

The Quasar Database

Rui Peng

College of Engineering and Computer Science
Syracuse University, Syracuse, NY 13244, USA

Abstract

With the increasing importance and popularity of E-commerce, it's essential to develop an efficient and durable E-commerce platform for customers, merchants, suppliers, and shippers to facilitate deals. In this project, we design, implement, and test a database for our brand new E-commerce platform called Quasar (a contraction of quasi-stellar radio source). We strongly believe that Quasar will surpass Amazon in the near future since we have the greatest database design, ever.



Image by ESO/M. Kornmesser, CC BY 4.0

Design

Information to Store

We design our database using an entity-relationship model powered by Vertabelo, a straightforward tool for database design. We first define 5 main entities (Customers, Products, Suppliers, Orders, Warehouses) and then determine relationships between them. If there's a many-to-many relationship, a linking table may be added to represent it.

E-R Diagram

We make Products the central table of our ER diagram since it is directly (or through an additional linking table) connected to all other main tables:

Customers wish for Products: Many-to-many, add a linking table CustomerWishes.

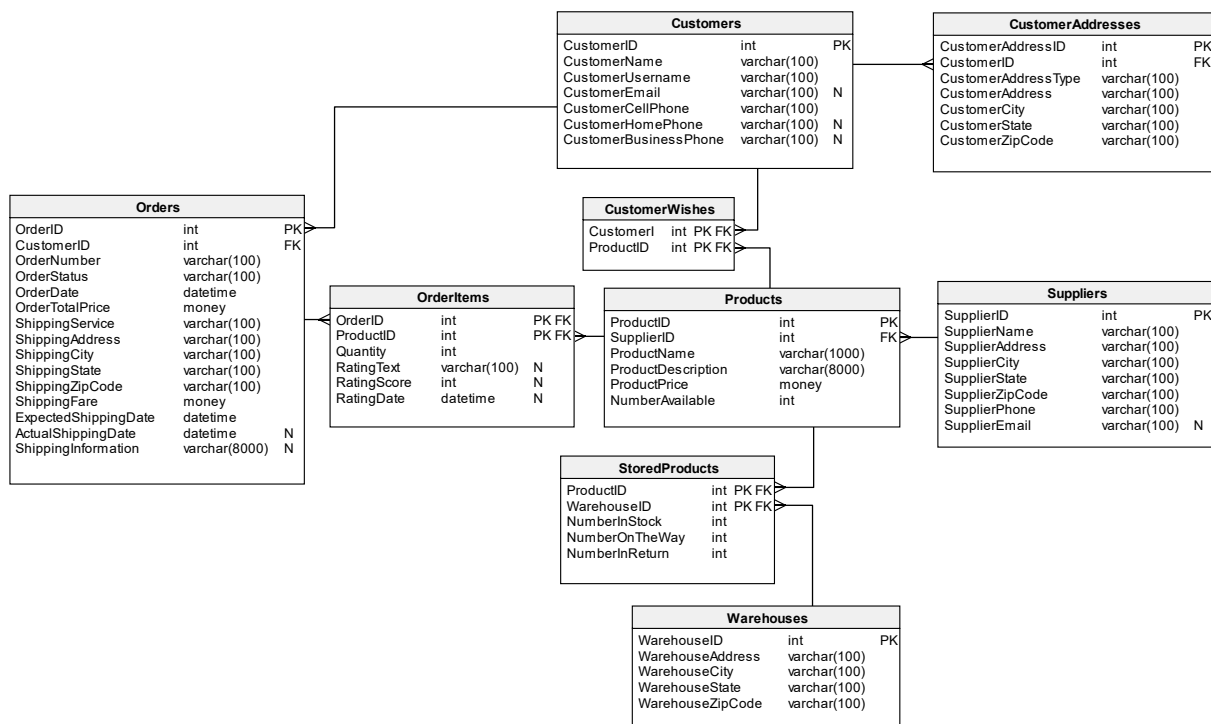
Orders contain Products: Many-to-many, add a linking table OrderItems which also contains product reviews.

Suppliers supply Products: One-to-many, add a foreign key in table Products.

Warehouses store Products: Many-to-many, add a linking table StoredProducts.

Customers place Orders: One-to-many, add a foreign key in table Orders.

We reserve 100 bytes for city, state, and zip code for possible international addresses, where the state shall be country/region name.



Column descriptions:

PK: Primary Key

FK: Foreign Key

N: Nullable, any column without N is non-null

Business Reports

We produce these business reports using views:

- Total sales for suppliers;
- Total money spent for customers;
- Average rating for each product;
- How many times each product is sold.

Business Logic

We add several constraints to enforce business logic:

- Almost all columns are required to be non-null, with few exceptions;
- Suppliers: Email can be null;
- Products: Check if price and number available are both ≥ 0 ;
- StoredProducts: Check if the number in stock, on the way, and in return are all ≥ 0 ;
- Customers: Email, home & business phone numbers can be null;
- Orders: Shipping information can be null, check if total price is ≥ 0 ;
- OrderItems: rating can be null, check if rating score is between 1 and 5;

We also add triggers to ensure correct action queries on our database:

- Suppliers on insert and update: check if email is valid;
- Customers on insert and update: check if email is valid;
- Orders on insert and update: order date (datetime) must be no later than current datetime
- Database: prevent dropping any table;

Security Levels

All views, triggers, stored procedures, and functions are created with encryptions to prevent users from accessing the underlying SQL code.

We create a login called Quasar with a must-change password 'MustChange' and assign it to server role dbcreator.

Performance and Efficiency Improvements

We create stored procedures and functions for the purpose of improving performance and efficiency.

We create 4 stored procedures to do these respectively:

- Customer purchases goods (Buy Now);
- Warehouse gets goods from supplier;
- Customer gives a rating;
- Customer returns an order.

We create 4 functions to do these respectively:

- Get overall total sales;
- Get total number in stock for a product;
- Get all warehouses in a state;
- Get all products whose price is lower than a given value.

Regular Business Transactions

Each stored procedure above is a regular business transaction. We've used SQL Server transactions to code all those stored procedures to enforce atomicity, consistency, isolation, and durability.

Implementation

Tables and Relationships

All source code is in the Appendix section of this report. A separate src file is also provided. In the Design part of this report, all tables, columns, keys, datatypes, nullabilities, relationships are determined. The screenshot of my design is the E-R diagram given above.

Normalization

The database satisfies BCNF constraints which is stricter than 3NF. More details about normalization are given in the Conclusion part.

Integrity and Security

All potential and security issues are addressed in the Design part. We have primary key in each table. We add a referential integrity constraint for each foreign key. We use password assignment, role, encryptions and triggers to enhance security and to prevent unauthorized access.

Testing

Create test data

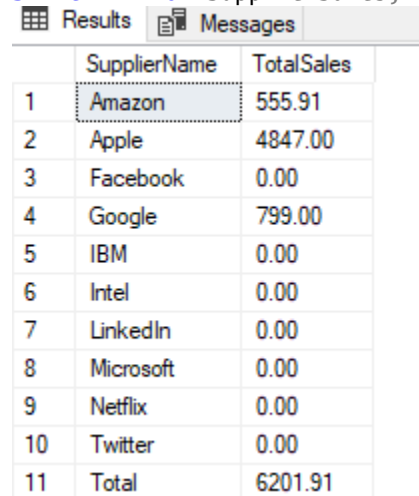
Each table is populated with test data in approximately 10 rows in the Implementation part.

Business Reports

We can get essential business reports simply by selecting from those views that we've created above.

```
-- show total sales for suppliers
```

```
SELECT * FROM SupplierSales;
```



The screenshot shows a database interface with a 'Results' tab selected. It displays a table with two columns: 'SupplierName' and 'TotalSales'. The table contains 11 rows of data, including 10 suppliers and a 'Total' row. The 'Amazon' row is highlighted.

	SupplierName	TotalSales
1	Amazon	555.91
2	Apple	4847.00
3	Facebook	0.00
4	Google	799.00
5	IBM	0.00
6	Intel	0.00
7	LinkedIn	0.00
8	Microsoft	0.00
9	Netflix	0.00
10	Twitter	0.00
11	Total	6201.91

```
-- show total money spent for customers
```

```
SELECT * FROM CustomerMoneySpent;
```

	CustomerName	TotalMoneySpent
1	Bike Day	0.00
2	Daisy Barbara	360.73
3	Haskell Day	2742.58
4	Juan Maria	23.76
5	Lemon Pie	0.00
6	Matt James	1635.69
7	Mozzarella Grant	1847.91
8	Round Robin	173.31
9	Salt Banana	0.00
10	Tony McDonald	0.00
11	Total	6783.98

-- show average rating for each product

SELECT * FROM ProductAverageRating;

	ProductName	SupplierName	AverageRating
1	All-new Echo (3rd Gen)	Amazon	No data
2	Echo Auto	Amazon	No data
3	Echo Dot (3rd Gen)	Amazon	3.33
4	Echo Show 5	Amazon	No data
5	Fire 7 Kids Edition Tablet	Amazon	No data
6	Fire 7 Tablet	Amazon	5.00
7	Kindle Paperwhite	Amazon	No data
8	iPad Pro	Apple	3.50
9	iPhone 11 Pro Max	Apple	No data
10	Pixel 4 XL	Google	1.00

-- show all customers who's spent more than \$1000

SELECT * FROM CustomerMoneySpent WHERE TotalMoneySpent > 1000;

	CustomerName	TotalMoneySpent
1	Haskell Day	2742.58
2	Matt James	1635.69
3	Mozzarella Grant	1847.91
4	Total	6783.98

-- show sales count for each product

SELECT * FROM ProductSalesCount;

	Product Name	Product Price	#Sold
1	Echo Dot (3rd Gen)	22.00	13
2	Echo Auto	29.99	3
3	Fire 7 Tablet	29.99	6
4	Echo Show 5	49.99	0
5	All-new Echo (3rd Gen)	59.99	0
6	Fire 7 Kids Edition Tablet	59.99	0
7	Kindle Paperwhite	84.99	0
8	Pixel 4 XL	799.00	1
9	iPhone 11 Pro Max	1449.00	1
10	iPad Pro	1699.00	2

Reliability

We test all triggers, functions, and then run a complex query that joins almost all tables (except CustomerWishes, since Customers is better to be linked to Products through Orders) together. We also join Customers, CustomerWishes, and Products together to ensure that CustomerWishes also works perfectly.

```
-- update a supplier with invalid email address
```

```
BEGIN TRY
    UPDATE Suppliers
    SET SupplierEmail = 'notvalid@.com'
    WHERE SupplierID = 3;
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE() AS ERROR;
END CATCH
```

	WARNING
1	SUPPLIERS: INVALID EMAIL

	ERROR
1	The transaction ended in the trigger. The batch has been aborted.

```
-- insert a customer with invalid email address
```

```
BEGIN TRY
    INSERT Customers VALUES (12, 'FAKE NAME', 'FAKE ID', '@FAKE.edu', '123-456-7890',
NULL, NULL);
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE() AS ERROR;
END CATCH
```

Results Messages	
WARNING	
1	CUSTOMERS: INVALID EMAIL

ERROR	
1	The transaction ended in the trigger. The batch has been aborted.

```
-- update an order with invalid order date
BEGIN TRY
    UPDATE Orders
    SET OrderDate = '2099-12-09'
    WHERE OrderID = 3;
```

```
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE() AS ERROR;
END CATCH
```

Results Messages	
WARNING	
1	ORDERS: INVALID DATE

ERROR	
1	The transaction ended in the trigger. The batch has been aborted.

```
-- attempt to drop a table
BEGIN TRY
    DROP TABLE CustomerAddresses;
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE() AS ERROR;
END CATCH
```

Results Messages	
WARNING	
1	SYSTEM: YOUR SUSPICIOUS BEHAVIOUR IS DENIED AND RECODED

ERROR	
1	The transaction ended in the trigger. The batch has been aborted.

```
-- get overall total sales
SELECT dbo.fnGetOverallTotalSales() AS OverallTotalSales;
```

Results	Messages
OverallTotalSales	
1	6201.91

```
-- get total number in stock for Echo Dot (3th Gen)
SELECT dbo.fnGetProductTotalInStock(1) AS EchoDotNumberInStore;
```

Results	Messages
EchoDotNumberInStore	
1	13702

```
-- get all California warehouses
SELECT * FROM dbo.fnGetWarehousesByState('CA');
```

Results Messages

	WarehouseID	WarehouseAddress	WarehouseCity	WarehouseState	WarehouseZipCode
1	2	9031 Lurline Ave	Los Angeles	CA	91311
2	8	16550 Via Esprillo	San Diego	CA	92127
3	10	96 E San Fernando St	San Jose	CA	95113

```
-- get all products who's cheaper than $100
SELECT * FROM dbo.fnGetProductsWithPriceBelow(100);
```

Results

Messages

	ProductID	SupplierID	ProductName	ProductDescription	ProductPrice	NumberAvailable
1	1	1	Echo Dot (3rd Gen)	Smart speaker with Alexa, Charcoal	22.00	77539
2	2	1	Fire 7 Tablet	7" display, 16 GB, Black	29.99	9349
3	3	1	Echo Auto	Add Alexa to your car	29.99	13775
4	4	1	Echo Show 5	Compact smart display with Alexa, Charcoal	49.99	10093
5	5	1	Kindle Paperwhite	Now waterproof with 2x the storage	84.99	14207
6	6	1	Fire 7 Kids Edition Tablet	7" display, 16 GB, Blue kid-proof case	59.99	2992
7	7	1	All-new Echo (3rd Gen)	Smart speaker with Alexa, Charcoal	59.99	480

```
-- join all tables but CustomerWishes
SELECT *
FROM Suppliers JOIN Products ON Suppliers.SupplierID = Products.SupplierID
JOIN StoredProducts ON StoredProducts.ProductID = Products.ProductID
JOIN Warehouses ON StoredProducts.WarehouseID = Warehouses.WarehouseID
JOIN OrderItems ON Products.ProductID = OrderItems.ProductID
JOIN Orders ON Orders.OrderID = OrderItems.OrderID
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
JOIN CustomerAddresses ON CustomerAddresses.CustomerID = Customers.CustomerID;
```

Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													
Customer: 10													

Query executed successfully.

Rui (14.0 RTM) | RUIV41248 (56) | Quasar | 00:00:00 | 70 rows


```
-- join Customers, CustomerWishes, and Products
SELECT CustomerName, ProductName AS Wish
FROM Customers JOIN CustomerWishes ON Customers.CustomerID = CustomerWishes.CustomerID
      JOIN Products ON CustomerWishes.ProductID = Products.ProductID;
```

	CustomerName	Wish
1	Daisy Barbara	Echo Dot (3rd Gen)
2	Daisy Barbara	Fire 7 Tablet
3	Daisy Barbara	Echo Show 5
4	Daisy Barbara	Kindle Paperwhite
5	Daisy Barbara	Pixel 4 XL
6	Matt James	Echo Dot (3rd Gen)
7	Matt James	Fire 7 Tablet
8	Juan Maria	Fire 7 Tablet
9	Mozzarella Grant	Echo Dot (3rd Gen)
10	Haskell Day	iPad Pro

Transactions

We test regular business transactions simply by executing their corresponding stored procedures.

```
-- customer purchase goods
SELECT ProductID, NumberAvailable FROM Products WHERE ProductID = 5;
SELECT Orders.OrderID, CustomerID, OrderStatus, OrderDate, OrderTotalPrice,
      ShippingAddress, ShippingCity, ShippingState, ShippingZipCode,
      ExceptedShippingDate, Quantity
FROM Orders JOIN OrderItems ON Orders.OrderID = OrderItems.OrderID
WHERE CustomerID = 6 AND ProductID = 5;
EXEC spBuyNow 6, 5;
SELECT ProductID, NumberAvailable FROM Products WHERE ProductID = 5;
SELECT Orders.OrderID, CustomerID, OrderStatus, OrderDate, OrderTotalPrice,
      ShippingAddress, ShippingCity, ShippingState, ShippingZipCode,
      ExceptedShippingDate, Quantity
FROM Orders JOIN OrderItems ON Orders.OrderID = OrderItems.OrderID
WHERE CustomerID = 6 AND ProductID = 5;
```

ProductID	NumberAvailable
1	5
5	14207

OrderID	CustomerID	OrderStatus	OrderDate	OrderTotalPrice	ShippingAddress	ShippingCity	ShippingState	ShippingZipCode	ExceptedShippingDate	Quantity
1	12	Ready to go	2019-12-01 17:06:36.370	97.7892	100 College Pl	Syracuse	NY	13210	2019-12-02 17:06:36.370	1

ProductID	NumberAvailable
1	5
5	14206

A new order is placed with order number, status, date, total price, shipping address, expected shipping date automatically filled. We only need CustomerID, ProductID, and quantity (optional) to place an “Buy Now” order. The NumberAvailable decremented.

```
-- warehouse gets goods from supplier
SELECT * FROM StoredProducts WHERE WarehouseID = 7;
EXEC spGetGoods 10, 7, 6;
SELECT * FROM StoredProducts WHERE WarehouseID = 7;
```

Results		Messages			
	ProductID	WarehouseID	NumberInStock	NumberOnTheWay	NumberInReturn
1	8	7	32	100	50

	ProductID	WarehouseID	NumberInStock	NumberOnTheWay	NumberInReturn
1	8	7	32	100	50
2	10	7	6	0	0

Now Warehouse 7 has gotten 6 iPad Pros.

```
-- customer gives a rating
SELECT * FROM OrderItems WHERE OrderID = 1 AND ProductID = 3;
EXEC spGiveRating 1, 3, 'Good Product. My favorite !', 5;
SELECT * FROM OrderItems WHERE OrderID = 1 AND ProductID = 3;
```

Results		Messages				
	OrderID	ProductID	Quantity	RatingText	RatingScore	RatingDate
1	1	3	1	NULL	NULL	NULL

	OrderID	ProductID	Quantity	RatingText	RatingScore	RatingDate
1	1	3	1	Good Product. My favorite !	5	2019-12-01 16:53:52.440

Now the new rating has been given.

```
-- customer returns an order
SELECT * FROM Orders WHERE OrderID = 7;
EXEC spReturnOrder 7;
SELECT * FROM Orders WHERE OrderID = 7;
```

Results		Messages								
	OrderID	CustomerID	OrderNumber	OrderStatus	OrderDate	OrderTotalPrice	ShippingService	ShippingAddress	ShippingCity	Ship
1	7	5	VBNMGHJ-733	Delivered	2019-11-21 00:00:00.000	1847.91	USPS	222 Waverly Ave	Syracuse	NY

<										
	OrderID	CustomerID	OrderNumber	OrderStatus	OrderDate	OrderTotalPrice	ShippingService	ShippingAddress	ShippingCity	Ship
1	7	5	VBNMGHJ-733	Return	2019-11-21 00:00:00.000	1847.91	USPS	222 Waverly Ave	Syracuse	NY

Now the status has been changed to Return. Further processing shall be done upon receiving the returned goods.

Conclusion

Analysis and Remarks

This project is time-consuming, during which I've written hundreds of codes on my own. Normalization is essential but we actually don't have to follow it extremely strictly. We also need to make the database

easy to use for both SQL and application programmers since sometimes over-normalization could make the database more difficult to understand.

The definitions of 3NF and BCNF in the textbook and slides are oversimplified or incorrect. I've checked some other materials and summarized some basic concepts in normalization.

A summary of basic concepts in normalization

Functional Dependency

Let R be a relational schema, X and Y be nonempty sets of attributes (columns) of R . A relational schema R satisfies the functional dependency $X \rightarrow Y$ (X functional determines Y) if the following holds for every pair of tuples (rows) t_1 and t_2 in every possible legal instance of R :

If $t_1.X = t_2.X$, then $t_1.Y = t_2.Y$.

If $t_1.X$ appears only once (for example, phone number, SSN, email address), there isn't such a pair and thus we can't say there's such a functional dependency even if it looks like.

Superkey and candidate key

Superkey: a set of attributes that uniquely specifies a tuple (row). No two distinct tuples (rows) have the same values in a superkey.

Candidate key: a minimal superkey (any proper subset of a candidate key is not a superkey).

Prime attribute: an attribute in a candidate key.

Let X be some set of attributes, Y be the set of all attributes. If $X \rightarrow Y$, then X is a superkey. If $X \rightarrow Y$ and there's no proper subset V of X such that $V \rightarrow Y$, then X is a candidate key.

Boyce-Codd Normal Form

Let R be a relational schema, F be the set of functional dependencies given to hold over R , X be a nonempty subset of attributes of R , A be an attribute of R . R is in Boyce-Codd Normal Form if for every functional dependency $X \rightarrow A$ in F , one of the following holds:

$A \in X$ (trivial functional dependency), or

X is a superkey.

Each tuple is an entity or relationship identified by a key and described by the remaining attributes. No redundancy can be detected using functional dependencies alone. No field of a tuple can be inferred by using only functional dependencies.

BCNF Decomposition

Let $X \rightarrow A$ be a functional dependency that violates BCNF. Decompose R into $R-A$ and XA . Do this recursively until BCNF is satisfied.

Future Work

Consider adding a shopping cart for each customer. Each customer can modify the shopping cart and place an order with all items in the cart.

Appendix

```
-- DB init
USE master;
GO
DROP DATABASE IF EXISTS Quasar;
GO
CREATE DATABASE Quasar;
```

```

GO
USE Quasar;
GO

-- create and init table Suppliers
DROP TABLE IF EXISTS Suppliers;
GO
CREATE TABLE Suppliers (
    SupplierID int PRIMARY KEY,
    SupplierName varchar(100) NOT NULL,
    SupplierAddress varchar(100) NOT NULL,
    SupplierCity varchar(100) NOT NULL,
    SupplierState varchar(100) NOT NULL,
    SupplierZipCode varchar(100) NOT NULL,
    SupplierPhone varchar(100) NOT NULL,
    SupplierEmail varchar(100)
);
GO
INSERT Suppliers VALUES
    (1, 'Amazon', '410 Terry Ave N', 'Seattle', 'WA', '98109', '800-280-4331',
    'primary@amazon.com'),
    (2, 'Google', '1600 Amphitheatre Pkwy', 'Mountain View', 'CA', '94043', '800-555-1212', NULL),
    (3, 'Apple', '1 Apple park Way', 'Cupertino', 'CA', '95014', '408-996-1010', NULL),
    (4, 'Microsoft', '1 Microsoft Way', 'Redmond', 'WA', '98052', '425-882-8080', NULL),
    (5, 'Facebook', '1 Hacker Way', 'Monlo Park', 'CA', '94025', '650-853-1300', NULL),
    (6, 'Twitter', '1355 Market St, Ste 900', 'San Francisco', 'CA', '94103', '415-222-9670',
    'support@twitter.com'),
    (7, 'LinkedIn', '605 W Maude Ave', 'Sunnyvale', 'CA', '94085', '650-687-3600',
    'press@linkedin.com'),
    (8, 'IBM', '1 New Orchard Rd', 'Armonk', 'NY', '10504', '914-765-1900', NULL),
    (9, 'Intel', '2200 Mission College Blvd', 'Santa Clara', 'CA', '95054', '408-765-8080', NULL),
    (10, 'Netflix', '100 Winchester Cir', 'Los Gatos', 'CA', '95032', '408-540-3700', NULL)
;
GO

-- create and init table Products
DROP TABLE IF EXISTS Products;
GO
CREATE TABLE Products (
    ProductID int PRIMARY KEY,
    SupplierID int NOT NULL REFERENCES Suppliers(SupplierID),
    ProductName varchar(1000) NOT NULL,
    ProductDescription varchar(8000) NOT NULL,
    ProductPrice money NOT NULL CHECK (ProductPrice >= 0),
    NumberAvailable int NOT NULL CHECK (NumberAvailable >= 0)
);
GO
INSERT Products VALUES

```

```

(1, 1, 'Echo Dot (3rd Gen)', 'Smart speaker with Alexa, Charcoal', 22.00, 77539), -- Amazon
(2, 1, 'Fire 7 Tablet', '7" display, 16 GB, Black', 29.99, 9349),
(3, 1, 'Echo Auto', 'Add Alexa to your car', 29.99, 13775),
(4, 1, 'Echo Show 5', 'Compact smart display with Alexa, Charcoal', 49.99, 10093),
(5, 1, 'Kindle Paperwhite', 'Now waterproof with 2x the storage', 84.99, 14207),
(6, 1, 'Fire 7 Kids Edition Tablet', '7" display, 16 GB, Blue kid-proof case', 59.99, 2992),
(7, 1, 'All-new Echo (3rd Gen)', 'Smart speaker with Alexa, Charcoal', 59.99, 480),
(8, 2, 'Pixel 4 XL', '6.3"', 'Clearly white, 128 GB', 799, 625), -- Google
(9, 3, 'iPhone 11 Pro Max', 'Gold, 512 GB capacity, unlocked and SIM-free', 1449, 3125), -- Apple
(10, 3, 'iPad Pro', '12.9"', 'Silver, 1 TB storage, Wi-Fi + Cellular connectivity', 1699, 572)
;
GO

```

-- create and init table Warehouses

```
DROP TABLE IF EXISTS Warehouses;
```

```
GO
```

```
CREATE TABLE Warehouses (
    WarehouseID int PRIMARY KEY,
    WarehouseAddress varchar(100) NOT NULL,
    WarehouseCity varchar(100) NOT NULL,
    WarehouseState varchar(100) NOT NULL,
    WarehouseZipCode varchar(100) NOT NULL

```

```
);
```

```
GO
```

```
INSERT Warehouses VALUES
```

```

(1, '6 W 35th St', 'New York', 'NY', '10001'),
(2, '9031 Lurline Ave', 'Los Angeles', 'CA', '91311'),
(3, '2801 S Western Ave', 'Chicago', 'IL', '60608'),
(4, '10550 Ella Blvd', 'Houston', 'TX', '77038'),
(5, '6835 W Buckeye Rd', 'Phoenix', 'AZ', '85043'),
(6, '2235 Castor Ave', 'Philadelphia', 'PA', '19134'),
(7, '1410 S Callaghan Rd', 'San Antonio', 'TX', '78227'),
(8, '16550 Via Esprillo', 'San Diego', 'CA', '92127'),
(9, '33333 Lyndon B Johnson Fwy', 'Dallas', 'TX', '75241'),
(10, '96 E San Fernando St', 'San Jose', 'CA', '95113')

```

```
;
```

```
GO
```

-- create and init linking table StoredProducts

```
DROP TABLE IF EXISTS StoredProducts;
```

```
GO
```

```
CREATE TABLE StoredProducts (
    ProductID int NOT NULL REFERENCES Products(ProductID),
    WarehouseID int NOT NULL REFERENCES Warehouses(WarehouseID),
    NumberInStock int NOT NULL CHECK (NumberInStock >= 0),
    NumberOnTheWay int NOT NULL CHECK (NumberOnTheWay >= 0),
    NumberInReturn int NOT NULL CHECK (NumberInReturn >= 0),
    PRIMARY KEY (ProductID, WarehouseID)

```

```

);
GO
INSERT StoredProducts VALUES
    (1, 1, 12345, 1234, 123), -- Amazon Echo Dot (3rd Gen) stored in Warehouse 1, 2, 3
    (1, 2, 1234, 123, 12),
    (1, 3, 123, 12, 1),
    (2, 2, 2345, 234, 23), -- Amazon Fire 7 Tabler stored in Warehouse 2, 3, 4
    (2, 3, 234, 23, 2),
    (2, 4, 199, 19, 8),
    (8, 7, 20, 100, 50), -- Google Pixel 4 XL stored in Warehouse 7, 8
    (8, 8, 219, 20, 2),
    (9, 8, 1863, 7, 109), -- Apple iPhone 11 Pro Max stored in Warehouse 8
    (10, 10, 300, 17, 5) -- Apple iPad Pro stored in Warehouse 10
;
GO

-- create and init table Customers
-- all customers here are not related to any real-world individual and are just for test purposes.
-- DON'T CALL THESE NUMBERS OR EMAIL THESE ADDRESSES IN YOUR REAL LIFE
DROP TABLE IF EXISTS Customers;
GO
CREATE TABLE Customers (
    CustomerID int PRIMARY KEY,
    CustomerName varchar(100) NOT NULL,
    CustomerUsername varchar(100) NOT NULL,
    CustomerEmail varchar(100),
    CustomerCellPhone varchar(100) NOT NULL,
    CustomerHomePhone varchar(100),
    CustomerBusinessPhone varchar(100)
)
GO
INSERT Customers VALUES
    (1, 'Daisy Barbara', 'dbarbara109', 'dbarbara109@syr.edu', '315-443-1234', NULL, NULL),
    (2, 'Matt James', 'mjames77', 'mjames77@syr.edu', '315-443-9699', NULL, '315-443-8063'),
    (3, 'Juan Maria', 'jmaria09', 'jmaria09@syr.edu', '315-443-2000', '315-443-2905', '315-443-2693'),
    (4, 'Lemon Pie', 'idonteatpizza', 'lpie01@syr.edu', '315-443-7356', NULL, '315-443-2123'),
    (5, 'Mozzarella Grant', 'mgrant01', 'mgrant01@syr.edu', '315-443-1836', NULL, NULL),
    (6, 'Salt Banana', 'sbanana03', 'sbanana03@syr.edu', '315-443-6333', NULL, NULL),
    (7, 'Round Robin', 'rrobin', 'rrobin@syr.edu', '315-443-0000', NULL, NULL),
    (8, 'Tony McDonald', 'greatestrecipe', 'tmcdonald@syr.edu', '315-443-2799', '315-443-7763', NULL),
    (9, 'Bike Day', 'bday693', 'bday693@syr.edu', '315-443-1693', NULL, NULL),
    (10, 'Haskell Day', 'hday355', 'hday355@syr.edu', '315-443-4903', NULL, NULL)
;
GO

-- create and init linking table CustomerWishes

```

```

DROP TABLE IF EXISTS CustomerWishes;
GO
CREATE TABLE CustomerWishes (
    CustomerID int NOT NULL,
    ProductID int NOT NULL,
    PRIMARY KEY (CustomerID, ProductID)
);
GO
INSERT CustomerWishes VALUES
    (1, 1), -- Daisy wants Echo Dot 3, Fire 7, Echo Show 5, Kindle Paperwhite, and Pixel 4 XL
    (1, 2),
    (1, 4),
    (1, 5),
    (1, 8),
    (2, 1), -- Matt wants Echo Dot 3 and Fire 7
    (2, 2),
    (3, 2), -- Juan wants Fire 7
    (5, 1), -- Mozzarella wants Echo Dot 3
    (10, 10) -- Haskell wants iPad Pro
;
GO

-- create and init table CustomerAddresses
DROP TABLE IF EXISTS CustomerAddresses
GO
CREATE TABLE CustomerAddresses (
    CustomerAddressID int PRIMARY KEY,
    CustomerID int NOT NULL REFERENCES Customers(CustomerID),
    CustomerAddressType varchar(100) NOT NULL,
    CustomerAddress varchar(100) NOT NULL,
    CustomerCity varchar(100) NOT NULL,
    CustomerState varchar(100) NOT NULL,
    CustomerZipCode varchar(100) NOT NULL
);
GO
INSERT CustomerAddresses VALUES
    (1, 1, 'Shipping', '111 College Pl', 'Syracuse', 'NY', '13210'),
    (2, 1, 'Billing', '721 Univ Ave', 'Syracuse', 'NY', '13210'),
    (3, 2, 'Shipping', '900 S Crouse Ave', 'Syracuse', 'NY', '13244'),
    (4, 2, 'Billing', '900 S Crouse Ave', 'Syracuse', 'NY', '13244'),
    (5, 3, 'Shipping', '721 Univ Ave', 'Syracuse', 'NY', '13244'),
    (6, 3, 'Billing', '900 Irving Ave', 'Syracuse', 'NY', '13244'),
    (7, 4, 'Shipping', '100 Crouse Dr', 'Syracuse', 'NY', '13244'),
    (8, 4, 'Billing', '100 Crouse Dr', 'Syracuse', 'NY', '13244'),
    (9, 5, 'Shipping', '222 Waverly Ave', 'Syracuse', 'NY', '13244'),
    (10, 5, 'Billing', '222 Waverly Ave', 'Syracuse', 'NY', '13244'),
    (11, 6, 'Shipping', '100 College Pl', 'Syracuse', 'NY', '13210'),
    (12, 6, 'Billing', '100 College Pl', 'Syracuse', 'NY', '13210'),

```

```

(13, 7, 'Shipping', '150 Sims Dr', 'Syracuse', 'NY', '13244'),
(14, 7, 'Billing', '150 Sims Dr', 'Syracuse', 'NY', '13244'),
(15, 8, 'Shipping', '111 College Pl', 'Syracuse', 'NY', '13210'),
(16, 8, 'Billing', '111 College Pl', 'Syracuse', 'NY', '13210'),
(17, 9, 'Shipping', '111 College Pl', 'Syracuse', 'NY', '13210'),
(18, 9, 'Billing', '111 College Pl', 'Syracuse', 'NY', '13210'),
(19, 10, 'Shipping', '130 Sims Dr', 'Syracuse', 'NY', '13244'),
(20, 10, 'Shipping', '303 Univ Pl', 'Syracuse', 'NY', '13210'),
(21, 10, 'Shipping', '215 Univ Pl', 'Syracuse', 'NY', '13210'),
(22, 10, 'Billing', '720 Univ Ave', 'Syracuse', 'NY', '13210'),
(23, 10, 'Billing', '111 College Pl', 'Syracuse', 'NY', '13244')
;
GO

-- create and init table Orders
DROP TABLE IF EXISTS Orders;
GO
CREATE TABLE Orders (
    OrderID int PRIMARY KEY,
    CustomerID int NOT NULL REFERENCES Customers(CustomerID),
    OrderNumber varchar(100) NOT NULL,
    OrderStatus varchar(100) NOT NULL,
    OrderDate datetime NOT NULL,
    OrderTotalPrice money NOT NULL CHECK (OrderTotalPrice >= 0), -- including discounts, tax, and
other fees
    ShippingService varchar(100) NOT NULL,
    ShippingAddress varchar(100) NOT NULL,
    ShippingCity varchar(100) NOT NULL,
    ShippingState varchar(100) NOT NULL,
    ShippingZipCode varchar(100) NOT NULL,
    ShippingFare money NOT NULL,
    ExceptedShippingDate datetime NOT NULL,
    ActualShippingDate datetime,
    ShippingInformation varchar(8000)
);
GO
INSERT Orders VALUES
(1, 1, 'AERGEB-445', 'Delivered', '11-21-2019', 263.47, 'UPS', '111 College Pl', 'Syracuse', 'NY',
'13210', 12.99, '11-22-2019', '11-22-2019', 'Secret Packaging'),
(2, 1, 'EARGERB-786', 'Delivered', '11-22-2019', 31.75, 'USPS', '111 College Pl', 'Syracuse', 'NY',
'13210', 7.99, '11-22-2019', '11-22-2019', 'Secret Packaging'),
(3, 1, 'DTREGE-453', 'Shipped', '11-27-2019', 65.51, 'FedEx', '111 College Pl', 'Syracuse', 'NY',
'13210', 17.99, '11-29-2019', '11-30-2019', 'Secret Packaging'),
(4, 2, 'SHTRHSD-789', 'Ready to go', '11-30-2019', 1570.91, 'UPS', '900 S Crouse Ave', 'Syracuse',
'NY', '13244', 5.99, '11-30-2019', '11-30-2019', NULL),
(5, 2, 'SFHSFHB-453', 'Delivered', '11-19-2019', 64.78, 'UPS', '900 S Crouse Ave', 'Syracuse', 'NY',
'13244', 0, '11-22-2019', '11-22-2019', NULL),

```



```

        (6, 3, 'SFDHSFD-862', 'Delivered', '11-18-2019', 23.76, 'UPS', '721 Univ Ave', 'Syracuse', 'NY',
        '13244', 0, '11-22-2019', '11-22-2019', 'Secret Packaging'),
        (7, 5, 'VBNMGHJ-733', 'Delivered', '11-21-2019', 1847.91, 'USPS', '222 Waverly Ave', 'Syracuse',
        'NY', '13244', 12.99, '11-22-2019', '11-22-2019', NULL),
        (8, 7, 'UGNNDFFS-945', 'Shipped', '11-22-2019', 173.31, 'UPS', '111 College Pl', 'Syracuse', 'NY',
        '13210', 6.99, '11-22-2019', '11-22-2019', 'Ship to Parcel Locker'),
        (9, 10, 'HOUIBCZ-991', 'Delivered', '11-23-2019', 31.75, 'FedEx', '111 College Pl', 'Syracuse', 'NY',
        '13210', 7.99, '11-25-2019', '11-25-2019', 'Care Handling'),
        (10, 10, 'DFSHSDG-338', 'Delivered', '11-17-2019', 2710.83, 'UPS', '111 College Pl', 'Syracuse',
        'NY', '13210', 12.99, '11-27-2019', '11-30-2019', 'Secret Packaging')
;
GO

```

```

-- create and init linking table OrderItems
-- all these ratings are only for testing purposes
-- they don't indicate whether this item is good in the real world
DROP TABLE IF EXISTS OrderItems;

```

```

GO
CREATE TABLE OrderItems (
    OrderID int NOT NULL REFERENCES Orders(OrderID),
    ProductID int NOT NULL REFERENCES Products(ProductID),
    Quantity int NOT NULL,
    RatingText varchar(100),
    RatingScore int CHECK (1 <= RatingScore AND RatingScore <= 5),
    RatingDate datetime,
    PRIMARY KEY (OrderID, ProductID)
);
GO

```

```

INSERT OrderItems VALUES
    (1, 1, 1, 'Good product', 5, '11-29-2019'),
    (1, 2, 6, NULL, 5, '11-30-2019'),
    (1, 3, 1, NULL, NULL, NULL),
    (2, 1, 1, 'Worst speaker ever ! Trash !', 1, '11-30-2019'),
    (3, 1, 2, NULL, NULL, NULL),
    (4, 9, 1, NULL, NULL, NULL),
    (5, 3, 2, NULL, NULL, NULL),
    (6, 1, 1, 'Good but not excellent', 4, '11-23-2019'),
    (7, 10, 1, 'Good tablet', 4, '11-23-2019'),
    (8, 1, 7, NULL, NULL, NULL),
    (9, 1, 1, NULL, NULL, NULL),
    (10, 8, 1, 'Slow delivery', 1, '11-30-2019'),
    (10, 10, 1, 'Good but way too expensive, Slow delivery', 3, '11-30-2019')
;
GO

```

```

-- create view to show total sales for each supplier
DROP VIEW IF EXISTS SupplierSales;
GO

```

```

CREATE VIEW SupplierSales WITH ENCRYPTION AS
SELECT ISNULL(SupplierName, 'Total') AS SupplierName, ISNULL(SUM(ProductPrice*Quantity), 0) AS
TotalSales
FROM Suppliers LEFT JOIN Products ON Suppliers.SupplierID = Products.SupplierID
      LEFT JOIN OrderItems ON OrderItems.ProductID = Products.ProductID
GROUP BY ROLLUP(SupplierName)
GO

```

```

-- create view to show money spent for each customer
DROP VIEW IF EXISTS CustomerMoneySpent;
GO
CREATE VIEW CustomerMoneySpent WITH ENCRYPTION AS
SELECT ISNULL(CustomerName, 'Total') AS CustomerName, ISNULL(SUM(OrderTotalPrice), 0) AS
TotalMoneySpent
FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
GROUP BY CustomerName WITH CUBE;
GO

```

```

-- create view to show avg rating
DROP VIEW IF EXISTS ProductAverageRating;
GO
CREATE VIEW ProductAverageRating WITH ENCRYPTION AS
SELECT ProductName, SupplierName, ISNULL(CONVERT(varchar(20), AVG(CONVERT(money,
RatingScore))), 'No data') AS AverageRating
FROM OrderItems RIGHT JOIN Products ON OrderItems.ProductID = Products.ProductID
      JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
GROUP BY ProductName, SupplierName
GO

```

```

-- create view to show price and how many times each product is sold
DROP VIEW IF EXISTS ProductSalesCount;
GO
CREATE VIEW ProductSalesCount WITH ENCRYPTION AS
SELECT ProductName, ProductPrice, ISNULL(SUM(Quantity), 0) AS '#Sold'
FROM Products LEFT JOIN OrderItems ON Products.ProductID = OrderItems.ProductID
GROUP BY ProductName, ProductPrice
GO

```

```

-- check if new email addresses for suppliers are valid
DROP TRIGGER IF EXISTS tgSuppliersInsertUpdate;
GO
CREATE TRIGGER tgSuppliersInsertUpdate
      ON Suppliers
      WITH ENCRYPTION
      AFTER INSERT, UPDATE
AS
      IF NOT EXISTS(SELECT * FROM inserted WHERE SupplierEmail NOT LIKE '_%@_%._%') -- all
emails are valid

```

```

        RETURN;
        SELECT 'SUPPLIERS: INVALID EMAIL' AS WARNING;
        ROLLBACK TRAN;
GO

-- check if new email addresses for customers are valid
DROP TRIGGER IF EXISTS tgCustomersInsertUpdate;
GO
CREATE TRIGGER tgCustomersInsertUpdate
    ON Customers
    WITH ENCRYPTION
    AFTER INSERT, UPDATE
AS
    IF NOT EXISTS(SELECT * FROM inserted WHERE CustomerEmail NOT LIKE '_%@_%._%') -- all
emails are valid
        RETURN;
        SELECT 'CUSTOMERS: INVALID EMAIL' AS WARNING;
        ROLLBACK TRAN;
GO

-- prevent any order date (datetime) from being later than current datetime
DROP TRIGGER IF EXISTS tgOrdersInsertUpdate;
GO
CREATE TRIGGER tgOrdersInsertUpdate
    ON Orders
    WITH ENCRYPTION
    AFTER INSERT, UPDATE
AS
    IF NOT EXISTS(SELECT * FROM inserted WHERE OrderDate > GETDATE()) -- all dates are valid
        RETURN;
        SELECT 'ORDERS: INVALID DATE' AS WARNING;
        ROLLBACK TRAN;
GO

-- prevent any table from being dropped
DROP TRIGGER IF EXISTS tgDropTable;
GO
CREATE TRIGGER tgDropTable
    ON DATABASE
    WITH ENCRYPTION
    AFTER DROP_TABLE
AS
    SELECT 'SYSTEM: YOUR SUSPICIOUS BEHAVIOUR IS DENIED AND RECOEDED' AS WARNING;
    ROLLBACK TRAN;
GO

-- create login Quasar and add it to server role dbcreator
IF EXISTS (SELECT * FROM sys.server_principals WHERE name = 'Quasar')
```

```

        DROP LOGIN Quasar;
GO
CREATE LOGIN Quasar WITH PASSWORD = 'MustChange' MUST_CHANGE, CHECK_EXPIRATION = ON;
GO
EXEC sp_addsrvrolemember 'Guest', 'dbcreator';
GO

-- customer purchases goods
DROP PROC IF EXISTS spBuyNow;
GO
CREATE PROC spBuyNow
    @CustomerID int,
    @ProductID int,
    @Quantity int = 1
    WITH ENCRYPTION
AS
    DECLARE @OrderID int = (SELECT MAX(OrderID) FROM Orders) + 1;
    DECLARE @OrderTotalPrice money = (SELECT ProductPrice FROM Products WHERE ProductID =
@ProductID) * @Quantity * 1.08 + 6.00;
    DECLARE @ShippingAddress varchar(100), @ShippingCity varchar(100), @ShippingState
varchar(100), @ShippingZipCode varchar(100);
    SELECT TOP 1 @ShippingAddress = CustomerAddress, @ShippingCity = CustomerCity,
@ShippingState = CustomerState, @ShippingZipCode = CustomerZipCode
    FROM CustomerAddresses
    WHERE CustomerID = @CustomerID AND CustomerAddressType = 'Shipping'
    ORDER BY CustomerAddressID;
    BEGIN TRY
        BEGIN TRAN
            INSERT Orders VALUES (
                @OrderID, @CustomerID, NEWID(), 'Ready to go', GETDATE(),
@OrderTotalPrice, 'UPS',
                @ShippingAddress, @ShippingCity, @ShippingState, @ShippingZipCode,
6.00, DATEADD(DAY, 1, GETDATE()), NULL, NULL
            );
            INSERT OrderItems VALUES (@OrderID, @ProductID, @Quantity, NULL, NULL,
NULL);

            UPDATE Products
            SET NumberAvailable = NumberAvailable - 1
            WHERE ProductID = @ProductID;
            -- the warehouse info shall be updated at the time of actual shipment
        COMMIT TRAN
    END TRY
    BEGIN CATCH
        SELECT 'ERROR PLACING AN ORDER' AS ERROR;
        ROLLBACK TRAN
    END CATCH
GO

```

```

-- warehouse gets goods from supplier
DROP PROC IF EXISTS spGetGoods;
GO
CREATE PROC spGetGoods
    @ProductID int,
    @WarehouseID int,
    @Quantity int
    WITH ENCRYPTION
AS
    BEGIN TRY
        BEGIN TRAN
            IF NOT EXISTS (SELECT * FROM StoredProducts WHERE ProductID = @ProductID
AND WarehouseID = @WarehouseID)
                INSERT StoredProducts VALUES (@ProductID, @WarehouseID,
@Quantity, 0, 0);
            ELSE
                UPDATE StoredProducts
                SET NumberInStock = NumberInStock + @Quantity
                WHERE ProductID = @ProductID AND WarehouseID = @WarehouseID;
            COMMIT TRAN
        END TRY
        BEGIN CATCH
            SELECT 'ERROR GETTING GOODS' AS ERROR;
            ROLLBACK TRAN
        END CATCH
GO

-- customer gives rating
DROP PROC IF EXISTS spGiveRating;
GO
CREATE PROC spGiveRating
    @OrderID int,
    @ProductID int,
    @RatingText varchar(100),
    @RatingScore int
    WITH ENCRYPTION
AS
    BEGIN TRY
        BEGIN TRAN
            IF NOT EXISTS (SELECT * FROM OrderItems WHERE OrderID = @OrderID AND
ProductID = @ProductID)
                THROW 65536, 'ERROR GIVING RATING', 255;
            UPDATE OrderItems
            SET RatingText = @RatingText, RatingScore = @RatingScore, RatingDate =
GETDATE()
            WHERE OrderID = @OrderID AND ProductID = @ProductID;
            COMMIT TRAN
        END TRY

```

```

        BEGIN CATCH
            SELECT ERROR_MESSAGE() AS ERROR;
            ROLLBACK TRAN
        END CATCH
GO

-- Customer returns an order
DROP PROC IF EXISTS spReturnOrder;
GO
CREATE PROC spReturnOrder
    @OrderID int
    WITH ENCRYPTION
AS
    BEGIN TRY
        BEGIN TRAN
            IF NOT EXISTS (SELECT * FROM Orders WHERE OrderID = @OrderID)
                THROW 65535, 'ERROR RETURNING ORDER', 255;
            UPDATE Orders
            SET OrderStatus = 'Return'
            WHERE OrderID = @OrderID;
            -- further processing shall be done upon receiving the returned goods
        COMMIT TRAN
    END TRY
    BEGIN CATCH
        SELECT ERROR_MESSAGE() AS ERROR;
        ROLLBACK TRAN
    END CATCH
GO

-- get overall total sales
DROP FUNCTION IF EXISTS fnGetOverallTotalSales;
GO
CREATE FUNCTION fnGetOverallTotalSales ()
    RETURNS money
    WITH ENCRYPTION
AS
    BEGIN
        RETURN (
            SELECT SUM(Quantity * ProductPrice)
            FROM OrderItems JOIN Products ON OrderItems.ProductID =
Products.ProductID
        );
    END
GO

-- get total number in stock for a product
DROP FUNCTION IF EXISTS fnGetProductTotalInStock;
GO

```

```

CREATE FUNCTION fnGetProductTotalInStock
    (@ProductID int)
    RETURNS int
    WITH ENCRYPTION
AS
    BEGIN
        RETURN (
            SELECT SUM(NumberInStock)
            FROM StoredProducts
            WHERE ProductID = @ProductID
        );
    END
GO

-- get all warehouses in a state
DROP FUNCTION IF EXISTS fnGetWarehousesByState;
GO
CREATE FUNCTION fnGetWarehousesByState
    (@WarehouseState varchar(100))
    RETURNS TABLE
    WITH ENCRYPTION
AS
    RETURN (
        SELECT *
        FROM Warehouses
        WHERE WarehouseState = @WarehouseState
    );
GO

-- get all products whose price is lower than a given value
DROP FUNCTION IF EXISTS fnGetProductsWithPriceBelow;
GO
CREATE FUNCTION fnGetProductsWithPriceBelow
    (@Price money)
    RETURNS TABLE
    WITH ENCRYPTION
AS
    RETURN (
        SELECT *
        FROM Products
        WHERE ProductPrice < @Price
    );
GO

-- show total sales for suppliers
SELECT * FROM SupplierSales;

-- show total money spent for customers

```

```

SELECT * FROM CustomerMoneySpent;

-- show average rating for each product
SELECT * FROM ProductAverageRating;

-- show all customers who's spent more than $1000
SELECT * FROM CustomerMoneySpent WHERE TotalMoneySpent > 1000;

-- show sales count for each product
SELECT * FROM ProductSalesCount;

-- update a supplier with invalid email address
BEGIN TRY
    UPDATE Suppliers
    SET SupplierEmail = 'notvalid@.com'
    WHERE SupplierID = 3;
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE() AS ERROR;
END CATCH

-- insert a customer with invalid email address
BEGIN TRY
    INSERT Customers VALUES (12, 'FAKE NAME', 'FAKE ID', '@FAKE.edu', '123-456-7890', NULL,
    NULL);
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE() AS ERROR;
END CATCH

-- update an order with invalid order date
BEGIN TRY
    UPDATE Orders
    SET OrderDate = '2099-12-09'
    WHERE OrderID = 3;
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE() AS ERROR;
END CATCH

-- attempt to drop a table
BEGIN TRY
    DROP TABLE CustomerAddresses;
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE() AS ERROR;
END CATCH

```



```

-- get overall total sales
SELECT dbo.fnGetOverallTotalSales() AS OverallTotalSales;

-- get total number in stock for Echo Dot (3th Gen)
SELECT dbo.fnGetProductTotalInStock(1) AS EchoDotNumberInStore;

-- get all California warehouses
SELECT * FROM dbo.fnGetWarehousesByState('CA');

-- get all products who's cheaper than $100
SELECT * FROM dbo.fnGetProductsWithPriceBelow(100);

-- join all tables but CustomerWishes
SELECT *
FROM Suppliers JOIN Products ON Suppliers.SupplierID = Products.SupplierID
      JOIN StoredProducts ON StoredProducts.ProductID = Products.ProductID
      JOIN Warehouses ON StoredProducts.WarehouseID = Warehouses.WarehouseID
      JOIN OrderItems ON Products.ProductID = OrderItems.ProductID
      JOIN Orders ON Orders.OrderID = OrderItems.OrderID
      JOIN Customers ON Orders.CustomerID = Customers.CustomerID
      JOIN CustomerAddresses ON CustomerAddresses.CustomerID = Customers.CustomerID;

-- join Customers, CustomerWishes, and Products
SELECT CustomerName, ProductName AS Wish
FROM Customers JOIN CustomerWishes ON Customers.CustomerID = CustomerWishes.CustomerID
      JOIN Products ON CustomerWishes.ProductID = Products.ProductID;

-- customer purchase goods
SELECT ProductID, NumberAvailable FROM Products WHERE ProductID = 5;
SELECT Orders.OrderID, CustomerID, OrderStatus, OrderDate, OrderTotalPrice,
      ShippingAddress, ShippingCity, ShippingState, ShippingZipCode, ExceptedShippingDate, Quantity
FROM Orders JOIN OrderItems ON Orders.OrderID = OrderItems.OrderID
WHERE CustomerID = 6 AND ProductID = 5;
EXEC spBuyNow 6, 5;
SELECT ProductID, NumberAvailable FROM Products WHERE ProductID = 5;
SELECT Orders.OrderID, CustomerID, OrderStatus, OrderDate, OrderTotalPrice,
      ShippingAddress, ShippingCity, ShippingState, ShippingZipCode, ExceptedShippingDate, Quantity
FROM Orders JOIN OrderItems ON Orders.OrderID = OrderItems.OrderID
WHERE CustomerID = 6 AND ProductID = 5;

-- warehouse gets goods from supplier
SELECT * FROM StoredProducts WHERE WarehouseID = 7;
EXEC spGetGoods 10, 7, 6;
SELECT * FROM StoredProducts WHERE WarehouseID = 7;

-- customer gives a rating
SELECT * FROM OrderItems WHERE OrderID = 1 AND ProductID = 3;
EXEC spGiveRating 1, 3, 'Good Product. My favorite !', 5;

```

```
SELECT * FROM OrderItems WHERE OrderID = 1 AND ProductID = 3;
```

```
-- customer returns an order
```

```
SELECT * FROM Orders WHERE OrderID = 7;
```

```
EXEC spReturnOrder 7;
```

```
SELECT * FROM Orders WHERE OrderID = 7;
```