Experiment 2

Aim: Mapping ER/EER to Relational schema model.

Hardware and Software Requirement: P-IV and above, Oracle

Theory:

The set of allowed values for each attribute is called the domain of the attribute

Attribute values are (normally) required to be atomic; that is, indivisible

The special value *null* is a member of every domain. Indicated that the value is "unknown"

The null value causes complications in the definition of many operations

$A_1, A_2, \ldots, A_n$ are *attributes*

$R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

Example:

instructor = (*ID, name, dept_name, salary*)

Formally, given sets D_1, D_2, …. D_n a relation r is a subset of $D_1 \times D_2 \times \ldots \times D_n$

Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$

The current values (relation instance) of a relation are specified by a table

An element t of r is a *tuple*, represented by a *row* in a table

Primary keys allow entity sets and relationship sets to be expressed uniformly as *relation schemas* that represent the contents of the database.

A database which conforms to an E-R diagram can be represented by a collection of schemas.

For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.

Each schema has a number of columns (generally corresponding to attributes), which have unique names.

Representing Entity Sets as Schemas

A strong entity set reduces to a schema with the same attributes.

A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set

payment = (*loan_number*, *payment_number*, *payment_date*, *payment_amount*)

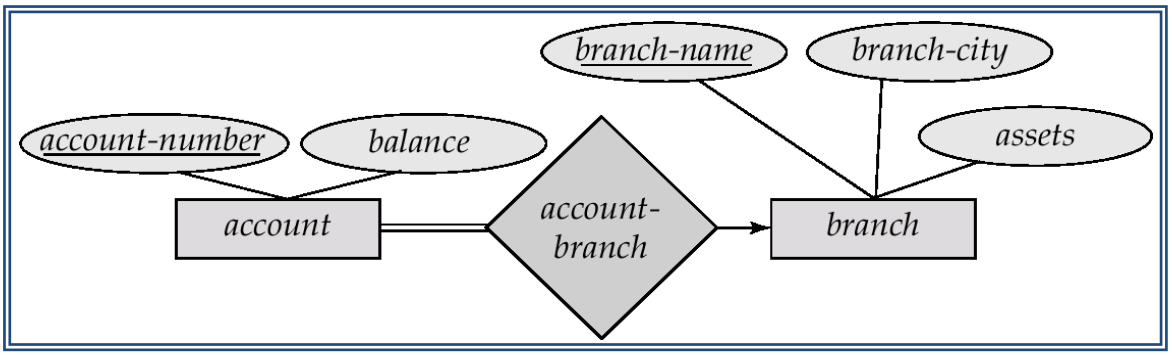
A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.

Example: schema for relationship set borrower

borrower = (*customer_id*, *loan_number*)

Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the "many" side, containing the primary key of the "one" side

Example: Instead of creating a schema for relationship set *account_branch*, add an attribute *branch_name* to the schema arising from entity set *account*



For one-to-one relationship sets, either side can be chosen to act as the "many" side

That is, extra attribute can be added to either of the tables corresponding to the two entity sets

If participation is *partial* on the "many" side, replacing a schema by an extra attribute in the schema corresponding to the "many" side could result in null values

The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.

Example: The *payment* schema already contains the attributes that would appear in the *loan_payment* schema (i.e., *loan_number* and *payment_number*).

Composite attributes are flattened out by creating a separate attribute for each component attribute

Example: given entity set *customer* with composite attribute *name* with component attributes *first_name* and *last_name* the schema corresponding to the entity set has two attributes

name.first_name and *name.last_name*

A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*

Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*

Example: Multivalued attribute *dependent_names* of *employee* is represented by a schema:

employee_dependent_names = (*employee_id*, *dname*)

Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*

For example, an employee entity with primary key 123-45-6789 and dependents Jack and Jane maps to two tuples:

(123-45-6789 , Jack) and (123-45-6789 , Jane)

Representing Specialization via Schemas

Form a schema for each entity set with all local and inherited attributes

schema	attributes
person	*name, street, city*
customer	*name, street, city, credit_rating*
employee	*name, street, city, salary*

If specialization is total, the schema for the generalized entity set (*person*) not required to store information

Can be defined as a "view" relation containing union of specialization relations

But explicit schema may still be needed for foreign key constraints

Drawback: *street* and *city* may be stored redundantly for people who are both customers and employees

Schemas Corresponding to Aggregation

To represent aggregation, create a schema containing

primary key of the aggregated relationship,

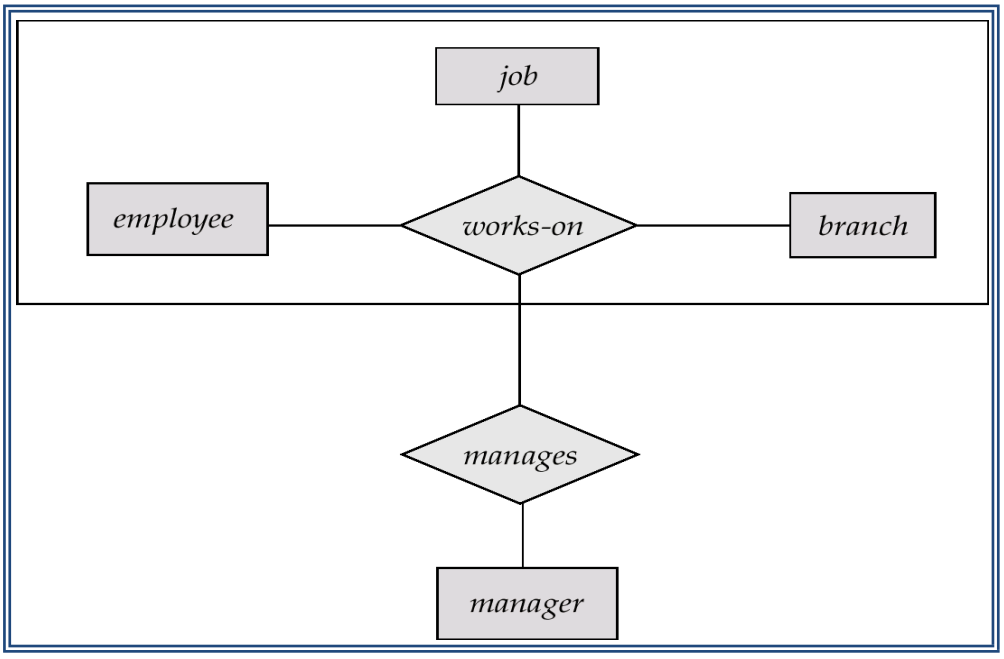
the primary key of the associated entity set

any descriptive attributes

For example, to represent aggregation manages between relationship works_on and entity set manager, create a schema

manages (*employee_id*, *branch_name*, *title*, *manager_name*)

Schema *works_on* is redundant provided we are willing to store null values for attribute *manager_name* in relation on schema *manages*



Conclusion: We have successfully mapped our ER model to Relational Schema.

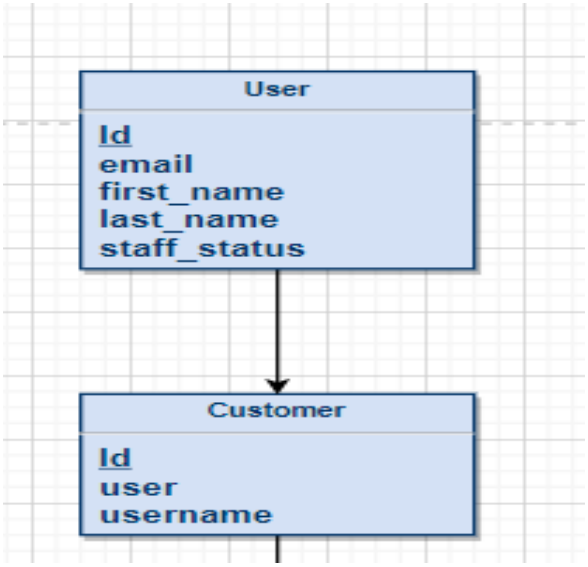
User-Customer Relation:

User = (id, email, first_name, last_name, staff_status)

id, email, first_name,last_name, staff_status all are attributes of User where id is a primary key

Customer = (id, user ,username)

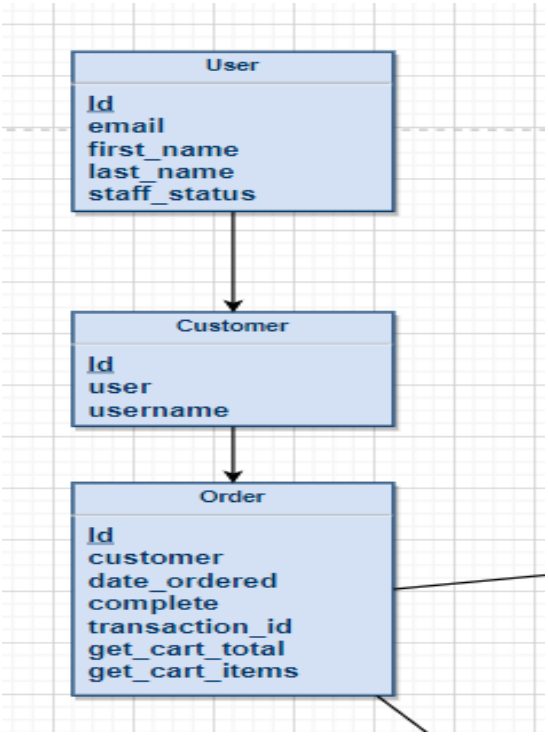
id is a primary key, user is foreign key referencing User model through User primary key(user.id) and username is a normal attribute.



Customer-Order relation:

Order = (id, Customer, date_ordered, complete, transaction_id)

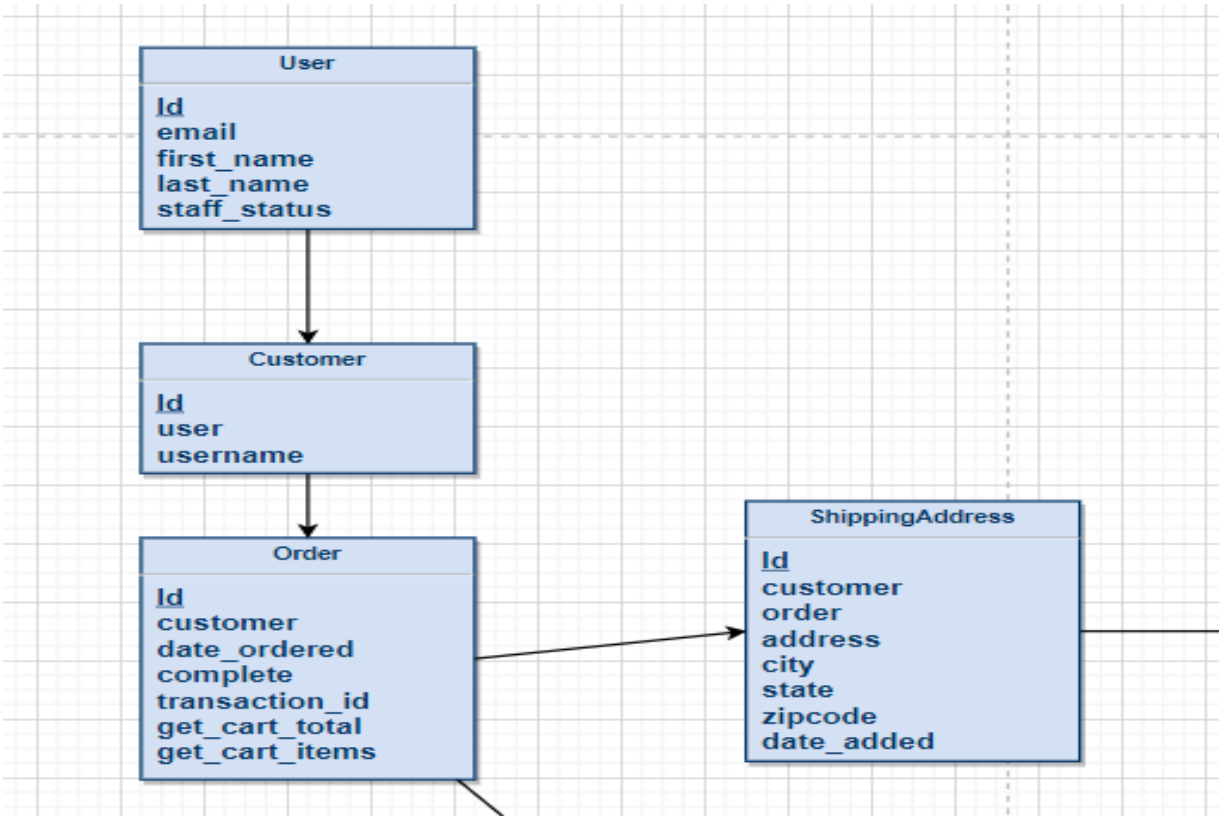
id is a primary key, Customer is a foreign key referencing Customer referencing through Customer primary key (customer.id) and other are normal attributes.



Order-Customer-Shipping Relation:

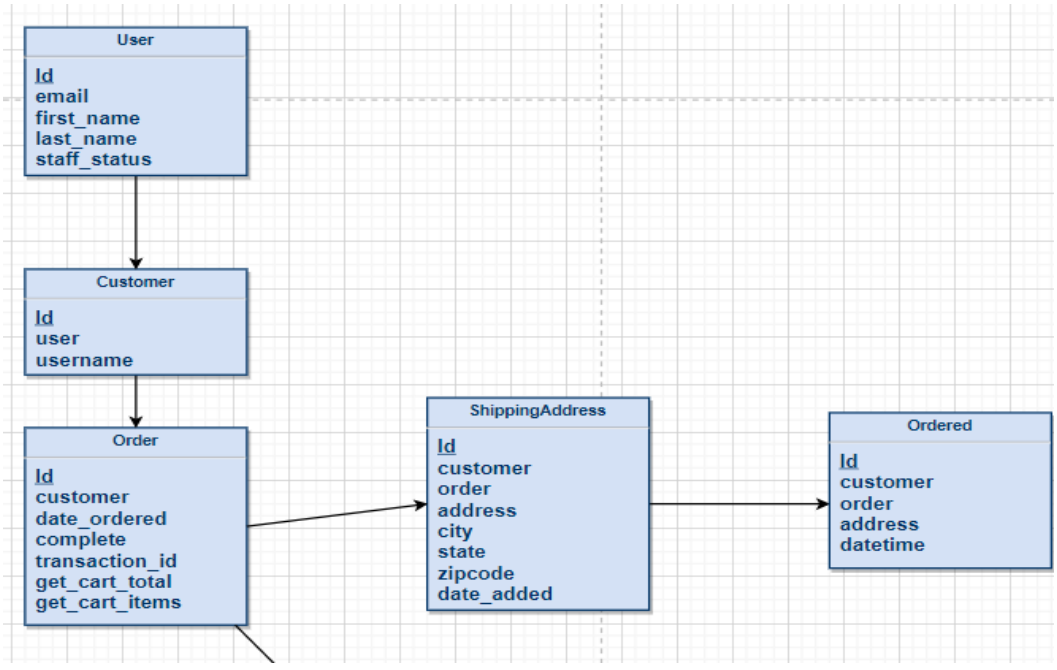
ShippingAddress = (id, customer, order, address, city, state, zip_code, date, added)

id is a primary key, customer and order are foreign keys referencing Customer and Order through their corresponding primary key(customer.id and order.id) and other are normal attribute.



Ordered-Order-Customer-Shipping Relation:

Ordered = id is a primary key, customer, order and address are foreign keys referencing Customer, Order and ShippingAddress through their corresponding primary key(customer.id, order.id and address.id) and other are normal attributes.



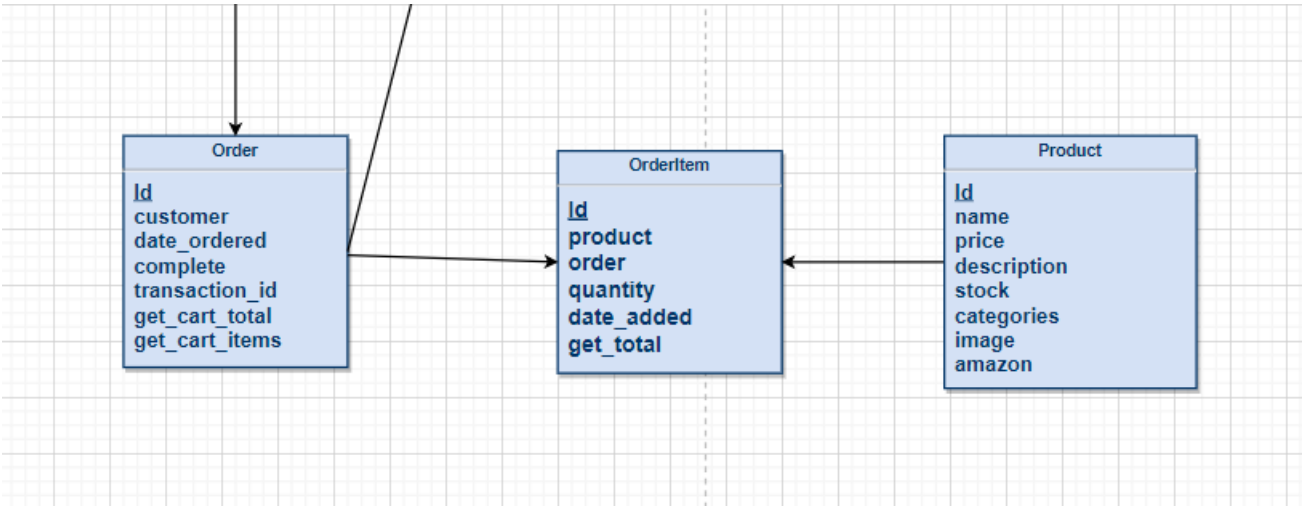
OrderItem-Order-Product Relation:

Product = (id, name, price, description, stock, categories, image, amazon)

id is a primary key and other are normal attributes.

OrderedItem = (id, product, order, quantity, date_added)

id is a primary key, product and order are foreign keys referencing Product and Order through their corresponding primary key(product.id and order.id) and other are normal attributes.



Final-Relational Schema: