# Design

System design is the process of defining the elements of a system such as the architecture, modules and components, the different interfaces of those components and the data that goes through that system. It is meant to satisfy specific needs and requirements of a business or organization through the engineering of a coherent and well-running system.

Systems design implies a systematic approach to the design of a system. It may take a bottom-up or top-down approach, but either way the process is systematic wherein it takes into account all related variables of the system that needs to be created—from the architecture, to the required hardware and software, right down to the data and how it travels and transforms throughout its travel through the system. Systems design then overlaps with systems analysis, systems engineering and systems architecture.

## Structural Design

Structural patterns ease the design of a program by identifying a way to realize relationships between entities. The difference between class patterns and object patterns is that class patterns describe how inheritance can be used to provide more useful program interfaces. Object patterns, on the other hand, describe how objects can be composed into larger structures using object composition, or the inclusion of objects within other objects.

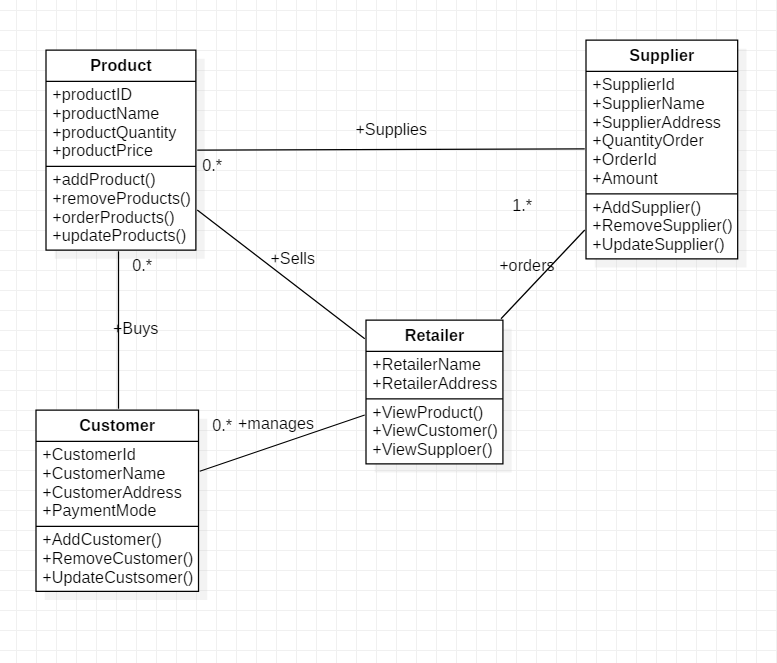
### Parts of Structural Design are

#### Class Diagram

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML). In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity.

Notations.

|  |  |  |
| --- | --- | --- |
| Diagram element | Graphical presentation | Description |
| Class | UML Class Diagram Notation - Class | Class represents a set of objects that have the same structure, behavior, and relationships with objects of other classes. |
| Attribute | UML Class Diagram Notation - Attributes | Attribute is a typed value that defines the properties and behavior of the object. |
| Operation | UML Class Diagram Notation - Operation | Operation is a function that can be applied to the objects of a given class. |
| Responsibility | UML Class Diagram Notation - Responsibility | Responsibility is a contract which the class must conform. |
| Interface | UML Class Diagram Notation - Interface | Interface is an abstract class that defines a set of operations that the object of the class associated with this interface provides to other objects. |
| Association | UML Class Diagram Notation - Association | Association is a relationship that connect two classes. |
| Aggregation | UML Class Diagram Notation - Aggregation | Aggregation is an association with the relation between the whole and its parts, the relation when one class is a certain entity that includes the other entities as components. |
| N-ary Association | UML Class Diagram Notation - N-ary Association | N-ary association represents two or more aggregations. |
| Composition | UML Class Diagram Notation - Composition | Composition is a strong variant of aggregation when parts cannot be separately of the whole. |
| Generalization | UML Class Diagram Notation - Generalization | Generalization ia an association between the more general classifier and the more special classifier. |
| Inheritance | UML Class Diagram Notation - Inheritance | Inheritance is a relationship when a child object or class assumes all properties of his parent object or class. |
| Realization | UML Class Diagram Notation - Realization | Realization is a relationship between interfaces and classes or components that realize them. |
| Dependency | UML Class Diagram Notation - Dependency | Dependency is a relationship when some changes of one element of the model can need the change of another dependent element. |
| << >> | UML Class Diagram Notation - Relationship | Allows to define the properties of the dependency relationship between classes or classes and packages. |
| { } | UML Class Diagram Notation - additional properties of association | Allows to indicate the additional properties of association. |
| Multiplicity | UML Class Diagram Notation - Multiplicity | Multiplicity shows the quantity of instances of one class that are linked to one instance of the other class. |
| Package | UML Class Diagram Notation - Package | Package groups the classes and other packages. |
| Note | UML Class Diagram Notation - Note | Note is a textual explication. |
| Note connector | UML Class Diagram Notation - Note connector | Note connector is a connection between the note and elements. |



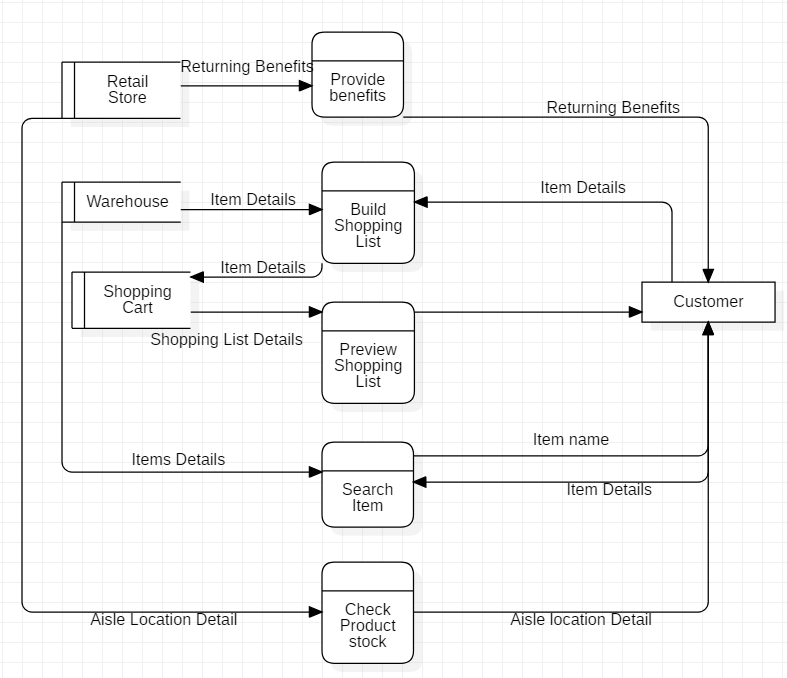
**IMAGE CLASS DIAGRAM**

Four individual classes are created supplier, retailer, product and customer. Each of them are interconnected to form an exchanging relationship. Here retailer orders products through suppliers and supplier supplies the product. Customer then look product details and is managed by the retailer to sell products and provide product description.

**Data Flow Diagram**

A data flow diagram (or DFD) is a graphical representation of the flow of data through an information system. It shows how information inputs to and output from the system, the sources and destinations of that information, and where that information is stored.

1. External entity: an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.
2. Process: any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules. A short label is used to describe the process, such as “Submit payment.”
3. Data store: files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label, such as “Orders.”
4. Data flow: the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like “Billing details.”



Here the data is customized and process from retail store to the customer in systematic method. To begin the retail store announces benefits if the customer is a frequent face to the store. Then the customer builds a cart full of product where the products are supplied from the warehouse. After the cart is complete and customer is provided with option to pay with cash or QR pay.

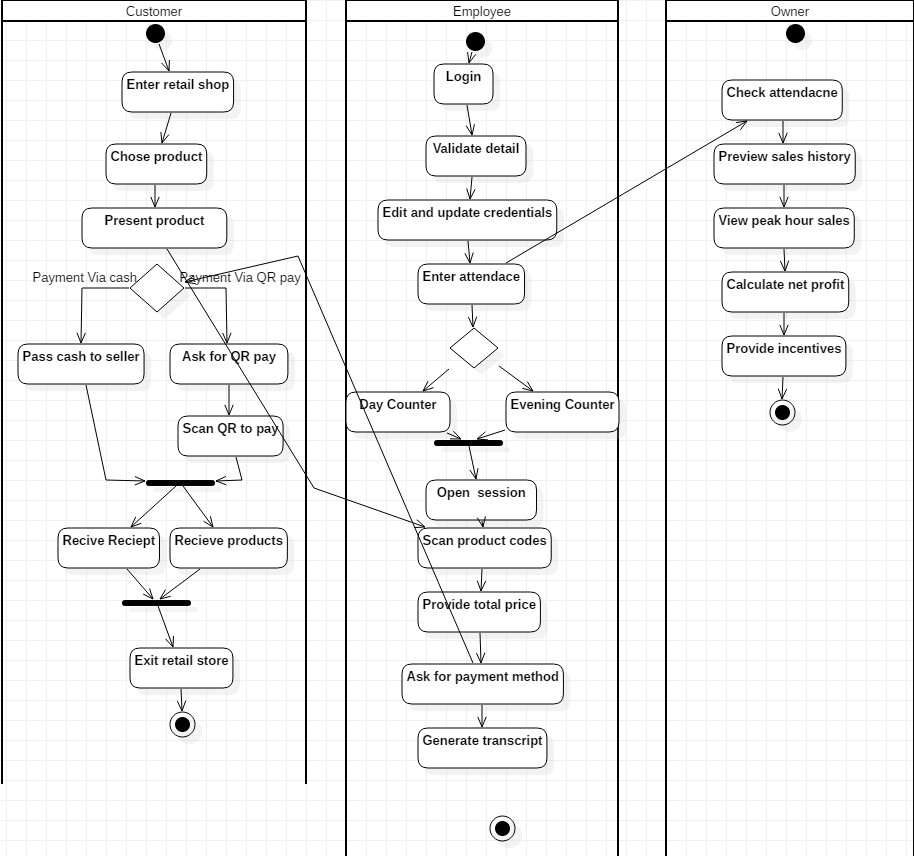
# Behaviour Design

In software engineering, behavioural design patterns are design patterns that identify common communication patterns among objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication.

## Activity Diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements as fork,join,etc.

|  |  |  |
| --- | --- | --- |
| start Symbol | Start symbol | Represents the beginning of a process or workflow in an activity diagram. It can be used by itself or with a note symbol that explains the starting point. |
| activity Symbol | Activity symbol | Indicates the activities that make up a modeled process. These symbols, which include short descriptions within the shape, are the main building blocks of an activity diagram. |
| connector Symbol | Connector symbol | Shows the directional flow, or control flow, of the activity. An incoming arrow starts a step of an activity; once the step is completed, the flow continues with the outgoing arrow. |
| joint Symbol | Joint symbol/ Synchronization bar | Combines two concurrent activities and re-introduces them to a flow where only one activity occurs at a time. Represented with a thick vertical or horizontal line. |
| fork Symbol | Fork symbol | Splits a single activity flow into two concurrent activities. Symbolized with multiple arrowed lines from a join. |
| decision Symbol | Decision symbol | Represents a decision and always has at least two paths branching out with condition text to allow users to view options. This symbol represents the branching or merging of various flows with the symbol acting as a frame or container. |
| note Symbol | Note symbol | Allows the diagram creators or collaborators to communicate additional messages that don't fit within the diagram itself. Leave notes for added clarity and specification. |
| send signal Symbol | Send signal symbol | Indicates that a signal is being sent to a receiving activity. |
| receive signal Symbol | Receive signal symbol | Demonstrates the acceptance of an event. After the event is received, the flow that comes from this action is completed. |
| shallow history pseudostate symbol | Shallow history pseudostate symbol | Represents a transition that invokes the last active state. |
| option loop symbol | Option loop symbol | Allows the creator to model a repetitive sequence within the option loop symbol. |
| flow final symbol | Flow final symbol | Represents the end of a specific process flow. This symbol shouldn’t represent the end of all flows in an activity; in that instance, you would use the end symbol. The flow final symbol should be placed at the end of a process in a single activity flow. |
| condition text | Condition text | Placed next to a decision marker to let you know under what condition an activity flow should split off in that direction. |
| end symbol | End symbol | Marks the end state of an activity and represents the completion of all flows of a process. |



**IMAGE ACTIVITY**

Customer activity is straight forward. In short steps include

* Select product
* Provide payment method
* Take receipt
* Checkout

Here the customer selects and browses through presented products. Finalizes decision and heads to the counter to process transaction. Then is prompted by the employee for a particular payment method. After a successful payment items are provided with a receipt of all the transaction amount.

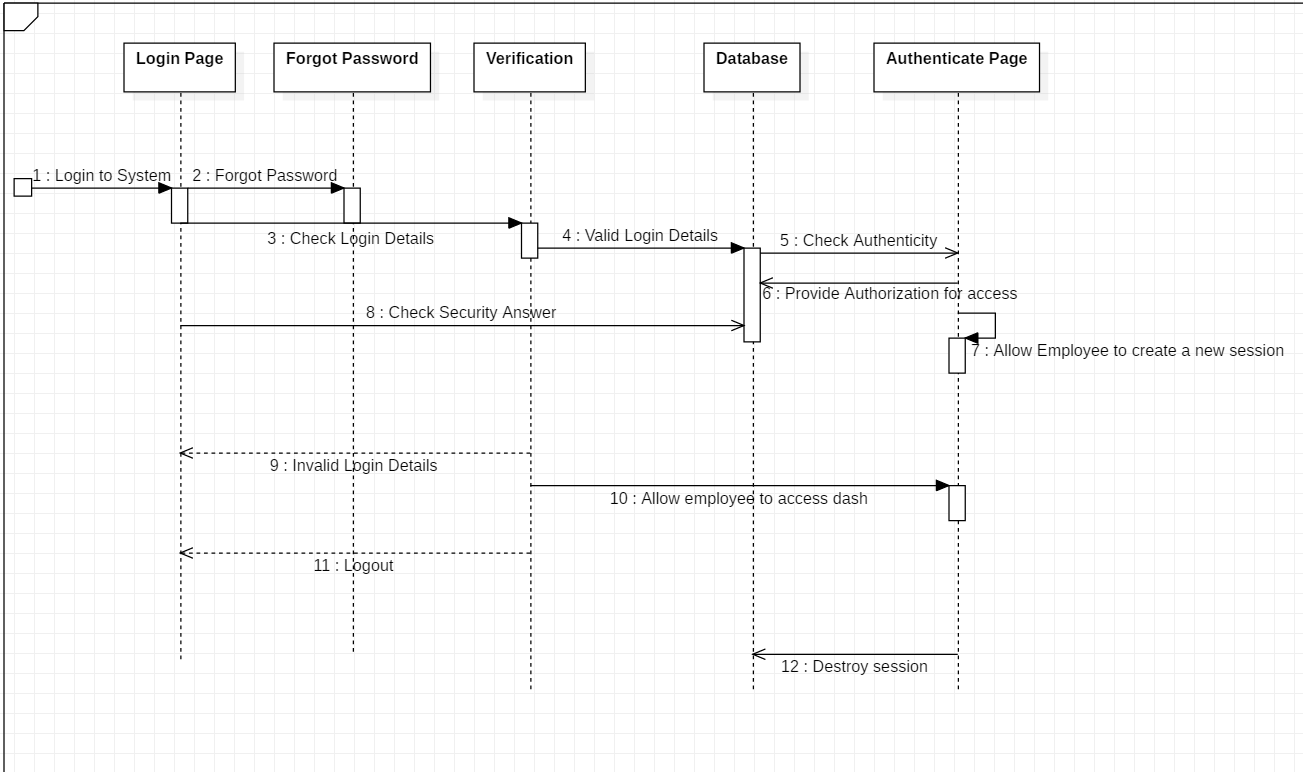
In the second diagram for the employee diagram processes employee login, attendance , counter managements and sessions to start working. Employees are validated through login. Then provided with a menu to choose session and the shifts begins.

In the third diagram the owner, is in the top level hierarchy flow. Owner manages employee details, generates profit per day in order to provide some additional benefit to the employee. He/she can also view peak hour sales to maximize number of sales per day.

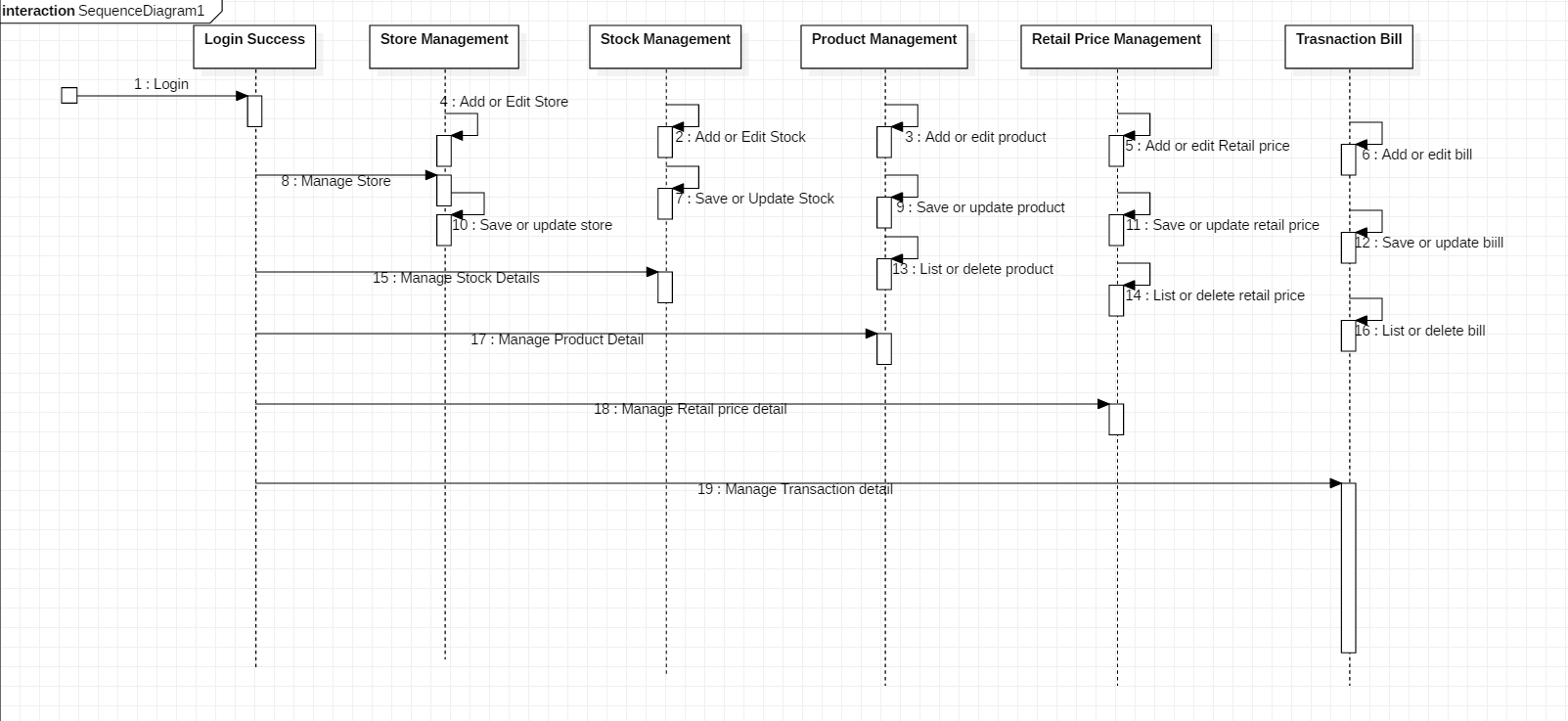
## Sequence diagram

Object interactions usually begin at the top of a diagram and end at the bottom. In a sequence diagram, object interaction occurs through messages on the vertical and horizontal dimensions and are designated by horizontal arrows and message names. The initial sequence diagram message begins at the top and is located on the diagram's left side. Subsequent messages are added just below previous messages. Sequence diagram messages may be subdivided by type, based on functionality.

|  |  |
| --- | --- |
| **Notation Description** | **Visual Representation** |
| **Actor**   * a type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data) * external to the subject (i.e., in the sense that an instance of an actor is not a part of the instance of its corresponding subject). * represent roles played by human users, external hardware, or other subjects.   Note that:   * An actor does not necessarily represent a specific physical entity but merely a particular role of some entity * A person may play the role of several different actors and, conversely, a given actor may be played by multiple different person. | Actor |
| **Lifeline**   * A lifeline represents an individual participant in the Interaction. | Lifeline |
| **Activations**   * A thin rectangle on a lifeline) represents the period during which an element is performing an operation. * The top and the bottom of the of the rectangle are aligned with the initiation and the completion time respectively | Activation |
| **Call Message**   * A message defines a particular communication between Lifelines of an Interaction. * Call message is a kind of message that represents an invocation of operation of target lifeline. | Call Message |
| **Return Message**   * A message defines a particular communication between Lifelines of an Interaction. * Return message is a kind of message that represents the pass of information back to the caller of a corresponded former message. | Return Message |
| **Self Message**   * A message defines a particular communication between Lifelines of an Interaction. * Self message is a kind of message that represents the invocation of message of the same lifeline. | Self-Message |
| **Recursive Message**   * A message defines a particular communication between Lifelines of an Interaction. * Recursive message is a kind of message that represents the invocation of message of the same lifeline. It's target points to an activation on top of the activation where the message was invoked from. | Recursive Message |
| **Create Message**   * A message defines a particular communication between Lifelines of an Interaction. * Create message is a kind of message that represents the instantiation of (target) lifeline. | Create Message |
| **Destroy Message**   * A message defines a particular communication between Lifelines of an Interaction. * Destroy message is a kind of message that represents the request of destroying the lifecycle of target lifeline. | Destroy Message |
| **Duration Message**   * A message defines a particular communication between Lifelines of an Interaction. * Duration message shows the distance between two time instants for a message invocation. | Duration Message |
| **Note**  A note (comment) gives the ability to attach various remarks to elements. A comment carries no semantic force, but may contain information that is useful to a modeler. | Note |



Here the employees are presented with a login system. They can either login or if they cannot remember their credential they can verify and reset in real time. After login the system validates and authenticates to provide access. Then employees are prompted to choose session settings and begin session. After the day of sales they can destroy their session to record the total sales of the day.



In this diagram the retailer/owner can add/edit stocks, price and transactions which will be synchronized system wide.

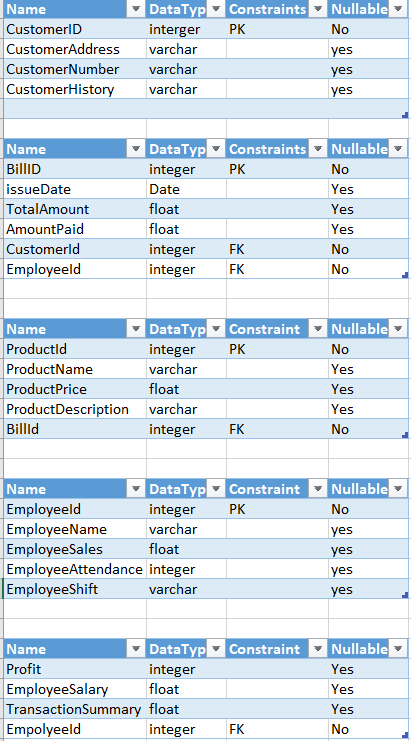
# Database design

A data dictionary is a file or a set of files that contains a database's metadata. The data dictionary contains records about other objects in the database, such as data ownership, data relationships to other objects, and other data.

The data dictionary is a crucial component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.

In a relational database, the metadata in the data dictionary includes the following:

* Names of all tables in the database and their owners
* Names of all indexes and the columns to which the tables in those indexes relate
* Constraints defined on tables, including primary keys, foreign-key relationships to other tables, and not-null constraints

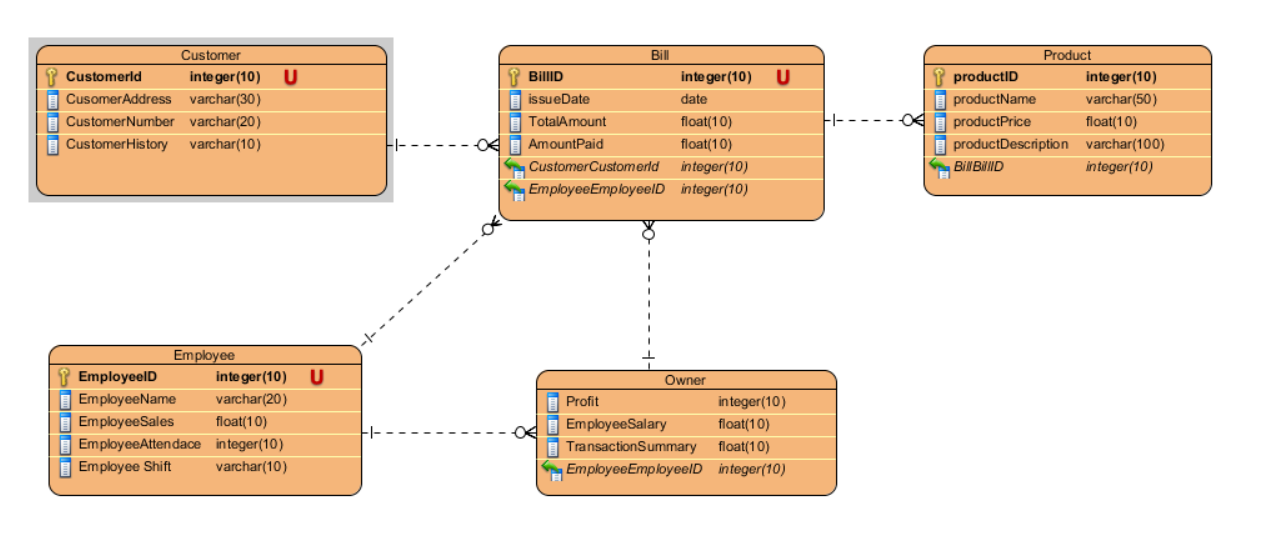


## ER Diagram

An entity-relationship diagram (ERD) is a data modelling technique that graphically illustrates an information system’s entities and the relationships between those entities. An ERD is a conceptual and representational model of data used to represent the entity framework infrastructure.

The elements of an ERD are:

* Entities
* Relationships
* Attributes



This ER diagram represents the connection and collection of multiple entities with relationships bounding them together. Here owner can handle multiple bills and customer at a time where employee can only listen orders from single owner. Customers can make much transaction and get receipt generated for every transaction but the customer credential are singular.

# Architectural Design

Architecture serves as a blueprint for a system. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

It defines a structured solution to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of −

Selection of structural elements and their interfaces by which the system is composed.

* Behavior as specified in collaborations among those elements.
* Composition of these structural and behavioral elements into large subsystem.
* Architectural decisions align with business objectives.
* Architectural styles guide the organization.

## UI Protyping

User interface (UI) prototyping is an iterative development technique in which users are actively involved in the mocking-up of the UI for a system. UI prototypes have several purposes:

* As an analysis artefact that enables you to explore the problem space with your stakeholders.
* As a design artefact that enables you to explore the solution space of your system.
* A basis from which to explore the usability of your system.
* A vehicle for you to communicate the possible UI design(s) of your system.
* A potential foundation from which to continue developing the system (if you intend to throw the prototype away and start over from scratch then you don’t need to invest the time writing quality code for your prototype)

