**Thinking About Tables SLIDE**

Now we look at a description of a product which will be a shoe in this situation. We define if it is business, casual or athletic. Brand, individual shoe name, size, color, price, discount, tax rate, and quantity.

**Create a Table product\_type**

This table will define if a product is considered business, casual or athletic

CREATE TABLE product\_type(

name VARCHAR(30) NOT NULL,

id SERIAL PRIMARY KEY);

**Create Product Table SLIDE**

We talked about how a primary key is used to uniquely identify a row in a table. A foreign key is used to identify 1 of a group of possible rows in another table.

If we create a product table and want to store a value from the product type table we can reference that information using a foreign key.

When creating a foreign key it has an integer type instead of a serial type. We can’t use serial because Postgres will try to assign a value to serial types.

-- type\_id references rows in the table product\_id and the row we are referencing matches the id

-- column

CREATE TABLE product(

type\_id INTEGER REFERENCES product\_type(id),

name VARCHAR(30) NOT NULL,

supplier VARCHAR(30) NOT NULL,

description TEXT NOT NULL,

id SERIAL PRIMARY KEY);

**Breaking Up Tables**

Table with Information that Differentiates Items of the Same Type NO SLIDE

This table describes just the quality of an item. If I were to list quantity here it would

make it hard to look at this as a single item. Quantity should be kept in a completely different

table if needed.

Anything that gets in the way of being able to model an individual object should be put in

another table.

When dealing with prices it is recommended to define Precision (Total number of digits) and Scale (How many digits in fraction).

The picture will be a url to the picture.

CREATE TABLE item(

product\_id INTEGER REFERENCES product(id),

size INTEGER NOT NULL,

color VARCHAR(30) NOT NULL,

picture VARCHAR(256) NOT NULL,

price NUMERIC(6,2) NOT NULL,

id SERIAL PRIMARY KEY);

**Sales Order Table**

Only information pertaining to the order is here aside from products and prices

It simulates 2 people agreeing to do business, the time of that event, a purchase order and

the means of payment

CREATE TABLE sales\_order(

cust\_id INTEGER REFERENCES customer(id),

sales\_person\_id INTEGER REFERENCES sales\_person(id),

time\_order\_taken TIMESTAMP NOT NULL,

purchase\_order\_number INTEGER NOT NULL,

credit\_card\_number VARCHAR(16) NOT NULL,

credit\_card\_exper\_month SMALLINT NOT NULL,

credit\_card\_exper\_day SMALLINT NOT NULL,

credit\_card\_secret\_code SMALLINT NOT NULL,

name\_on\_card VARCHAR(100) NOT NULL,

id SERIAL PRIMARY KEY

);

**Sales Item Table SLIDE**

Each item that is part of an order goes in its own table. It is linked to the order with

sales\_order\_id

This simulates picking up a quantity of an individual item, with a certain discount and tax rate

The item itself is hidden in the item table and 100% defined specifically there in a way that

makes it easy to refer to it just by its id

If I were to list color, size or anything else here we would break that ability to consider

items in a self contained way

CREATE TABLE sales\_item(

item\_id INTEGER REFERENCES item(id),

sales\_order\_id INTEGER REFERENCES sales\_order(id),

quantity INTEGER NOT NULL,

discount NUMERIC(3,2) NULL DEFAULT 0,

taxable BOOLEAN NOT NULL DEFAULT FALSE,

sales\_tax\_rate NUMERIC(5,2) NOT NULL DEFAULT 0,

id SERIAL PRIMARY KEY

);

**Foreign & Primary Keys SLIDE**

You can see here how foreign keys allow us to merge our data. When we start issuing queries it will become more clear how to use these keys.

Product type is linked to the product. The product is linked to the item which is a more specific version of our product. Then both the item and sales order is linked to the sales item table. There are many other foreign keys linking tables, but I think this is enough for now.