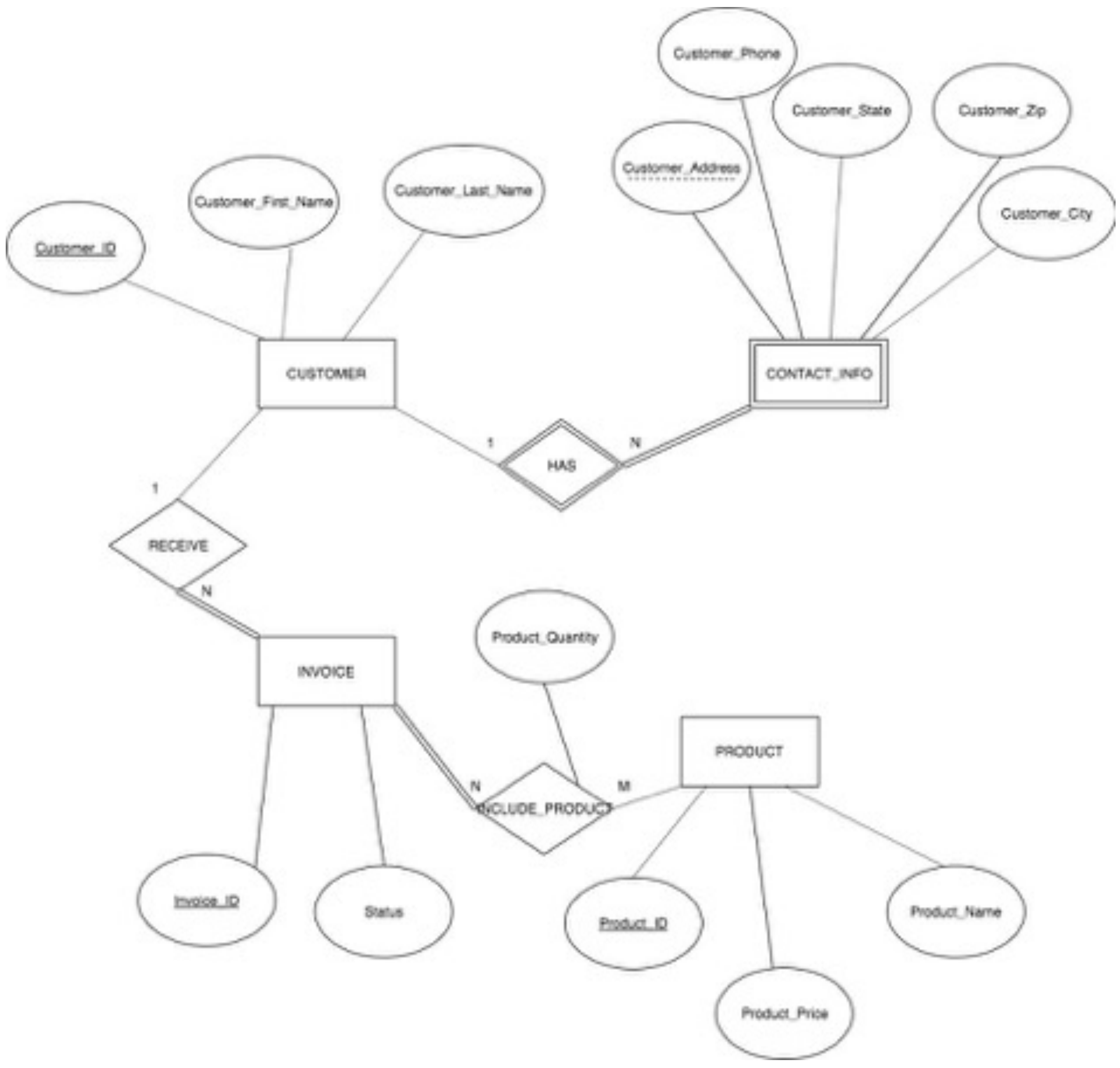
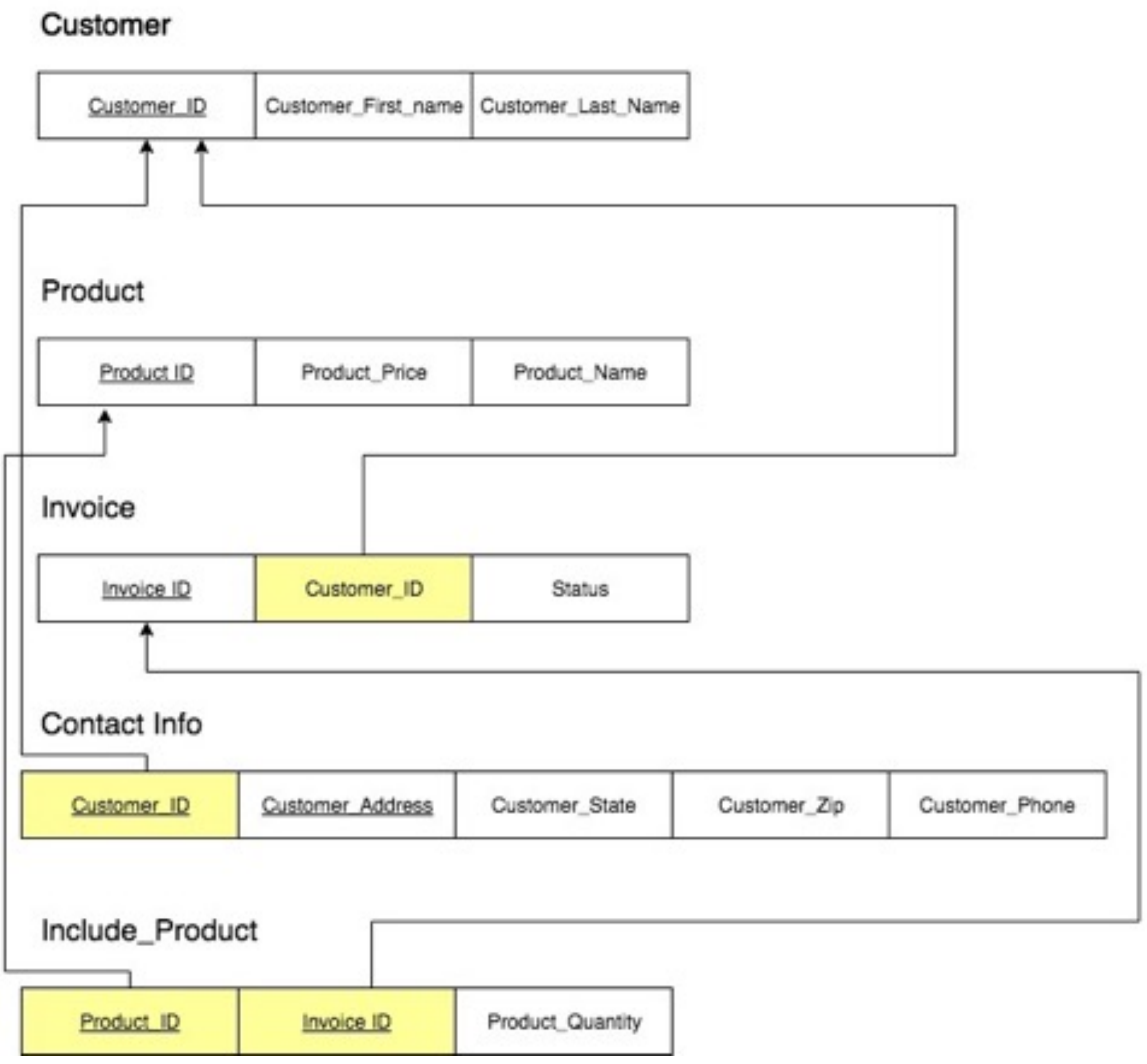
Name of project: Smart Invoice
Team ID: 32
Team member ID: YIWEN CAI(30648449), YINGTAO LIU(29988259), YUJIE WANG(30017429)
Team Member Username: ycai21/yliu157/ywang240

Yes there is a weak relation/entity, namely HAS/CONTACT_INFO.
CONTACT_INFO stores a customer's address, yet itself does not contain a key that can fully identify the row. There are chances for repeated address, city, zip, and phone to occur. It must, therefore, use the primary key of CUSTOMER, Customer_ID, combining with the partial key Customer_Address to identify a specific row.
For relationship HAS, we have entity CUSTOMER and CONTACT_INFO. We assume that a consumer can create a account without entering the address info, therefore customer has a partial participation. Since the consumer can have a minimum of 0 address and max of infinite address in theory, then it will have a (min, max) = (0,n). The (min, max) value of CONTACT_INFO is (1,1) and it has a total participation since for each e in E, we must be able to find one and only one customer. Therefor the cardinality for HAS is 1:N
For relationship RECEIVE. we have entity CUSTOMER and INVOICE. We assume that one customer can create an account with no purchase or multiple purchases. Therefore, the (min, max) value of customer should be (0,n) and CUSTOMER should have a partial participation. For INVOICE, there could be multiple invoice under the same person and every invoice must correspond to one customer. Therefore it has a (min, max) value of (1,n) and total participation. The cardinality for RECEIVE is 1:N.
For relation INCLUDE_PRODUCT, we have entities INVOICE and Product. we assume that every invoice contain arbitrary kind of products. One product can be in N invoice and one invoice can have m product. Therefore product has a (min, max) value of (0,n) and partial participation; invoice has a (min, max) value of (1,n) and total participation; the cardinality for RECEIVE is N:M.

Step1: Create relations CUSTOMER, INVOICE, PRODUCT correspond to regular entity types customer, invoice, products. Choose Customer_ID, Invoice_ID, and Product_ID as primary keys for the relations CUSTOMER, INVOICE, PRODUCT, respectively.
Step2: create the relation CONTACT_INFO in this step to correspond to the weak entity type contact_info. We include the primary key Customer_ID of the CUSTOMER relation—which corresponds to the owner entity type—as a foreign key attribute of CONTACT_INFO.The primary key of the CONTACT_INFO relation is the combination {Customer_Address, Customer_ID}, because Customer_Address is the partial key of CONTACT_INFO.
Step3: There is no one to one relation here.
Step4: we now map the 1:N relationship types HAS and RECEIVE. Those two are 1:N relationship because one customer can have multiple contact info stored yet

not one contact info can be inferred to the same customer, and one customer can have multiple invoice while one invoice can only be created for one buyer. For HAS we include the primary key Customer_ID of the CUSTOMER relation as foreign key in the CONTACT_INFO relation and call it Customer_ID. For RECEIVE we include the primary key of the CUSTOMER relation as foreign key in the INVOICE relation and call it Customer_ID.
Step5: We map the M:N relationship type INCLUDE_PRODUCT by
creating the relation INCLUDE_PRODUCT. We include the primary keys of the INVOICE and PRODUCT relations as foreign keys in INCLUDE_PRODUCT and call them Invoice_ID and Product_ID, respectively. We also add the attribute of INCLUDE_PRODUCT. The primary key of the INCLUDE_PRODUCT relation is the combination of the foreign key attributes {Invoice_ID, Product_ID}.
Step6: There is no multivalued attributes
Step7: There is no N-array relationship
Step8: There is no need for specialization or generalization
Step9: There is no category.





Relation Name	ER diagram components
CUSTOMER	E(CUSTOMER)
PRODUCT	E(PRODUCT)
INVOICE	E(INVOICE) + R(RECEIVE)
CONTACT_INFO	E(CONTACT_INFO) +R(HAS)
INCLUDE_PRODUCT	R(INCLUDE_PRODUCT)

For 2: We answer the question in a mixed manner because we find this way most efficient. foreign keys are indicated by yellow and primary keys are underscored. The schema is provided above.

For CUSTOMER:
Customer_ID is a unique number that assign to each customer. It has a data type of integer. Start with 0, the value of the Customer_ID will be increased by 1 each time we add a new customer. This servers as the primary key of the relation and can be used to identify each customer. We choose this attribute to serve as the primary key since it guarantees uniques.
Customer_First_Name: This is a string value that stores the customer's first name. It has no default value and can not be set to null since it is mandatory for customer to enter a first name.
Customer_Last_Name: This is a string value that stores the customer's last name. It has no default value and can not be set to null since it is mandatory for customer to enter a last name.

For CONTACT_INFO:
We separate contact information from CUSTOMER because we assume that Invoice does not need to have contact information in it. The contact info is for shipping and customer service purpose. Therefore, separating the contact info will improve the operating speed and space used.
The set of customer ID and Customer_address will be the primary key. Since one customer could have multiple address, we can not use customer ID along as the primary key. State, Zip will have a larger chance of repetition. Therefore we choose the set of those two attribute to be primary key. Customer ID is also a foreign key. If for some reason the corresponding tuple is deleted, then we should delete all the row in row that contain the some Customer ID in the primary key set.
Custom_Address is a string value that stores the address of the customer. It has no default value and can be set to null.
Custom_State is a string value that stores the state of the customer's home. It has no default value and can be set to null.
Custom_Zip is a string value that stores the zip code of the customer's home. It has no default value and can be set to null.
Custom_Phone is a integer value that stores the phone number of the customer. It has no default value and can be set to null.

For PRODUCT:
Product_ID is a unique number that assign to each product. It has a data type of integer. Start with 0, the value of the Product_ID will be increased by 1 each time we add a new product. This servers as the primary key of the relation and can be used to identify each customer. We choose this attribute to serve as the primary key since it guarantees uniques.

Product_Price is a floating number that record the price of the product. It has no default value and can not be set to null since it is mandatory to enter a price when add new product.
Product_Name is a string that record the name of the product. It has no default value and can not be set to null since it is mandatory to enter a name when add new product.

For INVOICE:
Invoice_ID is a unique number that assign to each product. It has a data type of integer. Start with 0, the value of the Invoice_ID will be increased by 1 each time we add a new product. This servers as the primary key of the relation and can be used to identify each customer. We choose this attribute to serve as the primary key since it guarantees uniques.
Customer_ID is a foreign key which refers to the Custom relation. It indicates who place the order. If the customer_ID in the referred relation is deleted, then we should delete all the raw containing this customer_ID. However, for the sake of record keeping, we should save the invoice fully in some way, for example, save it to local drive before deleting.
Status is string which has a domain with (normal, flagged). This attribute offers the specialist to quickly find all the problematic invoice and resolve the problem with efficiency instead of having to allocate each by Invoice_ID. It has a default value of "normal".

For INCLUDE_PRODUCT
It is a junction table, meaning it purpose is to connect Product_ID and Invoice_ID. Since one invoice can have arbitrary kinds of products with arbitrary amount. We use the set {Product_ID, Invoice_ID} as a primary key. Product_ID and Invoice_ID are also foreign keys. If the either element in the set is deleted in the referred relation, then we should delete all the raw containing this primary key. However, for the sake of record keeping, we should save the invoice fully in some way, for example, save it to local drive before deleting.

Product_Quality is a integer. We used this value to record the quantity of a particular product the customer ordered. It has no default value and can not be set to null since it is mandatory to enter the amount when placing an order.