE_CODE);
   UPDATE REWARD SET RR_VERSION=RR_VERSION+1 where  B_ID=bid and
RR_RULE_CODE=rrRuleCode;
```

```
END;
/
```

The procedure above performs the same function as mentioned in the RE_RULE table for the RR_RULE table.

9. addRERule sequence

```
CREATE PROCEDURE addRERule(reRuleCode VARCHAR, activityType VARCHAR, points
NUMBER, bid VARCHAR) AS
BEGIN
INSERT INTO RE_RULE
SELECT B_ID,
    LP_ID,
   reRuleCode as RE_RULE_CODE,
   1 as RE_VERSION,
   points as POINTS,
   (select A_CODE from ACTIVITY_TYPE where A_NAME=activityType) as A_CODE
   FROM LOYALTY_PROGRAM
   WHERE B_ID=bid;
END;
/
```

The procedure adds a new RE_RULE for a given brand and loyalty program.

10. addRRRule sequence

```
CREATE PROCEDURE addRRRule(rrRuleCode VARCHAR, rewardType VARCHAR, points
NUMBER, bid VARCHAR) AS
BEGIN
INSERT INTO RR_RULE
SELECT B_ID,
    LP_ID,
   rrRuleCode as RR_RULE_CODE,
   1 as RR_VERSION,
   points as POINTS
   FROM LOYALTY_PROGRAM
   WHERE B_ID=bid;
END;
/
```

The takeprocedure adds a new RR_RULE for a given brand and loyalty program.

RELATIONAL SCHEMA, FUNCTIONAL DEPENDENCIES AND CREATE STATEMENTS

1. **ADMIN(**ADMIN_ID, ADMIN_NAME, ADMIN_NUMBER)

ADMIN_ID -> ADMIN_NAME, ADMIN_NUMBER

The dependency holds true since each administrator is uniquely identified by an Admin_ID, which determines their phone number and name. Thus, the functional dependency is in **BCNF**.

```
create table ADMIN (
   ADMIN_ID varchar(15) primary key,
   ADMIN_NAME varchar(20),
   ADMIN_NUMBER number(10)
);
```

2. **ACTIVITY_TYPE**(A_CODE, A_NAME, ADMIN_ID)

A_CODE -> A_NAME
A_CODE -> ADMIN_ID

The relation is in BCNF since the super key A_CODE determines every other attribute in the table. The A_CODE determines the name of the activity mapped to it and also determines the administrator that adds the activity to the database(Since there cannot be duplicates of types of activities). Thus, the relation is in **BCNF**.

```
create table ACTIVITY_TYPE (
   A_CODE varchar(20) primary key,
   A_NAME varchar(50) not null,
   ADMIN_ID varchar(15),
   constraint fk_admin_id_act foreign key (ADMIN_ID) references ADMIN(ADMIN_ID) ON
DELETE SET NULL
);
```

3. **REWARD_TYPE(R_CODE, R_NAME, ADMIN_ID)**

R_CODE -> R_NAME
ADMIN_ID -> R_CODE
The relation is in BCNF since the super key R_CODE determines every other attribute in the table. The R_CODE determines the name of the reward mapped to it and also determines the administrator that adds the reward to the database(Since there cannot be duplicates of types of rewards). Thus, the relation is in **BCNF**.

```
create table REWARD_TYPE (
   R_CODE varchar(20) primary key,
```

```
    R_NAME varchar(50) not null,
    ADMIN_ID varchar(15),
    constraint fk_admin_id_rwd foreign key (ADMIN_ID) references ADMIN(ADMIN_ID) ON
DELETE SET NULL
);
```

4. **BRAND**(B_ID, B_NAME, JOIN_DATE, B_ADDRESS, ADMIN_ID)

B_ID ->B_NAME, JOIN_DATE, B_ADDRESS
B_ID -> ADMIN_ID

The dependency holds true since each brand is uniquely identified by a B_ID. Thus, the functional dependency is in **BCNF**,.i.e each relation has a super key on the right-hand side.

```
create table BRAND (
    B_ID varchar(15) primary key,
    B_NAME varchar(100) unique,
    JOIN_DATE date,
    B_ADDRESS varchar(200),
    ADMIN_ID varchar(15),
    constraint fk_admin_id_brand foreign key (ADMIN_ID) references ADMIN(ADMIN_ID) ON
DELETE SET NULL
);
```

5. **LOYALTY_PROGRAM**(B_ID, LP_ID, LP_NAME)

LP_ID ,B_ID -> LP_NAME

The dependency holds true since each LOYALTY_PROGRAM is uniquely identified by an LP_ID and B_ID. Also, each B_ID can have only one LP_ID associated with it. Thus, the functional dependency is in **BCNF**, i.e each relation has a super key on the right-hand side.

```
create table LOYALTY_PROGRAM (
    B_ID varchar(15),
    LP_ID varchar(15),
    LP_name varchar(100),
    STATE char(10),
    constraint fk_b_id_lp foreign key (B_ID) references BRAND(B_ID) ON DELETE CASCADE,
    constraint pk_lp primary key (B_ID, LP_ID)
);
```

6. **TIER**(B_ID, LP_ID, TIER_ID, TIER_NAME, POINTS_THRESHOLD, MULTIPLIER)

B_ID, LP_ID,TIER_ID -> POINTS, THRESHOLD, MULTIPLIER

The dependency holds true since each TIER is uniquely identified by a superkey consisting of LP_ID, B_ID, TIER_ID. Thus, the functional dependency is in BCNF.

```
create table TIER (
   B_ID varchar(15),
   LP_ID varchar(15),
   TIER_ID number(1),
   TIER_NAME varchar(50) not null,
   POINTS_THRESHOLD number(5) default '0',
   MULTIPLIER number(5) default '1',
   constraint fk_lp_tier foreign key (B_ID,LP_ID) references LOYALTY_PROGRAM(B_ID,LP_ID)
ON DELETE CASCADE,
   constraint pk_tier primary key (B_ID, LP_ID, TIER_ID),
   constraint check_tier check(TIER_ID<4) -- NEED TO ENFORCE CONSTRAINT (TIER_ID<4)
);
```

7. **ACTIVITY**(B_ID, LP_ID, A_CODE)

```
create table ACTIVITY (
   B_ID varchar(15),
    LP_ID varchar(15),
   --ACTIVITY_ID varchar(10),
   A_CODE varchar(20) not null,
     constraint fk_activity foreign key (B_ID,LP_ID) references
LOYALTY_PROGRAM(B_ID,LP_ID) ON DELETE CASCADE,
   constraint fk_a_code_act foreign key (A_CODE) references ACTIVITY_TYPE(A_CODE) ON
DELETE CASCADE,
   constraint pk_act primary key (B_ID,LP_ID, A_CODE)
);
```

8. **RE_RULE**(B_ID, LP_ID, RE_RULE_CODE, RE_VERSION, POINTS, A_CODE)

B_ID, LP_ID, RE_RULE_CODE, RE_VERSION, A_CODE -> POINTS

For a given BRAND_ID, LP_ID, RE_RULE_CODE, RE_VERSION and A_CODE, we can determine the number of points a customer can earn performing a given activity. This is the way we can map an activity to a given RE_RULE.

Since B_ID, LP_ID, RE_RULE_CODE, RE_VERSION, A_CODE is a superkey for the given relation, it follows **BCNF**.

```
create table RE_RULE (
   B_ID varchar(15),
   LP_ID varchar(15),
   RE_RULE_CODE varchar(10), --not required
   RE_VERSION number(5) default '1',
   POINTS number(5),
   A_CODE varchar(20) not null,
   constraint fk_a_code_re foreign key (B_ID,LP_ID,A_CODE) references
ACTIVITY(B_ID,LP_ID,A_CODE) ON DELETE CASCADE,
   constraint pk_re primary key (B_ID, LP_ID, RE_RULE_CODE, RE_VERSION)
);
```

9. **RR_RULE**(B_ID, LP_ID, RR_RULE_CODE, RR_VERSION, POINTS, REWARD_ID)

B_ID, LP_ID, RR_RULE_CODE, RR_VERSION, REWARD_ID -> POINTS

For a given BRAND_ID, LP_ID, RR_RULE_CODE, REWARD_ID, and RR_VERSION, we can determine the number of points a customer can use to redeem a reward. This is the way we can map a reward to a given RR_RULE.

Since B_ID, LP_ID, RR_RULE_CODE, and RR_VERSION are a superkey for the given relation, it follows **BCNF**.

```
create table RR_RULE (
   B_ID varchar(15),
   REWARD_ID varchar(10),
   LP_ID varchar(15),
   RR_RULE_CODE varchar(10),
   RR_VERSION number(5) not null,
   POINTS number(5),
   --constraint fk_lp_rr foreign key (B_ID,LP_ID) references LOYALTY_PROGRAM(B_ID,LP_ID)
ON DELETE CASCADE,
   constraint fk_lp_rr foreign key (B_ID,LP_ID,REWARD_ID) references
REWARD(B_ID,LP_ID,REWARD_ID) ON DELETE CASCADE,
   constraint pk_rr primary key (B_ID, LP_ID, RR_RULE_CODE, RR_VERSION)
);
```

10. **REWARD**(B_ID, LP_ID, REWARD_ID, GIFT_CARD_AMT, FREE_PROD_NAME, GIFT_CARD_EXPIRY_DATE, R_CODE, NO_OF_REWARDS)

B_ID, LP_ID, R_CODE, REWARD_ID -> GIFT_CARD_AMT, NO_OF_REWARDS, FREE_PROD_NAME

Assumption: We ignore the attribute GIFT_CARD_EXPIRY_DATE since the attribute has become redundant with respect to the project requirements and sample data.

The combination of attributes B_ID, LP_ID, R_CODE, REWARD_ID determines the value of the GIFT_CARD_AMT, NO_OF_REWARDS, and FREE_PROD_NAME.

The attributes GIFT_CARD_AMT, FREE_PROD_NAME can take null values. For example, if the REWARD_ID attribute corresponds to a gift card, then the FREE_PROD_NAME attribute will be NULL and vice-versa.

```
create table REWARD (
   B_ID varchar(15),
   LP_ID varchar(15),
   REWARD_ID varchar(10),
   GIFT_CARD_AMT number(5),
   FREE_PROD_NAME varchar(50),
   GIFT_CARD_EXPIRY_DATE date,
   R_CODE varchar(20) not null,
        NO_OF_REWARDS number(5),
   constraint fk_rwd foreign key (B_ID,LP_ID) references LOYALTY_PROGRAM(B_ID,LP_ID)
ON DELETE CASCADE,
   constraint fk_r_code_rwd foreign key (R_CODE) references REWARD_TYPE(R_CODE) ON
DELETE CASCADE,
   --constraint fk_rr_rule foreign key (B_ID,LP_ID,RR_RULE_CODE,RR_VERSION) references
RR_RULE(B_ID,LP_ID,RR_RULE_CODE,RR_VERSION) ON DELETE CASCADE,
   constraint pk_rwd primary key (B_ID,LP_ID, REWARD_ID)
);
```

11. **CUSTOMER**(<u>CUST_ID</u>, CUST_NAME, C_ADDRESS, C_PH_NO, ADMIN_ID, WALLLET_ID)

CUST_ID -> CUST_NAME, C_ADDRESS, C_PH_NO, WALLET_ID
CUST_ID -> ADMIN_ID

The CUST_ID can uniquely identify each customer in the database. Thus, we can obtain the attributes CUST_NAME, C_ADDRESS, C_PH_NO, WALLET_ID when we have the CUST_ID. CUST_ID also determines the administrator that adds the customer to the database.

```
create table CUSTOMER (
   CUST_ID varchar(15) ,
   CUST_NAME varchar(100) not null,
   C_ADDRESS varchar(200),
   C_PH_NO number(10) not null,
```

```
    ADMIN_ID varchar(15),
    WALLET_ID varchar(10),
    constraint fk_admin_id_cust foreign key (ADMIN_ID) references ADMIN(ADMIN_ID) ON
DELETE SET NULL,
    constraint pk_cust primary key (WALLET_ID,CUST_ID)

);
```

12. **WALLET**(WALLET_ID, CUST_ID, B_ID, LP_ID, CUST_TIER_STATUS, E_POINTS, R-POINTS)

WALLET_ID, CUST_ID, B_ID, LP_ID, CUST_TIER_STATUS -> E_POINTS, R_POINTS

Given a customer's WALLET_ID,, CUST_ID, B_ID, LP_ID, and CUST_TIER_STATUS we can calculate the number of points that a customer has earned and the number of points a customer has redeemed. Although CUST_TIER_STATUS changes when the customer spends or earns points, it is mainly dependent on the threshold set by the brand for a said tier and loyalty program. The CUST_TIER_STATUS update is also handled in the application logic so the Functional Dependency does not hold. Thus, the relation is in **BCNF** since all the elements on the left-hand side are superkeys.

```
create table WALLET (
    WALLET_ID varchar(10) ,
    CUST_ID varchar(15) not null,
    B_ID varchar(15) ,
    LP_ID varchar(15) ,
    CUST_TIER_STATUS number(1) default '0',
    E_POINTS number(10) default '0',
    R_POINTS number(10) default '0',
    constraint fk_cust_id_wallet foreign key (WALLET_ID,CUST_ID) references
CUSTOMER(WALLET_ID,CUST_ID) ON DELETE CASCADE,
   -- constraint fk_cust_wallet foreign key (B_ID,LP_ID) references
LOYALTY_PROGRAM(B_ID,LP_ID),
    constraint fk_tier_wallet foreign key (B_ID,LP_ID,CUST_TIER_STATUS) references
TIER(B_ID,LP_ID,TIER_ID) ON DELETE CASCADE,
    constraint pk_wallet primary key (WALLET_ID, LP_ID,B_ID)
);
```

13. **CUSTOMER_ACTIVITIES**(CA_ID, WALLET_ID, B_ID, LP_ID, RE_RULE_CODE, RE_VERSION, RE_RULE_CODE, RR_VERSION, REVIEW, GIFT_AMT, GIFT_CARD_USED, PROD_NAME, ACTIVITY_DATE, PURCHASE_AMT)

CA_ID, WALLET_ID, B_ID, LP_ID, RE_RULE_CODE, RE_VERSION, RE_RULE_CODE, RR_VERSION -> REVIEW, GIFT_AMT,GIFT_CARD_USED, PROD_NAME, ACTIVITY_DATE, PURCHASE AMOUNT

The combination of CA_ID, WALLET_ID, B_ID, LP_ID, RE_RULE_CODE, RE_VERSION, RE_RULE_CODE and RR_VERSION determines REVIEW, GIFT_AMT,GIFT_CARD_USED, PROD_NAME, ACTIVITY_DATE, PURCHASE AMOUNT. The customer activities table keeps a log of all the transactions that take place in the marketplace application. This is why a large combination of attributes determines the details concerning a transaction log. If the transaction involved a reward, then the fields that concern an activity are NULL and vice-versa if the transaction involves an activity. Thus, the relation is in **BCNF** since all the elements on the left-hand side are superkeys.

```
create table CUSTOMER_ACTIVITIES (
   CA_ID varchar(20) primary key,
   WALLET_ID varchar(10) not null,
   B_ID varchar(15) not null,
   LP_ID varchar(15) not null,
   RE_RULE_CODE varchar(10),
   RE_VERSION number(5),
   RR_RULE_CODE varchar(10),
   RR_VERSION number(5),
   REVIEW varchar(500),
   GIFT_AMT number(5),
   GIFT_CARD_USED number(1), -- 0 if not used. 1 if used
   PROD_NAME varchar(50),
   ACTIVITY_DATE date not null,
   PURCHASE_AMT number(5),
   constraint fk_wallet_ca foreign key (WALLET_ID,LP_ID,B_ID) references
WALLET(WALLET_ID,LP_ID,B_ID) ON DELETE CASCADE,
   constraint fk_rr_ca foreign key (B_ID, LP_ID, RR_RULE_CODE, RR_VERSION) references
RR_RULE(B_ID, LP_ID, RR_RULE_CODE, RR_VERSION) ON DELETE CASCADE,
   constraint fk_re_ca foreign key (B_ID, LP_ID, RE_RULE_CODE, RE_VERSION) references
RE_RULE(B_ID, LP_ID, RE_RULE_CODE, RE_VERSION) ON DELETE CASCADE
);
```

14. **CREDENTIALS**(USERNAME, U_PASSWORD, CUST_ID, WALLET_ID, B_ID, ADMIN_ID)

USERNAME, U_PASSWORD -> CUST_ID,WALLET_ID, B_ID, ADMIN_ID

The username and password determine the WALLET_ID, B_ID, and ADMIN_ID. The credentials of a person operating a BRAND account will have an empty CUST_ID and WALLET_ID field and vice-versa for a person operating a customer account. Each credential

instance can only be added by a single admin, hence USERNAME and U_PASSWORD determine the ADMIN_ID as well.

```
create table CREDENTIALS(
USERNAME varchar(50),
U_PASSWORD varchar(15) default 'abcd1234',
CUST_ID varchar(15),
WALLET_ID varchar(15),
B_ID varchar(15),
ADMIN_ID varchar(15),
constraint fk_cust_id_cred foreign key (CUST_ID, WALLET_ID) references
CUSTOMER(CUST_ID, WALLET_ID) ON DELETE CASCADE,
constraint fk_admin_id_cred foreign key (ADMIN_ID) references ADMIN(ADMIN_ID) ON
DELETE CASCADE,
constraint fk_brand_id_cred foreign key (B_ID) references BRAND(B_ID) ON DELETE
CASCADE,
constraint pk_cred primary key (USERNAME,U_PASSWORD)
);
```

SHOW QUERIES

1. List all customers that are not part of Brand02's program.

```
SELECT CUST_ID FROM CUSTOMER WHERE CUST_ID NOT IN ( SELECT DISTINCT
C.CUST_ID FROM CUSTOMER C, WALLET W
WHERE C.WALLET_ID=W.WALLET_ID AND W.B_ID  IN ('BRAND02'));
```

Output:

```
CUST_ID
---------------
C0004
C0002
```

2. List customers that have joined a loyalty program but have not participated in any activity in that program (list the customerid and the loyalty program id).

```
SELECT W.CUST_ID, W.LP_ID FROM WALLET W
WHERE NOT EXISTS(
        SELECT * FROM  CUSTOMER_ACTIVITIES CA
        WHERE W.WALLET_ID=CA.WALLET_ID AND W.LP_ID=CA.LP_ID);
```

Output:

```
no rows selected
```

3. List the rewards that are part of Brand01 loyalty program.

SELECT * FROM REWARD R WHERE B_ID = 'Brand01';

Output:

B_ID	LP_ID	REWARD_ID	GIFT_CARD_AMT	FREE_PROD_NAME	GIFT_CARD_EXPIRY_DATE	R_CODE	NO_OF_REWARDS
Brand01	TLP01	R1	50			R01	40
Brand01	TLP01	R2		FP1		R02	25

4. List all the loyalty programs that include “refer a friend” as an activity in at least one of their reward rules.

SELECT RE_RULE.LP_ID
FROM RE_RULE, ACTIVITY_TYPE
WHERE ACTIVITY_TYPE.A_NAME = 'Refer a friend' AND ACTIVITY_TYPE.A_CODE = RE_RULE.A_CODE;

Output:

LP_ID
RLP01
TLP02

5. For Brand01, list for each activity type in their loyalty program, the number of instances that have occurred.

SELECT RE.A_CODE, COUNT(*) AS CT
FROM RE_RULE RE, CUSTOMER_ACTIVITIES CA
WHERE RE.B_ID='Brand01' AND CA.B_ID=RE.B_ID AND
CA.RE_RULE_CODE=RE.RE_RULE_CODE AND CA.RE_VERSION=RE.RE_VERSION
GROUP BY RE.A_CODE;

Output:

A_CODE	COUNT(*)
A02	11
A01	14

6. List customers of Brand01 that have redeemed at least twice.

SELECT WALLET_ID, COUNT(*) AS CT
FROM CUSTOMER_ACTIVITIES WHERE RR_RULE_CODE <> 'null' AND B_ID='Brand01'
GROUP BY WALLET_ID HAVING COUNT(*) >=2 ;

Output:

```
WALLET_ID    COUNT(*)
---------- ----------
W0005               2
```

7. All brands where the total number of points redeemed overall is less than 500 points

SELECT B_ID,SUM(R_POINTS) AS PTS FROM WALLET GROUP BY B_ID HAVING SUM(R_POINTS) < 500;

Output:

```
B_ID            SUM(R_POINTS)
--------------- -------------
Brand01                   300
Brand03                     0
```

8. For Customer C0003, and Brand02, the number of activities they have done in the period of 08/1/2021 and 9/30/2021.

SELECT COUNT(*) AS CT FROM CUSTOMER_ACTIVITIES
WHERE B_ID = 'Brand02' AND RE_RULE_CODE<>'NULL' AND ACTIVITY_DATE>=DATE'2021-08-01' AND ACTIVITY_DATE<=DATE'2021-09-30'
   AND WALLET_ID IN (SELECT WALLET_ID FROM WALLET WHERE CUST_ID = 'C0003');

Output:

```
  COUNT(*)
----------
         4
```

TECHNOLOGY STACK

Database: OracleSQL
Language to implement backend logic: Java
Frameworks: JDBC