EXPRESSFOOD
UML

<u>Textural use case</u>

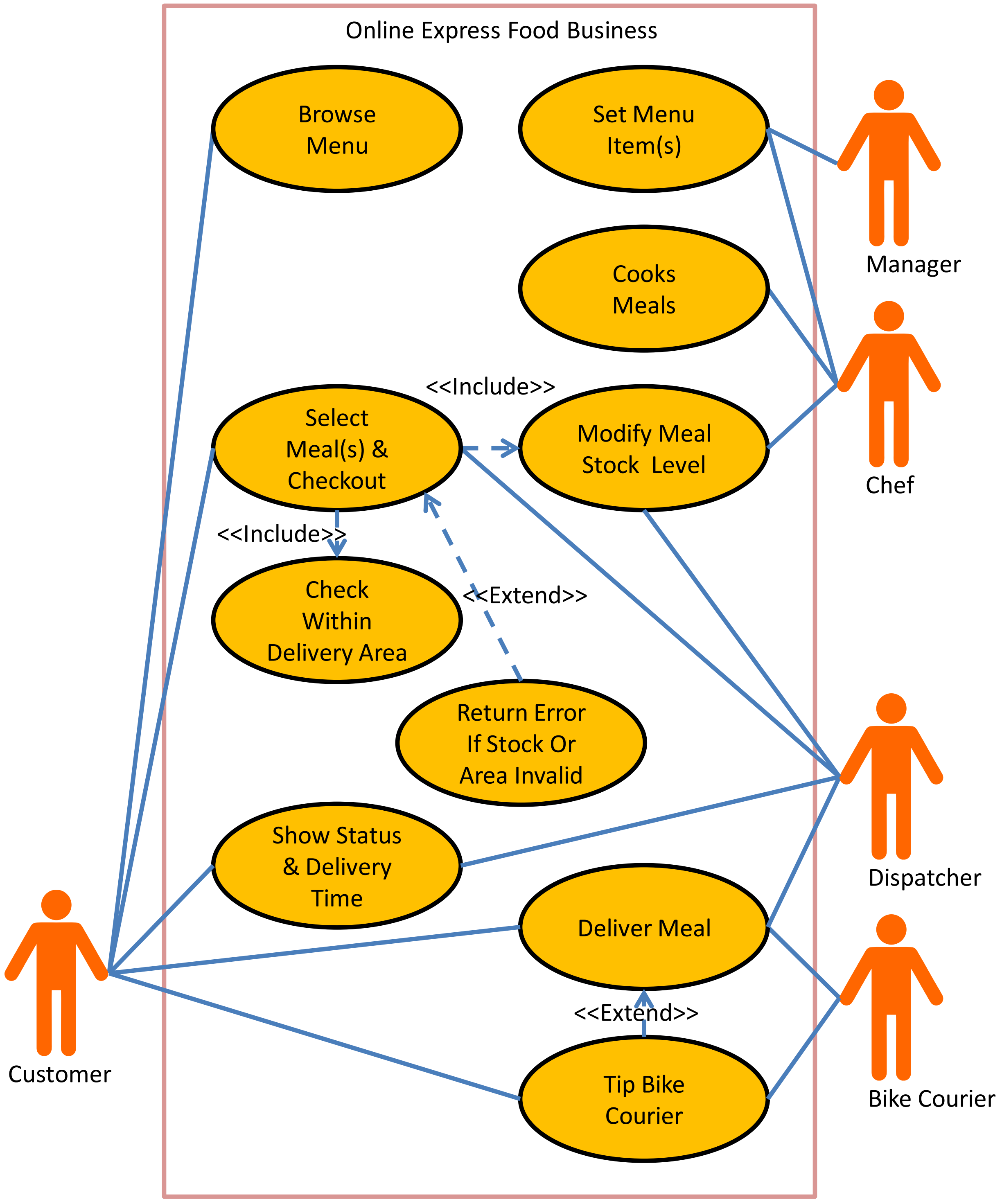
A customer logs on to ExpressFood website to browse the menu and/or select a daily menu item that they want to buy. If the item is in stock, they place an order of one or more of the 2 main daily dishes and/or one or more of the 2 daily desserts. On checkout, the customer’s delivery postcode is used to confirm if they are in the delivery area served by ExpressFood. If within the target area, the customer is shown a page with a Boolean delivery flag and an estimated time until delivery. ExpressFood selects a bike courier to deliver to the customer’s chosen delivery address in less than 20 minutes.

The menu changes daily, delivery is free.

Notes: Changing menu items on a daily basis is an unusual concept in the food business and results in a set of assumptions with varying levels of complexity with respect to the database design. These assumptions are discussed in the Domain Diagram pages.

<u>Use case diagram (see over page)</u>

EXPRESSFOOD
Use Case Diagram

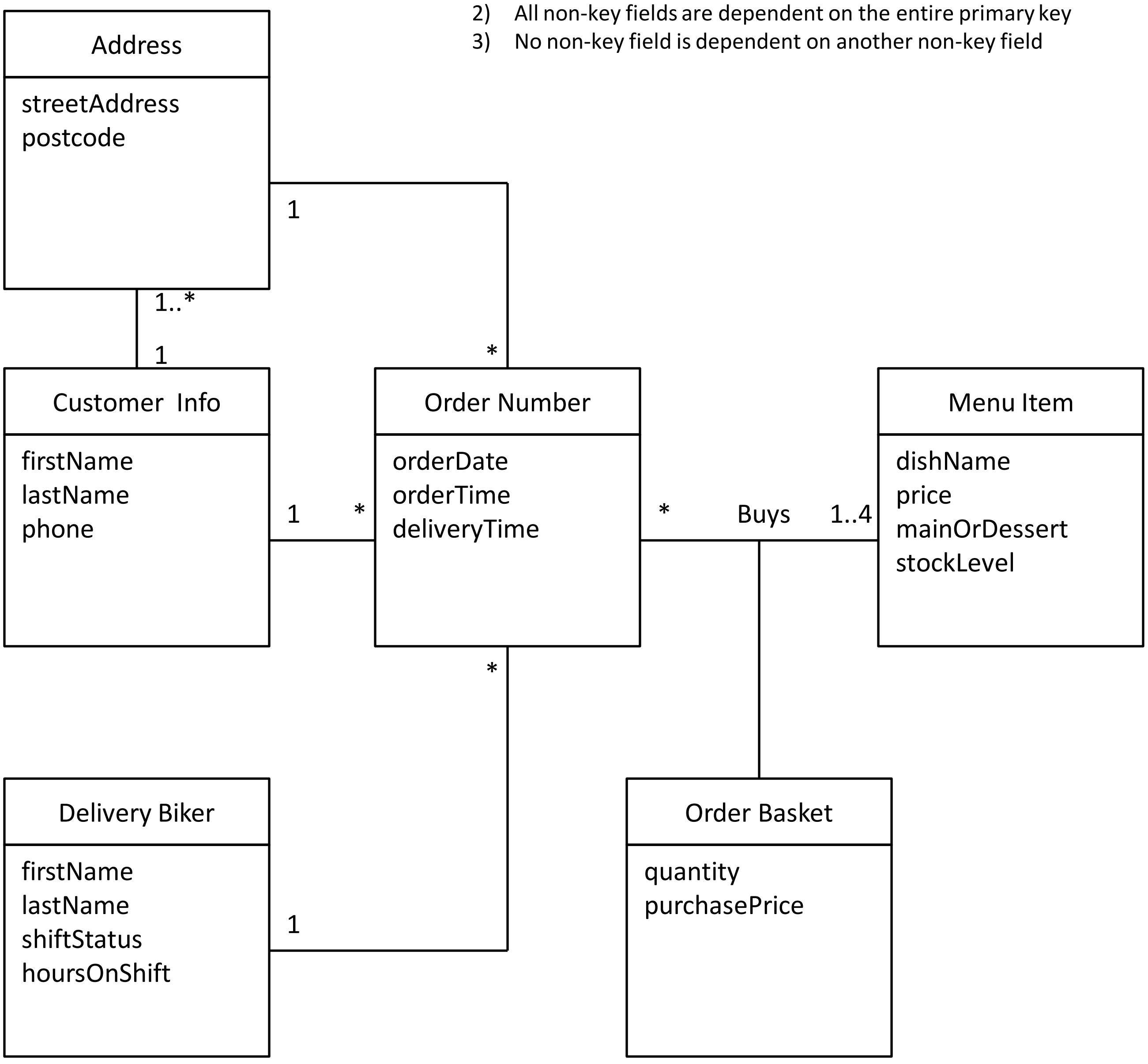


Notes: The use diagram is mostly self-explanatory. Modification of stock level is actor controlled (by the chef adding to, the dispatcher removing from). However, the system automatically deducts the stock from the database before it is physically removed to prevent more than one person ordering the same item. The delivery area check is automatically carried out by the system, there are no actors involved. Equally a return stock error and/or delivery area check error are returned automatically by the system to prevent futile orders.

ExpressFood Simple Domain Diagram

Normal Forms Check

1) Each field only contains atomic parts
2) All non-key fields are dependent on the entire primary key
3) No non-key field is dependent on another non-key field

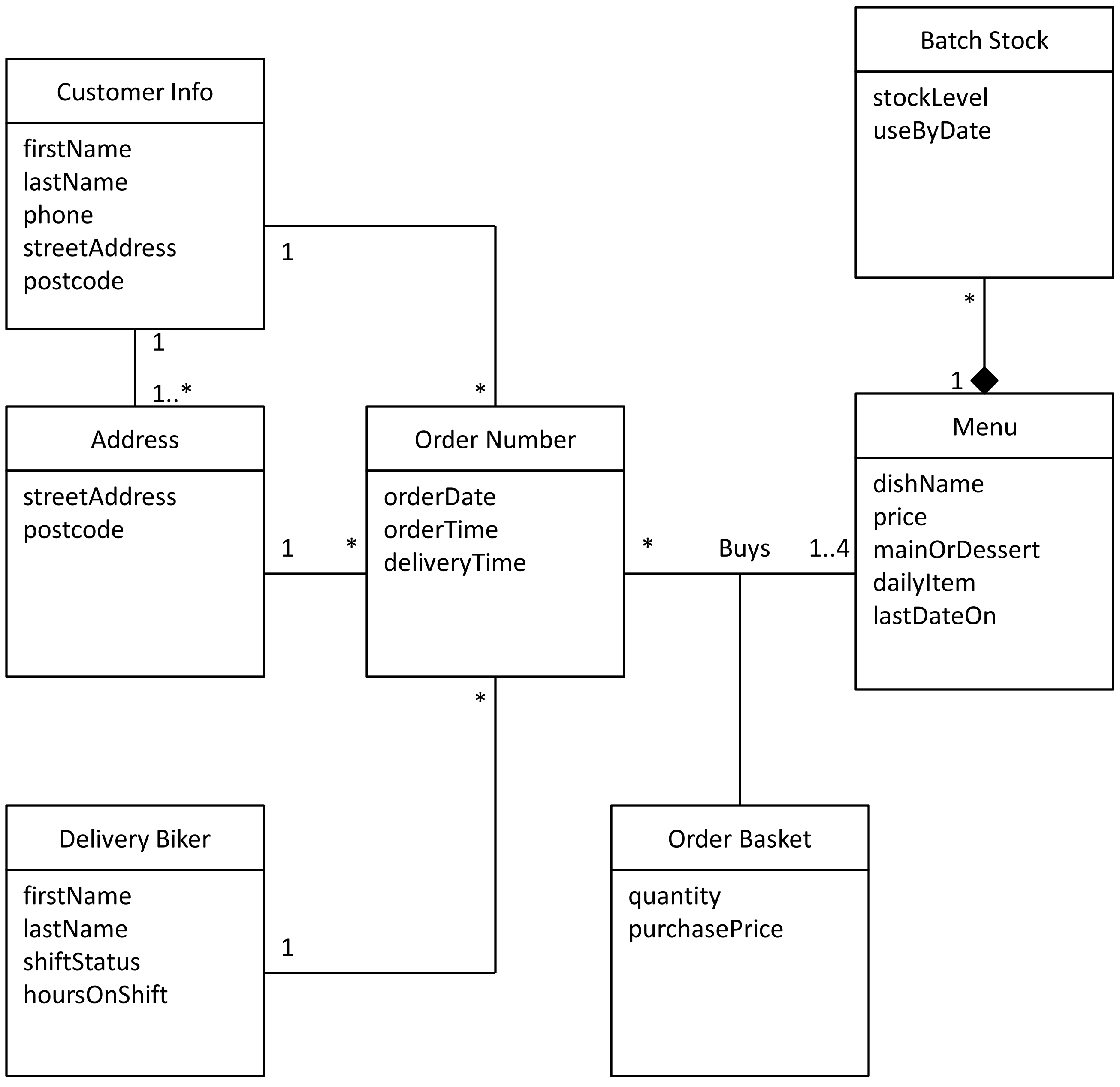


Notes: In this simple Domain Diagram, the assumption has been made that different “Menu Items” are cooked and cold stored for that day only (i.e. they can never be reused). This is unrealistic because there are a finite number of different dishes and it would be economically wasteful to throw away viable stock. However, as described in the project brief, and for a small start-up, this may be the initial case due to its simplicity.

Delivery time allows the company to track whether each delivery biker is meeting company targets i.e. under 20 minutes. Actual delivery time is automatically transferred to the database when a customer signs for delivery on the mobile phone app, or other such device, carried by the biker. Unreliable bikers can be identified for discussion of employment (improvement, warning, dismissal).

The biker “shiftStatus” is one of offShift/onShift/onDelivery represented by 0,1,2 in the system so that only current “onShift” bikers are selected. The attribute “hoursOnShift” is used to fairly apportion shifts between bikers, and could be used for wage calculation if by hourly rate as opposed to gig work of per delivery.

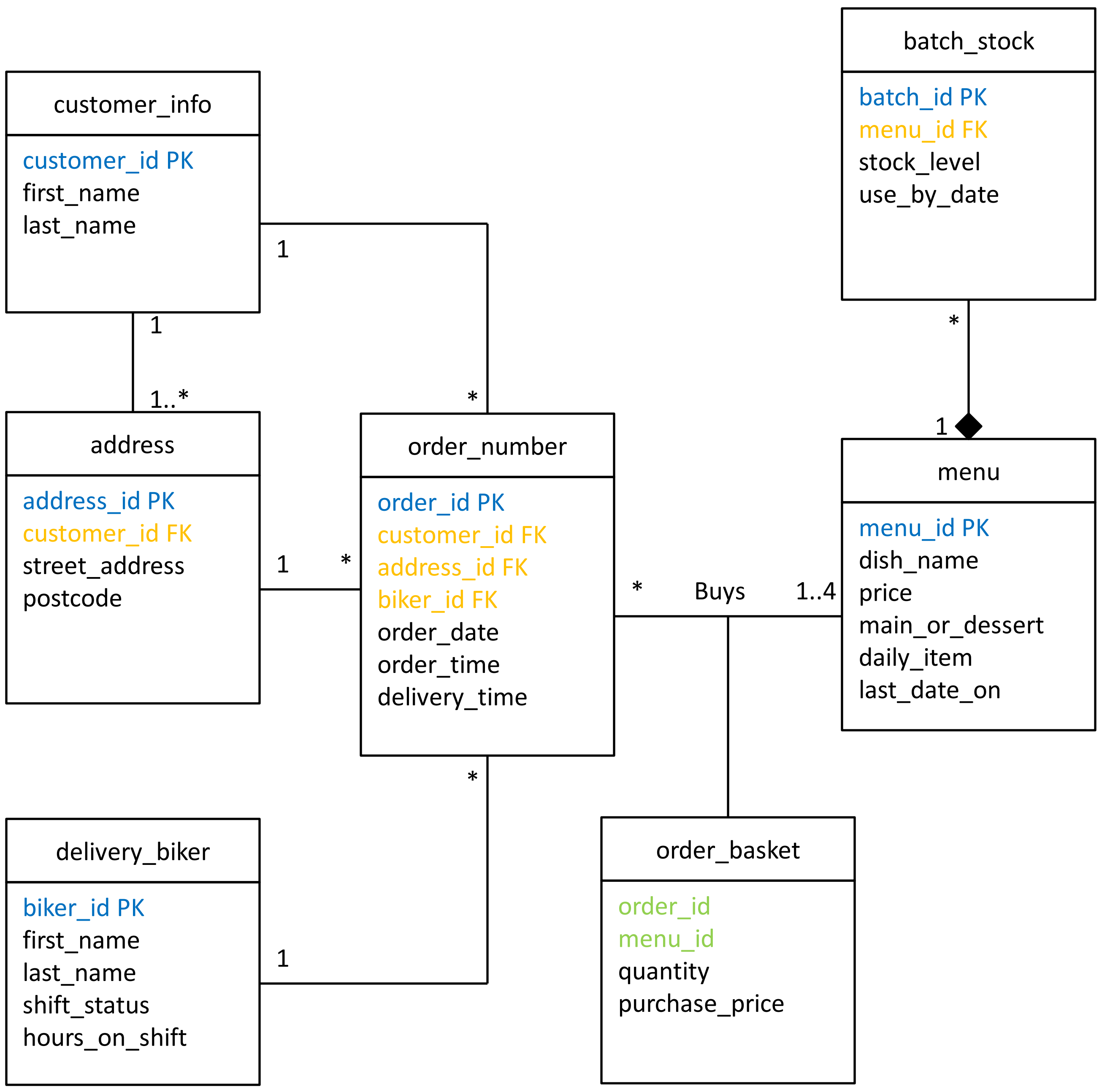
Express Food Domain Diagram (Alternate assuming food cold stores for more than 1 day)



Notes: As stated previously, it would be most efficient to sell all stock items as long as they are still within their use by date. In this diagram, menu items are stored for more than 1 day, with "useByDate" representing the last date that food can be sold. Thus, menu items could be reused on another day, even weeks later if cold stored frozen. The "lastDateOn" attribute can be used to prevent regular repeats of the same menu item so that, for example, an item could be repeated only after *n* number of days. So, for example, if ExpressFood had five "caramel slices" in stock, it would be advisable to cook more for daily item selection, thus on any day multiple batches could be available for the same dish type. The "useByDate" attribute also allows the company to easily rotate stock. The "dailyItem" attribute is Boolean and shows which dishes are on the menu for the current day.

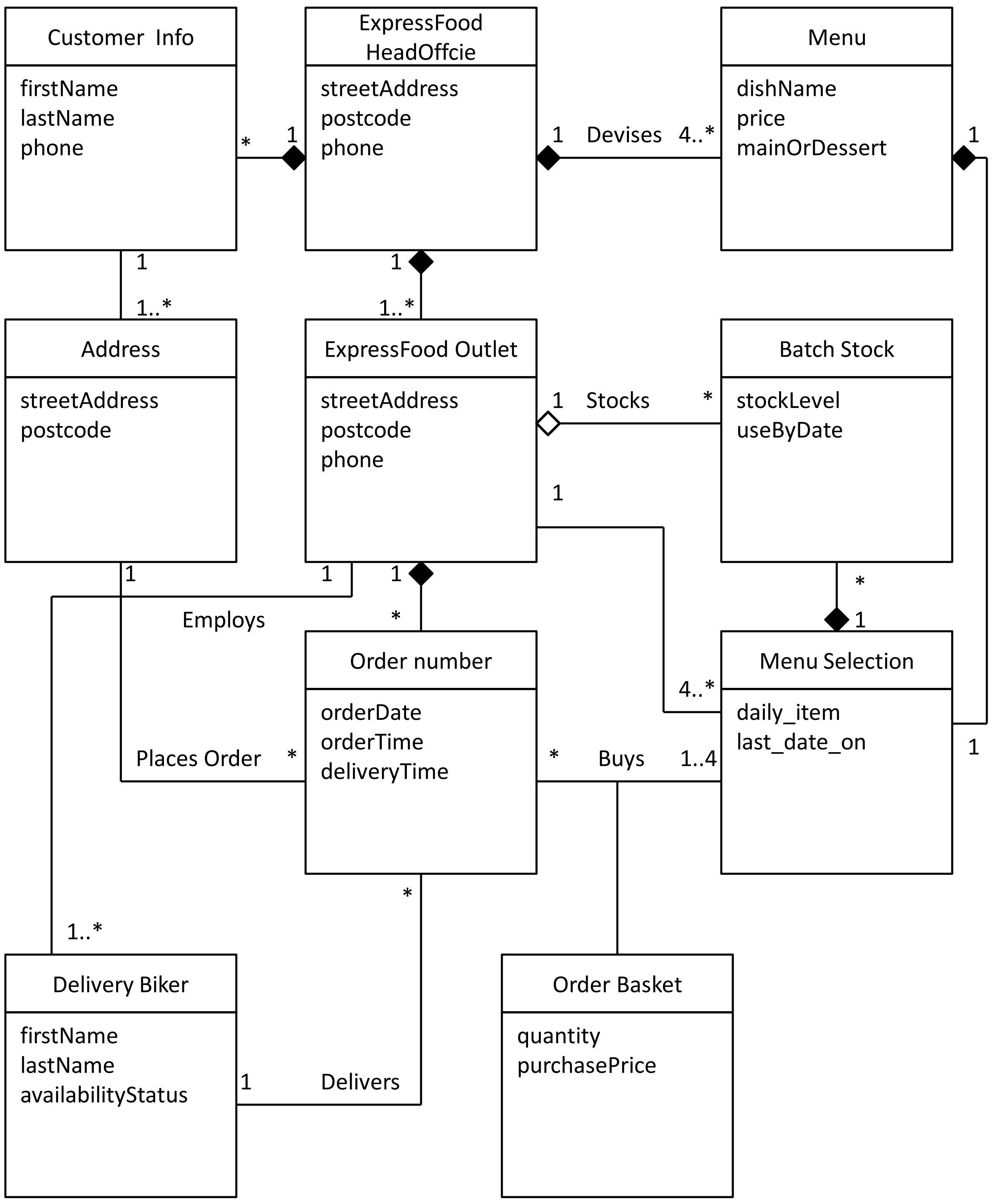
As before, the association table "Order Basket" separates which order and stock item correspond in any one order number. The purchase price in the "Order Basket" is dependent on the dish price at the time of purchase, but becomes independent thereafter. The dish price may change week to week, but the actual purchase price needs to be recorded for company accounting.

Express Food Domain Diagram (modified for database schema)



Notes: The same Domain Diagram as per previous slide but with Primary/Foreign Key identifiers added for database schema. This schema is the preferred schema because it allows expansion of the database when the company grows, whether through franchise or through direct company expansion.

Express Food Domain Diagram (Alternate with Store Class for *n* Outlets)



Notes: Example schema showing how the previous Domain Diagram, and database structure, can be scaled as the business grows. In this diagram, head office sets the types of dishes available (in the same way as the menus in all McDonalds are the same), but outlets control their stock and daily menu item selection. However, if cooking facilities were not available at an outlet, potentially stored food could be transferred from one outlet to another, hence the aggregation relationship.

SQL Database Queries

The database structure created allows the system user to retrieve a range of detailed data. This is especially useful for determining how many dishes were sold on any night, the remaining stock levels, use by dates etc. The below examples illustrate some useful SQL queries.

To find the quantity of a particular menu item sold on any given night, the following SQL query is used.

```
SELECT menu.dish_name, order_number.order_date, SUM(order_basket.quantity) total
FROM order_basket
JOIN order_number
ON order_number.order_id = order_basket.order_id
JOIN menu
ON menu.menu_id = order_basket.menu_id
WHERE menu.menu_id = 2;
```

The above query adds up all the sales of menu items with id = 2 (“Boeuf Bourguignon”) on each day and lists the sales by date.

The following SQL query lists the values of all sales on all days grouped by dish name and ordered by order date.

```
SELECT menu.dish_name,  order_number.order_date,
order_(basket.purchase_price*order_basket.quantity) total_daily_sales
FROM order_basket
JOIN order_number
ON order_number.order_id = order_basket.order_id
JOIN menu
ON menu.menu_id = order_basket.menu_id
GROUP BY menu.dish_name
ORDER BY order_number.order_date;
```

The following SQL query returns the summed sales on the date specified in “WHERE”.

```
SELECT order_number.order_date,
SUM(order_basket.purchase_price*order_basket.quantity) total
FROM order_basket
JOIN order_number
ON order_number.order_id = order_basket.order_id
WHERE order_number.order_date = '2018-12-19';
```