Many of the demos will use the following sets of tables belong to AltgeldMart. The tables will be in three different databases. With MySQL we can create relationships across databases.

I suggested creating the following databases in document 01-02. If you did not do that then, create them now. Login as root and create the three databases: employees, customers, orderEntry, and product. Then give your regular user account permissions to those databases.

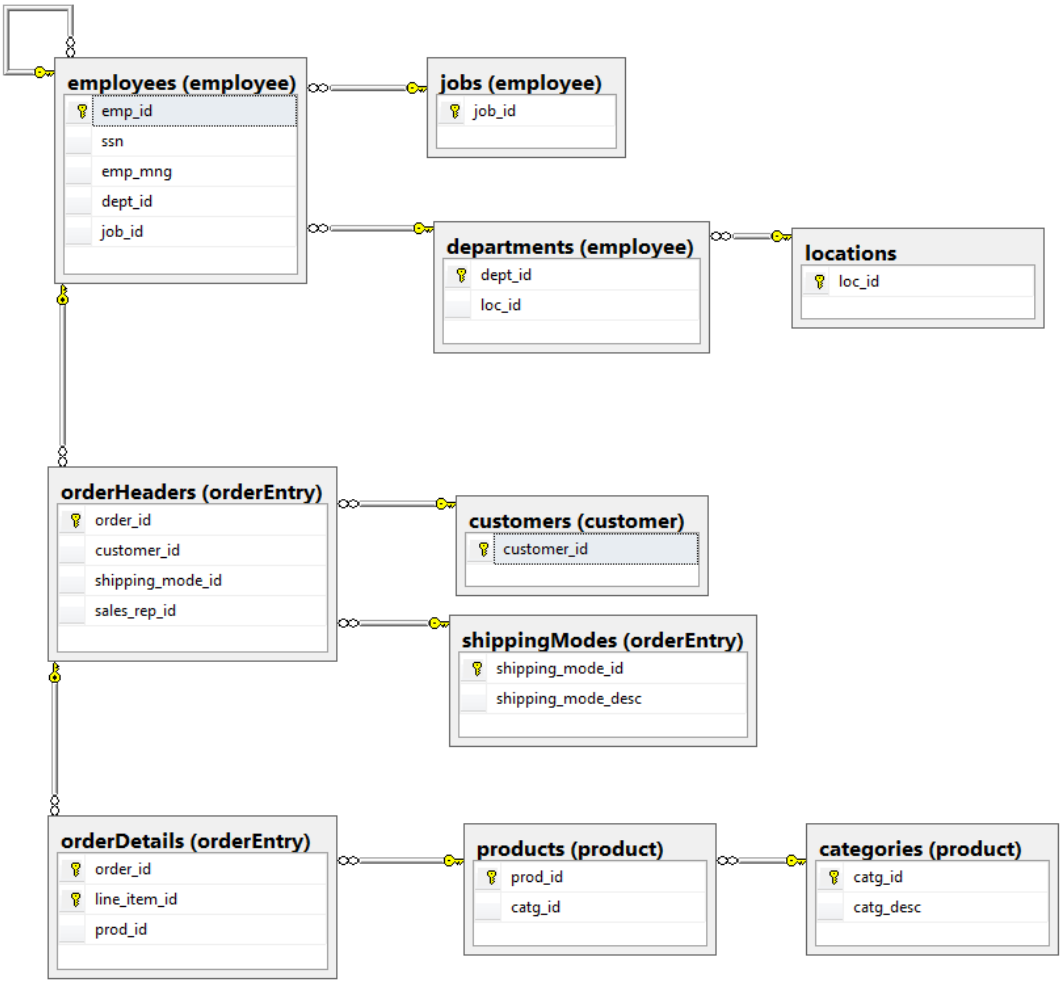
- The schema/database employee will store tables relating to employees
- The schema/database product will store tables relating to products we sell.
- The schema/database orderEntry will store tables relating to orders placed
- The schema/database customer will store tables relating to customers

This discusses the business rules and the relationships. You should read the Create Table statements for the details. Those statements are in the supplied scripts.

Our company (AltgeldMart) stores data about employees and the products sold and customer orders. These tables store only some of the data that would be needed by a company.

I am going to use diagrams produced by SQL Server because they have better diagramming tools. The table names will be the same; the database for the table is shown in parentheses. The column names are the same and the data types are compatible with MySQL.

This first diagram shows the relations between the tables. I have limited the display to show only the table names and the keys (pk and fk) attributes.



PRODUCT

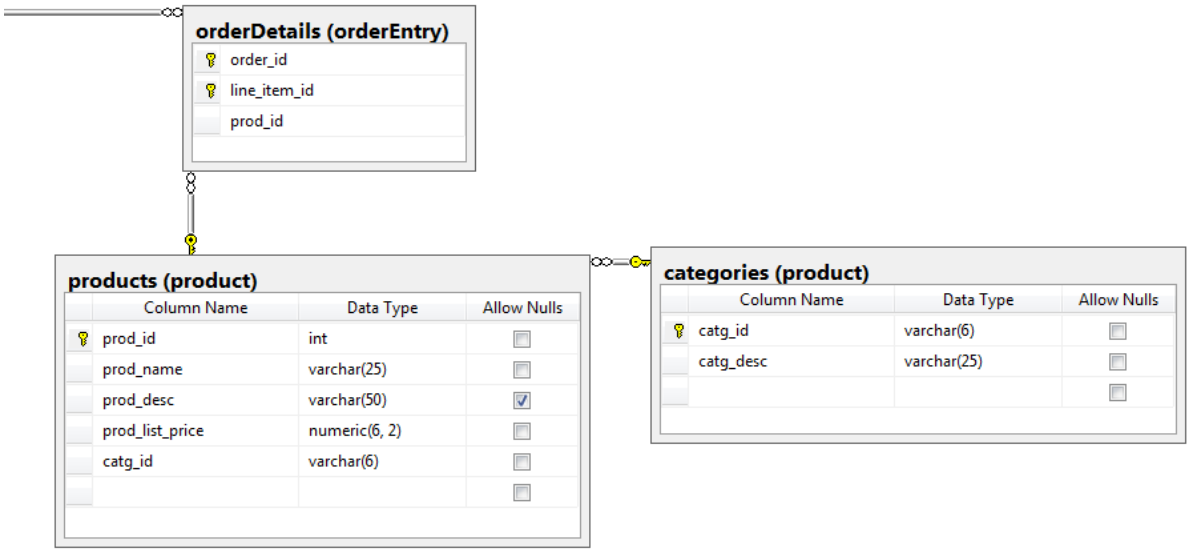
AltgeldMart sells a variety of products.

categories: Each product is classified as being in a category such as Housewares, Pet Supplies, Sporting Goods. This table is used to limit the values that can be entered as a category for a product. The table has a short value for each category as the primary key and a longer descriptive value. The descriptive values are unique. For example we have the rows:

catg_id	catg_desc
APL	APPLIANCES
BK	BOOKS

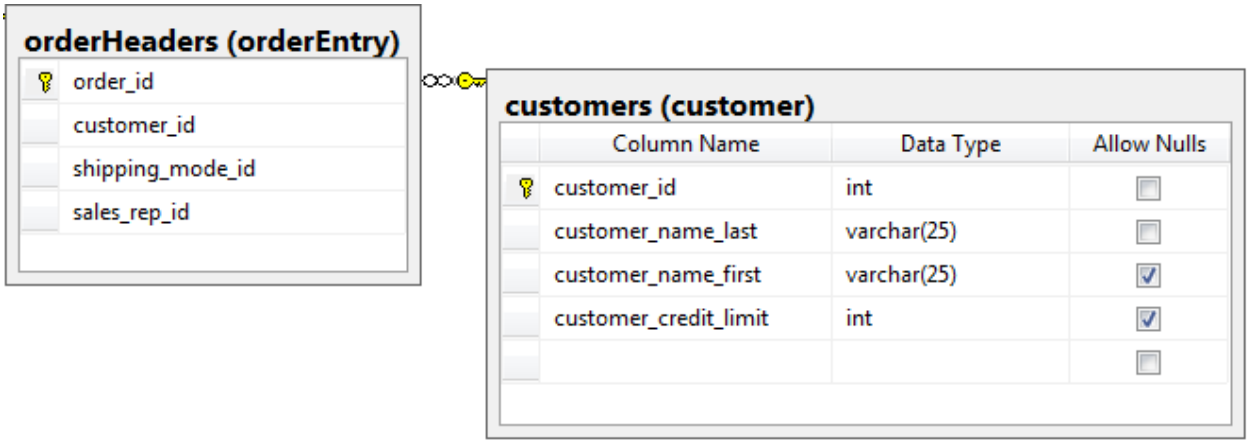
products: A product has a name (fairly short) and a description (which could be longer). A product has only a single category. Products also have a list price but we do not always sell items at their list price.

The diagram shows all of the columns and the data type and the nullability of the columns. It also shows how these tables relate to the orderDetails table



CUSTOMER

customers: For customers we are storing only the customer name and credit limit to keep the tables smaller.



ORDERENTRY

These are the tables associated with the order entry component.

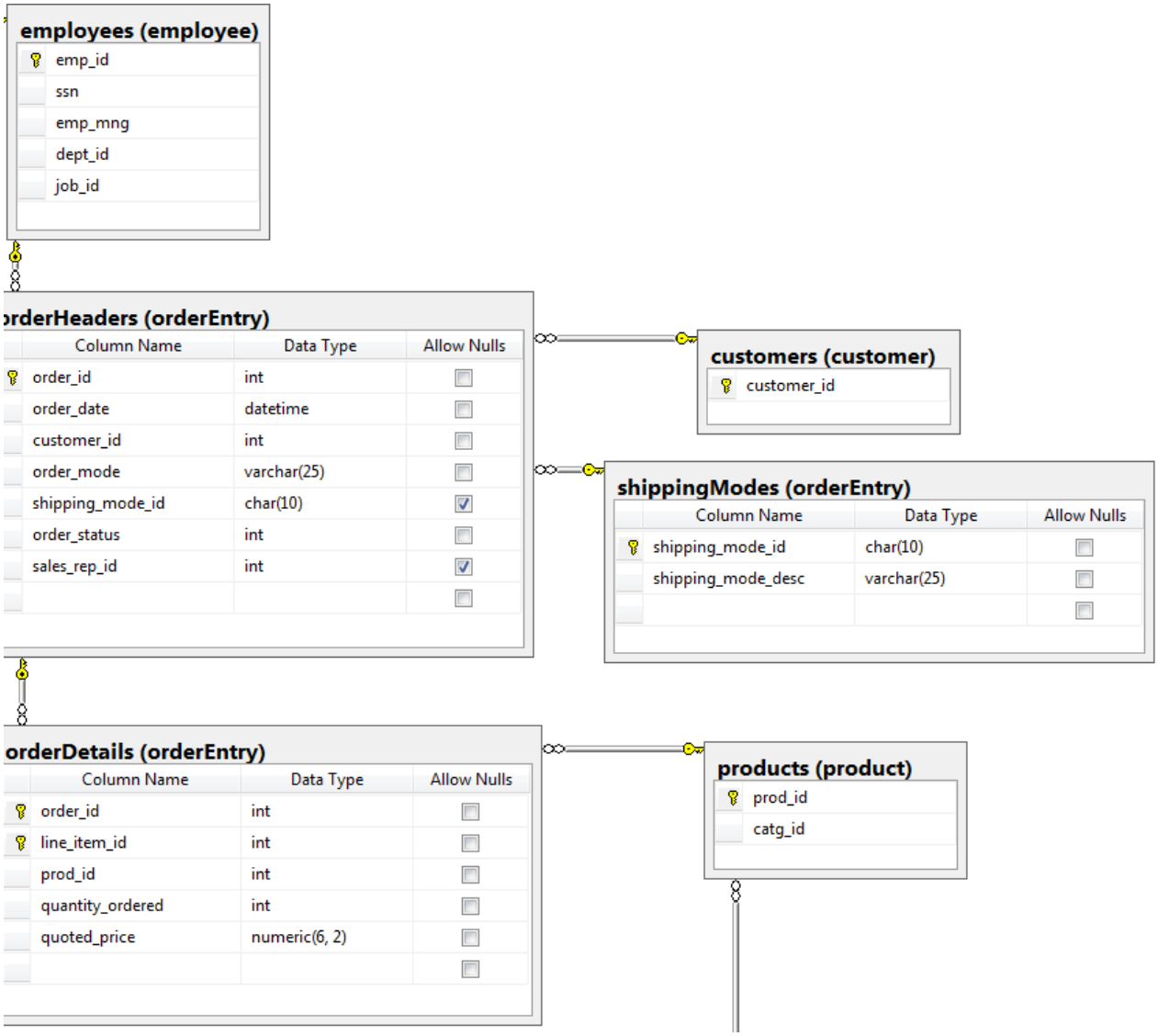
orderHeaders: Each sales order belongs to a single customer(customer_id) ; an order has an order_date, a order mode, a shipping mode and a numeric order status. For some sales we know the sales rep (employee) who handled the order.

orderDetails: The orderDetails tables includes the product being ordered, the quantity and the price at which the item was actually sold. By putting the actual sales price here, we could sell a product for less or more than its list price. If the list price changes, the actual price to the customer is not changed. The product id references the products table in the previous diagram.

The price in the orderDetails table is the price per item. For example, on order 114 the customer bought 5 mini freezers for $125.00 each. For this order the customer is charged $625.00 (5 * 125) plus any tax and shipping.

The orderDetails table include the order_id to link to the orderHeaders table. It also has a line_item_id- usually numbered 1,2,3, etc. Together order_id and line_item_id make up the primary key for the orderDetails table.

shippingModes: this table is similar to the product categories table. It lists the various types of shipping modes we use and each order is limited to one of these shipping modes.



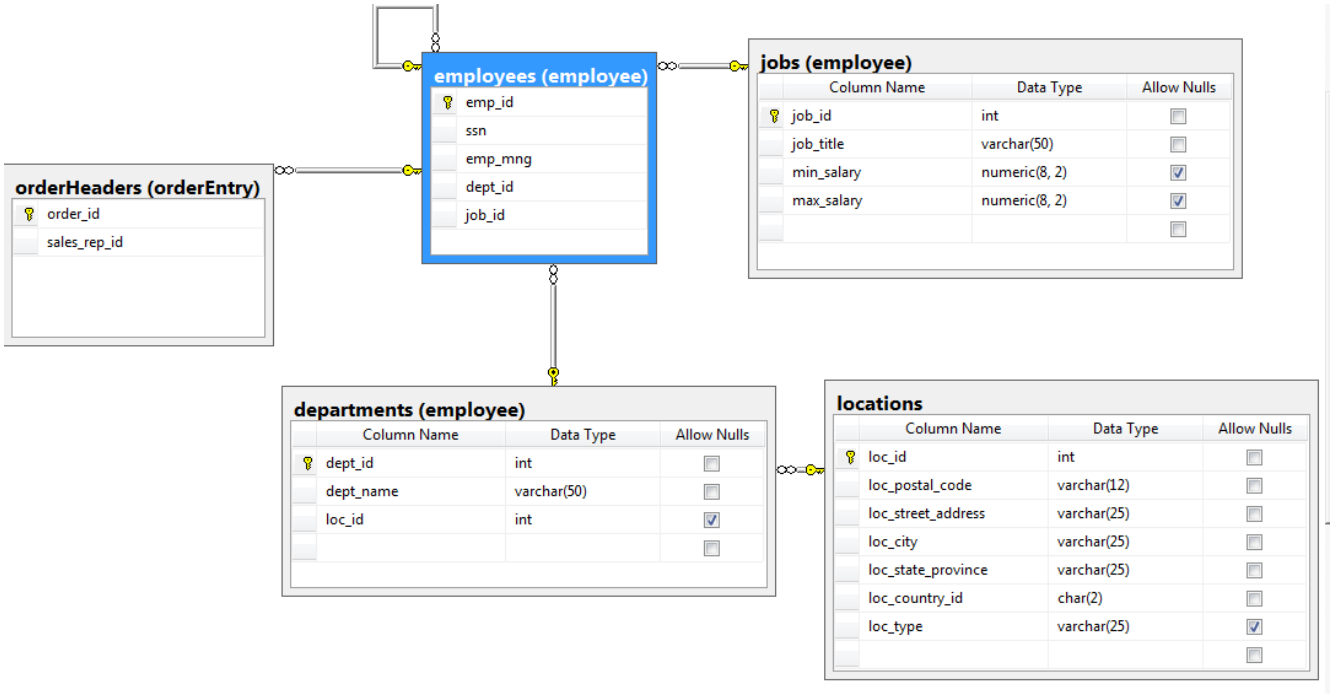
EMPLOYEE

employees: We store an employee's name. An employee has a single job title and is assigned to a single department. An employee may have a manager who is also an employee. This results in a relationship from the employees table back to itself. We store the employee's hire date and current salary.

jobs: Each employee has a single job title and there is a table of the valid job titles. This also contains a minimum and maximum salary for that job

departments: For each department, we store a department name and a single location.

locations: The locations for the company sites are stored in a table of locations; we store the address information for each location.



You might notice the looped relation from employees to employees. This is called a pig's ear. What this relation says is that an employee generally has a manager who is also an employee.

```
, constraint mng_emp_fk   foreign key (emp_mng) references
employee.employees(emp_id)
```