CS6360.001 Project: Target 4

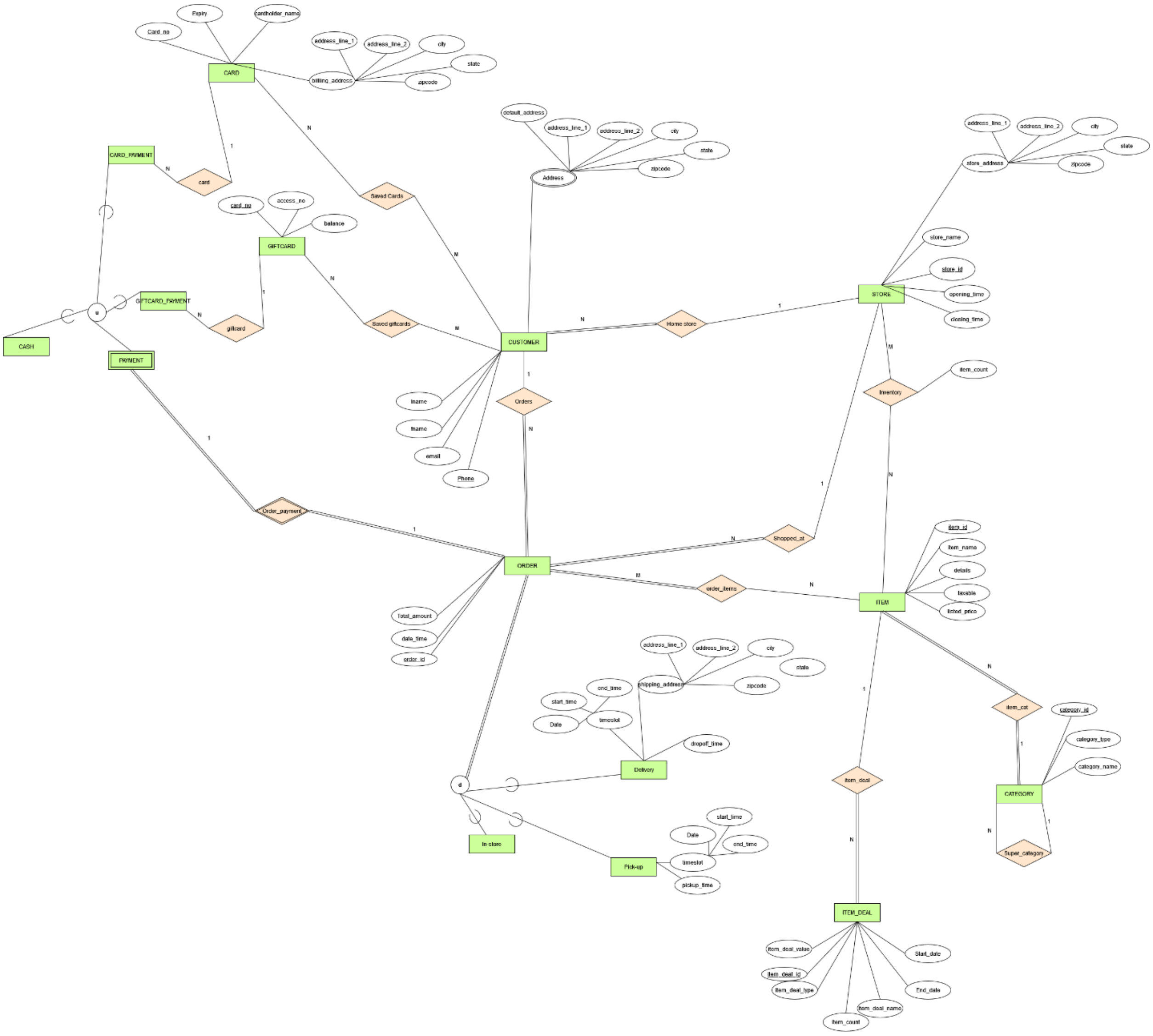
Team 9: Nikhil Rongala (NXR200017)

Goal: To design a database system for Target.

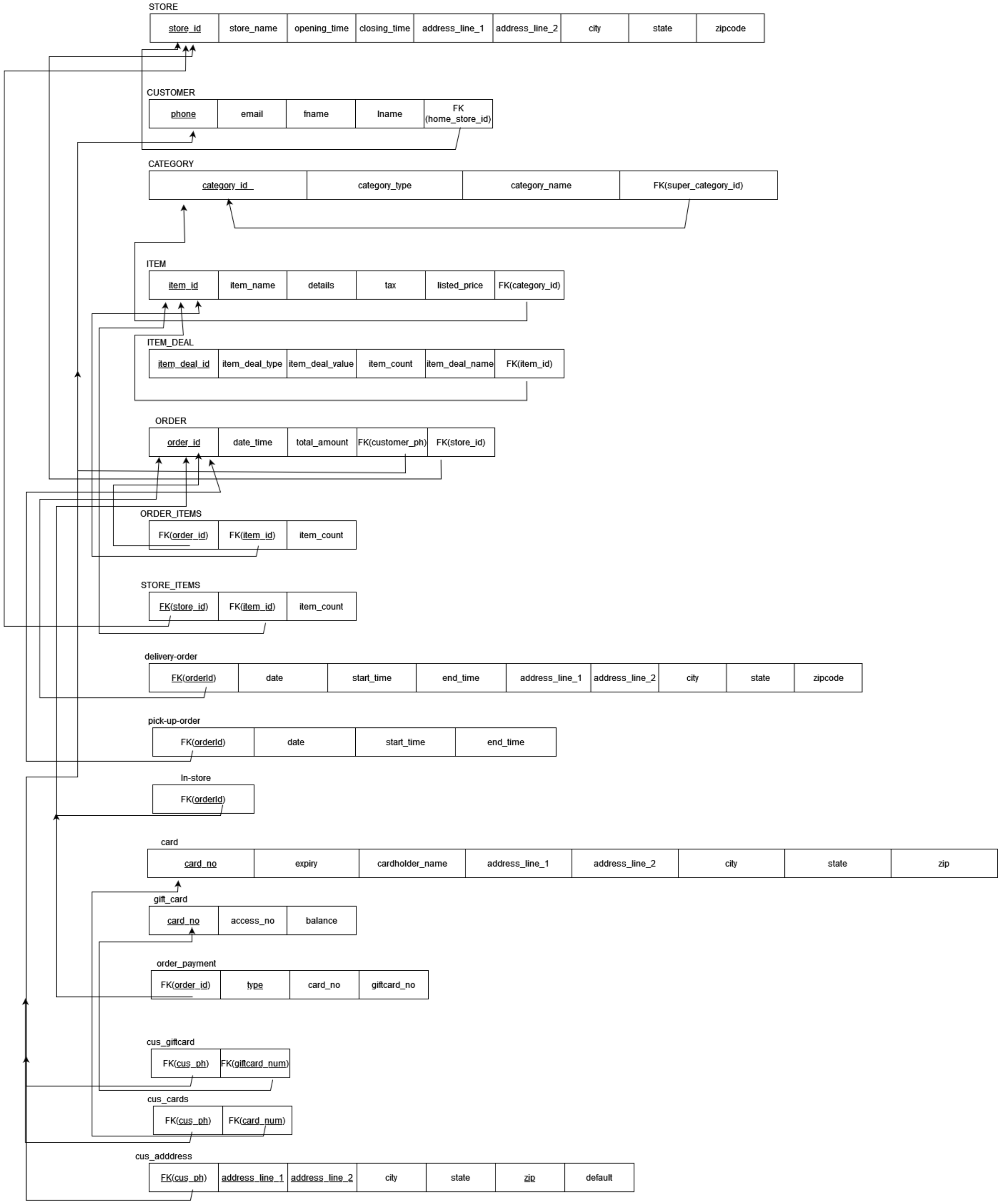
Data requirements:

1. The system has a collection of STORE(s). Each store has a store_id, store_name , address, opening and closing times.
2. Every store has an inventory of ITEMs. Each item has an item_id, item_name, details, tax on the item, and the listed price. The items are also grouped into categories. There is a hierarchy of categories. Each item belongs to only one base category.
3. There is a collection of DEAL(s). Many deals may be mapped to one item, but at any point of time, only one deal must be active for the item. Deals may be percent based or absolute value based.
4. The system should also store the collection of CUSTOMER(s). Each customer has a name, address(es), email and phone. Phone no. is unique for a customer account. Each customer also has a home store, saved cards and giftcards.
5. A customer can make ORDER(s) at any store and pay using any type of payment – card, giftcard or cash or any combination of the three. An order must have an orderId, time of purchase, the total amount, list of items and type of order- in-store, pickup or delivery. If it’s a pickup or delivery, the time slot allotted for the pickup or drop-off must be stored.
6. The system should support storing data pertaining to CARDs such as expiry date, billing address, cardholder name.
7. The system should store for each GIFTCARDs, a card_no, access_no, and remaining balance in the card.

ER diagram:



Relational Schema:



STORE

store_id	store_name	opening_time	closing_time	address_line_1	address_line_2	city	state	zipcode

CUSTOMER

phone	email	fname	lname	FK (home_store_id)

CATEGORY

category_id_	category_type	category_name	FK(super_category_id)

ITEM

item_id	item_name	details	tax	listed_price	FK(category_id)

ITEM_DEAL

item_deal_id	item_deal_type	item_deal_value	item_count	item_deal_name	FK(item_id)

ORDER

order_id	date_time	total_amount	FK(customer_ph)	FK(store_id)

ORDER_ITEMS

FK(order_id)	FK(item_id)	item_count

STORE_ITEMS

FK(store_id)	FK(item_id)	item_count

delivery-order

FK(orderId)	date	start_time	end_time	address_line_1	address_line_2	city	state	zipcode

pick-up-order

FK(orderId)	date	start_time	end_time

In-store

FK(orderId)

card

card_no	expiry	cardholder_name	address_line_1	address_line_2	city	state	zip

gift_card

card_no	access_no	balance

order_payment

FK(order_id)	type	card_no	giftcard_no

cus_giftcard

FK(cus_ph)	FK(giftcard_num)

cus_cards

FK(cus_ph)	FK(card_num)

cus_adddress

FK(cus_ph)	address_line_1	address_line_2	city	state	zip	default

Normalization:

Functional dependencies:

- STORE:
    - Store_id -> store_name, opening_time, closing_time, address_line_1, address_line_2, city, state, zipcode
    - Zipcode->city, state
- CUSTOMER:
    - Phone-> email, fname, lname, home_store_id
- CATEGORY:
    - Category_id-> category_type, category_name, super_category_id
    - Category_name -> super_category_id
- ITEM:
    - Item_id -> item_name, details, tax, listed_price, category_id
- ITEM_DEAL:
    - Item_deal_id -> item_deal_type, item_deal_value, item_count, item_deal_name, item_id
- ORDER:
    - Order_id ->date_time, total_amount, customer_ph, store_id
- ORDER_ITEMS:
    - Order_id -> item_id, item_count
- STORE_ITEMS:
    - DStore_id -> item_id, item_count
- DELIVERY_ORDER:
    - orderId-> date, start_time, end_time, address_line_1, address_line_2, city, state, zip
    - zip->city, state
- PICKUP-ORDER:
    - orderId-> date, start_time, end_time
- CARD:
    - card_no->expiry, cardholder_name, address_line_1, address_line_2, city, state, zip
    - zip->city, state
- GIFTCARD:
    - card_no -> access_no, balance
- ORDER_PAYMENT:
    - order_id-> type, card_no, giftcard_no
- CUS_ADDRESES:
    - zip->city, state

1NF: Since the relational schema is created from ER diagram using the transformation rules, its already in 1NF i.e. all attributes depend on the key.

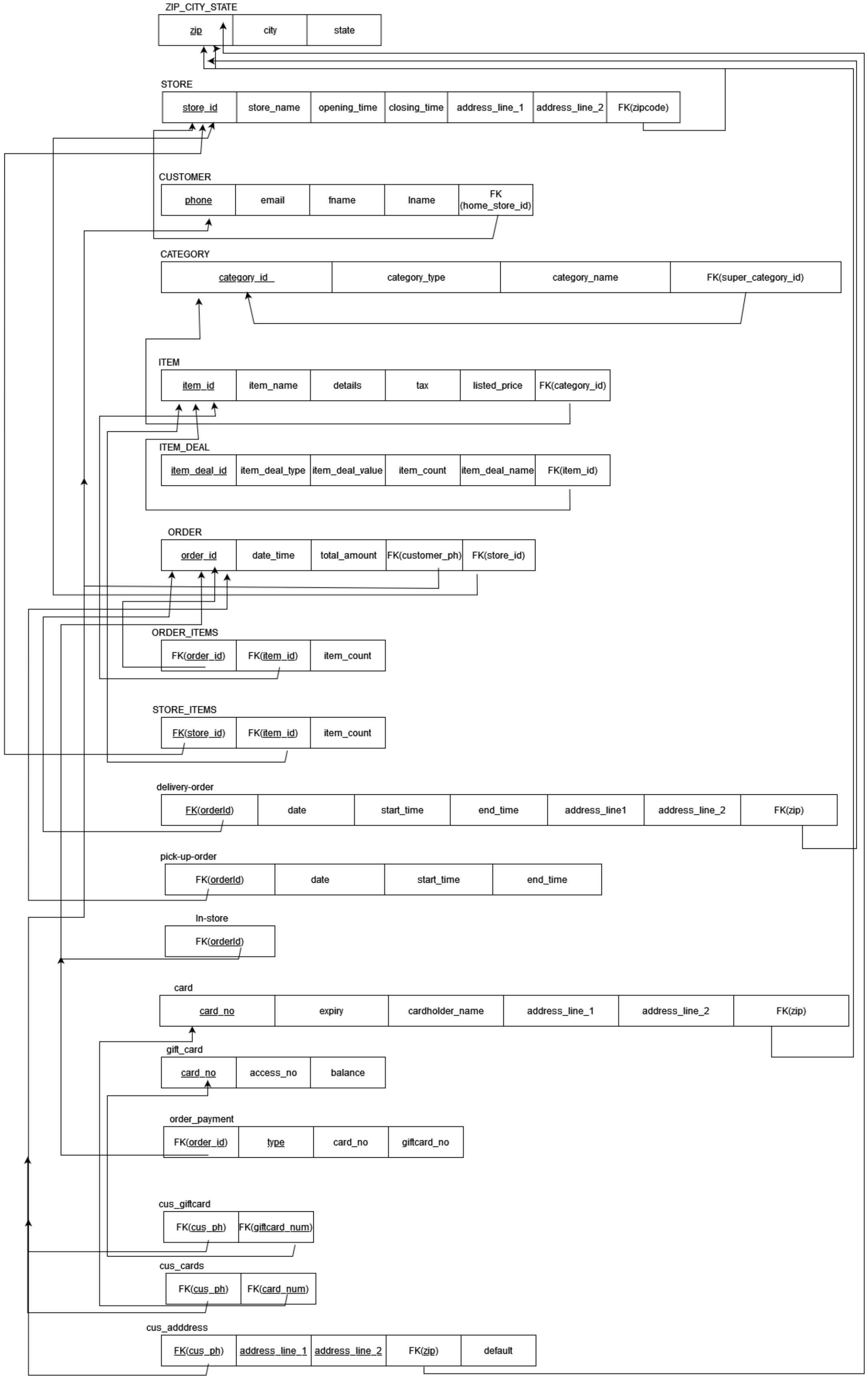
2NF: Partial dependencies: The only partial dependency is

zip-> city, state

in CUS_ADDRESSES table. This can be normalized by setting up a new table called ZIP_CITY_STATE with zip as PK.

3NF: zip-> city, state in tables CARD, STORE, DELIVERY-ORDER violate 3NF as neither zip is a superkey nor are city and state prime attributes. These tables can be split and the ZIP_CITY_STATE table created before can be used to normalize into 3NF as well.

Final relational schema:



ZIP_CITY_STATE

zip	city	state

STORE

store_id	store_name	opening_time	closing_time	address_line_1	address_line_2	FK(zipcode)

CUSTOMER

phone	email	fname	lname	FK (home_store_id)

CATEGORY

category_id_	category_type	category_name	FK(super_category_id)

ITEM

item_id	item_name	details	tax	listed_price	FK(category_id)

ITEM_DEAL

item_deal_id	item_deal_type	item_deal_value	item_count	item_deal_name	FK(item_id)

ORDER

order_id	date_time	total_amount	FK(customer_ph)	FK(store_id)

ORDER_ITEMS

FK(order_id)	FK(item_id)	item_count

STORE_ITEMS

FK(store_id)	FK(item_id)	item_count

delivery-order

FK(orderId)	date	start_time	end_time	address_line1	address_line_2	FK(zip)

pick-up-order

FK(orderId)	date	start_time	end_time

In-store

FK(orderId)

card

card_no	expiry	cardholder_name	address_line_1	address_line_2	FK(zip)

gift_card

card_no	access_no	balance

order_payment

FK(order_id)	type	card_no	giftcard_no

cus_giftcard

FK(cus_ph)	FK(giftcard_num)

cus_cards

FK(cus_ph)	FK(card_num)

cus_adddress

FK(cus_ph)	address_line_1	address_line_2	FK(zip)	default

SQL code to create tables, foreign key constraints:

See attached files.

Triggers:

1. Check_order_anomaly: When a user adds an item to an order, this trigger checks if the associated store has sufficient inventory for that item to process the order.
2. Check_item_deal_validity: When an item deal is inserted or its validity dates are changed, this trigger checks if the new deal is in conflict with any other deal since our requirement is only one promotion/deal at any point of time.

Stored Procedures:

1. getItemPrice: This procedure get the price of an item after applying promotions and tax based on the item count.
2. setOrderAmount: This procedure call the above procedure for each item in the order and computes the final order amount.