ALBANY ARTS GALLERY

Presented By: Olivier Bouan, Dawn Choo, Darya Kashirtseva, Claire Parker, Bridget Roell, Sharvani Srivastava, Elan Weiner, Jiaxi Zhu

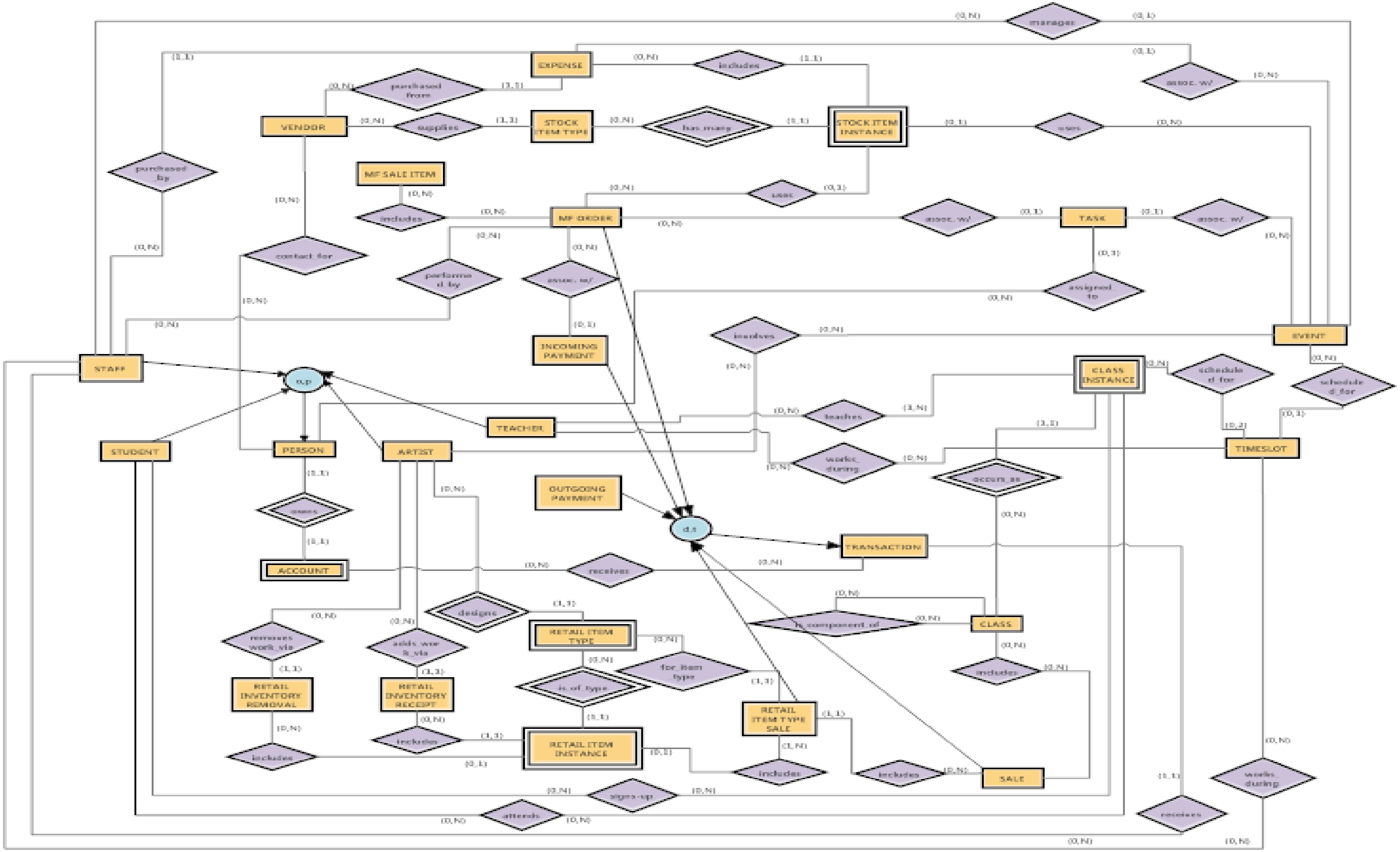
Agenda

- Client Introduction
- Simplified EER Diagram
- Relational Design Schema
- Queries
- Normalization

Client Introduction

- Albany Arts Gallery is a primarily consignment-based retail store on Solano Avenue in Albany, California.
- Most of the artwork in the gallery is made by local artists, or artists who originated in the Bay Area.
- In addition to retail, Albany Arts Gallery also has instructional classes. These are taught by multiple different people, including the owners, who are then paid out for their work based on an hourly rate.
- The gallery is currently organized with pen/paper, excel, and quickbooks.

Simplified EER Diagram

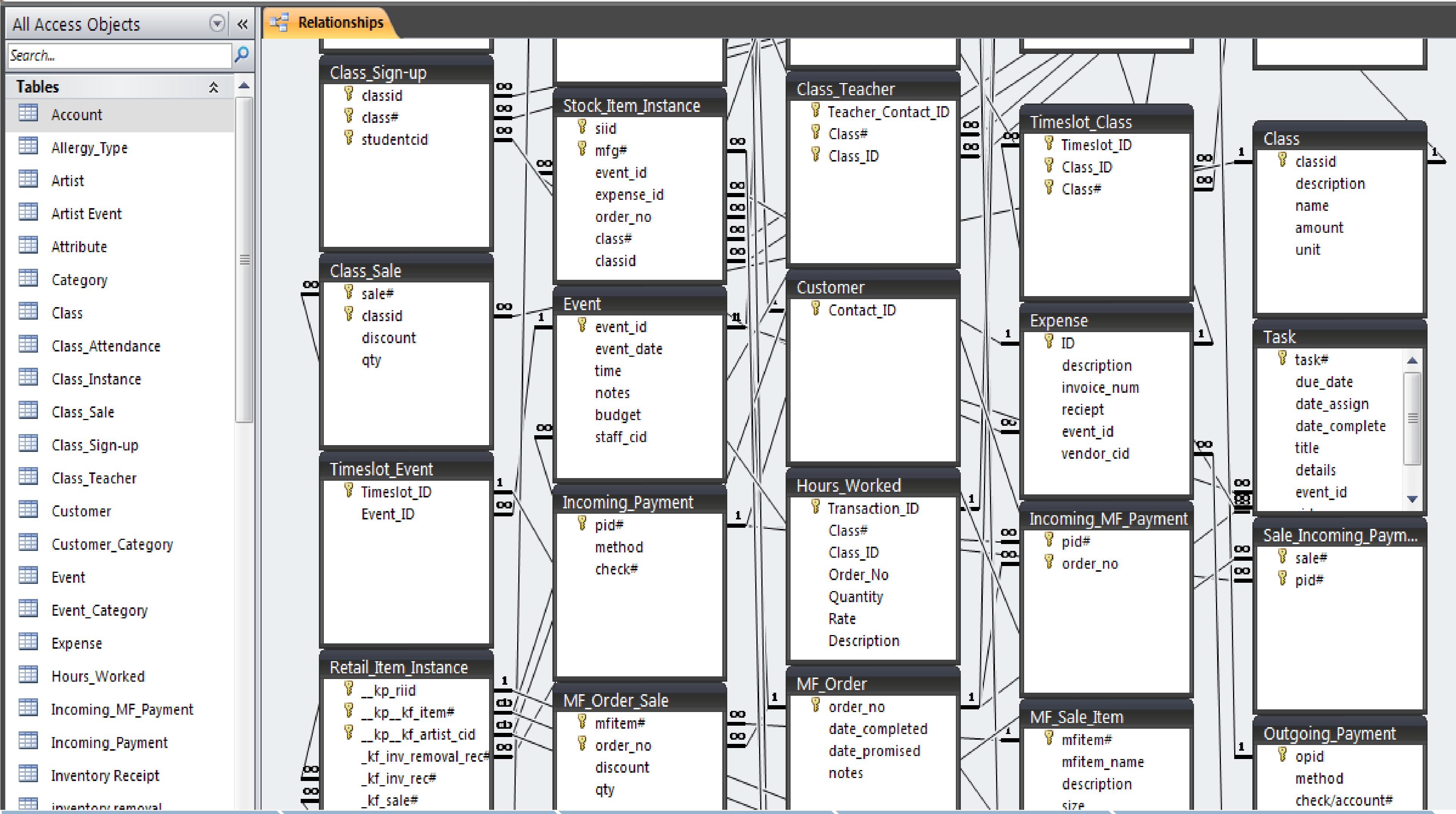


Relational Design Schema

1. Event(event_id, date, time, notes, budget, staff.cid[7b])
2. Task(task#, due_date, date_assign, date_complete, title, details, event_id[1], cid[7], order_no[3])
3. MF_Order(order_no, date_completed, date_placed, date_promised, notes)
4. Stock_Item_Instance(siid, event_id[1], expense_id[5], mfg#[14], order_no[3], staff.cid[7b], class#[22], classid[21])
5. Expense(expense_id, description, amount, vendor_invoice_no, receipt, date, event_id[1], account.cid[6], vendor_cid[39])
6. Account(balance, cid[7])
7. Person(fname, mi, lname, email1, work_phone, mobile_phone, home_phone, cid, street, city, state, zip, notes, account_ID[6], website)
7a. Teacher(cid[7])
7b. Staff(cid[7])
7c. Student(cid[7], waiver, dob)
7d. Customer(cid[7])
7e. Artist(cid[7], artist_id, bio, picture, artist_share, discount_policy, consignment_agreement, A_type, mark_up)
8. Outgoing_Payment(opid, date, amount, method, check/acct#, account.cid[6])
10. Retail_Item_Type(item#, price, item_name, description, artist.cid[7e], R_type, R_Category)
11. Inventory_Removal(inv_removal_rec#, date, staff.cid[7b], artist.cid[7e])
12. Inventory_Receipt(inv_rec#, date, account.cid[6], staff.cid[7b], artist.cid[7e])
13. Retail_Item_Instance(riid, item#[10], artist.cid[7e], inv_removal_rec#[11], inv_rec#[12], sale#[25])
14. Stock_Item_Type(mfg#, mfg_name, size, description, cost)
15. Labor_Hours(rate, qty, account.cid[6], order_no[3], start_time, end_time, date)
16. MF_Sale_Item(mfitem#, name, description, size, price)
17. Incoming_Payment(pid#, amount, payment_date, method, check#, staff.cid[7b], account.cid[7d])
18. Incoming_MF_Payment(pid#[17], order_no[3])
19. MF_Order_Sale(mfitem#[16], order_no[3], discount, qty)
20. Vendor_Stock_Type(mfg#[14], vendor_id[39])
21. Class(classid, description, name, mount, unit)
22. Class_Instance(class#, classid[21], length)

23. Class_Attendance(classid[21], class#[22], student.cid[7c], pickup.cid[7])
23. Class_Balance(classid[21], qty, student.cid[7c])
24. Class_Balance_Instance(class#[22], classid[21], student.cid[7c], qty)
24. Class_Sign-up(classid[21], class#[22], student.cid[7c])
25. Sale(sale#, date, tax, total, staff.cid[7b], account.cid[6])
26. Retail_Item_Type_Sale(sale#[25], item#[10], qty, discount, unit_price, total)
27. Class_Sale(sale#[25], classid[21], discount, qty)
27. Timeslot(TS_id, date, start_time)
28. Sale_Class_Balance(sale#[25], classid[21], student.cid[7c], qty)
29. Vendor_Contact(Vendor_id[39], contact.cid[7])
30. Attribute(attributeID, riid, item#[10], artist.cid[7e], type_type, detail)
31. Sale_Incoming_Payment(sale#[25], pid#[17])
32. Account_Sale(account.cid[6], sale#[25])
33. Artist_Event(artist.cid[7e], event_id[1])
34. Group_Contact(name[9], cid[7])
35. Student_Guardian(student.cid[7c], cid[7])
36. Subject_Type(teacher.cid[7a], subject)
37. Allergy_Type(student.cid[7c], allergy)
38. Transaction(tid, date, amount, account.cid[6])
38a. Incoming_Payment(tid[38], method, check#)
38b. Outgoing_Payment(tid[38], method, check/acct#)
38c. Hours_Worked(tid[38], qty, rate, description, class#, classid[21], order_no[3])
38d. Sale(tid[38], sale#[25])
38e. Retail_Item_Sale(tid[38], sale#[25], item#[10], qty, unit_price, discount)
39. Vendor(vendor_id, company_name, street, city, state, zip, notes, website, email, phone)
40. Timeslot_Class(TS_id[28], classid[21], class#[22])
41. Timeslot_Worked(TS_id[28], cid[7])
42. Timeslot_Event(TS_id[28], event_id[1])
43. MF_Staff(order_no[3], Staff.cid[7b])
44. Class_Teacher(teacher.cid[7a], class#[22], classid[21])

Access Implementation of Relational Design



All Access Objects

Search...

Tables

- Account
- Allergy_Type
- Artist
- Artist Event
- Attribute
- Category
- Class
- Class_Attendance
- Class_Instance
- Class_Sale
- Class_Sign-up
- Class_Teacher
- Customer
- Customer_Category
- Event
- Event_Category
- Expense
- Hours_Worked
- Incoming_MF_Payment
- Incoming_Payment
- Inventory Receipt
- inventory removal

Relationships

Class_Sign-up
- classid
- class#
- studentcid

Class_Sale
- sale#
- classid
- discount
- qty

Timeslot_Event
- Timeslot_ID
- Event_ID

Retail_Item_Instance
- __kp_riid
- __kp__kf_item#
- __kp__kf_artist_cid
- _kf_inv_removal_rec#
- _kf_inv_rec#
- _kf_sale#

Stock_Item_Instance
- siid
- mfg#
- event_id
- expense_id
- order_no
- class#
- classid

Event
- event_id
- event_date
- time
- notes
- budget
- staff_cid

Incoming_Payment
- pid#
- method
- check#

MF_Order_Sale
- mfitem#
- order_no
- discount
- qty

Class_Teacher
- Teacher_Contact_ID
- Class#
- Class_ID

Customer
- Contact_ID

Hours_Worked
- Transaction_ID
- Class#
- Class_ID
- Order_No
- Quantity
- Rate
- Description

MF_Order
- order_no
- date_completed
- date_promised
- notes

Timeslot_Class
- Timeslot_ID
- Class_ID
- Class#

Expense
- ID
- description
- invoice_num
- reciept
- event_id
- vendor_cid

Incoming_MF_Payment
- pid#
- order_no

MF_Sale_Item
- mfitem#
- mfitem_name
- description
- size

Class
- classid
- description
- name
- amount
- unit

Task
- task#
- due_date
- date_assign
- date_complete
- title
- details
- event_id

Sale_Incoming_Paym...
- sale#
- pid#

Outgoing_Payment
- opid
- method
- check/account#

Query 1: Determine the best items to put into the display case to maximize total value of items.

- **Assumptions and Constraints:**
  - The display case can hold a maximum of 25 items.
  - Any artist represented in the case has to have at least 5 items in the case for cohesion.
  - There can be more of the same item in the display case, because each item instance is unique.
- **Implementation Steps**
  - Run a SQL query to determine all items in the inventory, their prices and artists.
  - Convert this information into a data file for AMPL.
  - Run AMPL model with the corresponding data file, which solves a linear program that maximizes the value of items in the case, subject to above constraints.

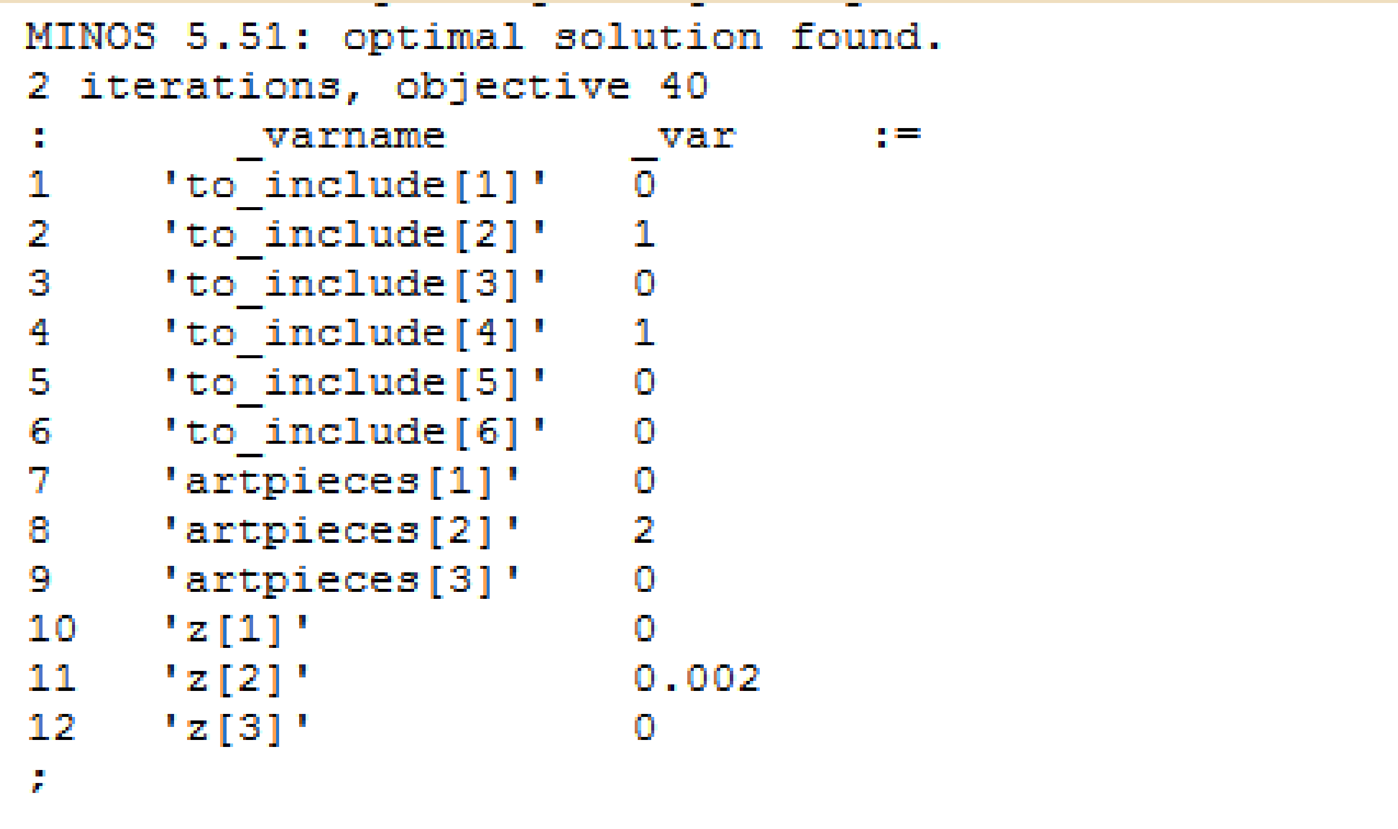
Query 1: Determine the best items to put into the display case to maximize total value of items.

- **SQL:**

```
SELECT i.riid, r.price, i.artist_cid,
r.item_num
FROM Retail_Item_Instance AS i,
Retail_Item_Type AS r
WHERE i.item_num = r.item_num
AND i.artist_cid = r.artist_cid AND
i.sale_num IS NULL;
```

- **AMPL:**

```
param R>=0;
param A>=0;
param price {1..R} >=0;
param belongs_to_artist{1..A,1..R}>=0;
var to_include {1..R} binary; #whether or not to include item in
case
var artpieces {1..A}; #number of pieces artist has in display
var z {1..A} binary; #used in cohesion_constraint
maximize total_value: sum {r in 1..R} (to_include[r]*price[r]);
subject to define_artpieces {a in 1..A} : artpieces[a] = (sum {r in
1..R} (belongs_to_artist[a,r]*to_include[r])); #find the number of
pieces belonging to each artist in the display case
subject to space_limitations: sum{r in 1..R} (to_include[r]) <= 25;
subject to cohesion_constraint1 {a in 1..A}: artpieces[a] <=
1000*z[a]; #arbitrary large number
subject to cohesion_constraint2 {a in 1..A}: 5-artpieces[a] <=
1000*(1-z[a]); #any artist represented has at least 5 items in the
case
```



```
MINOS 5.51: optimal solution found.
2 iterations, objective 40
:       _varname        _var     :=
1     'to_include[1]'   0
2     'to_include[2]'   1
3     'to_include[3]'   0
4     'to_include[4]'   1
5     'to_include[5]'   0
6     'to_include[6]'   0
7     'artpieces[1]'    0
8     'artpieces[2]'    2
9     'artpieces[3]'    0
10    'z[1]'            0
11    'z[2]'            0.002
12    'z[3]'            0
;
```

Query 1 Sample Results

Significance: Artwork in the display case tends to get the most attention from customers. Thus, this query aims to improve sales by helping the gallery determine which set of items are maximize the value of the display case.

Query 2: List Most Profitable Artists

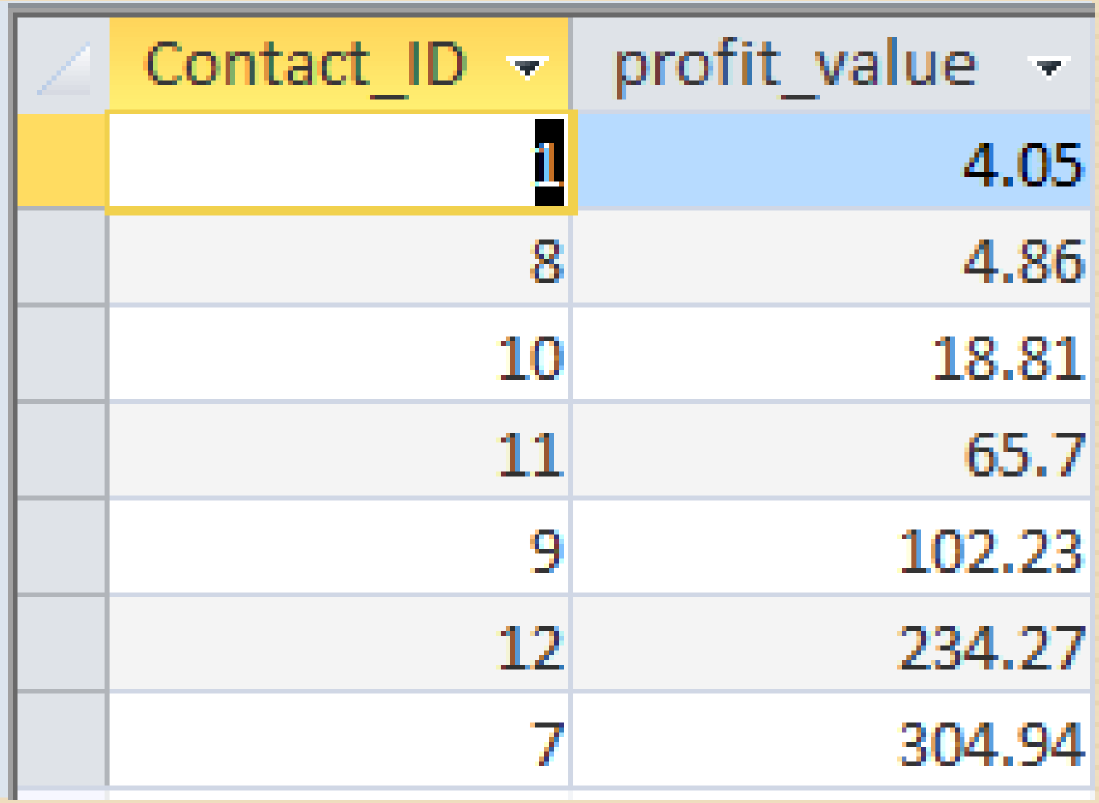
- **Implementation Steps**
  - Run the SQL code in Access
  - No other software is needed

- **Mathematical Principles**
  - Depreciation Formula: $V=P*(1-R)^n$
  - V = final value of product after depreciation
  - P = price of product
  - R = depreciation rate
  - N number of months

Query 2: List Most Profitable Artists

- **SQL:**

```
SELECT a.Contact_ID, round(sum(t.price *
(.9^DATEDIFF("m",r.date,s.Transaction_Date)))) AS
profit_value
FROM Artist AS a, Inventory_Receipt AS r,
Retail_Item_Instance AS i, Retail_Item_Type AS t,
[Transaction] AS s
WHERE a.contact_ID = t.artist_cid AND t.artist_cid =
i.artist_cid AND i.item_num =t.item_num AND i.rec_num =
r.inv_receipt_num AND s.TID = i.sale_num
GROUP BY a.Contact_ID
ORDER BY sum(t.price *
(.9^DATEDIFF("m",r.date,s.Transaction_Date)));
```



Contact_ID	profit_value
1	4.05
8	4.86
10	18.81
11	65.7
9	102.23
12	234.27
7	304.94

Query 2 Results

Significance: Lists most profitable artists taking depreciation of items into account. Depreciation is calculated as a function of time that an art piece has been in stock in the gallery.

Query 3: Forecast class attendance by class type using moving averages

- **Implementation Steps**
  - Extract attendance data by class instance: created an additional table listing all class instances with their respective attendance
  - Compute moving averages for class attendance: wrote Visual Basic codes in an Access module to extract data from the table and compute moving averages
  - Display results: (1) created another table to display moving averages along with their respective class instance information; (2) Exported data to Excel and created moving averages plot for each class type
- **Mathematical Principles and Assumptions**
  - Moving averages: taking the average attendance for the past 3 weeks for each data point, each week is given equal weight in this model
  - Allows more accurate illustration of recent attendance trends
  - Assuming weekly classes

Query 3: Forecast class attendance by class type using moving averages (Continued)

- SQL for new table ("Attendance_Lists")

```
SELECT c.classid AS Class_ID, ci.c_date AS
Class_Date, Count(ca.studentcid) AS
Attendance INTO Attendance_Lists
FROM class AS c, class_attendance AS ca,
class_instance AS ci
WHERE ((([ca.class#])=[ci.class#]) AND
(([ca.classid])=[ci.classid]) AND
(([ci.classid])=[c.classid]))
GROUP BY c.classid, ci.c_date
ORDER BY c.classid, ci.c_date;
```

- SQL for table to display results ("Attendance_MAvgs")

```
SELECT Attendance_Lists.Class_ID,
Attendance_Lists.Class_Date,
Attendance_Lists.Attendance,
MAvgs(3,[Class_Date],[Class_ID]) AS
MAvgs INTO Attendance_MAvgs
FROM Attendance_Lists;
```

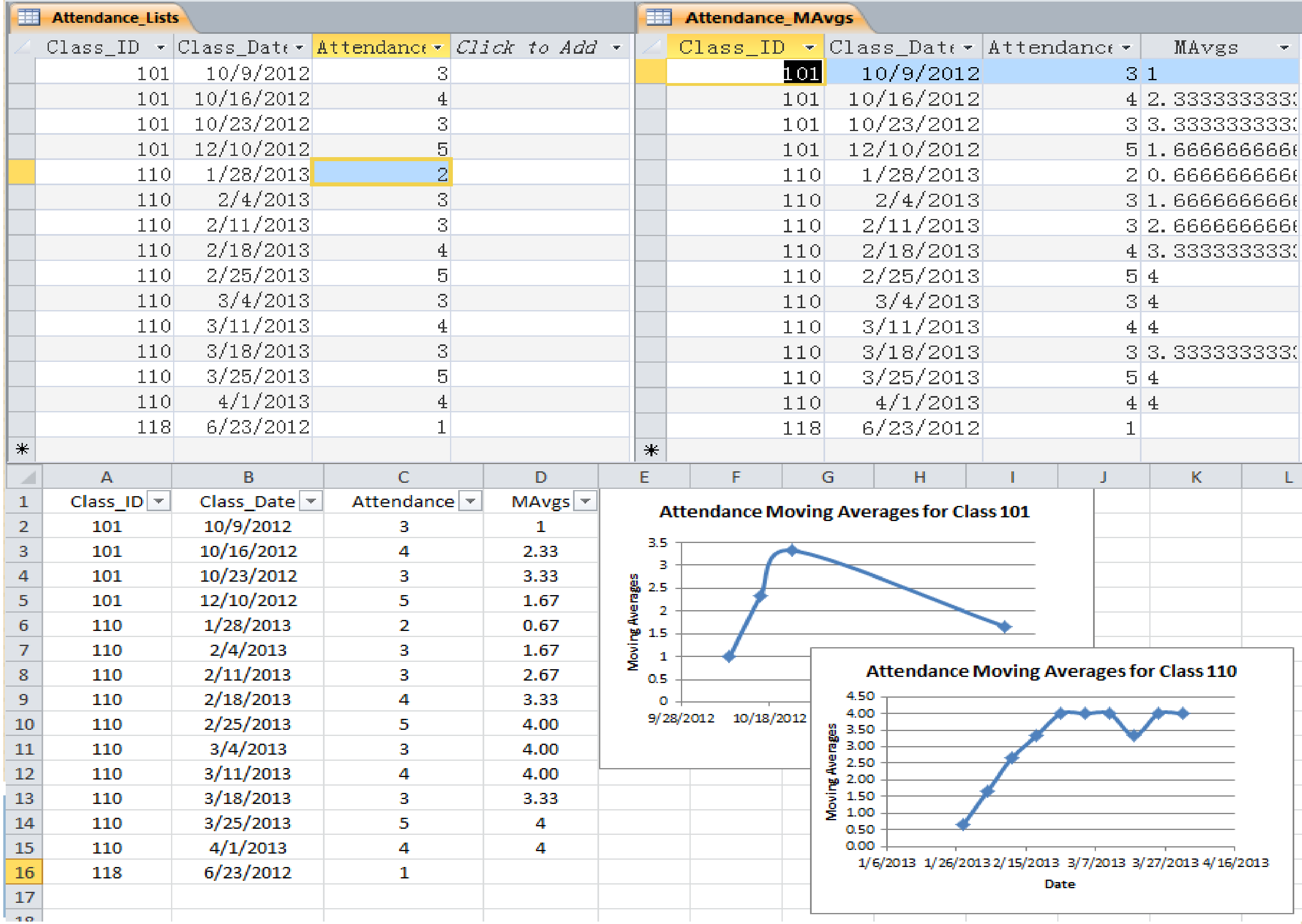
Query 3: Forecast class attendance by class type using moving averages (Continued)

Visual Basic code for computing moving averages

```
Option Compare Database
Option Explicit
Function MAvgs(Periods As Integer, Class_Date, Class_ID)
          Dim MyDB As Database, MyRST As Recordset, MySum
As Double
          Dim i, x
          Set MyDB = CurrentDb()
          Set MyRST = MyDB.OpenRecordset("Attendance_Lists")

          On Error Resume Next

          MyRST.Index = "PrimaryKey“
          x = Periods - 1
          ReDim Store(x)
          MySum = 0

          For i = 0 To x
                    MyRST.MoveFirst
                    MyRST.Seek "=", Class_ID, Class_Date
                    Store(i) = MyRST![Attendance]
```

```
          If i <> x Then Class_Date = Class_Date - 7
          ' The 7 here assumes weekly data.

          If Class_Date < #6/23/2012# Then MAvgs = Null:
Exit Function
          MySum = Store(i) + MySum

          Next i

          MAvgs = MySum / Periods
          MyRST.Close

End Function
```

Query 3 Results



Attendance_Lists

Class_ID	Class_Date	Attendance	Click to Add
101	10/9/2012	3	
101	10/16/2012	4	
101	10/23/2012	3	
101	12/10/2012	5	
110	1/28/2013	2	
110	2/4/2013	3	
110	2/11/2013	3	
110	2/18/2013	4	
110	2/25/2013	5	
110	3/4/2013	3	
110	3/11/2013	4	
110	3/18/2013	3	
110	3/25/2013	5	
110	4/1/2013	4	
118	6/23/2012	1	

Attendance_MAvgs

Class_ID	Class_Date	Attendance	MAvgs
101	10/9/2012	3	1
101	10/16/2012	4	2.333333333
101	10/23/2012	3	3.333333333
101	12/10/2012	5	1.666666666
110	1/28/2013	2	0.666666666
110	2/4/2013	3	1.666666666
110	2/11/2013	3	2.666666666
110	2/18/2013	4	3.333333333
110	2/25/2013	5	4
110	3/4/2013	3	4
110	3/11/2013	4	4
110	3/18/2013	3	3.333333333
110	3/25/2013	5	4
110	4/1/2013	4	4
118	6/23/2012	1	

	A	B	C	D
1	Class_ID	Class_Date	Attendance	MAvgs
2	101	10/9/2012	3	1
3	101	10/16/2012	4	2.33
4	101	10/23/2012	3	3.33
5	101	12/10/2012	5	1.67
6	110	1/28/2013	2	0.67
7	110	2/4/2013	3	1.67
8	110	2/11/2013	3	2.67
9	110	2/18/2013	4	3.33
10	110	2/25/2013	5	4.00
11	110	3/4/2013	3	4.00
12	110	3/11/2013	4	4.00
13	110	3/18/2013	3	3.33
14	110	3/25/2013	5	4
15	110	4/1/2013	4	4
16	118	6/23/2012	1	
17				

Significance: Helps to predict which classes will become more popular so as to allocate appropriate numbers of time slots, instructors, rooms, and other resources

Query 4: Determine which classes make the best customers.

- **Implementation Steps**
  - Run the SQL code in Access

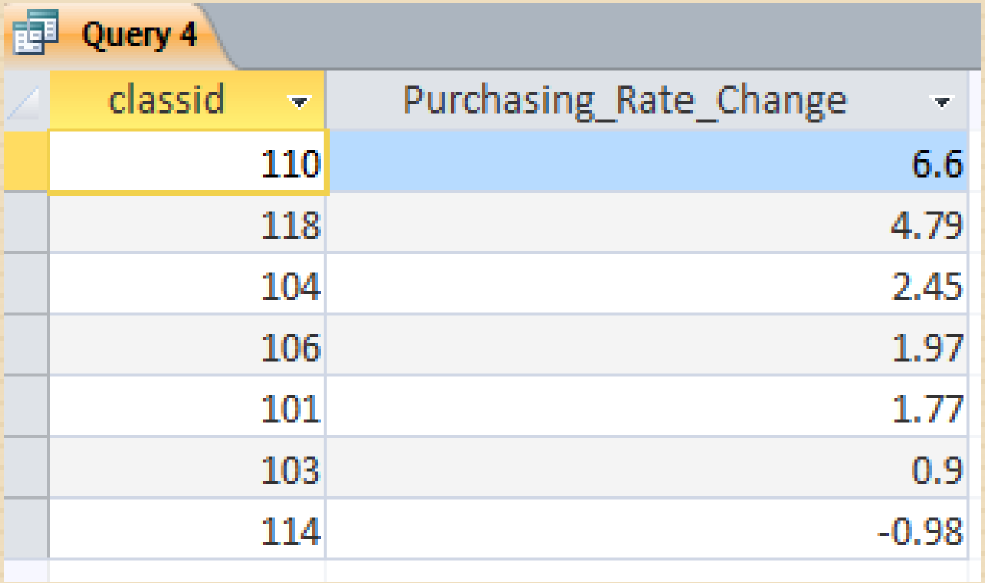
- **Mathematical Principles**
  - Rate Difference Formula: $TP_A/T_A\text{-}TP_B/T_B$
  - $TP_{B/A}$: sum of purchases before/after class instance
  - T_B: time between first purchase and class instance
  - T_A: time between most recent purchase and class instance

- **Table Selection**
  - Find rate for every customer for ever class instance for every class
  - Group by class
  - Display max rate per class

Query 4: Determine which classes make the best customers.

SQL:

```
SELECT a.classid, round(max(a.rate_change),2) AS Purchasing_Rate_Change

FROM    (SELECT s.account_id, ci.classid, ci.class_num, Sum(IIf(s.Date>=ci.c_date,s.total,
        0))/ datediff('d',max(s.date),ci.c_date)- Sum(IIf(s.Date<ci.c_date,s.total,0))/
        datediff('d',ci.c_date,min(s.date)) AS rate_change

        FROM class_instance AS ci, sale AS s, class_attendance AS ca

        WHERE ca.studentcid=s.account_id and ci.class_num=ca.class_num and
        ci.classid=ca.classid GROUP BY s.account_id, ci.classid, ci.class_num,
        ci.c_date)  AS a

GROUP BY a.classid

ORDER BY round(max(a.rate_change),2) DESC;
```



classid	Purchasing_Rate_Change
110	6.6
118	4.79
104	2.45
106	1.97
101	1.77
103	0.9
114	-0.98

Query 4 Results

Significance: The most profitable classes are those that increase customer engagement and purchasing with the gallery. We want to identify these classes and offer more like them.

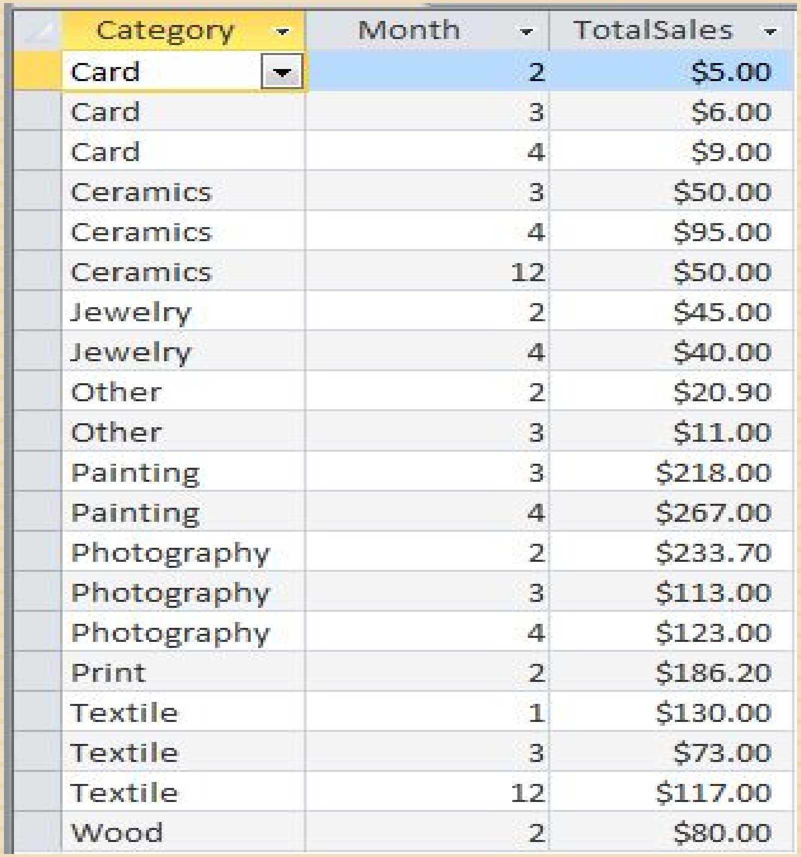
Query 5: Forecast best art categories to focus future displays on based on past sales data

- **Implementation Steps**
  - Extract total monthly sales by art category using SQL
  - Export results to Excel to use Winter’s Method to incorporate seasonality in forecasting future sales for each category
  - Initialize the data using 2013 monthly sales data for each category, generate seasonal factors, and forecast the 2014 monthly sales for each category
- **Mathematical Principles**
  - $F_{t,t+n} = (S_t - mG_t)c_{t+m-N}$
  - $S_t = \alpha \frac{D_t}{c_{t-N}} + (1-\alpha)(S_{t-1} - G_{t-1})$
  - $G_t = \beta(S_{t-1} - G_{t-1}) + (1-\beta)G_{t-1}$
  - $c_t = \gamma \frac{D_t}{S_t} + (1-\gamma)c_{t-N}$

Query 5: Forecast best art categories to focus future displays on based on past sales data

- **SQL:**

```
SELECT [c].Category, Month(t.Transaction_Date) AS
Month, Sum([r].Total) AS TotalSales
FROM [Item_Category] AS [c],
[Retail_Item_Type_Sale] AS [r], Transaction AS t
WHERE [c.Item#]=[r.item#] AND [c.Artist_CID] =
[r.artist_cid] AND t.TID=[r.sale#]
GROUP BY [c.Category], Month(t.Transaction_Date)
ORDER BY [c.Category] ASC ,
Month(t.Transaction_Date) ASC;
```



Category	Month	TotalSales
Card	2	$5.00
Card	3	$6.00
Card	4	$9.00
Ceramics	3	$50.00
Ceramics	4	$95.00
Ceramics	12	$50.00
Jewelry	2	$45.00
Jewelry	4	$40.00
Other	2	$20.90
Other	3	$11.00
Painting	3	$218.00
Painting	4	$267.00
Photography	2	$233.70
Photography	3	$113.00
Photography	4	$123.00
Print	2	$186.20
Textile	1	$130.00
Textile	3	$73.00
Textile	12	$117.00
Wood	2	$80.00

Query 5 Results

Significance: Knowing which art categories will sell best in the coming quarters accounting for seasonality will allow the gallery to focus displays and upcoming events on the relevant best-selling categories.

Normalization

- **Original Relation – 1NF**
  - Retail_Item_Type_Sale (Sale_Num, Item_Num, TID_Pmt_to_Artist, artist_cid, qty, discount, unit_price, total)
- **2NF, 3NF**
  - Retail_Item_Type_sale (Sale_Num, Item_Num, Artist_Cid, TID_Pmt_to_Artist, qty, discount, unit_price)
  - Retail_Sale_total (Qty, Discount, Price, Total)
- **BCNF**
  - R1(Sale_Num, Item_Num, Artis_CID, TID_pmt_to_artist)
  - R2(Sale_Num, Item_Num, Artist_Cid, qty, discount, unit_price)
  - Retail_Sale_total (Qty, Discount, Price, Total)

Normalization

- **Original Relation – 1NF, 2NF, 3NF**
  - Retail_Item_Instance (riid, item_number, artist_cid, inv_removal_rec_num, inv_rec_num, sale_num)
- **BCNF**
  - R1 (Inv_rec_num, artist_cid)
  - R2 (riid, item_number, inv_removal_rec_num, inv_rec_num, sale_num)

Normalization

- **Original Relation – 1NF**
  - Class_Instance (Class_Num, Class_ID, length, date)
- **2NF, 3NF, BCNF**
  - Class_Inst (Class_Num, Class_id, date)
  - Class_Len (Class_id, length)

Thank you!