Cis 111 Chapter 5 Student Lecture Notes

Topic #1: Summary Queries

Information extracted from a database often tries to answer questions such as the following:

- How many customers are from the state of Michigan?
- What is the average total of all orders for a specific customer?
- What is the highest customer total for the month of December?

Notice for each example above that a summary will need to be done in order to answer the above questions. The Select statement allows you to use SQL Server intrinsic functions that allow you to summarize data. The following is a list of **aggregate functions** used for summary queries:

```
AVG([ALL|DISTINCT] expression)
SUM([ALL|DISTINCT] expression)
MIN([ALL|DISTINCT] expression)
MAX([ALL|DISTINCT] expression)
COUNT([ALL|DISTINCT] expression)
COUNT(*)
```

AVG and **SUM** functions only work with numeric expressions.

Aggregate example	Meaning
Select **Avg(InvoiceTotal)** as AvgTotal From Invoices	A single value will be returned to display the average invoice total for all invoices.
Select **Sum(InvoiceTotal-CreditTotal-PaymentTotal)** as Balance From Invoices Where Month(InvoiceDate)=3	Return a single value that represents the total sum of the balance due for the month of March.
Select **Min(InvoiceTotal)** as Lowest, **Max(InvoiceTotal)** as Highest, **Count(*)** as InvoiceCount From Invoices Where Year(InvoiceDate)=2016	Return 1 row with 3 columns: column 1 is the lowest invoice total, column 2 is the highest invoice total and column 3 is the invoice count for invoices received for the 2016 year.
Select **Count(PaymentDate)** as InvoicesPaid From Invoices	A single value is returned which represents the total number of invoices that have been paid. **NOTE**: If PaymentDate is null it is NOT COUNTED.
SELECT **MIN(VendorName)** AS FirstVendor, **MAX(VendorName)** AS LastVendor, **COUNT(VendorName)** AS NumberOfVendors FROM Vendors;	Return 1 row with 3 columns: column 1 is the Vendor with the lowest name, column 2 is the Vendor with the highest name, column 3 is the vendor count.

MIN, MAX and **COUNT** can have work with several different data types for arguments, not just numeric.

Topic #2: Grouping Summary Queries

The examples you reviewed in the previous topic only returned one row of information with one or more columns. Summary queries can also be coded to produce subtotals. Please review the following examples…

Example #1:

```
SELECT VendorState, SUM(InvoiceTotal) AS [State Invoice Total],
       COUNT(*) AS [State Invoice Count],
       AVG(InvoiceTotal) AS [State Invoice Avg]
FROM   Vendors INNER JOIN
        Invoices ON Vendors.VendorID = Invoices.VendorID
GROUP BY VendorState
```

Example #1 Result set:

This summary query displays an invoice total, an invoice count and an average invoice total by vendor state.

	VendorState	State Invoice Total	State Invoice Count	State Invoice Avg
▶	AZ	662.0000	1	662.0000
	CA	37453.3500	40	936.3337
	DC	600.0000	1	600.0000
	MA	1367.5000	1	1367.5000
	MI	141819.7200	7	20259.9600
	NV	25362.4600	10	2536.2460
	OH	227.6800	3	75.8933
	PA	265.3600	2	132.6800

1 of 10 Cell is Read Only.

Example #2:

```
SELECT      VendorState, VendorCity, SUM(InvoiceTotal) AS [City Invoice Total]]],
                COUNT(*) AS [City Invoice Count], AVG(InvoiceTotal) AS [City Invoice Avg]
FROM   Vendors INNER JOIN  Invoices ON Vendors.VendorID = Invoices.VendorID
WHERE   (VendorState = 'CA')
GROUP BY Vendors.VendorCity, Vendors.VendorState
```

Example #2 Result set:

This summary query displays an invoice total, an invoice count and an average invoice total by vendor state/vendor city for the state of California.

	VendorState	VendorCity	City Invoice Total]	City Invoice Co	City Invoice Avg
▶	CA	Fresno	22966.1700	19	1208.7457
	CA	Los Angeles	503.2000	1	503.2000
	CA	Oxnard	564.0000	3	188.0000
	CA	Pasadena	980.6000	5	196.1200
	CA	Sacramento	1771.0100	7	253.0014
	CA	San Francisco	3633.1200	3	1211.0400
	CA	Turlock	95.0000	1	95.0000
	CA	Valencia	6940.2500	1	6940.2500

1 of 8 | Cell is Read Only.

Topic #3: Aggregate Function Distinct Clause

The aggregate functions you reviewed on the first page allow you to code a Distinct clause within the argument. Review the following examples:

Aggregate examples	Meaning
Select **Count(DISTINCT** VendorID**)** as VendorCount From Invoices	A single value will be returned to display the unique number of vendors found in the invoices table, meaning, if VendorID 12 is found multiple times in the Invoices table it will only be counted once.
Select **Count(DISTINCT** InvoiceDate**)** as MarchDaysCount From Invoices Where Month(InvoiceDate)=3 And Year(InvoiceDate)=2016	A single value will be returned to display the total number of days an invoice was received for the month of March in the year 2016.
Select **Distinct Count(**VendorID**)** as VendorCount From Invoices	A single value will be returned to display the number of invoices found in the Invoice table. This IS NOT the same result as the first example above because the Distinct clause is not coded within the Count argument.

Example #1:

A summary query with the DISTINCT keyword

```
SELECT COUNT(DISTINCT VendorID) AS NumberOfVendors,
    COUNT(VendorID) AS NumberOfInvoices,
    AVG(InvoiceTotal) AS AverageInvoiceAmount,
    SUM(InvoiceTotal) AS TotalInvoiceAmount
FROM Invoices
WHERE InvoiceDate > '2015-09-01';
```

The result set

	NumberOfVendors	NumberOfInvoices	AverageInvoiceAmount	TotalInvoiceAmount
1	34	114	1879.7413	214290.51

Topic #4: Having Clause

The **Having clause** is like a Where clause because it allows you to code a filter statement, the difference is when the filter statement is applied. A Where clause is used as records are being selected from one or more tables. The Having clause is applied after an interim table has been created using a Group by clause.

Where clause	Having clause
Can use any field that is found in tables referenced in the From clause.	Can only use fields that are found in the Group by clause.
Cannot use alias names or aggregate functions.	Can use alias names or aggregate functions

Example #1:

A summary query that calculates the average invoice amount by vendor

```
SELECT VendorID, AVG(InvoiceTotal) AS AverageInvoiceAmount
FROM Invoices
GROUP BY VendorID
HAVING AVG(InvoiceTotal) > 2000
ORDER BY AverageInvoiceAmount DESC;
```

The result set

	VendorID	AverageInvoiceAmount
1	110	23978.482
2	72	10963.655
3	104	7125.34
4	99	6940.25
5	119	4901.26
6	122	2575.3288
7	86	2433.00
8	100	2184.50

Example #2:
This example displays the invoice quantity (count) and the average invoice total grouped by vendor state/city who have at least 2 or more invoices in the Invoices table.

```
SELECT VendorState, VendorCity, COUNT(*) AS InvoiceQty,
    AVG(InvoiceTotal) AS InvoiceAvg
FROM Invoices JOIN Vendors
    ON Invoices.VendorID = Vendors.VendorID
GROUP BY VendorState, VendorCity
HAVING COUNT(*) >= 2
ORDER BY VendorState, VendorCity;
```

The result set

	VendorState	VendorCity	InvoiceQty	InvoiceAvg
1	CA	Fresno	19	1208.7457
2	CA	Oxnard	3	188.00
3	CA	Pasadena	5	196.12
4	CA	Sacramento	7	253.0014
5	CA	San Francisco	3	1211.04

```
(12 rows)
```

Example #3:
Why do these next two query examples return a different number of rows?

```
SELECT VendorName, COUNT(*) AS InvoiceQty,
    AVG(InvoiceTotal) AS InvoiceAvg
FROM Vendors JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
GROUP BY VendorName
HAVING AVG(InvoiceTotal) > 500
ORDER BY InvoiceQty DESC;
```

The result set

	VendorName	InvoiceQty	InvoiceAvg
1	United Parcel Service	9	2575.3288
2	Zylka Design	8	867.5312
3	Malloy Lithographing Inc	5	23978.482
4	Data Reproductions Corp	2	10963.655
5	IBM	2	600.06

(19 rows)

```
SELECT VendorName, COUNT(*) AS InvoiceQty,
    AVG(InvoiceTotal) AS InvoiceAvg
FROM Vendors JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
WHERE InvoiceTotal > 500
GROUP BY VendorName
ORDER BY InvoiceQty DESC;
```

The result set

	VendorName	InvoiceQty	InvoiceAvg
1	United Parcel Service	9	2575.3288
2	Zylka Design	7	946.6714
3	Malloy Lithographing Inc	5	23978.482
4	Ingram	2	1077.21
5	Pollstar	1	1750.00

(20 rows)

Example #4:
The next two queries return the same result set but do it differently.

```
SELECT InvoiceDate, COUNT(*) AS InvoiceQty,
SUM(InvoiceTotal) AS InvoiceSum
FROM Invoices
GROUP BY InvoiceDate
HAVING InvoiceDate BETWEEN '2016-01-01' AND '2016-01-31'
   AND COUNT(*) > 1
   AND SUM(InvoiceTotal) > 100
ORDER BY InvoiceDate DESC;
```

The result set

	InvoiceDate	InvoiceQty	InvoiceSum
1	2016-01-31 00:00:00	2	453.75
2	2016-01-25 00:00:00	3	2201.15
3	2016-01-23 00:00:00	2	347.75
4	2016-01-21 00:00:00	2	8078.44
5	2016-01-13 00:00:00	3	1888.95
6	2016-01-11 00:00:00	2	5009.51
7	2016-01-03 00:00:00	2	866.87

```
SELECT InvoiceDate, COUNT(*) AS InvoiceQty,
SUM(InvoiceTotal) AS InvoiceSum
FROM Invoices
WHERE InvoiceDate BETWEEN '2016-01-01' AND '2016-01-31'
GROUP BY InvoiceDate
HAVING COUNT(*) > 1
   AND SUM(InvoiceTotal) > 100
ORDER BY InvoiceDate DESC;
```

The same result set

	InvoiceDate	InvoiceQty	InvoiceSum
1	2016-01-31 00:00:00	2	453.75
2	2016-01-25 00:00:00	3	2201.15
3	2016-01-23 00:00:00	2	347.75
4	2016-01-21 00:00:00	2	8078.44
5	2016-01-13 00:00:00	3	1888.95
6	2016-01-11 00:00:00	2	5009.51
7	2016-01-03 00:00:00	2	866.87

Topic #4: Rollup Operator

The Rollup operator adds a summary row for each group specified. It also adds a summary row to the end of the result set that summarizes the entire result set. If you use the Rollup operator you cannot use the Distinct keyword in any of the aggregate functions.

Example #1

```
SELECT VendorID, COUNT(*) AS InvoiceCount,
          SUM(InvoiceTotal) AS InvoiceTotal
      FROM Invoices
      GROUP BY VendorID WITH ROLLUP;
```

Results | Messages

	VendorID	InvoiceCount	InvoiceTotal
29	115	4	43.67
30	117	1	16.62
31	119	1	4901.26
32	121	8	6940.25
33	122	9	23177.96
34	123	47	4378.02
35	NULL	114	214290.51

There is only one summary row since there is only one field used in the grouping, which is Vendor ID.

Example #2

```
SELECT VendorState, VendorCity, COUNT(*) AS QtyVendors
      FROM Vendors
      WHERE VendorState IN ('IA', 'NJ')
      GROUP BY VendorState, VendorCity WITH ROLLUP
```

Results | Messages

	VendorState	VendorCity	QtyVendors
1	IA	Fairfield	1
2	IA	Washington	1
3	IA	NULL	2
4	NJ	East Brunswick	2
5	NJ	Fairfield	1
6	NJ	Washington	1
7	NJ	NULL	4
8	NULL	NULL	6

This example contains 3 summary rows. One summary for each state (Iowa and New Jersey) and another summary for all states.

Topic #5: Cube Operator

This operator is similar to the Rollup operator, except that it adds summary rows for every combination of groups.

Example #1:

A summary query with a summary row for each set of groups

```
SELECT VendorState, VendorCity, COUNT(*) AS QtyVendors
FROM Vendors
WHERE VendorState IN ('IA', 'NJ')
GROUP BY VendorState, VendorCity WITH CUBE
ORDER BY VendorState DESC, VendorCity DESC;
```

	VendorState	VendorCity	QtyVendors
1	NJ	Washington	1
2	NJ	Fairfield	1
3	NJ	East Brunswick	2
4	NJ	NULL	4
5	IA	Washington	1
6	IA	Fairfield	1
7	IA	NULL	2
8	NULL	Washington	2
9	NULL	Fairfield	2
10	NULL	East Brunswick	2
11	NULL	NULL	6

Summary for New Jersey

Summary for Iowa

Summary row for each city.

Final summary row

Why is the result set sorted by VendorState and VendorCity in descending order?

Here is the result without the Order by clause:

Results | Messages

	VendorState	VendorCity	QtyVendors
1	NJ	East Brunswick	2
2	NULL	East Brunswick	2
3	IA	Fairfield	1
4	NJ	Fairfield	1
5	NULL	Fairfield	2
6	IA	Washington	1
7	NJ	Washington	1
8	NULL	Washington	2
9	NULL	NULL	6
10	IA	NULL	2
11	NJ	NULL	4

Topic #6: Grouping Sets Operator

This operator is similar to the Rollup and Cube operators, however, the **Grouping Sets** operator only includes summary rows, it only adds those summary rows for each specified group.

Example #1:

A summary query with two groups

```
SELECT VendorState, VendorCity, COUNT(*) AS QtyVendors
FROM Vendors
WHERE VendorState IN ('IA', 'NJ')
GROUP BY GROUPING SETS(VendorState, VendorCity)
ORDER BY VendorState DESC, VendorCity DESC;
```

The result set

	VendorState	VendorCity	QtyVendors
1	NJ	NULL	4
2	IA	NULL	2
3	NULL	Washington	2
4	NULL	Fairfield	2
5	NULL	East Brunswick	2

Example #2:

A summary query with a composite grouping

```
SELECT VendorState, VendorCity, VendorZipCode,
       COUNT(*) AS QtyVendors
FROM Vendors
WHERE VendorState IN ('IA', 'NJ')
GROUP BY GROUPING SETS((VendorState, VendorCity),
VendorZipCode, ())
ORDER BY VendorState DESC, VendorCity DESC;
```

The result set

	VendorState	VendorCity	VendorZipCode	QtyVendors
1	NJ	Washington	NULL	1
2	NJ	Fairfield	NULL	1
3	NJ	East Brunswick	NULL	2
4	IA	Washington	NULL	1
5	IA	Fairfield	NULL	1
6	NULL	NULL	07004	1
7	NULL	NULL	07882	1
8	NULL	NULL	08810	1
9	NULL	NULL	08816	1
10	NULL	NULL	52353	1
11	NULL	NULL	52556	1
12	NULL	NULL	NULL	6

Topic #7: Over Clause

The advantage of using the **Over clause** is that you can display summarized data along with the detailed records that were used in the summary.

Example #1:

A query that groups the summary data by date

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
  SUM(InvoiceTotal) OVER (PARTITION BY InvoiceDate) AS DateTotal,
  COUNT(InvoiceTotal) OVER (PARTITION BY InvoiceDate) AS DateCount,
  AVG(InvoiceTotal) OVER (PARTITION BY InvoiceDate) AS DateAvg
FROM Invoices;
```

The result set

	InvoiceNumber	InvoiceDate	InvoiceTotal	DateTotal	DateCount	DateAvg
1	989319-457	2015-12-08 00:00:00	3813.33	3813.33	1	3813.33
2	263253241	2015-12-10 00:00:00	40.20	40.20	1	40.20
3	963253234	2015-12-13 00:00:00	138.75	138.75	1	138.75
4	2-000-2993	2015-12-16 00:00:00	144.70	202.95	3	67.65
5	963253251	2015-12-16 00:00:00	15.50	202.95	3	67.65
6	963253261	2015-12-16 00:00:00	42.75	202.95	3	67.65

The first 3 columns display the invoice detail that were used in the summarized data.

The Sum, Count and Avg columns display summarized data that is partitioned by the InvoiceDate. Focus on the invoice detail for 12-16-2015. There are 3 invoices with a total sum of 202.95 and whose average is 67.65.

Lab Activity

Students will work on exercises for chapter 5 problems #1 through #9.