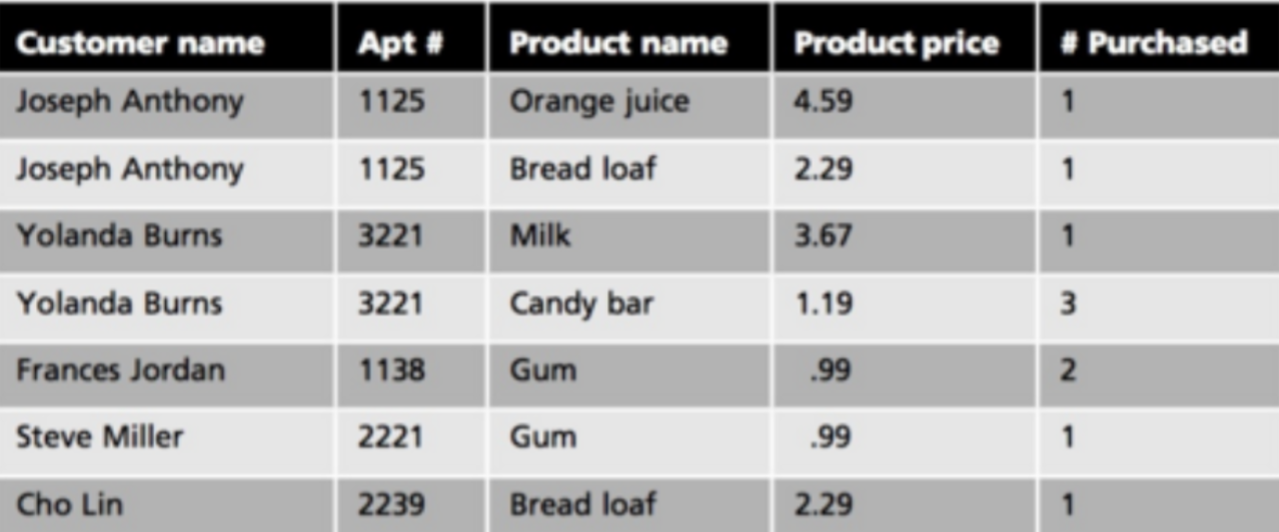
**Scope:** The scope of this project was to redesign a stores database to make it more efficient and normalized. Specific capabilities of queries were expected to work as the outcome. New tables and relations were created to be able to perform these queries.

**Original Store Database:**



**Current Problem:**

- Arguably “Customer name” should be atomic, first name and last name should be separated.

-Definity Data should be Isolated, customer information should not be with product information, and with the transactions. They should not have representation of each other in a single table and all should be isolated.

- Arguably does not satisfy first normalization because “# Purchased” column sometimes contains numbers greater than one, indicating multiple values for one record. Each record of “# Purchased” should be atomically separated individually.

- If one does consider the above table to be in first normal form even with the “# Purchase” being higher than one, then it does not satisfy the law of second normalization form. This is because the primary key “Apt #” in this situation, appears multiple times and not just once, this dissatisfies 2NF’s law of having a single primary key value in the table.

**Proposed Solution:**

To resolve these issues, I began by isolating the data into 3 new tables instead of one.

* Created new tables:
  + Customer.
  + Product.
  + Transaction.
* Each table has a new corresponding Primary key.
  + Customer table PK = customerId.
  + Product table PK = productId.
  + Transaction table PK = txId.
* The transaction table has two foreign keys.
  + Transaction table has FK to product table with productId.
  + Transaction table has FK to customer table with customerId.

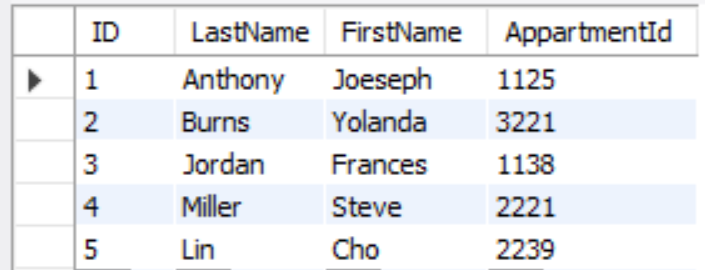
**New Relations:**

* One customer will have many transactions creating a 1:N relationship.
* One product will have many transactions creating a 1:N relationship.

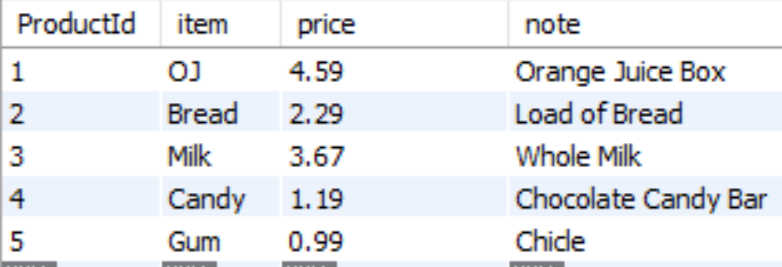
**New Tables:**

*\*\*Please see end of document for table creation and data insert statements of customer, product, and transaction tables to reflect original database\*\**

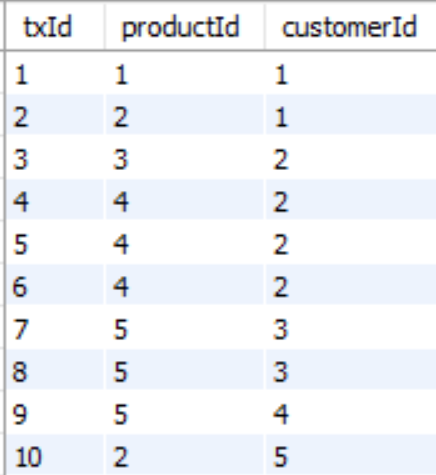
**Customer**



**Product**



**Transaction:**



**New Entity Relationship Diagram:**

|  |  |
| --- | --- |
|  | * Note transaction.productId is FK reference to product.productId. * Note transaction.customerId is FK reference to customer.cusomterId. |

**Queries:**

* This query returns a list of products that were sold in the transactions table, does not return duplicates. This is done with the EXTINCT syntax combined with an INNER JOIN on products in both transaction table and product table.

|  |  |
| --- | --- |
| SELECT DISTINCT item  FROM storedb.transaction  INNER JOIN storedb.product  ON storedb.transaction.ProductId = storedb.product.ProductId; |  |

* This query returns a list of all transactions of a specific customer. This is performed using two inner joins on all three tables and creating a WHERE clause with customerId = ‘{{customerId }}’

|  |  |
| --- | --- |
| SELECT customerId,FirstName,LastName,txId,product.item,product.price  FROM storedb.transaction  INNER JOIN storedb.customer  ON storedb.transaction.customerId = storedb.customer.ID  INNER JOIN storedb.product ON storedb.transaction.ProductId = storedb.product.ProductId where customerId = '1'; | Note this is example for customerId =’1’    Note this is example for customerId =’2’ |

* This query returns the total spent transactions for all of the transactions. This is essentially measuring the volume of all the transactions. This is done with the two inner joins on the 3 tables and using the SUM aggregate function to add all the prices found in the transactions table.

|  |  |
| --- | --- |
| total apartment purchases  SELECT sum(product.price) as totalprice  FROM storedb.transaction  INNER JOIN storedb.customer  ON storedb.transaction.customerId = storedb.customer.ID  INNER JOIN storedb.product ON storedb.transaction.ProductId = storedb.product.ProductId; |  |

* This query returns the total spent per each apartment. This essentially tallies everyone’s individual expenses by apartment Id, this is done with the two INNER JOINS, the SUM aggregate function, and implementing a WHERE clause on customer.AppartmentId=’{{AppartmentID}}’

|  |  |
| --- | --- |
| SELECT AppartmentId,FirstName,LastName, sum(product.price) as totalprice  FROM storedb.transaction  INNER JOIN storedb.customer  ON storedb.transaction.customerId = storedb.customer.ID  INNER JOIN storedb.product ON storedb.transaction.ProductId = storedb.product.ProductId where customer.AppartmentId = '3221'; | Note this for aprtment 3221.    Note this for aprtment 1125. |

**Conclusion:**

The above narrative demonstrates taking an original sales database table and normalizing it with atomic properties. We can see how this database now satisfies more normalization dependencies then it once did, it shows new relationships formed from new tables, and how queries can be performed to display data. This database went from failing the 2NF to now satisfying it by having a single primary key in each table.

**CUSTOMER TABLE CREATE AND INSERT SQL**

CREATE TABLE IF NOT EXISTS `storedb`.`customer` (

`customerId` INT NOT NULL,

`LastName` VARCHAR(255) NOT NULL,

`FirstName` VARCHAR(255) NOT NULL,

`AppartmentId` VARCHAR(255) NOT NULL,

PRIMARY KEY (`customerId`));

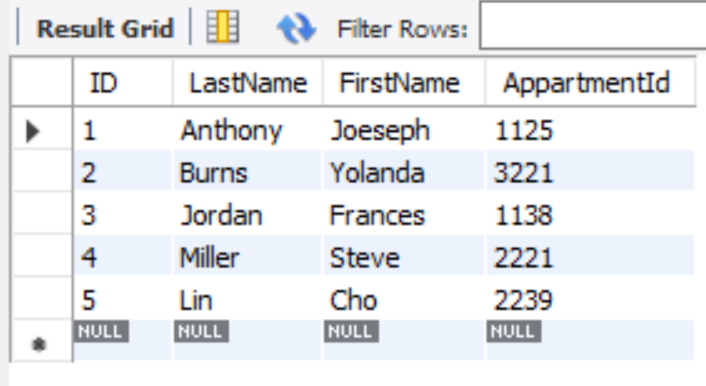
INSERT INTO `storedb`.`customer` (`ID`, `LastName`, `FirstName`, `AppartmentId`) VALUES ('1', 'Anthony', 'Joeseph', '1125');

INSERT INTO `storedb`.`customer` (`ID`, `LastName`, `FirstName`, `AppartmentId`) VALUES ('2', 'Burns', 'Yolanda', '3221');

INSERT INTO `storedb`.`customer` (`ID`, `LastName`, `FirstName`, `AppartmentId`) VALUES ('3', 'Jordan', 'Frances', '1138');

INSERT INTO `storedb`.`customer` (`ID`, `LastName`, `FirstName`, `AppartmentId`) VALUES ('4', 'Miller', 'Steve', '2221');

INSERT INTO `storedb`.`customer` (`ID`, `LastName`, `FirstName`, `AppartmentId`) VALUES ('5', 'Lin', 'Cho', '2239');



**PRODUCT TABLE CREATE AND INSERT SQL**

CREATE TABLE IF NOT EXISTS `storedb`.`product` (

`ProductId` int NOT NULL,

`item` varchar(255) NOT NULL,

`price` decimal(15,2) NOT NULL,

`note` varchar(255) DEFAULT NULL,

PRIMARY KEY (`ProductId`)

);

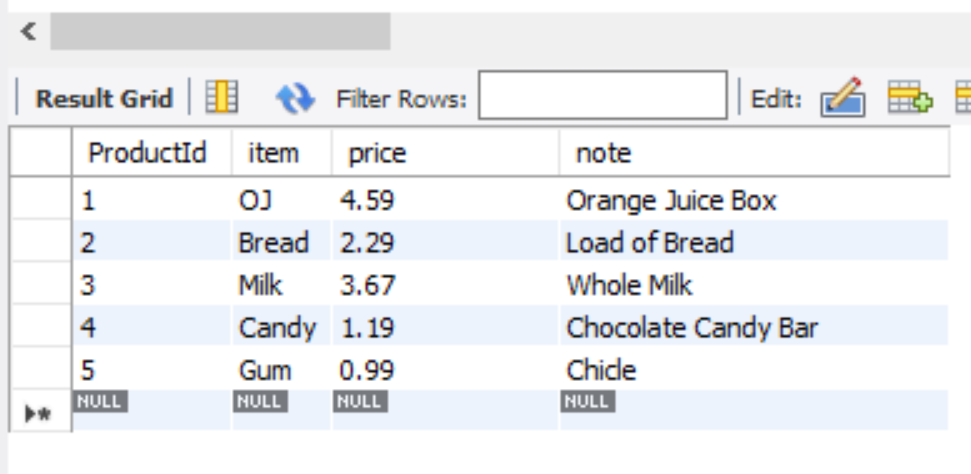
INSERT INTO `storedb`.`product` (`ProductId`, `item`, `price`, `note`) VALUES ('1', 'OJ', '4.59', 'Orange Juice Box');

INSERT INTO `storedb`.`product` (`ProductId`, `item`, `price`, `note`) VALUES ('2', 'Bread', '2.29', 'Load of Bread');

INSERT INTO `storedb`.`product` (`ProductId`, `item`, `price`, `note`) VALUES ('3', 'Milk', '3.67', 'Whole Milk');

INSERT INTO `storedb`.`product` (`ProductId`, `item`, `price`, `note`) VALUES ('4', 'Candy', '1.19', 'Chocolate Candy Bar');

INSERT INTO `storedb`.`product` (`ProductId`, `item`, `price`, `note`) VALUES ('5', 'Gum', '.99', 'Chicle');



**TRANSACTION TABLE CREATE AND INSERT SQL**

CREATE TABLE IF NOT EXSISTS ‘storedb`.`transaction` (

`txId` int NOT NULL,

`productId` int NOT NULL,

`customerId` int DEFAULT NULL,

PRIMARY KEY (`txId`),

KEY `productId` (`productId`),

KEY `customerId` (`customerId`),

CONSTRAINT `transaction\_ibfk\_1` FOREIGN KEY (`productId`) REFERENCES `product` (`ProductId`),

CONSTRAINT `transaction\_ibfk\_2` FOREIGN KEY (`customerId`) REFERENCES `customer` (`ID`)

) ;

INSERT INTO `storedb`.`transaction` (`txId`, `productId`, `customerId`) VALUES ('1', '1', '1');

INSERT INTO `storedb`.`transaction` (`txId`, `productId`, `customerId`) VALUES ('2', '2', '1');

INSERT INTO `storedb`.`transaction` (`txId`, `productId`, `customerId`) VALUES ('3', '3', '2');

INSERT INTO `storedb`.`transaction` (`txId`, `productId`, `customerId`) VALUES ('4', '4', '2');

INSERT INTO `storedb`.`transaction` (`txId`, `productId`, `customerId`) VALUES ('5', '4', '2');

INSERT INTO `storedb`.`transaction` (`txId`, `productId`, `customerId`) VALUES ('6', '4', '2');

INSERT INTO `storedb`.`transaction` (`txId`, `productId`, `customerId`) VALUES ('7', '5', '3');

INSERT INTO `storedb`.`transaction` (`txId`, `productId`, `customerId`) VALUES ('8', '5', '3');

INSERT INTO `storedb`.`transaction` (`txId`, `productId`, `customerId`) VALUES ('9', '5', '4');

INSERT INTO `storedb`.`transaction` (`txId`, `productId`, `customerId`) VALUES ('10', '2', '5');

