**E-Commerce Database**

**Goals & Overview**

* To understand the requirements given by the stakeholders.
* To identify what entities are required and the attributes corresponding to them.
* To plan on which relational database management to develop e-Commerce project.
* To design the database with the core entities and establish the relations between them.
* To ensure that the data stored is accurate and consistent by implementing validation rules and quality checks.
* To implement the customer queries to work efficiently with the DDL and DML commands
* To document the result in the form reports by implementing tools for data mining and data visualization.
* Overall goal of database project is to create a system that meets the specific needs of the users and help them to manage and make use of their data more effectively.

The steps to design and implement a database are as follows

1. **Requirement Specification & Analysis**:

This phase involves gathering information from stakeholders and analyzing their needs. Which includes functional and nonfunctional requirements for the database system.

The business here is an ecommerce website and where product & customer information is stored and dummy orders are created to understand database concepts.

The database has to store Customer Information, Product details they have in the inventory, Order details, Payment & Shipping information.

After the analysis, entities that are identified based on the requirements are

CUSTOMER, OREDER, PRODUCT, SHIPMENT.

1. **Conceptual Design:**

The next step is to create a conceptual schema for the database, using a high-level conceptual data model. The result of this phase is an Entity-Relationships (ER) diagram or UML class diagram. It is a high-level data model of the application. It describes how different entities (object,items) are related to each other. It also describes what attributes (features) each entity has. It includes the relationship between entities, attributes of the application.

**ENTITIES:**

* CUSTOMER – customer\_id, first\_name, last\_name, email, address, city, state, zip
* ORDER -- order\_id, order\_date, order\_cost, order\_status
* PRODUCT -- product\_id, product\_name, product\_description, price, quantity
* SHIPMENT -- shipment\_id, ship\_date, courier\_name, ship\_address

**CUSTOMER** : This entity will store the information about the customers who shop at the online store. We declare “customer\_id” to be primary key that uniquely identifies every record of the table. The attribute “email” can’t be a primary key since it takes more space and more time for processing large text than an integer datatype.

As index been created on primary key by default. Most RDBMSs build a clustered index on the primary key to facilitate fast search and retrieval. The other attributes in this entity are customer name, address, city, state and zip.

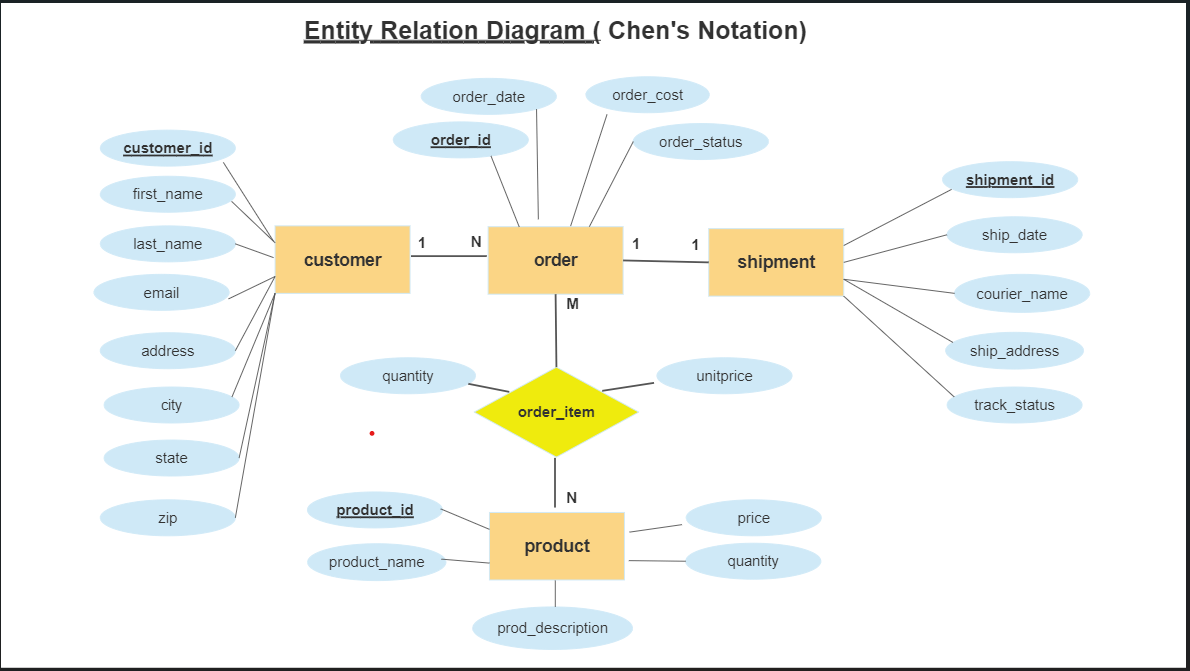
**ORDER**: This entity will store information about the orders placed by the customers. We declare “order\_id” to be primary key. The other attributes are the date of the order , cost of the order and the status of order pertains to ‘confirmed’/’canceled’.

Order entity will have

**PRODUCT:** This entity will store information about the products that the online store sells. We declare “product\_id” to be primary key. The other attributes are name of the product, description of the product , product price and quantity. The datatype of product description is ‘text’ as it can store large amount of text data than a varchar datatype.

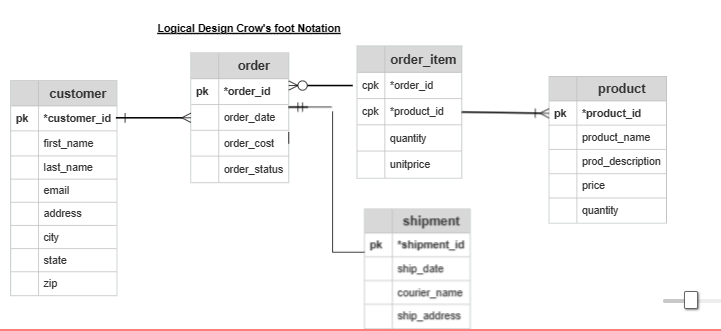
**SHIPMENT:** This entity will store information about the shipment of orders such as “shipment\_id” to be primary key and datatype is varchar instead of integer because we would like store hexadecimal values, shipment date, courier name and ship address.



****

**3. Logical Design**

The result of the logical design phase (or data model mapping phase) is a set of relational schemas. The ER diagram or class diagram is the basis for these. In this phase, the primary keys and foreign keys are defined.



The most crucial aspect in designing a relational database is to identify the relationships among tables. The types of relationship include:

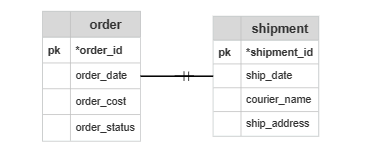
* one-to-one
* one-to-many
* many-to-many

one-to-one: In database one-to-one relationship is a type of relationship between 2 entities or tables where one record in the first table is associated with exactly one record in the second table, and vice versa. Typically, one table will have a primary key that uniquely identifies each record , and the other table will have a foreign key that references the primary key of the first table.

For example, if we take the entities **country** and **capital**. A country will have one capital and a capital refers to single country. This exhibits one-to-one relationship.

In our **e-commerce** application, we assume that an order will be shipped to one customer’s address. The two entities Order and Shipment exhibit a one-to-one relationship. That is for every row in parent table , there is one corresponding row in the child table.

The **order\_id** attribute of order entity is a **primary key** and it becomes the **foreign key** column in the **shipment** entity.



**One-to-many:**

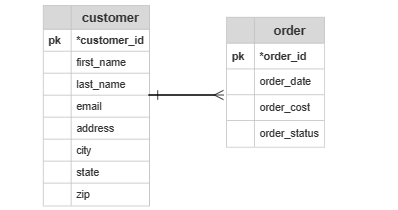
In database design, a one-to-many relationship is a type of relationship between two entities or tables where one record in the first table is associated with zero or more records in the second table, but each record in the second table is associated with exactly one record in the first table.

One-to many relatioships are widely used in database design,as they allow for efficient organization and management of related data.They enable the representation of complex relationships and provide flexibility in querying and retrieving data.

For example, in entities like **movie** and **director** , one director can direct one or more movies and a movie will be directed by one director.

In e-commerce database, the **customer** table may have **one-to-many** relationship with **order** table. Where each order belongs to only one customer and a customer can place one or multiple orders. The **customer** entity has **customer\_id** as **primary key** and the same **customer\_id** acts as **foreign\_key** in **order** table.

The **order** table may have **one-to-many** relationship with **order-item** table. Each order can have multiple items, but each item belongs to only order.



**many-to-many:**

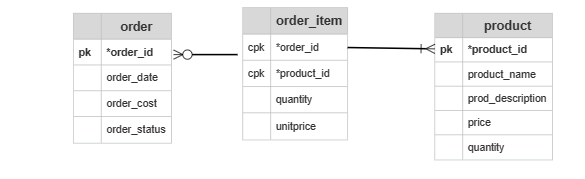
A many-to-many relationship is a type of relationship between two entities or tables where multiple records in the first table are associated with multiple records in the second table, and vice versa. It means that each record in one table can be related to multiple records in other table, and vice versa.

For example, we take entities like **movie** and **actor**. Where a movie can have one or more actors and an actor can act in one or more movies. Since both entities can’t have references of one another as foreign keys , we need to derive a new table called **movie\_actor** . Which can store the **movie\_id** as **foreign key** references movie\_id(primary key ) of movie table and **actor\_id as foreign key** references actor\_id(primary key) in **actor** table.

In e-commerce database, the relationship between **order** entity and **product** entity is a many-to-many relationship. A customer’s order may contain one or more products and a product can appear in many orders. The product table contains information about the products (such as product\_id as primary key , name, description, price and quantity). The table order contains customer’s orders (order\_id as primary key , customer\_id as foreign key, order date, cost and order status).

We cannot store the items ordered inside order table , as we do not know how many columns to reserve for the items. We also cannot store the order information in the products table. As an order can have multiple products and a product can be part of multiple orders.

To support many-to-many relationship, we need to create a third table (known as a junction table), say **order\_item,** where each row represents an item of a particular order. For this order\_item entity ,the primary key consists of **two** columns: **order\_id and product\_id** as **composite primary key** ,that uniquely identify each row. The columns order\_id and product\_id in order\_items table are used to reference orders and product tables. Hence, they are also the foreign keys in order\_items table. The other attributes in order\_item table are quantity and unitprice.( see below figure)



**4.Normalization**

Normalization is the last part of the logical design. The goal of normalization is to eliminate redundancy and potential update anomalies. Redundancy means that the same data is saved more than once in a database. Update anomaly is a consequence of redundancy. If a piece of data is saved in more than one place, the same data must be updated in more than one place. Normalization is a technique by which one can modify the relational schema to reduce the redundancy. Each normalization phase adds more relations (tables) into the database

Apply the normalization rules to check whether your database is structurally correct and optimal.

• **First Normal Form (1NF):** A table is 1NF if every cell contains a single value, not a list of values. This properties is known as atomic. 1NF also prohibits repeating group of columns such as item1, item2,.., itemN. Instead, you should create another table using one-to-many relationship .

In our e-commerce database design ,

Customer entity with attributes – **customer\_id, first\_name, last\_name, email, address, state, zipcode** has a primary key, no repeating groups and contains atomic values.

Order entity with attributes**- order\_id, order\_date, order\_cost, order\_status** has a primary key, no repeating groups and contains atomic values.

Product entity has attributes- **product\_id, product\_name, product\_description, price, quantity status** has a primary key, no repeating groups and contains atomic values.

Shipment entity has attributes- **shipment\_id, ship\_date, courier\_name, ship\_address** has a primary key, no repeating groups and contains atomic values.

Thus, all the entities have satisfied 1NF. Since there are no repeated groups and the primary keys are defined for each entity.

• **Second Normal Form (2NF):** Tables are in 2NF, they confirm to 1NF and every column that is not a primary key of the table is dependent on the whole of the primary key. Furthermore, if the primary key is made up of several columns, every non-key column shall depend on the entire set and not part of it.

In our e-commerce database design , we ensured that the entities have satisfied the rules of 1NF.

Customer entity with attributes – **customer\_id, first\_name, last\_name, email, address, state, zipcode** is in 1NF. Every non primary key column is fully dependent on primary key attribute with no partial dependency.

Order entity with attributes**- order\_id, order\_date, order\_cost, order\_status** has a primary key, is in 1NF. Every non primary key column is fully dependent on primary key attribute with no partial dependency.

Product entity has attributes- **product\_id, product\_name, product\_description, price, quantity status** is in 1NF. Every non primary key column is fully dependent on primary key attribute with no partial dependency.

Shipment entity has attributes- **shipment\_id, ship\_date, courier\_name, ship\_address** has a primary key, is in 1NF. Every non primary key column is fully dependent on primary key attribute with no partial dependency.

Hence, all the entities satisfies 2NF.

• **Third Normal Form (3NF):** Tables are in 3NF, if they confirm to 2NF and the non-primary key columns are fully dependent on primary key and nothing else. That is unrelated data is being removed.

In our e-commerce database design , we ensured that the entities have satisfied the rules of 2NF. In order to satisfy the 3NF, we have to split the customer table into 2 tables as there is a transitive dependency.

Customer entity with attributes – **customer\_id, first\_name, last\_name, email, address, state, zipcode.** In this, zipcode (non-primary key attribute) is transitively dependent on primary key (customer\_id). So, for customer table to be in 3NF, we split the customer table and derive a new table by name **city\_code** table having attributes – city, zip.



Order entity with attributes**- order\_id, order\_date, order\_cost, order\_status**  is in 3NF. Every non primary key column is fully dependent on primary key attribute with no transitive dependency.

Product entity has attributes- **product\_id, product\_name, product\_description, price, quantity available** is in 1NF. Every non primary key column is fully dependent on primary key attribute with no transitive dependency.

Shipment entity has attributes- **shipment\_id, ship\_date, courier\_name, ship\_address** has a primary key, is in 1NF. Every non primary key column is fully dependent on primary key attribute with no transitive dependency.

Thus, all entities are in 3NF.

• **Higher Normal Form**: 3NF has its inadequacies, which leads to higher Normal form, such as Boyce-Codd Normal form, Fourth Normal Form (4NF) and Fifth Normal Form (5NF). At times, you may decide to break some of the normalization rules, for performance reason or because the end-user requested for it.

**5.Physical Design:**

The goal of the last phase of database design, physical design, is to implement the database. At this phase one must know which database management system (DBMS) is used. For example, different DBMS's have different names for datatypes and have different datatypes. If we decide to go with SQL, the SQL clauses to create the database are written. The indexes, the integrity constraints (rules) and the users' access rights are defined