More SQL Practice Problems (Microsoft SQL Server version)

40 intermediate and advanced challenges for you to solve using a "learn-by-doing" approach

Sylvia Moestl Vasilik

Table of Contents

How to use this book 6

Setup 8

Intermediate Problems 10

1. Cost changes for each product 10
2. Customers with total orders placed 11
3. Products with first and last order date 11
4. Products with first and last order date, including name 12
5. Product cost on a specific date 13
6. Product cost on a specific date, part 2 14
7. Product List Price: how many price changes? 15
8. Product List Price: months with no price changes? 16
9. Current list price of every product 17
10. Products without a list price history 18
11. Product cost on a specific date, part 3 18
12. Products with multiple current list price records 19
13. Products with their first and last order date, including name and subcategory 20
14. Products with list price discrepancies 21
15. Orders for products that were unavailable 22
16. Orders for products that were unavailable: details 23
17. OrderDate with time component 24
18. Fix this SQL! Number 1 25
19. Raw margin quartile for products 26
20. Customers with purchases from multiple sales people 27
21. Fix this SQL! Number 2 27

22. Duplicate product 29
23. Duplicate product: details 29
Advanced Problems 31
24. How many cost changes do products generally have? 31
25. Size and base ProductNumber for products 32
26. Number of sizes for each base product number 33
27. How many cost changes has each product really had? 34
28. Which products had the largest increase in cost? 35
29. Fix this SQL! Number 3 36
30. History table with start/end date overlap 38
31. History table with start/end date overlap, part 2 39
32. Running total of orders in last year 40
33. Total late orders by territory 41
34. OrderDate with time component—performance aspects 42
35. Customer's last purchase—what was the product subcategory? 44
36. Order processing: time in each stage 46
37. Order processing: time in each stage, part 2 48
38. Order processing: time in each stage, part 3 49
39. Top three product subcategories per customer 50
40. History table with date gaps 51
Answers: Intermediate Problems 55
1. Cost changes for each product 55
2. Customers with total orders placed 55
3. Products with first and last order date 56
4. Products with first and last order date, including name 57
5. Product cost on a specific date 57
6. Product cost on a specific date, part 2 58
7. Product List Price: how many price changes? 58
8. Product List Price: months with no price changes? 59
9. Current list price of every product 60
10. Products without a list price history 60
11. Product cost on a specific date, part 3 62

12. Products with multiple current list price records 62
13. Products with their first and last order date, including name and subcategory 63
14. Products with list price discrepancies 64
15. Orders for products that were unavailable 64
16. Orders for products that were unavailable: details 65
17. OrderDate with time component 66
18. Fix this SQL! Number 1 66
19. Raw margin quartile for products 67
20. Customers with purchases from multiple sales people 68
21. Fix this SQL! Number 2 68
22. Duplicate product 69
23. Duplicate product: details 69
Answers: Advanced Problems 70
24. How many cost changes do products generally have? 70
25. Size and base ProductNumber for products 71
26. Number of sizes for each base product number 72
27. How many cost changes has each product really had? 73
28. Which products had the largest increase in cost? 74
29. Fix this SQL! Number 3 74
30. History table with start/end date overlap 75
31. History table with start/end date overlap, part 2 76
32. Running total of orders in last year 77
33. Total late orders by territory 78
34. OrderDate with time component—performance aspects 80
35. Customer's last purchase—what was the product subcategory? 80
36. Order processing: time in each stage 82
37. Order processing: time in each stage, part 2 83
38. Order processing: time in each stage, part 3 84
39. Top three product subcategories per customer 86
40. History table with date gaps 87

How to use this book

I was happy to get many emails from purchasers of my first book, SQL Practice Problems, telling my how useful the problems were in helping them learn SQL. And many of them asked for more practice problems! This book was written because of these repeated requests.

Is this an introductory book for SQL learners? No, it isn't. This book assumes that you've worked through the problems in SQL Practice Problems recently, or have equivalent skills. The basic concepts of SQL Select statements (including joins, left joins, grouping, where clauses, etc.) should be familiar to you. There are a few easier questions at the start of this book, but there are no beginner level questions.

The database used for these practice problems is based on the AdventureWorks database, which is one of the standard Microsoft SQL Server sample databases. However, I've made numerous changes to it, to make it suitable for the questions, and to create interesting data problems. Do *not* try to use the original AdventureWorks database for the problems, you will not get the right results.

What's the best way to work through this book?

I suggest that you not look at the hints at all, unless you're stuck.

Why is this? The reason is that if you need to struggle before coming up with the answer, that struggle will make it *much* more likely that you'll remember how to solve similar problems in real life. And in real life, there are no hints!

I've specifically created another version of this book (available as part of the same download) without hints and answers. I suggest that you use this no hint version, and only refer to the full version if you need to.

But if you're completely stuck and need some direction, please do go ahead and look at the hints. They are designed to gradually walk you through the problem, without giving too much away. The hints will guide you through one specific approach to the problem, though there may be many different answers.

You will need to research online, using your favorite search engine. The technical question and answer website https://stackoverflow.com/ is an outstanding resource. If you're stuck, searching online using regular language will usually be helpful. For instance, try searching for this string:

> SQL how to do an if then statement in a where clause

The first results that come up explain the Case statement, which is what you would use in this situation.

My comment from the previous book on research online still applies:

> Should you search online for answers, examples, etc.? Absolutely. I expect you to do research online as you work through the problems. I do not include all the syntax in this book. In my day-to-day work as a data engineer, I would be lost without being able to do online research. Sometimes I search online for a reminder of a certain syntax, sometimes for examples of a particular type of code, and sometimes for approaches to specific problems. Learning to find answers online effectively can cut your problem-solving time dramatically.

The answer to each problem, and frequently a discussion on the answer, are available in the back of the book. The same recommendation applies to the answer as well as the hints – don't use them unless you need to! But once you have an answer that solves the problem, do compare it to the answer that I've provided, just to get some perspective on alternative answers.

Thank you for purchasing this book!

For any questions or issues, please send email to
feedback@SQLPracticeProblems.com.
I will be happy to respond.

Setup

This section will help you install Microsoft SQL Server 2017 Express Edition, SQL Server Management Studio (SSMS) and walk you through setting up the practice database.

Important note: If you already have a recent version of Microsoft SQL Server installed (SQL Server 2012 and up), as well as SQL Server Management Studio, you do *not* need to reinstall them. You can jump directly to Step 3, setting up the practice database.

The setup of Microsoft SQL Server Express and SSMS will take about 45 minutes, mostly hands-off, with about 5 minutes of interaction here and there. It may take one or two reboots of your system, depending on which version of certain support files you have (dot.net framework).

SQL Server Express Edition will run on computers with more recent versions of Windows, including Windows 8 and Windows 10. Please review this requirements page for full details: https://docs.microsoft.com/en-us/sql/sql-server/install/hardware-and-software-requirements-for-installing-sql-server.

Setup Steps

1. Install MS SQL Server Express Edition 2017

Download and install MS SQL Server Express Edition 2017 from this website: https://www.microsoft.com/en-us/sql-server/sql-server-editions-express.

Feel free to do the Basic install (which is the default) unless you have special requirements.

2. Install SQL Server Management Studio (SSMS) 2017

Download and install SQL Server Management Studio (SSMS) 2017 (https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms).This is the tool that allows you to interact with SQL Server. You can either do this as a part of the MS SQL Server Express Edition 2017 install (there's a link at the bottom), or download it directly.

3. Set up the practice database

Here's a video walk-through on how to set up the practice database:

https://youtu.be/YTYC5-ffB64

You should be able to step through the video, creating the populating the practice database.

Questions or problems with the setup? Email me at feedback@SQLPracticeProblems.com

Intermediate Problems

1. Cost changes for each product

There's a table called ProductCostHistory which contains the history of the cost of the product. Using that table, get the total number of times the product cost has changed.

Sort the results by ProductID

Expected Results

ProductID	TotalPriceChanges
707	3
708	3
709	1
710	1
711	3
712	3
713	3
714	3
715	3
716	4
717	3
718	3
719	3
720	3
721	3

(not all rows shown)

2. Customers with total orders placed

We want to see a list of all the customers that have made orders, and the total number of orders the customer has made.

Sort by the total number of orders, in descending order

Expected Results

CustomerID	TotalOrders
11711	5
12166	4
29586	4
29745	4
29837	4
29812	4
29951	3
30115	3
29675	3
29676	3
11276	3
11247	3
11091	3
11997	3
12216	2

(not all rows shown)

3. Products with first and last order date

For each product that was ordered, show the first and last date that it was ordered.

In the previous problem I gave you the table name to use. For this problem, look at the list of tables, and figure out which ones you need to use.

Sort the results by ProductID.

Expected Results

ProductID	FirstOrder	LastOrder
707	2011-05-31	2014-06-28
708	2011-05-31	2014-06-25
709	2011-05-31	2012-03-30
710	2011-10-01	2011-10-01
711	2011-05-31	2014-06-30
712	2011-05-31	2014-06-22
713	2013-07-17	2014-05-06
714	2011-05-31	2014-06-29
715	2011-05-31	2014-06-07
716	2011-05-31	2014-06-30
717	2011-10-01	2014-03-31
718	2012-03-30	2014-03-31
719	2012-05-30	2013-06-30
722	2011-05-31	2014-05-01
723	2012-05-30	2013-06-30

(not all rows shown)

4. Products with first and last order date, including name

For each product that was ordered, show the first and last date that it was ordered. This time, include the name of the product in the output, to make it easier to understand.

Sort the results by ProductID.

Expected Results

ProductID	ProductName	FirstOrder	LastOrder
707	Sport-100 Helmet, Red	2011-05-31	2014-06-28
708	Sport-100 Helmet, Black	2011-05-31	2014-06-25
709	Mountain Bike Socks, M	2011-05-31	2012-03-30
710	Mountain Bike Socks, L	2011-10-01	2011-10-01
711	Sport-100 Helmet, Blue	2011-05-31	2014-06-30
712	AWC Logo Cap	2011-05-31	2014-06-22
713	Long-Sleeve Logo Jersey, S	2013-07-17	2014-05-06
714	Long-Sleeve Logo Jersey, M	2011-05-31	2014-06-29
715	Long-Sleeve Logo Jersey, L	2011-05-31	2014-06-07
716	Long-Sleeve Logo Jersey, XL	2011-05-31	2014-06-30

717	HL Road Frame - Red, 62	2011-10-01	2014-03-31
718	HL Road Frame - Red, 44	2012-03-30	2014-03-31
719	HL Road Frame - Red, 48	2012-05-30	2013-06-30
722	LL Road Frame - Black, 58	2011-05-31	2014-05-01
723	LL Road Frame - Black, 60	2012-05-30	2013-06-30

(not all rows shown)

Hint

You may have gotten an error like the following:

```
Msg 8120, Level 16, State 1, Line 3
Column 'Product.ProductName' is invalid in the select list because it is not
contained in either an aggregate function or the GROUP BY clause.
```

The fix is to put the ProductName field in the Group By clause, as well as the ProductID.

5. Product cost on a specific date

We'd like to get a list of the cost of products, as of a certain date, 2012-04-15. Use the ProductCostHistory to get the results.

Sort the output by ProductID.

Expected Results

ProductID	StandardCost
707	12.0278
708	12.0278
709	3.3963
710	3.3963
711	12.0278
712	5.7052
713	31.7244
714	31.7244
715	31.7244
716	31.7244
717	747.9682
718	747.9682

719	747.9682
720	747.9682
721	747.9682

(not all rows shown)

6. Product cost on a specific date, part 2

It turns out that the answer to the above problem has a problem. Change the date to 2014-04-15. What are your results?

If you use the SQL from the answer above, and just change the date, you won't get the results you want.

Fix the SQL so it gives the correct results with the new date. Note that when the EndDate is null, that means that price is applicable into the future.

Expected Results

ProductID	StandardCost
707	13.0863
708	13.0863
711	13.0863
712	6.9223
713	38.4923
714	38.4923
715	38.4923
716	38.4923
717	868.6342
718	868.6342
719	868.6342
720	868.6342
721	868.6342
722	204.6251
723	204.6251

(not all rows shown)

Hint

Instead of comparing against the EndDate directly, use the IsNull function against EndDate. Look online for some examples of how to use the function.

7. Product List Price: how many price changes?

Show the months from the ProductListPriceHistory table, and the total number of changes made in that month.

Expected Results

ProductListPriceMonth	TotalRows
2011/05	72
2012/05	128
2013/05	203

Hint

There's a function introduced in SQL Server 2012 called Format. Look at some examples online to help you easily create the date format shown in the Expected Results.

Note that in the format pattern part of the Format function, capitalization is important. The online reference for the Format function is confusing. I suggest looking at this web page for details on the format pattern:

https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings.

8. Product List Price: months with no price changes?

After reviewing the results of the previous query, it looks like price changes are made only in one month of the year.

We want a query that makes this pattern very clear. Show all months (within the range of StartDate values in ProductListPriceHistory). This includes the months during which no prices were changed.

Expected Results

CalendarMonth	TotalRows
2011/05 - May	72
2011/06 - Jun	0
2011/07 - Jul	0
2011/08 - Aug	0
2011/09 - Sep	0
2011/10 - Oct	0
2011/11 - Nov	0
2011/12 - Dec	0
2012/01 - Jan	0
2012/02 - Feb	0
2012/03 - Mar	0
2012/04 - Apr	0
2012/05 - May	128
2012/06 - Jun	0
2012/07 - Jul	0
2012/08 - Aug	0
2012/09 - Sep	0
2012/10 - Oct	0
2012/11 - Nov	0
2012/12 - Dec	0
2013/01 - Jan	0
2013/02 - Feb	0
2013/03 - Mar	0
2013/04 - Apr	0
2013/05 - May	203

Hint

There's a Calendar table that can be used for this problem.

Hint

First do the appropriate joins to the calendar table. Then set up the grouping by month.

Hint

If you're having difficulties with the where clause, look at the problem “High freight charges – last year” in my previous book, SQL Practice Problems, for an example.

9. Current list price of every product

What is the current list price of every product, using the ProductListPrice history?

Order by ProductID

Expected Results

ProductID	ListPrice
707	34.99
708	34.99
711	34.99
712	8.99
713	49.99
713	49.99
714	49.99
715	49.99
716	49.99
717	1431.50
718	1431.50
719	1431.50
720	1431.50
721	1431.50
722	337.22

(not all rows shown)

10. Products without a list price history

Show a list of all products that do not have any entries in the list price history table.

Sort the results by ProductID

Expected Results

ProductID	ProductName
1	Adjustable Race
2	Bearing Ball
3	BB Ball Bearing
4	Headset Ball Bearings
316	Blade
317	LL Crankarm
318	ML Crankarm
319	HL Crankarm
320	Chainring Bolts
321	Chainring Nut
322	Chainring
323	Crown Race
324	Chain Stays
325	Decal 1
326	Decal 2

(not all rows shown)

11. Product cost on a specific date, part 3

In the earlier problem “Product cost on a specific date, part 2”, this answer was given:

```
Select
    ProductID
    ,StandardCost
From ProductCostHistory
Where
    '2014-04-15' Between StartDate and IsNull(EndDate, getdate())
Order By ProductID
```

However, there are many ProductIDs that exist in the ProductCostHistory table that don’t show up in this list.

Show every ProductID in the ProductCostHistory table that does not appear when you run the above SQL.

Expected Results

ProductID
709
710
725
726
727
728
729
730
731
732
733
734
735
740
741

(not all rows shown)

Hint

There are two datasets that you need to combine. One is the list of distinct ProductIDs from ProductCostHistory. The other is the results from the answer to "Product cost on a specific date, part 2". How could you combine them?

12. Products with multiple current list price records

There should only be one current price for each product in the ProductListPriceHistory table, but unfortunately some products have multiple current records.

Find all these, and sort by ProductID

Expected Results

ProductID
713
837
868
899
930
961
992

13. Products with their first and last order date, including name and subcategory

In the problem “Products with their first and last order date, including name", we looked only at product that have been ordered.

It turns out that there are many products that have never been ordered.

This time, show all the products, and the first and last order date. Include the product subcategory as well.

Sort by the ProductName field.

Expected Results

ProductID	ProductName	ProductSubCategoryName	FirstOrder	LastOrder
1	Adjustable Race	NULL	NULL	NULL
879	All-Purpose Bike Stand	Bike Stands	2013-07-23	2014-06-19
712	AWC Logo Cap	Caps	2011-05-31	2014-06-22
3	BB Ball Bearing	NULL	NULL	NULL
2	Bearing Ball	NULL	NULL	NULL
877	Bike Wash - Dissolver	Cleaners	2013-06-13	2014-06-19
316	Blade	NULL	NULL	NULL
843	Cable Lock	Locks	2012-05-30	2013-09-30
952	Chain	Chains	2013-06-30	2014-05-01
324	Chain Stays	NULL	NULL	NULL
322	Chainring	NULL	NULL	NULL
320	Chainring Bolts	NULL	NULL	NULL
321	Chainring Nut	NULL	NULL	NULL

866	Classic Vest, L	Vests	2013-10-26	2014-04-25
865	Classic Vest, M	Vests	2013-06-30	2014-06-26

(not all rows shown)

Hint

Earlier answers included a left join. The answer to this problem will use multiple left joins, because we want to show all products, yet show fields from related tables (SalesOrderDetail, ProductSubCategory) that may not have related records.

14. Products with list price discrepancies

It's astonishing how much work with SQL and data is in finding and resolving discrepancies in data. Some of the salespeople have told us that the current price in the price list history doesn't seem to match the actual list price in the Product table.

Find all these discrepancies. Sort the results by ProductID.

Expected Results

ProductID	ProductName	Prod_ListPrice	PriceHist_LatestListPrice	Diff
792	Road-250 Red, 58	2443.35	2442.35	1.00
858	Half-Finger Gloves, S	24.49	23.49	1.00
891	HL Touring Frame - Blue, 50	1003.91	1004.91	-1.00
924	LL Mountain Frame - Black, 42	249.79	248.79	1.00
957	Touring-1000 Yellow, 60	2384.07	2385.07	-1.00
990	Mountain-500 Black, 42	539.99	538.99	1.00

Hint

As a first step, you could get the current price from the ProductListPriceHistory table. Then, join to the Product table, and calculate the differences between the list prices.

Finally, only show those rows for which there is a difference.

15. Orders for products that were unavailable

It looks like some products were sold before or after they were supposed to be sold, based on the SellStartDate and SellEndDate in the Product table. Show a list of these orders, with details.

Sort the results by ProductID, then OrderDate.

Expected Results

ProductID	OrderDate	ProductName	Qty	SellStartDate	SellEndDate
726	2013-08-30	LL Road Frame - Red, 48	1	2011-05-31	2013-05-29
726	2014-02-28	LL Road Frame - Red, 48	1	2011-05-31	2013-05-29
729	2013-06-30	LL Road Frame - Red, 60	2	2011-05-31	2013-05-29
730	2011-05-31	LL Road Frame - Red, 62	2	2011-06-01	2013-05-29
730	2011-05-31	LL Road Frame - Red, 62	6	2011-06-01	2013-05-29
730	2011-05-31	LL Road Frame - Red, 62	2	2011-06-01	2013-05-29
730	2011-05-31	LL Road Frame - Red, 62	2	2011-06-01	2013-05-29
730	2011-05-31	LL Road Frame - Red, 62	2	2011-06-01	2013-05-29
730	2013-06-30	LL Road Frame - Red, 62	1	2011-06-01	2013-05-29
730	2014-03-30	LL Road Frame - Red, 62	1	2011-06-01	2013-05-29
760	2013-06-30	Road-650 Red, 60	3	2011-05-31	2013-05-29
760	2013-12-31	Road-650 Red, 60	2	2011-05-31	2013-05-29
760	2014-02-28	Road-650 Red, 60	1	2011-05-31	2013-05-29
760	2014-03-30	Road-650 Red, 60	5	2011-05-31	2013-05-29
761	2013-05-30	Road-650 Red, 62	2	2011-05-31	2013-05-29

(not all rows shown)

Hint

The first step is to join the SalesOrderHeader and SalesOrderDetail tables, in order to get the ProductID and the OrderDate.

Once you have those fields, you can join to the Product table, and then figure out whether the OrderDate is between the SellStartDate and SellEndDate.

16. Orders for products that were unavailable: details

We'd like to get more details on when products that were supposed to be unavailable were ordered.

Create a new column that shows whether the product was ordered before the sell start date, or after the sell end date.

Sort the results by ProductID and OrderDate.

Expected Results

ProductID	OrderDate	Qty	SellStartDate	SellEndDate	ProblemType
726	2013-08-30	1	2011-05-31	2013-05-29	Sold after end date
726	2014-02-28	1	2011-05-31	2013-05-29	Sold after end date
729	2013-06-30	2	2011-05-31	2013-05-29	Sold after end date
730	2011-05-31	2	2011-06-01	2013-05-29	Sold before start date
730	2011-05-31	6	2011-06-01	2013-05-29	Sold before start date
730	2011-05-31	2	2011-06-01	2013-05-29	Sold before start date
730	2011-05-31	2	2011-06-01	2013-05-29	Sold before start date
730	2011-05-31	2	2011-06-01	2013-05-29	Sold before start date
730	2013-06-30	1	2011-06-01	2013-05-29	Sold after end date
730	2014-03-30	1	2011-06-01	2013-05-29	Sold after end date
760	2013-06-30	3	2011-05-31	2013-05-29	Sold after end date
760	2013-12-31	2	2011-05-31	2013-05-29	Sold after end date
760	2014-02-28	1	2011-05-31	2013-05-29	Sold after end date
760	2014-03-30	5	2011-05-31	2013-05-29	Sold after end date
761	2013-05-30	2	2011-05-31	2013-05-29	Sold after end date

(not all rows shown)

Hint

A reminder – if you need if/then functionality in SQL Server, you need to use the Case statement. Look online for examples.

17. OrderDate with time component

How many OrderDate values in SalesOrderHeader have a time component to them?

Show the results as below.

Expected Results

TotalOrderWithTime	TotalOrders	PercentOrdersWithTime
140	1561	0.089686098654

Hint

One way to determine whether a datetime field actually has a time component is to use the Convert function to convert it to a date field, and then compare that converted column to the original column, to see if it's identical.

Hint

There are many ways of getting the answer to this problem. Search for the keyword "derived tables" online for some examples of one option for solving this problem.

18. Fix this SQL! Number 1

We want to show details about certain products (name, subcategory, first order date, last order date), similar to what we did in a previous query.

This time, we only want to show the data for products that have Silver in the color field. You know, by looking at the Product table directly, that there are many products that have that color.

A colleague sent you this query, and asked you to look at it. It seems correct, but it returns no rows. What's wrong with it?

```
Select
    Product.ProductID
    ,ProductName
    ,ProductSubCategoryName
    ,FirstOrder = Convert(date, Min(OrderDate))
    ,LastOrder = Convert(date, Max(OrderDate))
From Product
    Left Join SalesOrderDetail  Detail
        on Product.ProductID = Detail.ProductID
    Left Join SalesOrderHeader  Header
        on Header.SalesOrderID = Detail .SalesOrderID
    Left Join ProductSubCategory
        on ProductSubCategory .ProductSubCategoryID = Product.ProductSubCategoryID
Where
    'Color' = 'Silver'
Group by
    Product.ProductID
    ,ProductName
    ,ProductSubCategoryName
Order by LastOrder desc
```

Hint

When troubleshooting problems in SQL, one of the first principles is to simplify. Make the SQL as short as you can, while still showing the problem. Then it should be obvious what the issue is.

19. Raw margin quartile for products

The product manager would like to show information for all products about the raw margin – that is, the price minus the cost. Create a query that will show this information, as well as the raw margin quartile.

For this problem, the quartile should be 1 if the raw margin of the product is in the top 25%, 2 if the product is in the second 25%, etc.

Sort the rows by the product name.

Expected Results

ProductID	ProductName	StandardCost	ListPrice	RawMargin	Quartile
879	All-Purpose Bike Stand	59.466	159.00	99.534	3
712	AWC Logo Cap	6.9223	8.99	2.0677	4
877	Bike Wash - Dissolver	2.9733	7.95	4.9767	4
843	Cable Lock	10.3125	25.00	14.6875	4
952	Chain	8.9866	20.24	11.2534	4
866	Classic Vest, L	23.749	63.50	39.751	3
865	Classic Vest, M	23.749	63.50	39.751	3
864	Classic Vest, S	23.749	63.50	39.751	3
878	Fender Set - Mountain	8.2205	21.98	13.7595	4
948	Front Brakes	47.286	106.50	59.214	3
945	Front Derailleur	40.6216	91.49	50.8684	3
863	Full-Finger Gloves, L	15.6709	37.99	22.3191	4
862	Full-Finger Gloves, M	15.6709	37.99	22.3191	4
861	Full-Finger Gloves, S	15.6709	37.99	22.3191	4
860	Half-Finger Gloves, L	9.1593	24.49	15.3307	4

(not all rows shown)

Hint

To get the quartile, there's a particular window function called Ntile that is commonly used. Look for some examples online.

20. Customers with purchases from multiple sales people

Show all the customers that have made purchases from multiple sales people.

Sort the results by the customer name (first name plus last name).

Expected Results

CustomerID	CustomerName	TotalDifferentSalesPeople
29675	Aaron Con	3
29637	Donna Carreras	2
29523	John Arthur	2
29486	Kim Abercrombie	2
29617	Lindsey Camacho	2
29508	Marvin Allen	2
29837	Onetha Higgs	2
29614	Ryan Calafato	2

Hint

As a first step, just do the grouping at the Customer/SalesOrderHeader level. The key field you need to work with is the SalesPersonEmployeeID.

Hint

Instead of a simple count on the total SalesPersonEmployeeID values, what additional keyword could you put on the Having clause?

21. Fix this SQL! Number 2

A colleague has sent you the following SQL, which causes an error:

```
Select top 100
    Customer.CustomerID
```

```
    ,CustomerName = FirstName + ' ' + LastName
    ,OrderDate
    ,SalesOrderHeader.SalesOrderID
    ,SalesOrderDetail.ProductID
    ,Product.ProductName
    ,LineTotal
From  SalesOrderHeader
    Join Product
        on Product.ProductID = SalesOrderDetail .ProductID
                                                              Join SalesOrderDetail
        on SalesOrderHeader .SalesOrderID = SalesOrderDetail .SalesOrderID
    Join Customer
        on Customer.CustomerID = SalesOrderHeader.CustomerID
Order by
    CustomerID
    ,OrderDate
```

The error it gives is this:

```
Msg 4104, Level 16, State 1, Line 11
The multi-part identifier "SalesOrderDetail.ProductID" could not be bound.
```

Fix the SQL so it returns the correct results without error.

Hint

Two things will help you here:

1. Make the SQL as short as possible, but so that it still produces the error when you run it. How short can it be, and still produce the error? Does this give you any hints as to what the problem is?

2. Search for the actual error message online, after removing for the details of the table/field name. So you would just search for this string:

> The multi-part identifier could not be bound

22. Duplicate product

It looks like the Product table may have duplicate records. Find the names of the products that have duplicate records (based on having the same ProductName).

Expected Results

ProductName
Mountain Pump
Touring Tire

23. Duplicate product: details

We'd like to get some details on the duplicate product issue. For each product that has duplicates, show the product name and the specific ProductID that we believe to be the duplicate (the one that's not the first ProductID for the product name).

Expected Results

PotentialDuplicateProductID	ProductName
1000	Mountain Pump
1001	Touring Tire

Hint

Search online for how to identify duplicates. This is a very common issue, and there's lots of information on it.

Hint

Using the window function Row_Number() is one of the ways of solving this problem. How could you use Row_Number(), along with Partition, to solve this?

Hint

The suggested answer for this includes a CTE (Common Table Expression). If you'd like more information on CTEs, please review problem 39 (Orders – accidental double-entry details) in my previous book, SQL Practice Problems, for some details, or look online for examples.

Congratulations! You've completed the intermediate problems.

Any questions or feedback on the problems, hints, or answers? I'd like to hear from you. Please email me at feedback@SQLPracticeProblems.com.

Advanced Problems

24. How many cost changes do products generally have?

We've worked on many problems based on the ProductCostHistory table. We know that the cost for some products has changed more than for other products. Write a query that shows how many cost changes that products have, in general.

For this query, you can ignore the fact that in ProductCostHistory, sometimes there's an additional record for a product where the cost didn't actually change.

Expected Results

TotalPriceChanges	TotalProducts
1	216
2	52
3	22
4	3

Hint

As a first step, look at the answer for the problem “Cost changes for each product”. How could you use it as the basis for another query?

25. Size and base ProductNumber for products

The ProductNumber field in the Product table comes from the vendor of the product. The size is sometimes a part of this field.

We need to get the base ProductNumber (without the size), and then the size separately. Some products do not have a size. For those products, the base ProductNumber will be the same as the ProductNumber, and the size field will be null.

Limit the results to those ProductIDs that are greater than 533. Sort by ProductID.

Expected Results

ProductID	ProductNumber	HyphenLocation	BaseProductNumber	Size
534	TO_2301	0	TO_2301	NULL
535	TP_0923	0	TP_0923	NULL
679	RC_0291	0	RC_0291	NULL
680	FR_R92B-58	8	FR_R92B	58
706	FR_R92R-58	8	FR_R92R	58
707	HL_U509R	0	HL_U509R	NULL
708	HL_U509	0	HL_U509	NULL
709	SO_B909-M	8	SO_B909	M
710	SO_B909-L	8	SO_B909	L
711	HL_U509B	0	HL_U509B	NULL
712	CA_1098	0	CA_1098	NULL
713	LJ_0192-S	8	LJ_0192	S
714	LJ_0192-M	8	LJ_0192	M
715	LJ_0192-L	8	LJ_0192	L
716	LJ_0192-X	8	LJ_0192	X

(not all rows shown)

Hint

Look at the ProductNumber field. What characteristic does it have that can help you determine if there's a size component?

Hint

You will be using multiple, nested string functions for this problem. Look online for information and examples on the Substring and Charindex functions. Build the new calculated fields one step at a time.

Hint

Remember that you can use the Case statement to return different outputs, depending on whether or not the ProductNumber field has a hyphen in it (indicating that it has a size).

26. Number of sizes for each base product number

Now we'd like to get all the base ProductNumbers, and the number of sizes that they have.

Use the output of the previous problem to get the results. However, do not use the filter from the previous problem (ProductIDs that are greater than 533). Instead of that filter, select only those products that are clothing (ProductCategory = 3).

Order by the base ProductNumber.

Expected Results

BaseProductNumber	TotalSizes
CA_1098	1
GL_F110	3
GL_H102	3
LJ_0192	4
SB_M891	3
SH_M897	4
SH_W890	3
SJ_0194	4
SO_B909	2
SO_R809	2
TG_W091	3
VE_C304	3

27. How many cost changes has each product really had?

A sharp-eyed analyst has pointed out that the total number of product cost changes (from the problem "Cost changes for each product" is not right. Why? Because sometimes, even when there's a new record in the ProductCostHistory table, the cost is not actually different from the previous record!

This eventually will require a fix to the database, to make sure that we do not allow a record like this to be entered. This could be done as a table constraint, or a change to the code used to insert the row.

However, for now, let's just get an accurate count of cost changes per product, where the cost has actually changed. Also include the initial row for a product, even if there's only 1 record.

Sort the output by ProductID.

Expected Results

ProductID	ProductName	TotalCostChanges
707	Sport-100 Helmet, Red	3
708	Sport-100 Helmet, Black	3
709	Mountain Bike Socks, M	1
710	Mountain Bike Socks, L	1
711	Sport-100 Helmet, Blue	3
712	AWC Logo Cap	3
713	Long-Sleeve Logo Jersey, S	3
714	Long-Sleeve Logo Jersey, M	3
715	Long-Sleeve Logo Jersey, L	3
716	Long-Sleeve Logo Jersey, XL	3
717	HL Road Frame - Red, 62	3
718	HL Road Frame - Red, 44	3
719	HL Road Frame - Red, 48	3
720	HL Road Frame - Red, 52	3
721	HL Road Frame - Red, 56	3

(not all rows shown)

Hint

As an intermediate step, try to get results like this:

ProductID	StartDate	StandardCost	PreviousStandardCost
707	2011-05-31	12.0278	NULL
707	2012-05-30	13.8782	12.0278
707	2013-05-30	13.0863	13.8782
708	2011-05-31	12.0278	NULL

708	2012-05-30	13.8782	12.0278
708	2013-05-30	13.0863	13.8782
709	2011-05-31	3.3963	NULL
710	2011-05-31	3.3963	NULL
711	2011-05-31	12.0278	NULL
711	2012-05-30	13.8782	12.0278
711	2013-05-30	13.0863	13.8782
712	2011-05-31	5.7052	NULL
712	2012-05-30	5.2297	5.7052
712	2013-05-30	6.9223	5.2297
713	2011-05-31	31.7244	NULL

(not all rows shown)

Hint

There's a window function introduced in SQL Server 2012 that does exactly what you need to get the above results.

Hint

Once you have the SQL to get the above results, you can compare the StandardCost with the PreviousStandardCost to check if it's different.

Note that you must also account for when the PreviousStandardCost is null.

28. Which products had the largest increase in cost?

We'd like to show which products have had the largest, one time increases in cost. Show all of the price increases (and decreases), in decreasing order of difference.

Don't show any records for which there is no price difference. For instance, if a product only has 1 record in the cost history table, you would not show it in the output, because there has been no change in the cost history.

Expected Results

ProductID	CostChangeDate	StandardCost	PreviousStandardCost	PriceDifference
717	2012-05-30	722.2568	747.9682	25.7114
718	2012-05-30	722.2568	747.9682	25.7114
719	2012-05-30	722.2568	747.9682	25.7114
720	2012-05-30	722.2568	747.9682	25.7114
721	2012-05-30	722.2568	747.9682	25.7114
722	2012-05-30	170.1428	176.1997	6.0569
723	2012-05-30	170.1428	176.1997	6.0569
724	2012-05-30	170.1428	176.1997	6.0569
736	2012-05-30	170.1428	176.1997	6.0569
737	2012-05-30	170.1428	176.1997	6.0569
738	2012-05-30	170.1428	176.1997	6.0569
713	2012-05-30	29.0807	31.7244	2.6437
714	2012-05-30	29.0807	31.7244	2.6437
715	2012-05-30	29.0807	31.7244	2.6437
716	2012-05-30	29.0807	31.7244	2.6437

(not all rows shown)

Hint

This problem is similar to the previous problem. You should be able to reuse some of the SQL.

29. Fix this SQL! Number 3

There's been some problems with fraudulent transactions. The data science team has requested, for a machine learning job, a unusual set of records. It should include data for 11 CustomerIDs that are specifically identified as fraudulent. It should also include a set of 100 random customers. The set of 100 random customers must exclude the 11 customers suspected of fraud.

The SQL below solves the problem. However, there's a problem with this SQL, which is that the list of bad customers is repeated twice. Having hard-coded numbers or lists of numbers in SQL is not a good idea in general. But duplicating them is even worse, because of the potential that they will get out of sync.

Improve this SQL by not repeating the hard-coded list of CustomerIDs that are fraud suspects.

```
;with FraudSuspects as (
    Select *
    From Customer
    Where
        CustomerID in (
            29401
            ,11194
            ,16490
            ,22698
            ,26583
            ,12166
            ,16036
            ,25110
            ,18172
            ,11997
            ,26731
        )
)
, SampleCustomers as (
    Select top 100 *
    From Customer
    Where
        CustomerID not in (
            29401
            ,11194
            ,16490
            ,22698
            ,26583
            ,12166
            ,16036
            ,25110
            ,18172
            ,11997
            ,26731
        )
    Order by
        NewID()
)
Select * From FraudSuspects
Union all
Select * From SampleCustomers
```

30. History table with start/end date overlap

There is a product that has an overlapping date ranges in the ProductListPriceHistory table.

Find the products with overlapping records, and show the dates that overlap.

Expected Results

CalendarDate	ProductID	TotalRows
2013-05-15	746	2
2013-05-16	746	2
2013-05-17	746	2
2013-05-18	746	2
2013-05-19	746	2
2013-05-20	746	2
2013-05-21	746	2
2013-05-22	746	2
2013-05-23	746	2
2013-05-24	746	2
2013-05-25	746	2
2013-05-26	746	2
2013-05-27	746	2
2013-05-28	746	2
2013-05-29	746	2

Hint

As an initial step, write a query that shows output like the following. We basically want the ProductID, and each date that is covered by the StartDate/EndDate fields.

ProductID	CalendarDate
707	2011-05-31
707	2011-06-01
707	2011-06-02
707	2011-06-03
707	2011-06-04
707	2011-06-05
707	2011-06-06
707	2011-06-07
707	2011-06-08
707	2011-06-09
707	2011-06-10
707	2011-06-11
707	2011-06-12
707	2011-06-13
707	2011-06-14

Hint

To get the above result, you'll need to join the ProductListPriceHistory table and Calendar table, using greater than/less than, instead of a join using "=".

Hint

Once you've gotten the output suggested in the initial hint, you can count the number of times a particular date shows up, per ProductID.

If it's more than 1, it's an overlap.

31. History table with start/end date overlap, part 2

It turns out that the SQL that was provided in the Answer section for the previous problem has an error. It's missing a ProductID that also has a date range overlap.

If you wrote SQL that actually showed 2 separate ProductIDs—great job!

If you didn't, then fix the SQL for the previous problem to show all date range overlaps

Sort the results by ProductID and CalendarDate.

Expected Results

CalendarDate	ProductID	TotalRows
2013-05-27	737	2
2013-05-28	737	2
2013-05-29	737	2
2013-05-15	746	2
2013-05-16	746	2
2013-05-17	746	2
2013-05-18	746	2
2013-05-19	746	2
2013-05-20	746	2
2013-05-21	746	2
2013-05-22	746	2
2013-05-23	746	2

2013-05-24	746	2
2013-05-25	746	2
2013-05-26	746	2
2013-05-27	746	2
2013-05-28	746	2
2013-05-29	746	2

Hint

The critical thing to note here is that the StartDate in the ProductListPriceHistory table is not nullable, whereas the EndDate is nullable. In fact, having a null EndDate is used to indicate the current price.

What happens when you compare a null with a non-null value?

32. Running total of orders in last year

For the company dashboard we'd like to calculate the total number of orders, by month, as well as the running total of orders.

Limit the rows to the last year of orders. Sort by calendar month.

Expected Results

CalendarMonth	TotalOrders	RunningTotal
2013/06 - Jun	13	13
2013/07 - Jul	86	99
2013/08 - Aug	88	187
2013/09 - Sep	89	276
2013/10 - Oct	99	375
2013/11 - Nov	105	480
2013/12 - Dec	102	582
2014/01 - Jan	105	687
2014/02 - Feb	88	775
2014/03 - Mar	120	895
2014/04 - Apr	105	1000
2014/05 - May	117	1117
2014/06 - Jun	47	1164

Hint

Searching online for running sum or running total will get you some good examples.

33. Total late orders by territory

Show the number of total orders, and the number of orders that are late.

For this problem, an order is late when the DueDate is before the ShipDate.

Group and sort the rows by Territory.

Expected Results

TerritoryID	TerritoryName	CountryCode	TotalOrders	TotalLateOrders
1	Northwest	US	233	78
2	Northeast	US	14	7
3	Central	US	20	11
4	Southwest	US	296	93
5	Southeast	US	23	6
6	Canada	CA	187	64
7	France	FR	148	57
8	Germany	DE	137	22
9	Australia	AU	324	100
10	United Kingdom	GB	179	62

Hint

There are multiple different ways of solving this problem. Try getting results like the below as an intermediate step:

SalesOrderID	TerritoryID	DueDate	ShipDate	OrderArrivedLate
43659	5	2011-06-12	2011-06-14	1
43662	6	2011-06-12	2011-06-07	0
43668	6	2011-06-12	2011-06-07	0

43680	4	2011-06-12	2011-06-07	0
43681	5	2011-06-12	2011-06-07	0
43689	4	2011-06-12	2011-06-07	0
43692	4	2011-06-12	2011-06-07	0
43701	9	2011-06-12	2011-06-07	0
43722	10	2011-06-18	2011-06-20	1
43743	9	2011-06-23	2011-06-18	0
43764	9	2011-06-28	2011-06-23	0
43785	7	2011-07-03	2011-07-05	1
43806	9	2011-07-06	2011-07-01	0
43827	9	2011-07-09	2011-07-04	0
43848	1	2011-07-13	2011-07-15	1

(not all rows shown)

Hint

Once you get the results from the above hint, you can use aggregate functions to get Territory level information.

34. OrderDate with time component—performance aspects

We don't go often get into performance issues in these practice problems. But there's many different ways of getting the answer to the problem “OrderDate with time component”. Looking at the different answers gives us a good opportunity to look at the performance implications of different strategies.

Below are 4 SQL statements to solve the problem of how many OrderDate values in the SalesOrderHeader table have a time component.

Figure out which of the below solutions is the most efficient in terms of performance. Performance testing has many different aspects, but for this problem, use the “logical reads” output that you get when using the “STATISTICS IO” option. The number of logical reads used by a SQL statement is a good metric for the amount of resources used.

First, read up on “STATISTICS IO” and “logical reads” online. Then, turn on Statistics IO in your query.

The answer to this problem is the number of the solution which consumes the least resources, based on logical reads.

```
----------------------------------------
-- Solution #1
----------------------------------------
Select
    OrdersWithTime.TotalOrderWithTime
    ,TotalOrders.TotalOrders
    ,PercentOrdersWithTime =
        OrdersWithTime.TotalOrderWithTime  * 1.0
        /
        TotalOrders.TotalOrders
From
    (Select TotalOrderWithTime = Count(*)
    from SalesOrderHeader
    where OrderDate <> Convert(date, OrderDate))
        OrdersWithTime
    full outer join
    (Select TotalOrders = Count(*)
    from SalesOrderHeader )
        TotalOrders
        on 1 = 1

----------------------------------------
-- Solution #2
----------------------------------------
;with OrdersWithTime as (
    Select TotalOrderWithTime = Count(*)
    from SalesOrderHeader
    where OrderDate <> Convert(date, OrderDate)
)
, TotalOrders as (
    Select TotalOrders = Count(*)
    from SalesOrderHeader
)
Select
    OrdersWithTime.TotalOrderWithTime
    ,TotalOrders.TotalOrders
    ,PercentOrdersWithTime =
        OrdersWithTime.TotalOrderWithTime  * 1.0
        /
        TotalOrders.TotalOrders
from OrdersWithTime
    full outer join TotalOrders
        on 1=1

----------------------------------------
-- Solution #3
----------------------------------------
Select
```

```
    TotalOrderWithTime = (Select Count(*)
    from SalesOrderHeader
    where OrderDate <> Convert(date, OrderDate))
    ,TotalOrders =
        (Select Count(*) from SalesOrderHeader )
    ,PercentOrdersWithTime =
        (Select Count(*) from
        SalesOrderHeader
        where OrderDate <> Convert(date, OrderDate)) * 1.0
        /
        (Select Count(*)
        from SalesOrderHeader )

----------------------------------------
-- Solution #4
----------------------------------------
;with Main as (
    Select
        SalesOrderID
        ,HasTimeComponent =
            case
                When OrderDate <> Convert(date, OrderDate)
                    then 1
                else 0
            end
    From SalesOrderHeader
)
Select
    TotalOrdersWithTime =Sum(HasTimeComponent )
    ,TotalOrders = Count(*)
    ,PercentOrdersWithTime =
        Sum(HasTimeComponent ) * 1.0
        /
        Count(*)
From Main
```

35. Customer's last purchase—what was the product subcategory?

For a limited list of customers, we need to show the product subcategory of their last purchase. If they made more than one purchase on a particular day, then show the one that cost the most.

Limit the customers to these customer IDs:

19500
19792
24409
26785

Expected Results

CustomerID	CustomerName	ProductSubCategoryName
19500	Russell Shan	Road Bikes
19792	Cristina Nara	Road Bikes
24409	Jackson Hernandez	Helmets
26785	Raul Raje	Tires and Tubes

Hint

As a starting point, try getting the following:

CustomerID	OrderDate	SalesOrderID	ProductID	LineTotal	ProductSubCategoryName
19500	2013-08-23	54896	793	2443.350000	Road Bikes
19500	2013-08-23	54896	707	34.990000	Helmets
19500	2013-04-21	50553	800	1000.437500	Road Bikes
19792	2013-10-16	58113	795	2443.350000	Road Bikes
19792	2013-10-16	58113	707	34.990000	Helmets
19792	2012-02-21	45735	753	3578.270000	Road Bikes
24409	2014-05-23	73593	708	34.990000	Helmets
24409	2014-05-23	73593	933	32.600000	Tires and Tubes
24409	2014-05-23	73593	922	3.990000	Tires and Tubes
26785	2013-10-17	58135	930	35.000000	Tires and Tubes
26785	2013-10-17	58135	711	34.990000	Helmets
26785	2013-10-17	58135	921	4.990000	Tires and Tubes

Hint

After you've gotten the above output, you want to limit the results to only show the product subcategory of their *last* purchase. Figure out which window function you need to do that.

36. Order processing: time in each stage

When an order is placed, it goes through different stages, such as processed, readied for pick up, in transit, delivered, etc.

How much time does each order spend in the different stages?

To figure out which tables to use, take a look at the list of tables in the database. You should be able to figure out the tables to use from the table names.

Limit the orders to these SalesOrderIDs:

68857
70531
70421

Sort by the SalesOrderID, and then the date/time.

Expected Results

SalesOrderID	EventName	TrackingEventDate	NextTrackingEventDate	HoursInStage
68857	Order processing	2014-03-23 02:00	2014-03-23 05:00	3
68857	Order processed	2014-03-23 05:00	2014-03-24 00:00	19
68857	In transit	2014-03-24 00:00	2014-03-24 03:00	3
68857	Arrived at facility	2014-03-24 03:00	2014-03-24 05:00	2
68857	Departed from facility	2014-03-24 05:00	2014-03-24 09:00	4
68857	Order delivered	2014-03-24 09:00	NULL	NULL
70421	Order processing	2014-04-11 02:00	2014-04-11 05:00	3
70421	Order processed	2014-04-11 05:00	2014-04-12 00:00	19
70421	In transit	2014-04-12 00:00	2014-04-12 03:00	3
70421	Arrived at facility	2014-04-12 03:00	2014-04-12 05:00	2
70421	Departed from facility	2014-04-12 05:00	2014-04-12 09:00	4
70421	Order delivered	2014-04-12 09:00	NULL	NULL
70531	Order processing	2014-04-13 02:00	2014-04-13 05:00	3
70531	Order processed	2014-04-13 05:00	2014-04-14 00:00	19
70531	In transit	2014-04-14 00:00	2014-04-14 03:00	3
70531	Arrived at facility	2014-04-14 03:00	2014-04-14 05:00	2
70531	Departed from facility	2014-04-14 05:00	2014-04-14 09:00	4
70531	Order delivered	2014-04-14 09:00	NULL	NULL

Hint

As an initial step, try to get results like these.

SalesOrderID	TrackingEventID	EventName	EventDateTime
43659	1	Order processing	2011-05-31 02:00:00.0000000
43659	2	Order processed	2011-05-31 04:00:00.0000000
43659	3	In transit	2011-06-01 00:00:00.0000000
43659	4	Arrived at facility	2011-06-01 03:00:00.0000000
43659	5	Departed from facility	2011-06-01 05:00:00.0000000
43659	6	Order delivered	2011-06-01 09:00:00.0000000
43660	1	Order processing	2011-05-31 02:00:00.0000000
43660	2	Order processed	2011-05-31 04:00:00.0000000
43660	3	In transit	2011-06-01 00:00:00.0000000
43660	4	Arrived at facility	2011-06-01 03:00:00.0000000
43660	5	Departed from facility	2011-06-01 05:00:00.0000000
43660	6	Order delivered	2011-06-01 09:00:00.0000000
43661	1	Order processing	2011-05-31 02:00:00.0000000
43661	2	Order processed	2011-05-31 04:00:00.0000000
43661	3	In transit	2011-06-01 00:00:00.0000000

(not all rows shown)

Hint

We've used a window function called Lag before. This answer to this problem uses a similar function, called Lead. It will allow you to get the time of the next tracking event.

Hint

As in previous problems, you can use the Format function to show the date/time fields in the correct format, for the final output.

Note that the case of the format string is important. If you're having problems with it, this is a good reference:

http://www.sql-server-helper.com/sql-server-2012/format-function-vs-convert-function.aspx

37. Order processing: time in each stage, part 2

Now we want to show the time spent in each stage of order processing. But instead of showing information for specific orders, we want to show aggregate data, by online vs offline orders.

Sort first by OnlineOfflineStatus, and then TrackingEventID.

Expected Results

OnlineOfflineStatus	EventName	AverageHoursSpentInStage
Offline	Order processing	2
Offline	Order processed	20
Offline	In transit	3
Offline	Arrived at facility	2
Offline	Departed from facility	4
Offline	Order delivered	NULL
Offline	Delivery error	6
Online	Order processing	2
Online	Order processed	19
Online	In transit	3
Online	Arrived at facility	2
Online	Departed from facility	4
Online	Order delivered	NULL
Online	Delivery error	6

Hint

A good first step is to get, at the SalesOrderID/TrackingEventID level, all the information you need to do the grouping and calculations. This includes calculated fields such as the OnlineOfflineStatus and NextTrackingEventDate, as well as base fields such as SalesOrderID, TrackingEventID, EventName, etc.

This means that for this first step, your output would have one record for each row in OrderTracking, so 188,790 records.

Hint

In the final output, you need to order the rows by TrackingEventID and not EventName. When you do that, you may get an error message like this:

Column "Main.TrackingEventID" is invalid in the ORDER BY clause because it is not contained in either an aggregate function or the GROUP BY clause

It should be fairly self-explanatory, but you cannot only have a field in the order by clause in this case. You also need to have it in the Group By clause.

38. Order processing: time in each stage, part 3

The previous query was very helpful to the operations manager, to help her get an overview of differences in order processing between online and offline orders.

Now she has another request, which is to have the averages for online/offline status on the same line, to make it easier to compare.

Expected Results

EventName	OfflineAvgHoursInStage	OnlineAvgHoursInStage
Order processing	2	2
Order processed	20	19
In transit	3	3
Arrived at facility	2	2
Departed from facility	4	4
Order delivered	NULL	NULL
Delivery error	2	8

Hint

Start out with the answer for the previous query. How could you treat the output of that as two separate tables, that could be joined on EventName?

Hint

Create one CTE for online transactions, then another for offline transactions.

Join both by EventName

39. Top three product subcategories per customer

The marketing department would like to have a listing of customers, with the top 3 product subcategories that they've purchased.

To define “top 3 product subcategories”, we'll order by the total amount purchased for those subcategories (i.e. the line total).

Normally we'd run the query for all customers, but to make it easier to view the results, please limit to just the following customers:

13763
13836
20331
21113
26313

Sort the results by CustomerID

Expected Results

CustomerID	CustomerName	TopProdSubCat1	TopProdSubCat2	TopProdSubCat3
13763	Stephanie Richardson	Fenders	Socks	NULL
13836	Dominic Sara	Road Bikes	Mountain Bikes	Tires and Tubes
20331	Jeremy Cox	Shorts	Vests	NULL
21113	Laura Cai	Tires and Tubes	NULL	NULL
26313	Dalton Simmons	Touring Bikes	Mountain Bikes	Jerseys

Hint

First try to get results like this:

CustomerID	CustomerName	ProductSubCat	LineTotal
13763	Stephanie Richardson	Fenders	21.980000
13763	Stephanie Richardson	Socks	8.990000
13836	Dominic Sara	Mountain Bikes	769.490000
13836	Dominic Sara	Road Bikes	3578.270000
13836	Dominic Sara	Tires and Tubes	32.280000
20331	Jeremy Cox	Shorts	69.990000
20331	Jeremy Cox	Vests	63.500000
21113	Laura Cai	Tires and Tubes	23.780000
26313	Dalton Simmons	Jerseys	53.990000
26313	Dalton Simmons	Mountain Bikes	2049.098200
26313	Dalton Simmons	Touring Bikes	2384.070000

Hint

The next step is to add a row number to the output. How do you need to partition/sort in order to get the correct row number?

Hint

You should now have one table with 11 rows and 5 different customers, with everything they've purchased in all the product subcategories, as well as a RowNumber.

For the next step, to get one row per customer, the answer is similar to what you did for the previous problem, “Order processing: time in each stage, part 3”. You need to use derived tables to create the equivalent of three separate tables, one for each of the top three product subcategories, joining on the CustomerID.

Note that it's important which derived table you use as the base (i.e., the one that provides the CustomerID and CustomerName). Since not every customer has purchased three separate different product subcategories, you will miss data if you use the wrong one.

40. History table with date gaps

It turns out that, in addition to overlaps, there are also some gaps in the ProductListPriceHistory table. That is, there are some date ranges for which there are no list prices. We need to find the products and the dates for which there are no list prices.

This is one of the most challenging and fun problems in this book, so take your time and enjoy it! Try doing it first without any hints, because even if you don't manage to solve the problem, you will have learned much more.

Expected Results

ProductID	DateWithMissingPrice
742	2013-05-21
742	2013-05-22
742	2013-05-23
742	2013-05-24

742	2013-05-25
742	2013-05-26
742	2013-05-27
742	2013-05-28
742	2013-05-29
747	2013-05-27
747	2013-05-28
747	2013-05-29

Hint

Online research for "SQL gaps and islands" will yield some results, but you may also get bogged down in details.

Try visualizing an intermediate result set, that may not get you exactly what you want, but is a step in the right direction. You will be using the Calendar table.

Hint

As an intermediate step, work towards a result set that shows you, for each product, a list of days that *should* be covered by the ProductListPriceHistory.

Once you have that, it'll be much easier to figure out how to join to the ProductListPriceHistory table to determine which days are not covered.

Since the calendar table goes far into the past and the future, limit the results by the first StartDate for each product, and one year after the last EndDate in ProductListPriceHistory. Depending on how you write this, you may need multiple CTEs.

Here's what the first few rows would look like:

ProductID	CalendarDate
707	2011-05-31
707	2011-06-01
707	2011-06-02
707	2011-06-03
707	2011-06-04
707	2011-06-05
707	2011-06-06
707	2011-06-07
707	2011-06-08
707	2011-06-09
707	2011-06-10
707	2011-06-11
707	2011-06-12
707	2011-06-13
707	2011-06-14

Hint

If you're having trouble getting results like the above, here's some SQL that you can use in order to get a list of products, with the date ranges for which they should have list prices:

```
Select
    ProductID
    ,FirstStartDate = Min(StartDate)
    ,LastEndDate   = Max(IsNull(EndDate, '2014-05-29'))
From ProductListPriceHistory
Group by ProductID
```

Now, join the results of the above SQL to the Calendar table. Obviously, the Calendar table does not have a ProductID, so you'll need to join on the CalendarDate with a "between".

In your output, you want to have a total of 144,175 rows. That's as many dates, per ProductID, as exist between the FirstStartDate and LastEndDate (from the CTE in the above hint).

Hint

Once you have the output you need (one row for each product, for each date that it should have an applicable price in ProductListPriceHistory) you're ready for the next step.

That would be a left join to ProductListPriceHistory, to show where there are no records in that table. You'd need to join both on the date (with a Between operator for the date) and the ProductID (with an Equals (=) operator).

Congratulations! You've completed the advanced problems.

Any questions or feedback on the problems, hints, or answers? I'd like to hear from you. Please email me at feedback@SQLPracticeProblems.com.