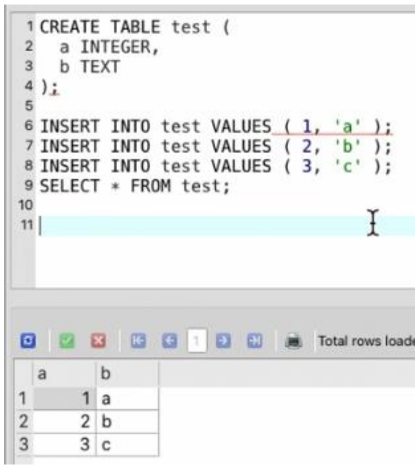
- single quotes are used for a literal string.
  most other things that require quotes use double quotes in standard SQL.
- The SELECT statement is used for all queries.

- **SELECT – FROM – WHERE – HAVING – ORDER BY – LIMIT - OFFSET**
- SQLiteStudio: F9 execute the SQL
- Count rows of a table: **SELECT COUNT(*) FROM** Country;
  (若选某一个变量，只会统计有数据的观测值！！)
- **SELECT DISTINCT** * FROM table; (without duplication)
- 新增或删除 Table:



```
CREATE TABLE test (
  a INTEGER,
  b TEXT
);

INSERT INTO test VALUES ( 1, 'a' );
INSERT INTO test VALUES ( 2, 'b' );
INSERT INTO test VALUES ( 3, 'c' );
SELECT * FROM test;
```

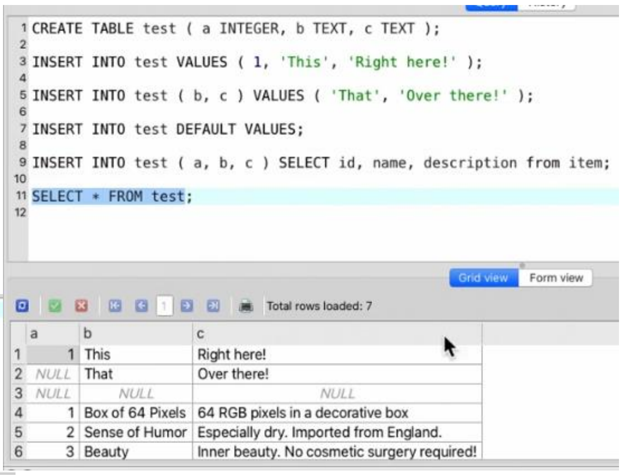
Total rows loade

	a	b
1	1	a
2	2	b
3	3	c

➔ CREAT TABLE test (
    id **INTEGER PRIMARY KEY**, → SQLite only
    b INTEGER **DEFAULT** '0',
    c TEXT **UNIQUE**,
    d TEXT **UNIQUE NOT NULL**
);

- SELECT * FROM test;
- DROP TABLE test;
  (如果已删除，再执行会报错)
- DROP TABLE IF EXISTS test;

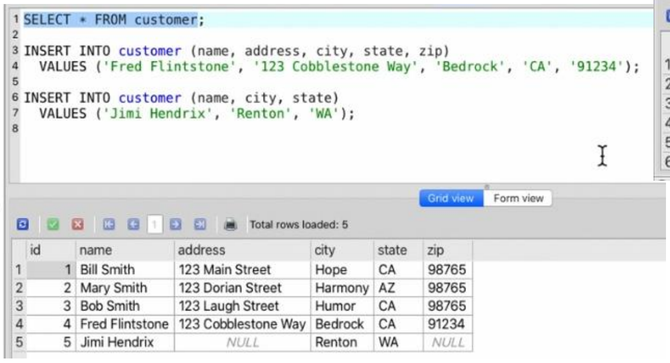
- 编辑 TABLE:
  - **INSERT INTO table_name**
    (col_name1, col_name2,
    col_name3)
    VALUES ('必须在引号里',' ',' ');



```
CREATE TABLE test ( a INTEGER, b TEXT, c TEXT );

INSERT INTO test VALUES ( 1, 'This', 'Right here!' );

INSERT INTO test ( b, c ) VALUES ( 'That', 'Over there!' );

INSERT INTO test DEFAULT VALUES;

INSERT INTO test ( a, b, c ) SELECT id, name, description from item;

SELECT * FROM test;
```

Total rows loaded: 7

	a	b	c
1	1	This	Right here!
2	NULL	That	Over there!
3	NULL	NULL	NULL
4	1	Box of 64 Pixels	64 RGB pixels in a decorative box
5	2	Sense of Humor	Especially dry. Imported from England.
6	3	Beauty	Inner beauty. No cosmetic surgery required!



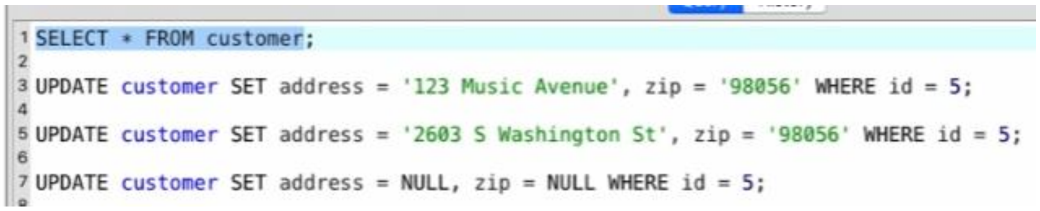
```
SELECT * FROM customer;

INSERT INTO customer (name, address, city, state, zip)
  VALUES ('Fred Flintstone', '123 Cobblestone Way', 'Bedrock', 'CA', '91234');

INSERT INTO customer (name, city, state)
  VALUES ('Jimi Hendrix', 'Renton', 'WA');
```

Total rows loaded: 5

	id	name	address	city	state	zip
1	1	Bill Smith	123 Main Street	Hope	CA	98765
2	2	Mary Smith	123 Dorian Street	Harmony	AZ	98765
3	3	Bob Smith	123 Laugh Street	Humor	CA	98765
4	4	Fred Flintstone	123 Cobblestone Way	Bedrock	CA	91234
5	5	Jimi Hendrix	NULL	Renton	WA	NULL

还可以用 INSERT INTO table1 +
SELECT .. FROM table2
纵向将(部分的)tbl2 添加到 tbl1 里！

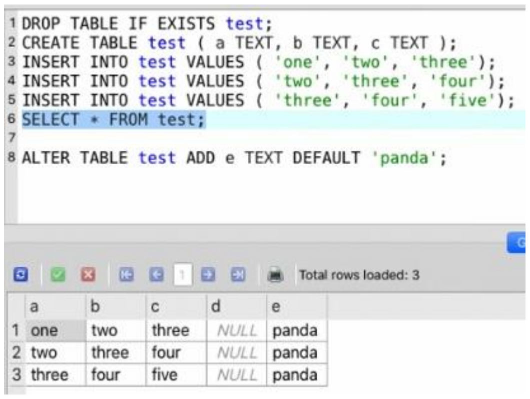
- UPDATE table_name SET col_name1 = ' ', col_name2 = ' ' WHERE ...;



```
SELECT * FROM customer;

UPDATE customer SET address = '123 Music Avenue', zip = '98056' WHERE id = 5;

UPDATE customer SET address = '2603 S Washington St', zip = '98056' WHERE id = 5;

UPDATE customer SET address = NULL, zip = NULL WHERE id = 5;
```

- DELETE ROWS: DELETE FROM table_name WHERE id=5;

- ALTER TABLE (增加列)



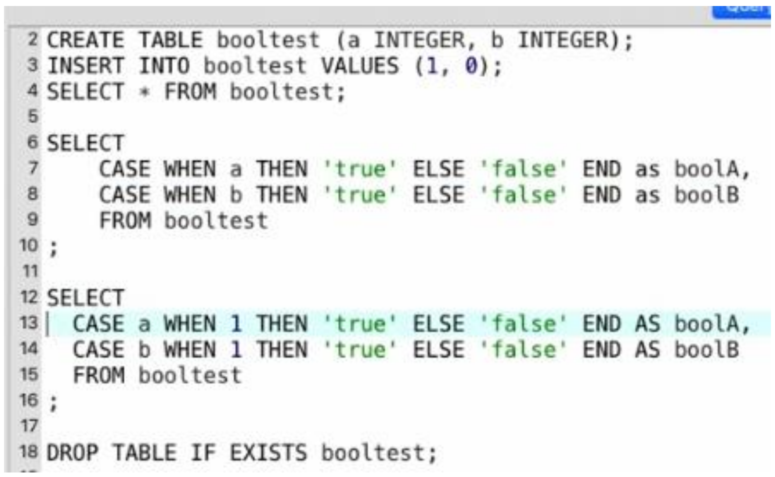
```
DROP TABLE IF EXISTS test;
CREATE TABLE test ( a TEXT, b TEXT, c TEXT );
INSERT INTO test VALUES ( 'one', 'two', 'three');
INSERT INTO test VALUES ( 'two', 'three', 'four');
INSERT INTO test VALUES ( 'three', 'four', 'five');
SELECT * FROM test;

ALTER TABLE test ADD e TEXT DEFAULT 'panda';
```

Total rows loaded: 3

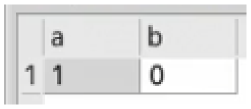
	a	b	c	d	e
1	one	two	three	NULL	panda
2	two	three	four	NULL	panda
3	three	four	five	NULL	panda

- Conditions:

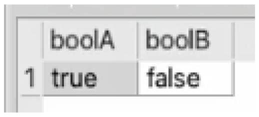
  In standard SQL, a zero (0) is considered false, and anything that's not a zero is considered true.



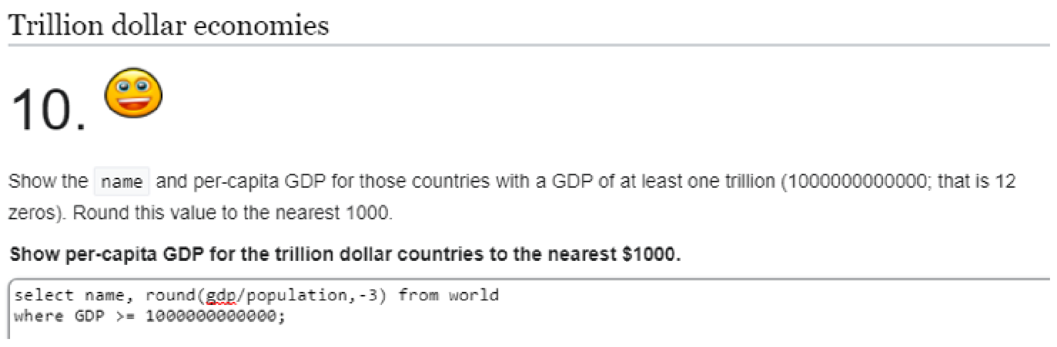
```
CREATE TABLE booltest (a INTEGER, b INTEGER);
INSERT INTO booltest VALUES (1, 0);
SELECT * FROM booltest;

SELECT
    CASE WHEN a THEN 'true' ELSE 'false' END as boolA,
    CASE WHEN b THEN 'true' ELSE 'false' END as boolB
    FROM booltest
;

SELECT
  CASE a WHEN 1 THEN 'true' ELSE 'false' END AS boolA,
  CASE b WHEN 1 THEN 'true' ELSE 'false' END AS boolB
  FROM booltest
;

DROP TABLE IF EXISTS booltest;
```



	a	b
1	1	0



	boolA	boolB
1	true	false



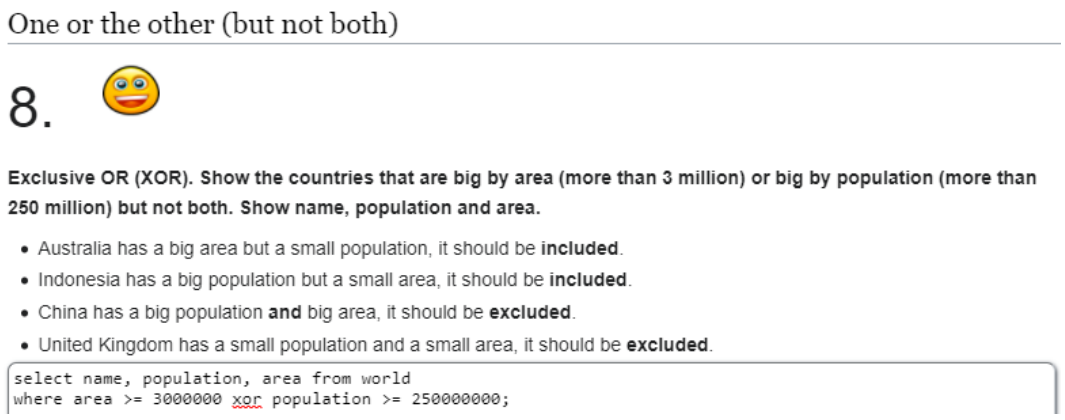
Trillion dollar economies

10.

Show the `name` and per-capita GDP for those countries with a GDP of at least one trillion (1000000000000; that is 12 zeros). Round this value to the nearest 1000.

Show per-capita GDP for the trillion dollar countries to the nearest $1000.

```
select name, round(gdp/population,-3) from world
where GDP >= 1000000000000;
```



One or the other (but not both)

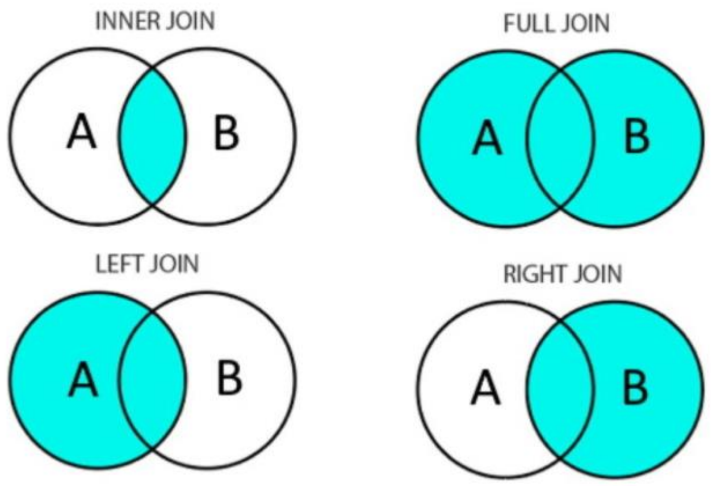
8.

Exclusive OR (XOR). Show the countries that are big by area (more than 3 million) or big by population (more than 250 million) but not both. Show name, population and area.

- Australia has a big area but a small population, it should be **included**.
- Indonesia has a big population but a small area, it should be **included**.
- China has a big population **and** big area, it should be **excluded**.
- United Kingdom has a small population and a small area, it should be **excluded**.

```
select name, population, area from world
where area >= 3000000 xor population >= 250000000;
```

- Join tables:



```
-- inner join
-- returns results only where the join condition is true
select top 1000 *
from FactInternetSales s
inner join DimProduct p on s.ProductKey = p.ProductKey
```

```
-- left join
-- returns all rows from sales, regardless of the join condition
select distinct EnglishProductName
from FactInternetSales s
left join DimProduct p on s.ProductKey = p.ProductKey
order by 1
```

- Filter data:

```
-- add filter conditions to join
select *
from FactInternetSales s
inner join DimProduct p
    on      s.ProductKey = p.ProductKey
    and     p.StartDate > '2013-01-01'

-- basic filter with WHERE
-- get sales of a specific product only
SELECT *
FROM FactInternetSales s
INNER JOIN DimProduct p ON s.ProductKey = p.ProductKey
WHERE p.EnglishProductName = 'Road-650 Black, 62'

-- non-equi-filters
-- get all orders for 2013
SELECT *
FROM FactInternetSales s
INNER JOIN DimProduct p ON s.ProductKey = p.ProductKey
WHERE       s.OrderDate >= '2013-01-01'
AND         s.OrderDate <= '2013-12-31'

-- also can use "between" for dates
SELECT *
FROM FactInternetSales s
INNER JOIN DimProduct p ON s.ProductKey = p.ProductKey
WHERE s.OrderDate BETWEEN '2013-01-01' AND '2013-12-31';

-- filter for multiple values using IN
SELECT *
FROM FactInternetSales s
INNER JOIN DimProduct p ON s.ProductKey = p.ProductKey
WHERE p.EnglishProductName in(
          'Mountain-400-W Silver, 38',
          'Mountain-400-W Silver, 40',
          'Mountain-400-W Silver, 42',
          'Mountain-400-W Silver, 46')

-- find all current and future matches with LIKE
-- % starts with ‘Mountain’ and ends with anything else

SELECT *
FROM FactInternetSales s
INNER JOIN DimProduct p ON s.ProductKey = p.ProductKey
WHERE p.EnglishProductName LIKE 'Mountain%'
```

- Aggregate data:

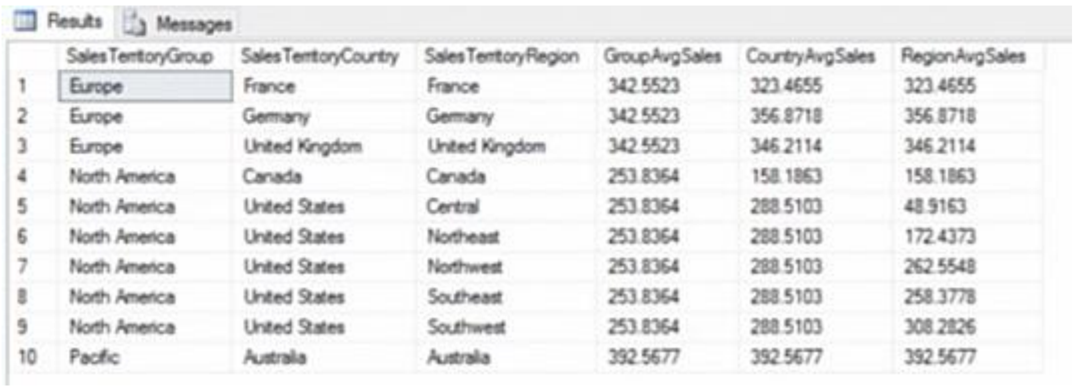
```
select OrderDate, sum(SalesAmount)
from FactInternetSales

select OrderDate, sum(SalesAmount)
from FactInternetSales
group by OrderDate
order by OrderDate
```

```
-- simple aggregations

-- Use additional aggregations to understand more about product
sales such as distribution of sales etc..
SELECT
    cat.EnglishProductCategoryName 'Category'
 ,  sub.EnglishProductSubcategoryName 'SubCategory'
 ,  count(1) 'Count' -- How many sales where there?
 ,  sum(s.SalesAmount) 'Sales' -- How much sales did we have?
 ,  avg(s.SalesAmount) 'Avg_SalesAmount' -- What was the Avg sale amount?
 ,  min(s.SalesAmount) 'Min_SaleAmount' -- What was the Min sale amount?
 ,  max(s.SalesAmount) 'Max_SaleAmount' -- What was the Max sale amount
FROM FactInternetSales s
LEFT JOIN DimProduct p ON s.ProductKey = p.ProductKey
LEFT JOIN DimProductSubcategory sub ON p.ProductSubcategoryKey =
sub.ProductSubcategoryKey
LEFT JOIN DimProductCategory cat ON sub.ProductCategoryKey =
cat.ProductCategoryKey
-- must use group by in order for aggregation to work properly
GROUP BY
    cat.EnglishProductCategoryName -- column aliases aren't allowed
 ,  sub.EnglishProductSubcategoryName
ORDER BY
    cat.EnglishProductCategoryName
,   sub.EnglishProductSubcategoryName


-- filter to 2013 with WHERE(Before GROUP BY)
SELECT
    YEAR(s.OrderDate) 'Year'
 ,  cat.EnglishProductCategoryName 'Category'
 ,  sub.EnglishProductSubcategoryName 'SubCategory'
 ,  count(1) 'Count' -- use 1 instead of a field for faster performance
 ,  sum(s.SalesAmount) 'Sales'
 ,  avg(s.SalesAmount) 'Avg_Quantity'
 ,  min(s.SalesAmount) 'Min_SaleAmount'
 ,  max(s.SalesAmount) 'Max_SaleAmount'

FROM FactInternetSales s
INNER JOIN DimProduct p ON s.ProductKey = p.ProductKey
INNER JOIN DimProductSubcategory sub ON p.ProductSubcategoryKey =
sub.ProductSubcategoryKey
INNER JOIN DimProductCategory cat ON sub.ProductCategoryKey =
cat.ProductCategoryKey
-- filter
WHERE YEAR(s.OrderDate) = 2013 --use date function to parse year
-- must use group by in order for aggregation to work properly
GROUP BY
    YEAR(s.OrderDate)
,   cat.EnglishProductCategoryName -- column aliases aren't allowed
,   sub.EnglishProductSubcategoryName
ORDER BY
    cat.EnglishProductCategoryName
,   sub.EnglishProductSubcategoryName
```

```sql
-- Only show products in 2013 that sold more than $1M USD
SELECT
    cat.EnglishProductCategoryName 'Category'
,   sub.EnglishProductSubcategoryName 'SubCategory'
,   count(1) 'Count' -- use 1 instead of a field for faster performance
,   sum(s.SalesAmount) 'Sales'
,   avg(s.SalesAmount) 'Avg_Quantity'
,   min(s.SalesAmount) 'Min_SaleAmount'
,   max(s.SalesAmount) 'Max_SaleAmount'
FROM FactInternetSales s
INNER JOIN DimProduct p ON s.ProductKey = p.ProductKey
INNER JOIN DimProductSubcategory sub ON p.ProductSubcategoryKey =
sub.ProductSubcategoryKey
INNER JOIN DimProductCategory cat ON sub.ProductCategoryKey =
cat.ProductCategoryKey
-- filter
WHERE YEAR(s.OrderDate) = 2013 --use date function to parse year
-- must use group by in order for aggregation to work properly
GROUP BY
    cat.EnglishProductCategoryName -- column aliases aren't allowed
,   sub.EnglishProductSubcategoryName
-- use HAVING to filter after the aggregate is computed
HAVING
    sum(s.SalesAmount) > 1000000
ORDER BY
    cat.EnglishProductCategoryName
 ,  sub.EnglishProductSubcategoryName
```

- Window functions: specify partitions and ordering for the purpose of aggregation.

```sql
-- Window Functions
/*
OVER()
    -- executes an aggregation over a given partition and sort order
    -- works with Ranking, Aggregate and Analytics functions
*/

-- Show each sales average for Group, Country, and Region all in one
query
SELECT DISTINCT
    t.SalesTerritoryGroup
,   t.SalesTerritoryCountry
,   t.SalesTerritoryRegion
,   AVG(s.SalesAmount) OVER(PARTITION BY t.SalesTerritoryGroup ) as
'GroupAvgSales'
,   AVG(s.SalesAmount) OVER(PARTITION BY t.SalesTerritoryCountry ) as
'CountryAvgSales'
,   AVG(s.SalesAmount) OVER(PARTITION BY t.SalesTerritoryRegion ) as
'RegionAvgSales'

FROM FactInternetSales s
JOIN DimSalesTerritory t ON
    s.SalesTerritoryKey = t.SalesTerritoryKey
WHERE
    YEAR(s.OrderDate) = 2013
ORDER BY
    1,2,3 – order by column 1 then 2, then 3.
```

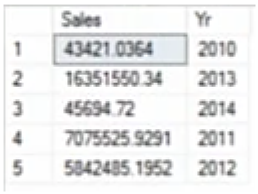


Results | Messages

	SalesTerritoryGroup	SalesTerritoryCountry	SalesTerritoryRegion	GroupAvgSales	CountryAvgSales	RegionAvgSales
1	Europe	France	France	342.5523	323.4655	323.4655
2	Europe	Germany	Germany	342.5523	356.8718	356.8718
3	Europe	United Kingdom	United Kingdom	342.5523	346.2114	346.2114
4	North America	Canada	Canada	253.8364	158.1863	158.1863
5	North America	United States	Central	253.8364	288.5103	48.9163
6	North America	United States	Northeast	253.8364	288.5103	172.4373
7	North America	United States	Northwest	253.8364	288.5103	262.5548
8	North America	United States	Southeast	253.8364	288.5103	258.3778
9	North America	United States	Southwest	253.8364	288.5103	308.2826
10	Pacific	Australia	Australia	392.5677	392.5677	392.5677

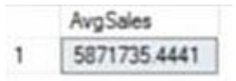
- Subqueries:

```
-- Use a sub-query to aggregate an underlying Table
select *
from (
    select sum(SalesAmount) as 'Sales', YEAR(OrderDate) as 'Yr'
    from FactInternetSales
    group by YEAR(OrderDate)
) YrSales
```



	Sales	Yr
1	43421.0364	2010
2	16351550.34	2013
3	45694.72	2014
4	7075525.9291	2011
5	5842485.1952	2012

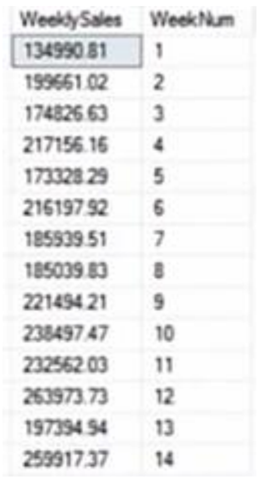
```
-- Create new aggregates on to of derived
select avg(Sales) as 'AvgSales'
from (
    select sum(SalesAmount) as 'Sales', YEAR(OrderDate) as 'Yr'
    from FactInternetSales
    group by YEAR(OrderDate)
) YrSales
```



	AvgSales
1	5871735.4441

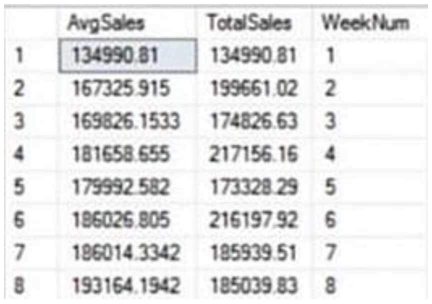
```
-- Use a subquery to test if values are IN another table
SELECT EnglishProductName 'Product'
FROM DimProduct p
WHERE p.ProductSubcategoryKey IN
    (SELECT sc.ProductSubcategoryKey
     FROM DimProductSubcategory sc
     WHERE sc.EnglishProductSubcategoryName = 'Wheels')
-- Re-write this as a Join instead (since the tables are in a same DB)
SELECT     p.EnglishProductName
FROM       DimProduct p
JOIN       DimProductSubcategory sc ON p.ProductSubcategoryKey =
sc.ProductSubcategoryKey
WHERE      sc.EnglishProductSubcategoryName = 'Wheels'

-- Use EXISTS to test if the outer queries value is present in the
sub-query
-- Sometimes this is the only way to express this join type

SELECT EnglishProductName 'Product'
FROM DimProduct p
WHERE EXISTS
    (SELECT * -- no data is returned, only a Boolean true/false
     FROM DimProductSubcategory sc
     WHERE p.ProductSubcategoryKey = sc.ProductSubcategoryKey
    AND   sc.EnglishProductSubcategoryName = 'Wheels')
```

- Rolling calculations: (Moving Averages, YTD totals, % change in growth)



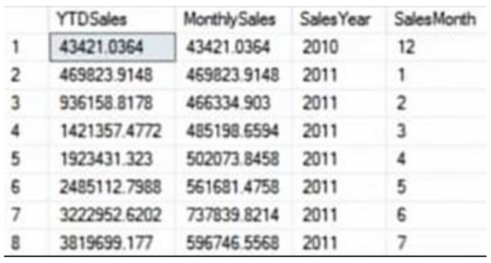
WeeklySales	WeekNum
134990.81	1
199661.02	2
174826.63	3
217156.16	4
173328.29	5
216197.92	6
185939.51	7
185039.83	8
221494.21	9
238497.47	10
232562.03	11
263973.73	12
197394.94	13
259917.37	14

```
-- Show a 6 week rolling average of Weekly Sales for 2013

-- first create weekly sales totals
SELECT SUM(s.SalesAmount) 'WeeklySales'
,       DATEPART(ww, s.OrderDate) as 'WeekNum'
FROM    FactInternetSales s
WHERE   YEAR(s.OrderDate) = 2013
GROUP BY
        DATEPART(ww, s.OrderDate)
ORDER BY
        DATEPART(ww, s.OrderDate) ASC
-- use that subquery as our source and calculate the
moving average (every 6 weeks)
SELECT
        AVG(WeeklySales) OVER (ORDER BY WeekNum ROWS BETWEEN 6 PRECEDING AND CURRENT
ROW) as AvgSales
,       WeeklySales as 'TotalSales'
,       WeekNum
FROM (– table of "weekly sales totals" we just created
        SELECT SUM(s.SalesAmount) 'WeeklySales'
        ,       DATEPART(ww, s.OrderDate) as 'WeekNum'
        FROM    FactInternetSales s
        WHERE   YEAR(s.OrderDate) = 2013
        GROUP BY
                DATEPART(ww, s.OrderDate)
        ) AS s
GROUP BY
        WeekNum, WeeklySales
ORDER BY
        WeekNum ASC
```



	AvgSales	TotalSales	WeekNum
1	134990.81	134990.81	1
2	167325.915	199661.02	2
3	169826.1533	174826.63	3
4	181658.655	217156.16	4
5	179992.582	173328.29	5
6	186026.805	216197.92	6
7	186014.3342	185939.51	7
8	193164.1942	185039.83	8

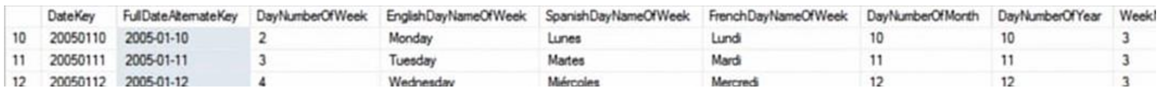
```
-- Running Total (Year-To-Date Total)
*  YTD refers to the period beginning the first day of the current calendar
year or fiscal year up to the current date.
SELECT
        SUM(MonthlySales) OVER (PARTITION BY SalesYear ORDER BY SalesMonth ROWS
UNBOUNDED PRECEDING) as YTDSales -PARTITION BY: reset by SalesYear!!
,       MonthlySales as 'MonthlySales'
,       SalesYear
,       SalesMonth
FROM (
        SELECT SUM(s.SalesAmount) 'MonthlySales'
        ,       MONTH(s.OrderDate) as 'SalesMonth'
        ,       year(s.OrderDate) as 'SalesYear'
        FROM    FactInternetSales s
        GROUP BY
                MONTH(s.OrderDate)
        ,       year(s.OrderDate)
        ) AS s
GROUP BY
        SalesMonth, SalesYear, MonthlySales
ORDER BY
        SalesYear, SalesMonth ASC
```



	YTDSales	MonthlySales	SalesYear	SalesMonth
1	43421.0364	43421.0364	2010	12
2	469823.9148	469823.9148	2011	1
3	936158.8178	466334.903	2011	2
4	1421357.4772	485198.6594	2011	3
5	1923431.323	502073.8458	2011	4
6	2485112.7988	561681.4758	2011	5
7	3222952.6202	737839.8214	2011	6
8	3819699.177	596746.5568	2011	7

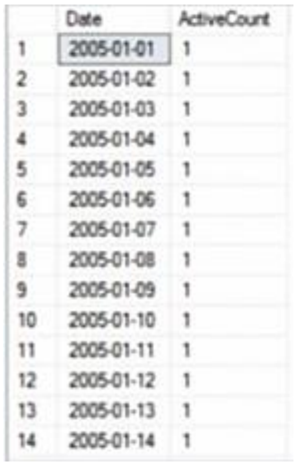
- Analyze employee data: with StartDate and EndDate
  (Active employee counts by given date, Attribution Rate, Active count rends)

```
-- Employee Table
select *
from DimEmployee

-- Analyzing Employee Data
-- How many active employees did we have on Nov 13th, 2013?
SELECT COUNT(1) – count the first column
FROM DimEmployee emp
WHERE StartDate <= '2013-11-13'
AND (
       EndDate > '2013-11-13'
    OR
       EndDate IS NULL
    )

-- start with dates table
select top 100 *
from DimDate
```



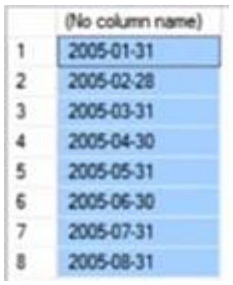
	DateKey	FullDateAlternateKey	DayNumberOfWeek	EnglishDayNameOfWeek	SpanishDayNameOfWeek	FrenchDayNameOfWeek	DayNumberOfMonth	DayNumberOfYear	WeekI
10	20050110	2005-01-10	2	Monday	Lunes	Lundi	10	10	3
11	20050111	2005-01-11	3	Tuesday	Martes	Mardi	11	11	3
12	20050112	2005-01-12	4	Wednesday	Miércoles	Mercredi	12	12	3

```
-- Show me a trend of active employees by Month
-- Start by getting the Daily count
SELECT
       dt.FullDateAlternateKey as 'Date'
,      count(1) as ActiveCount
FROM DimDate dt
LEFT JOIN     (SELECT 'Active' as 'EmpStatus', * FROM DimEmployee) emp
       -- add a new column ‘EmpStatus’ filled up with ‘Active’
       ON (dt.FullDateAlternateKey between emp.StartDate and
ISNULL(emp.EndDate,'9999-12-31'))
       -- 有多少Active的员工，同一天就会变成多少行，  then Count(1) GROUP BY date
GROUP BY
       dt.FullDateAlternateKey
ORDER BY
       1
```



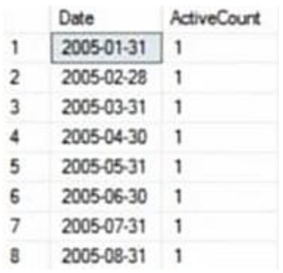
	Date	ActiveCount
1	2005-01-01	1
2	2005-01-02	1
3	2005-01-03	1
4	2005-01-04	1
5	2005-01-05	1
6	2005-01-06	1
7	2005-01-07	1
8	2005-01-08	1
9	2005-01-09	1
10	2005-01-10	1
11	2005-01-11	1
12	2005-01-12	1
13	2005-01-13	1
14	2005-01-14	1

```sql
-- Show EOM Function (the ending of the month)
select DISTINCT top 20 EOMONTH(FullDateAlternateKey)
from DimDate d
order by 1
```



	(No column name)
1	2005-01-31
2	2005-02-28
3	2005-03-31
4	2005-04-30
5	2005-05-31
6	2005-06-30
7	2005-07-31
8	2005-08-31

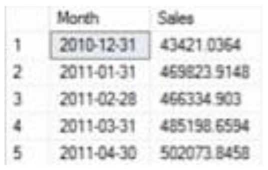
```sql
-- These counts are cumulative, so for monthly totals take
the last day of the month
SELECT
        dt.FullDateAlternateKey as 'Date'
,       count(1) as ActiveCount
FROM DimDate dt
LEFT JOIN       (SELECT 'Active' as 'EmpStatus', * FROM DimEmployee) emp
        -- regular active employees
        ON (dt.FullDateAlternateKey between emp.StartDate and
ISNULL(emp.EndDate,'9999-12-31'))
WHERE
        dt.FullDateAlternateKey = EOMONTH(dt.FullDateAlternateKey)
GROUP BY
        dt.FullDateAlternateKey
ORDER BY
        1
```



	Date	ActiveCount
1	2005-01-31	1
2	2005-02-28	1
3	2005-03-31	1
4	2005-04-30	1
5	2005-05-31	1
6	2005-06-30	1
7	2005-07-31	1
8	2005-08-31	1

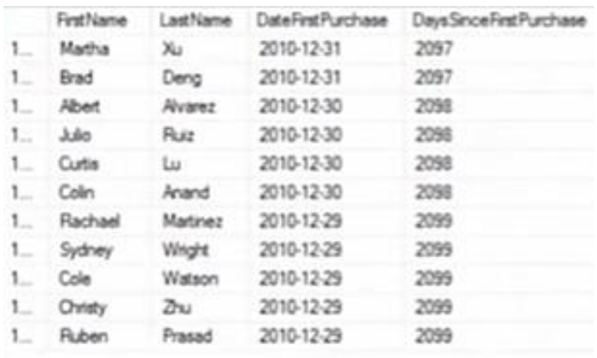
- Date and time functions:

```sql
-- Date & Time Functions
EOMONTH – END OF THE MONTH
-- Get total sales for the month and show the last day of the month
SELECT
        EOMONTH(OrderDate) as 'Month' -- SQL Server Only
,       SUM(SalesAmount) as 'Sales'
FROM FactInternetSales
GROUP BY
        EOMONTH(OrderDate)
ORDER BY 1
```



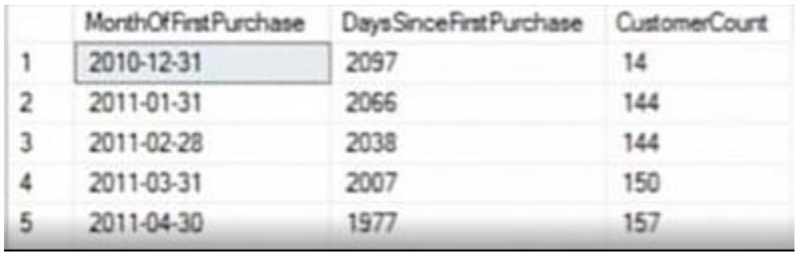
	Month	Sales
1	2010-12-31	43421.0364
2	2011-01-31	469823.9148
3	2011-02-28	466334.903
4	2011-03-31	485198.6594
5	2011-04-30	502073.8458

```sql
-- Calculate the customer acquisition funnel
SELECT
        c.FirstName
,       c.LastName
,       c.DateFirstPurchase
,       DATEDIFF(d,c.DateFirstPurchase,getdate()) as 'DaysSinceFirstPurchase'
-- How long have they been a customer?
-- Getdate()/now()/today() returns today’s date.
FROM DimCustomer c
ORDER BY 3 DESC
```



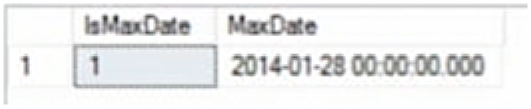
	FirstName	LastName	DateFirstPurchase	DaysSinceFirstPurchase
1...	Martha	Xu	2010-12-31	2097
1...	Brad	Deng	2010-12-31	2097
1...	Albert	Alvarez	2010-12-30	2098
1...	Julio	Ruiz	2010-12-30	2098
1...	Curtis	Lu	2010-12-30	2098
1...	Colin	Anand	2010-12-30	2098
1...	Rachael	Martinez	2010-12-29	2099
1...	Sydney	Wright	2010-12-29	2099
1...	Cole	Watson	2010-12-29	2099
1...	Christy	Zhu	2010-12-29	2099
1...	Ruben	Prasad	2010-12-29	2099

```
-- Calculate a Monthly average of customer tenure
SELECT
        EOMONTH(c.DateFirstPurchase) as 'MonthOfFirstPurchase'
        -- What month did they become a customer?
,       DATEDIFF(d,EOMONTH(c.DateFirstPurchase),getdate()) as
'DaysSinceFirstPurchase'
        -- How long have they been a customer?
,       COUNT(1) as 'CustomerCount' -- How Many customers are there for this month?
FROM DimCustomer c
GROUP BY EOMONTH(c.DateFirstPurchase)
ORDER BY 2 DESC
```

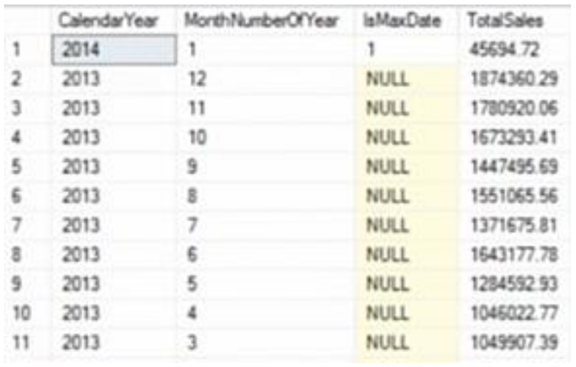


	MonthOfFirstPurchase	DaysSinceFirstPurchase	CustomerCount
1	2010-12-31	2097	14
2	2011-01-31	2066	144
3	2011-02-28	2038	144
4	2011-03-31	2007	150
5	2011-04-30	1977	157

```
-- The data might not always be updated, so lets find the latest
monthly sales amount

-- Get the most recent month
SELECT
        d.CalendarYear
,       d.MonthNumberOfYear
,       mdt.IsMaxDate
,       sum(s.SalesAmount) as 'TotalSales'

FROM DimDate d
JOIN FactInternetSales s ON d.DateKey = s.OrderDateKey
LEFT JOIN (-- SELF JOIN
        SELECT
                1 as 'IsMaxDate',
                MAX(OrderDate) as 'MaxDate'
        FROM
                FactInternetSales
    ) mdt
    ON
        d.CalendarYear = YEAR(mdt.MaxDate)
AND
        d.MonthNumberOfYear = MONTH(mdt.MaxDate)

GROUP BY
        d.CalendarYear,
        d.MonthNumberOfYear,
        mdt.IsMaxDate

ORDER BY
        1 DESC,2 DESC
```



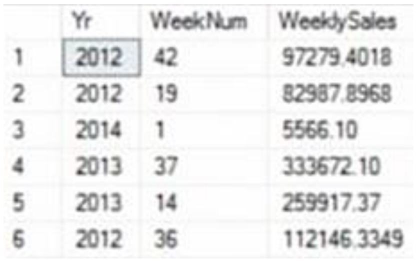
	IsMaxDate	MaxDate
1	1	2014-01-28 00:00:00.000



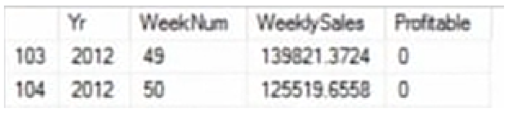
	CalendarYear	MonthNumberOfYear	IsMaxDate	TotalSales
1	2014	1	1	45694.72
2	2013	12	NULL	1874360.29
3	2013	11	NULL	1780920.06
4	2013	10	NULL	1673293.41
5	2013	9	NULL	1447495.69
6	2013	8	NULL	1551065.56
7	2013	7	NULL	1371675.81
8	2013	6	NULL	1643177.78
9	2013	5	NULL	1284592.93
10	2013	4	NULL	1046022.77
11	2013	3	NULL	1049907.39

- Common table expressions:

```
-- use a CTE to get an aggregate of an aggregate
-- Show number of profitable weeks
WITH Sales_CTE (Yr, WeekNum, WeeklySales)
AS
(
    SELECT YEAR(OrderDate) as Yr,
           DATEPART(wk,OrderDate) as WeekNum,
           sum(SalesAmount) as WeeklySales
    FROM FactInternetSales
    GROUP BY YEAR(OrderDate), DATEPART(wk,OrderDate)
)
SELECT *,
CASE WHEN WeeklySales > 140000 THEN 1 ELSE 0 END as 'Profitable'
FROM Sales_CTE
ORDER BY 1,2
GO
```



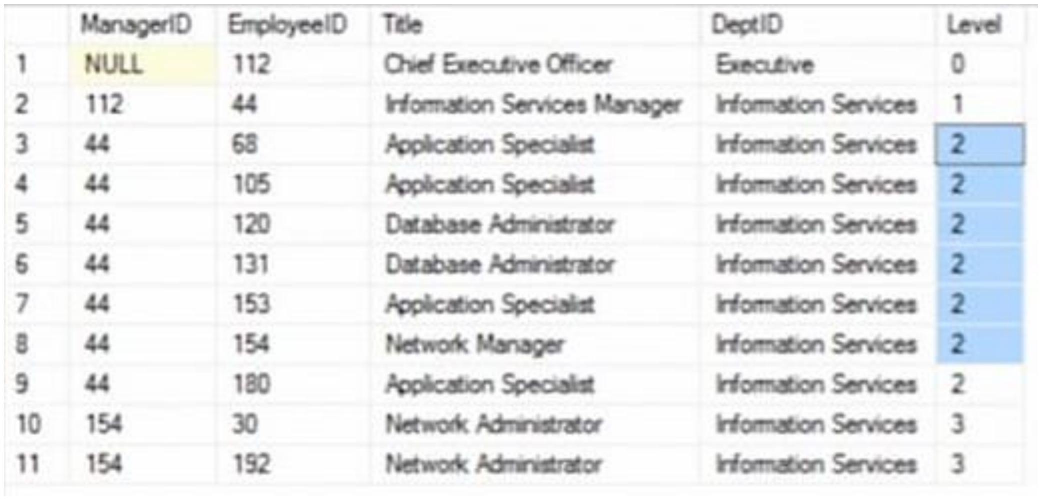
	Yr	WeekNum	WeeklySales
1	2012	42	97279.4018
2	2012	19	82987.8968
3	2014	1	5566.10
4	2013	37	333672.10
5	2013	14	259917.37
6	2012	36	112146.3349



	Yr	WeekNum	WeeklySales	Profitable
103	2012	49	139821.3724	0
104	2012	50	125519.6558	0

```
-- Summarize by Year
WITH Sales_CTE (Yr, WeekNum, WeeklySales)
AS
(
    SELECT YEAR(OrderDate) as Yr,
           DATEPART(wk,OrderDate) as WeekNum,
           sum(SalesAmount) as WeeklySales
    FROM FactInternetSales
    GROUP BY YEAR(OrderDate), DATEPART(wk,OrderDate)
)
SELECT Yr, SUM(CASE WHEN WeeklySales > 140000 THEN 1 ELSE 0 END) as 'Profitable'
FROM Sales_CTE
GROUP BY Yr
ORDER BY 1
GO
```

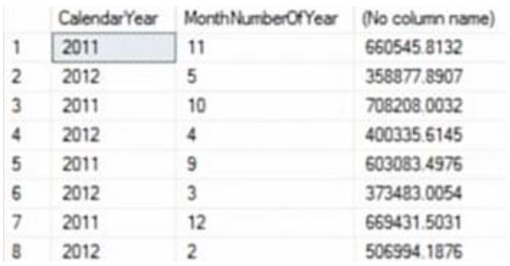
```
-- Use CTE to navigate employee hierarchy  (构建人事汇报等级关系)
WITH DirectReports (ManagerID, EmployeeID, Title, DeptID, Level)
AS
(
-- Anchor member definition
    SELECT e.ParentEmployeeKey, e.EmployeeKey, e.Title, e.DepartmentName,
        0 AS Level
    FROM DimEmployee AS e
    WHERE e.ParentEmployeeKey IS NULL -- CEO level, 无上级
    UNION ALL
-- extracts all the rows including the duplicates (repeated values) from both the
queries.
-- Recursive member definition
    SELECT e.ParentEmployeeKey, e.EmployeeKey, e.Title, e.DepartmentName,
        Level + 1   -- no.44 的Level在no.112的基础上+1, 以此类推
    FROM DimEmployee AS e
    INNER JOIN DirectReports AS d
        ON e.ParentEmployeeKey = d.EmployeeID
)
-- Statement that executes the CTE
SELECT ManagerID, EmployeeID, Title, DeptID, Level
FROM DirectReports
WHERE DeptID = 'Information Services' OR Level = 0
```



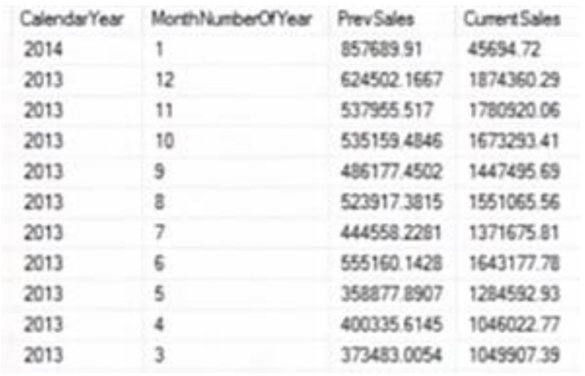
	ManagerID	EmployeeID	Title	DeptID	Level
1	NULL	112	Chief Executive Officer	Executive	0
2	112	44	Information Services Manager	Information Services	1
3	44	68	Application Specialist	Information Services	2
4	44	105	Application Specialist	Information Services	2
5	44	120	Database Administrator	Information Services	2
6	44	131	Database Administrator	Information Services	2
7	44	153	Application Specialist	Information Services	2
8	44	154	Network Manager	Information Services	2
9	44	180	Application Specialist	Information Services	2
10	154	30	Network Administrator	Information Services	3
11	154	192	Network Administrator	Information Services	3

- **Year-over-year calculations: YoY Analysis - remove seasonality**

```sql
-- Get Prev Year Sales
WITH MonthlySales (YearNum, MonthNum, Sales)
AS
(
   SELECT d.CalendarYear, d.MonthNumberOfYear,
          SUM(s.SalesAmount)
   FROM DimDate d
   JOIN FactInternetSales s ON d.DateKey = s.OrderDateKey
   GROUP BY d.CalendarYear, d.MonthNumberOfYear
)
-- Get Current Year and join to CTE for previous year
SELECT
       d.CalendarYear
,      d.MonthNumberOfYear
,      ms.Sales PrevSales
,      SUM(s.SalesAmount) CurrentSales
FROM DimDate d
JOIN FactInternetSales s ON
   d.DateKey = s.OrderDateKey
JOIN MonthlySales ms ON
   d.CalendarYear-1 = ms.YearNum AND
   d.MonthNumberOfYear = ms.MonthNum
GROUP BY
       d.CalendarYear
,      d.MonthNumberOfYear
,      ms.Sales
ORDER BY
       1 DESC, 2 DESC
```

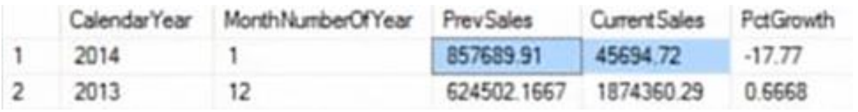


	CalendarYear	MonthNumberOfYear	(No column name)
1	2011	11	660545.8132
2	2012	5	358877.8907
3	2011	10	708208.0032
4	2012	4	400335.6145
5	2011	9	603083.4976
6	2012	3	373483.0054
7	2011	12	669431.5031
8	2012	2	506994.1876



CalendarYear	MonthNumberOfYear	PrevSales	CurrentSales
2014	1	857689.91	45694.72
2013	12	624502.1667	1874360.29
2013	11	537955.517	1780920.06
2013	10	535159.4846	1673293.41
2013	9	486177.4502	1447495.69
2013	8	523917.3815	1551065.56
2013	7	444558.2281	1371675.81
2013	6	555160.1428	1643177.78
2013	5	358877.8907	1284592.93
2013	4	400335.6145	1046022.77
2013	3	373483.0054	1049907.39

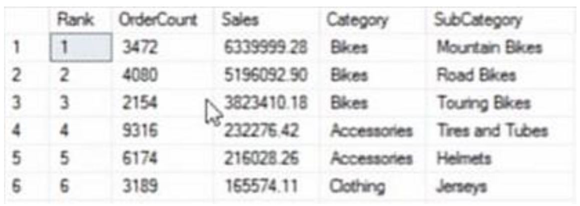
```sql
-- Now calculate the % change Year over Year
WITH MonthlySales (YearNum, MonthNum, Sales)
AS
(
       SELECT d.CalendarYear, d.MonthNumberOfYear, SUM(s.SalesAmount)
       FROM DimDate d
       JOIN FactInternetSales s ON d.DateKey = s.OrderDateKey
       GROUP BY d.CalendarYear, d.MonthNumberOfYear
)
SELECT -- Get Current Year and join to CTE for previous year
       d.CalendarYear
,      d.MonthNumberOfYear
,      ms.Sales PrevSales
,      SUM(s.SalesAmount) CurrentSales
,      (SUM(s.SalesAmount) - ms.Sales) / SUM(s.SalesAmount) 'PctGrowth'
FROM DimDate d
JOIN FactInternetSales s ON d.DateKey = s.OrderDateKey
JOIN MonthlySales ms ON
       d.CalendarYear-1 = ms.YearNum AND
       d.MonthNumberOfYear = ms.MonthNum
GROUP BY
       d.CalendarYear
,      d.MonthNumberOfYear
,      ms.Sales
ORDER BY
       1 DESC, 2 DESC
```



	CalendarYear	MonthNumberOfYear	PrevSales	CurrentSales	PctGrowth
1	2014	1	857689.91	45694.72	-17.77
2	2013	12	624502.1667	1874360.29	0.6668

- Finding Ranks: (Top-selling products overall/ Top products for each subcategory)
  - ✓ RANK()
  - ✓ DENSE_RANK()
  - ✓ ROW_NUMBER() : returns the index of each row
  - ✓ PERCENT_RANK()

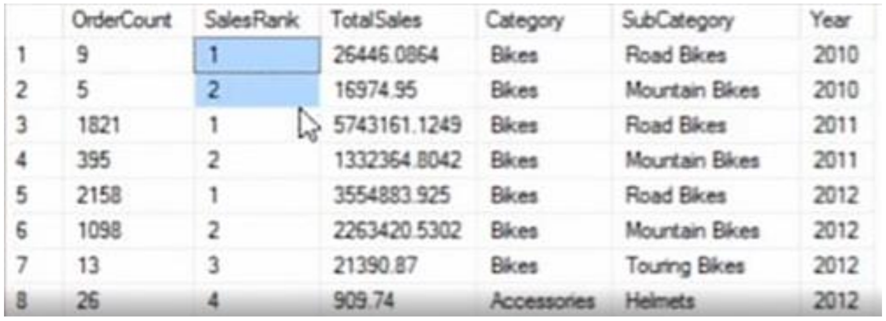
```
-- Find the top products of 2013
-- using ROW_NUMBER() as a Rank function
-- fragile solution
SELECT
       ROW_NUMBER() OVER (ORDER BY sum(s.SalesAmount) DESC)  AS 'Rank'
,      count(DISTINCT s.SalesOrderNumber) 'OrderCount'
       -- use 1 instead of a field for faster performance
,      sum(s.SalesAmount) 'Sales'
,      cat.EnglishProductCategoryName 'Category'
,      sub.EnglishProductSubcategoryName 'SubCategory'
FROM FactInternetSales s
INNER JOIN DimProduct p ON s.ProductKey = p.ProductKey
INNER JOIN DimProductSubcategory sub ON
           p.ProductSubcategoryKey = sub.ProductSubcategoryKey
INNER JOIN DimProductCategory cat ON
           sub.ProductCategoryKey = cat.ProductCategoryKey
-- filter
WHERE YEAR(s.OrderDate) = 2013 --use date function to parse year
-- must use group by in order for aggregation to work properly
GROUP BY
       cat.EnglishProductCategoryName
                -- column aliases aren't allowed
,      sub.EnglishProductSubcategoryName
ORDER BY 3 DESC;
```



	Rank	OrderCount	Sales	Category	SubCategory
1	1	3472	6339999.28	Bikes	Mountain Bikes
2	2	4080	5196092.90	Bikes	Road Bikes
3	3	2154	3823410.18	Bikes	Touring Bikes
4	4	9316	232276.42	Accessories	Tires and Tubes
5	5	6174	216028.26	Accessories	Helmets
6	6	3189	165574.11	Clothing	Jerseys

```
-- use RANK() function instead
-- when RANK() and ROW_NUBER() have the same order by the results are the same
SELECT
       ROW_NUMBER() OVER (ORDER BY sum(s.SalesAmount) DESC)  AS 'Rank'
,      count(DISTINCT s.SalesOrderNumber)
,      RANK() OVER (ORDER BY sum(s.SalesAmount) DESC) 'SalesRank'
,      sum(s.SalesAmount) 'TotalSales'
,      cat.EnglishProductCategoryName 'Category'
,      sub.EnglishProductSubcategoryName 'SubCategory'
FROM FactInternetSales s
INNER JOIN DimProduct p ON s.ProductKey = p.ProductKey
INNER JOIN DimProductSubcategory sub ON
           p.ProductSubcategoryKey = sub.ProductSubcategoryKey
INNER JOIN DimProductCategory cat ON
           sub.ProductCategoryKey = cat.ProductCategoryKey
-- filter
WHERE YEAR(s.OrderDate) = 2013 --use date function to parse year
GROUP BY -- must use group by in order for aggregation to work properly
       cat.EnglishProductCategoryName -- column aliases aren't allowed
,      sub.EnglishProductSubcategoryName
ORDER BY cat.EnglishProductCategoryName, sub.EnglishProductSubcategoryName;
```

```
-- Show the top product Sub Categories for each year
SELECT
        count(DISTINCT s.SalesOrderNumber) 'OrderCount'
,       RANK() OVER (PARTITION BY YEAR(s.OrderDate) ORDER BY sum(s.SalesAmount)
DESC) 'SalesRank'  -- Each year starts all over again
,       sum(s.SalesAmount) 'TotalSales'
,       cat.EnglishProductCategoryName 'Category'
,       sub.EnglishProductSubcategoryName 'SubCategory'
,       YEAR(s.OrderDate) 'Year'
FROM FactInternetSales s
INNER JOIN DimProduct p ON s.ProductKey = p.ProductKey
INNER JOIN DimProductSubcategory sub ON
           p.ProductSubcategoryKey = sub.ProductSubcategoryKey
INNER JOIN DimProductCategory cat ON
           sub.ProductCategoryKey = cat.ProductCategoryKey
GROUP BY -- must use group by in order for aggregation to work properly
        cat.EnglishProductCategoryName -- column aliases aren't allowed
,       sub.EnglishProductSubcategoryName
,       YEAR(s.OrderDate)
ORDER BY YEAR(s.OrderDate), SUM(s.SalesAmount) DESC;
```



	OrderCount	SalesRank	TotalSales	Category	SubCategory	Year
1	9	1	26446.0864	Bikes	Road Bikes	2010
2	5	2	16974.95	Bikes	Mountain Bikes	2010
3	1821	1	5743161.1249	Bikes	Road Bikes	2011
4	395	2	1332364.8042	Bikes	Mountain Bikes	2011
5	2158	1	3554883.925	Bikes	Road Bikes	2012
6	1098	2	2263420.5302	Bikes	Mountain Bikes	2012
7	13	3	21390.87	Bikes	Touring Bikes	2012
8	26	4	909.74	Accessories	Helmets	2012