1. DATABASE APPLICATION DESCRIPTION:

The **Pizza Shop Database** is designed to manage customer orders, track order status, and maintain records of available pizzas. The system ensures efficient processing of orders, automatic status updates, and accurate pricing. The application provides:

- Customer management (details, orders, and contact information)
- Order tracking and history
- Menu with pizza options and prices
- Automated order status progression

Key Features

1. Menu Display & Selection

- Customers can view a list of available pizzas, including name, description, price.
- The menu data is stored in a database and fetched by the frontend.

2. Order Placement & Processing

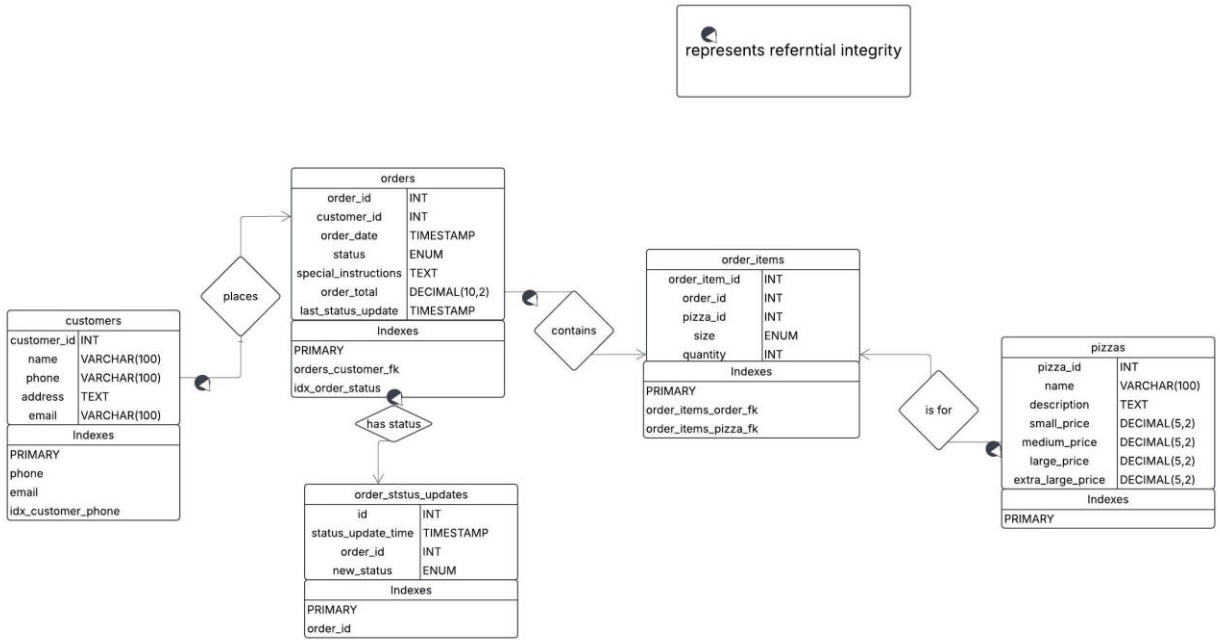
- Customers provide their name, phone number, and address before placing an order.
- They select pizzas, specify quantities, and confirm their order.
- Each order is assigned a unique order ID for tracking.
- Order details are stored in a cloud-based database.

3. Order Status Tracking

- Orders have predefined statuses: Pending, Preparing, Out for Delivery, Delivered.
- Customers can check their order status, order info and customer info using the assigned order ID.
- A database event is implemented to update the status when certain criteria are met (e.g., automatically moving from Pending to Preparing after a set time).
- The frontend fetches the latest order status along with the other information using API calls and displays it to the user.

2. ER DIAGRAM: (App used to draw: Lucidchart)

Note: Coundn’t find curved arrow to represent referential integrity constraint, so a small dot circle is placed to represent referential integrity constraint



Customers → Orders

Relation: One-to-Many (1:M) has “places” relationship
Foreign Key: orders.customer_id references customers.customer_id

Each customer can place multiple orders, but each order belongs to only **one customer**

Orders → Order_Items

Relation: One-to-Many (1:M)
Foreign Key: order_items.order_id references orders.order_id

Each order can contain multiple pizzas, but each **pizza entry in order_items belongs to a single order**.

Orders → Order_Status_Updates

Relation: One-to-Many (1:M)
Foreign Key: order_status_updates.order_id references orders.order_id
Each order can have multiple status updates, but each status update belongs to only **one order**.

Order_Items → Pizzas

Relation: Many-to-One (M:1)
Foreign Key: order_items.pizza_id references pizzas.pizza_id
Multiple order items can refer to the **same pizza**, but each order item references only **one pizza**.

3. SCHEMA AND NORMALIZATION TO 4NF:

1NF:

Goal: Remove **duplicate values** and ensure each column contains atomic values.

- We split the data into separate tables:
    - customers (customer details)
    - orders (order details)
    - pizzas (menu items)
    - order_items (association table between orders and pizzas)
    - order_status_updates (tracks status changes)
- Each row now has a **unique identifier (Primary Key)**.

2NF (Second Normal Form)

Goal: Remove **partial dependencies**.

- The composite keys in the order_items table had dependencies on order_id and pizza_id.
- We ensured that all **non-key attributes depend on the whole primary key** by restructuring relationships.

3 NF (Third Normal Form)

Goal: Remove **transitive dependencies**.

- The orders table had customer_name, phone, and address, which **depend on customer_id**.
- We moved these to the customers table.
- Ensured **only foreign keys** remained in referencing tables.

4NF (Fourth Normal Form)

Goal: Remove **multi-valued dependencies**.

- Each order previously had **multiple independent status updates**.
- We created a **separate `order_status_updates` table** to store each status update as a **single row**.
- Ensured **one fact per table** to eliminate redundancy.

4. Final Normalized Schema

Table for customers:

```
CREATE TABLE `customers` (
  `customer_id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(100) NOT NULL,
  `phone` VARCHAR(20) NOT NULL,
  `address` TEXT NOT NULL,
  `email` VARCHAR(100) DEFAULT NULL,

  PRIMARY KEY (`customer_id`),
  UNIQUE KEY `phone` (`phone`),
  UNIQUE KEY `email` (`email`)
```

```
);
```

Table for orders:

```
CREATE TABLE `orders` (
  `order_id` INT NOT NULL AUTO_INCREMENT,
  `customer_id` INT NOT NULL,
  `order_date` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
  `status` ENUM('Pending','Preparing','Out for Delivery','Delivered') DEFAULT
'Pending',
  `special_instructions` TEXT,
  `order_total` DECIMAL(10,2) DEFAULT '0.00',
  `last_status_update` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

  PRIMARY KEY (`order_id`),
  CONSTRAINT `orders_customer_fk` FOREIGN KEY (`customer_id`) REFERENCES
`customers` (`customer_id`)
);
```

Table for pizzas:

```
CREATE TABLE `pizzas` (
  `pizza_id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(100) NOT NULL,
  `description` TEXT,
  `small_price` DECIMAL(5,2) NOT NULL DEFAULT '0.00',
  `medium_price` DECIMAL(5,2) NOT NULL DEFAULT '0.00',
  `large_price` DECIMAL(5,2) NOT NULL DEFAULT '0.00',
  `extra_large_price` DECIMAL(5,2) NOT NULL DEFAULT '0.00',

  PRIMARY KEY (`pizza_id`)
);
```

Table for order_items:

```
CREATE TABLE `order_items` (
  `order_item_id` int NOT NULL AUTO_INCREMENT,
  `order_id` int NOT NULL,
  `pizza_id` int NOT NULL,
  `size` enum('Small','Medium','Large','Extra Large') NOT NULL,
  `quantity` int NOT NULL,
```

```
  PRIMARY KEY (`order_item_id`),
  CONSTRAINT `order_items_ibfk_1` FOREIGN KEY (`order_id`) REFERENCES `orders`
(`order_id`),
  CONSTRAINT `order_items_ibfk_2` FOREIGN KEY (`pizza_id`) REFERENCES `pizzas`
(`pizza_id`)
);
```

Table for order_status_updates:

```
CREATE TABLE `order_status_updates` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `order_id` INT NOT NULL,
  `status_update_time` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
  `new_status` ENUM('Pending','Preparing','Out for Delivery','Delivered') NOT
NULL,

  PRIMARY KEY (`id`),
  CONSTRAINT `order_status_updates_fk` FOREIGN KEY (`order_id`) REFERENCES
`orders` (`order_id`)
);
```