Performance-test for lookup in table with 7.5 million records in SQL-DB

Use MySQL running XAMPP a 'pc-server' program on my own small pc,

XAMPP free open source package with web-server, database

DataModel has fact table with Sale and TimeDim table, later I added a product table

The first test was for Fact-sales-table with 2.6 mio records

```
SELECT * FROM `fact`
```

id	felt1	felt2	sale	date	idTid	time
1	0	felt2-0	60	2010-01-01	1	00:00:00
2	1	felt2-1	60	2010-01-01	1	00:00:00
3	2	felt2-2	60	2010-01-01	1	00:00:00
4	3	felt2-3	60	2010-01-01	1	00:00:00
5	4	felt2-4	60	2010-01-01	1	00:00:00
6	5	felt2-5	60	2010-01-01	1	00:00:00
7	6	felt2-6	60	2010-01-01	1	00:00:00

In performance test 2, I use Fact2-sales-table, connected up to Product-table, with 7.5 mio+ records

Tabel: fact2=fact-sale table

#	Navn	Datatype	Tegnsæt (sortering)	Attributter	Nulværdi	Standardværdi	Kommentarer	Ekstra
1	**id**	int(11)			Nej	*Ingen*		AUTO_INCREMENT
2	ProductID	int ->Product table			lej	*Ingen*		
3	**felt2**	varchar(30)	utf8mb4_general_ci		Nej	*Ingen*		
4	**sale**	float			Ja	*NULL*		
5	**date**	date			Nej	*Ingen*		
6	**idTid**	int(11)			Ja	*NULL*		
7	**time**	time			Nej	*Ingen*		

TimeDim table with 5500 records, from 15*365 days

Tabel: timedim

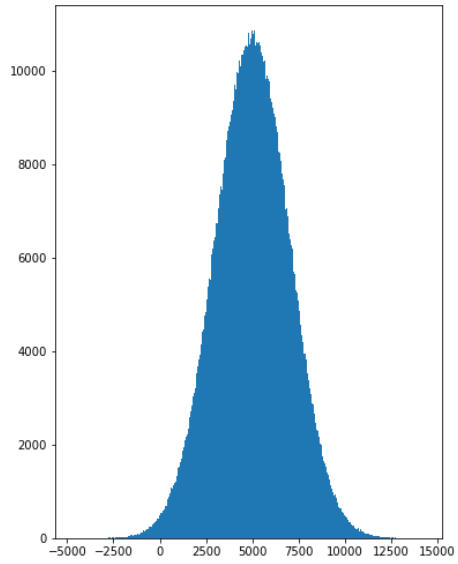
#	Navn	Datatype	Tegnsæt (sortering)	Attributter	Nulværdi	Standardværdi	Kommentarer	Ekstra
1	**id**	int(11)			Nej	*Ingen*		AUTO_INCREMENT
2	**Date**	date			Nej	*Ingen*		
3	**Day_in_week**	int(11)			Ja	*NULL*		
4	**Week_in_year**	int(11)			Ja	*NULL*		
5	**Quarter**	int(11)			Ja	*NULL*		
6	**year**	int(11)			Nej	*Ingen*		
7	**month**	int(11)			Nej	*Ingen*		
8	**day_in_month**	int(11)			Nej	*Ingen*		

Product table

```
SELECT * FROM `product` WHERE 1;
```

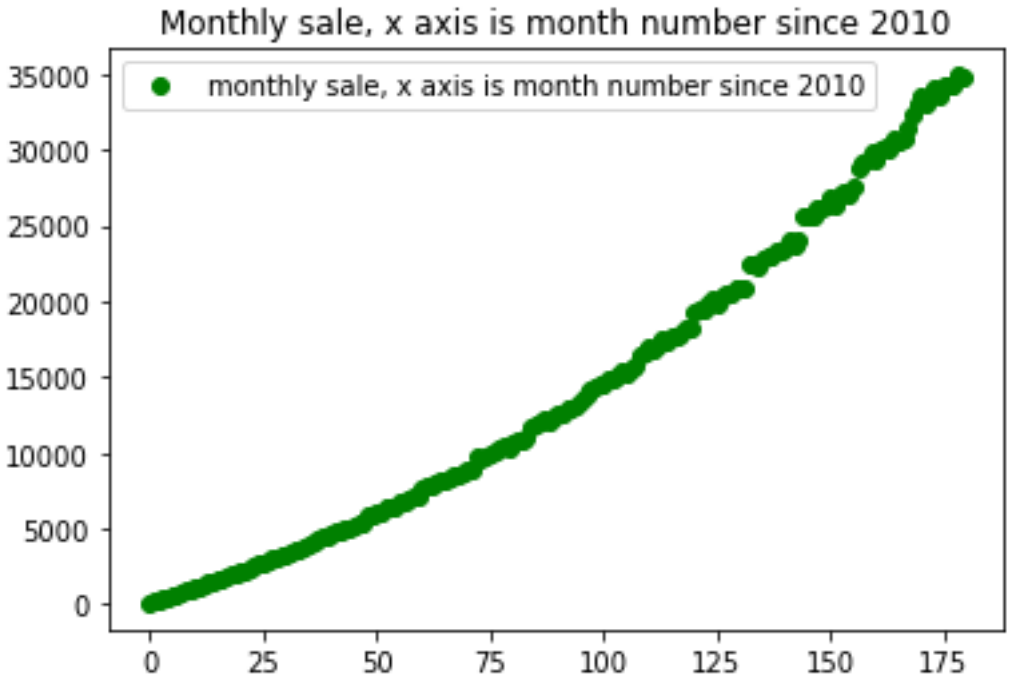
	id	name	price
Ret Kopi Slet	1	name-1	1
Ret Kopi Slet	2	name-2	2
Ret Kopi Slet	3	name-3	3
Ret Kopi Slet	4	name-4	4
Ret Kopi Slet	5	name-5	5
Ret Kopi Slet	6	name-6	6

First I generate product table with 10000 rows, product nr X is defined with: productID=X, name='name'-X, price=X



product distribution in the fact2-sale-table: Normal-distribution (mean=5000,sigma=2000)

I have used this model: +100 each month ,5% increase each year



Fact-sale-table is generated with this python code

```
############### Fact-sale-tabel-datagenerering
def generate_fact2():
   q="truncate fact2"  # auto-íncrement key resets
   mycursor.execute(q)
   normaldistributed_productIDs()
   i=0
   val = []
   idTid=1   # key ->TimeDIM.ID
   day_in_months=[[0,31,28,31,30,31,30,31,31,30,31,30,31],[0,31,29,31,30,31,30,31,31,30,31,30,31]]
   Day_in_week =datetime.date(2010,1,1).isoweekday()-1  # isoweek starts sunday, dk starts monday
   month_no_since_start=1
   print('Day_in_week:',Day_in_week)
   for y in range(2010,2025):
      for m in range(1,13):
         #for d in range(1,29):
         for d in range(1,day_in_months[y%4==0][m]+1):
            Day_in_week=Day_in_week%7
            _date=datetime.datetime(y, m, d).strftime('%Y-%m-%d')

            # for each of the 24 hours, 60 values are calculated in the j-loop, with samme sale while  productID
and felt2 are changed
            for h in range(0,23):
            #for h in range(0,1):
               tid=str(h)+':00:00'
               sale=(100+4*(h-11.5)+(Day_in_week-3)*6)*month_no_since_start*pow(1.05,(y-2010))
               # <4*(h-11.5)>=0 over hver dag, <(Day_in_week-3)*6>=0 : gennem ugen vokser værdien med 6
```

```
            # for hver mnd vokser værdien ca 100 +- lille forskel pga ugedage i mnd; 10% salgsstigning årligt
            for j in range(0,60):
                _productID=int(productID[i%10000])  # folded around 10000, from the productID distribution
                felt2='felt2-'+str(i%100)
                val.append((_productID,felt2,sale,str(_date),str(tid),idTid))
                i=i+1
        idTid+=1
        Day_in_week+=1
    month_no_since_start+=1

    sql = "INSERT INTO fact2 (productID, felt2,sale,date,time,idTid) VALUES (%s, %s, %s, %s, %s, %s)"
    mycursor.executemany(sql, val)
    mydb.commit()
    print(mycursor.rowcount, "was inserted.")  #
    val = []
print('antal rækker:',i)
```

Graph is made with this code

```
q=" SELECT MONTH(fact.date) as m, YEAR(fact.date) as y, COUNT(*), sum(fact.sale) as sum, Avg(fact.sale) as
avg FROM fact GROUP BY y,m;"
tic = time.perf_counter()
mycursor.execute(q)
myresult = mycursor.fetchall()
tic2 = time.perf_counter()
print('Dperf_counter():',tic2-tic)
print('antal:',len(myresult) )
monthly_sale=myresult
#graf:
x=[]
y=[]
m=0
for r in monthly_sale:
  x.append(m)
  y.append(r[4])   #avg sale
  m+=1
import matplotlib.pyplot as plt
plt.plot(x,y, 'go', label='monthly sale, x axis is month number since 2010') # green circles
plt.legend()
plt.title('Monthly sale, x axis is month number since 2010')
plt.show()
```

Test search for felt1 in fact-table, int record, time ca 1.2s

100000 different values in fact, 26-27 records for each value, with 2.6mio+ rows

```
perf_cnt=0
```

```
import random
for i in range(0,10):
    tal=random.randint(0, 100000)
    tic = time.perf_counter()
    q='SELECT * FROM fact where felt1='+str(tal)
    mycursor.execute(q)
    myresult = mycursor.fetchall()
    tic2 = time.perf_counter()
    print('Dperf_counter():',tic2-tic)
    perf_cnt+=tic2-tic
    print('tal:',tal,' antal:',len(myresult))
    #Dperf_counter(): 1.2s, uden index på felt1 ; 26 eller 27 per value, passer med lidt mere end 2.6 mio+,
der er en hver 100000 i fact-table
print('avg-time: ', perf_cnt/10) # 1.2
```

Test search for felt2, string record, ca 2.1s without index, 0.8s with index

100 different values for felt2

```
perf_cnt=0
import random
for i in range(0,10):
    tal=random.randint(0, 100)

    tic = time.perf_counter()
    mycursor.execute("SELECT * FROM fact where felt2='felt2-"+str(tal)+"'")
    myresult = mycursor.fetchall()
    tic2 = time.perf_counter()
    print('Dperf_counter():',tic2-tic) # 840
    perf_cnt+=tic2-tic
    print('tal:',tal,' antal:',len(myresult))
print('avg time: ', perf_cnt/10) # 2.1s  uden index på felt2; ca 26300 rows for hver, paser med 1/100 af
rækkerne, for fact-table
# 0.8s efter indførelse af index på felt2
```

Find 100 most sold products, 2.4s for fact2 with 7.5 mio records

```
Counting no of product sold, expected top near mean for product_distribution, 5000
# product - distribution
def product_distribution()   :
    perf_cnt=0
    tic = time.perf_counter()
    q="select productid,count(*) as cnt from fact2 group by productid order by cnt desc"  #  7.5 mio rows
2.4s
```

```
   #q="select productid,count(*) as cnt,avg(sale) from fact2 group by productid order by cnt"  # 1mio tager
11100s, so this can't be used
   mycursor.execute(q)
   myresult = mycursor.fetchall()
   print('antal:',len(myresult))
   tic2 = time.perf_counter()
   print('Dperf_counter():',tic2-tic)

   for x in myresult:
     print(x)
   productdistribution= myresult
   avg=0
   i=0
   for x in myresult:
     print(x)
     avg+=x[0]*x[1]
   print(avg/7.5E6)    # ca 5000, ok
```

productdistribution=list of (productID,count) for most sold, is used in a lookup in the Product table to get name for each product:

```
   i=0
   antal=100   #list of 100 most sold products
   print('most sold products\nproductid,       name,  price, count')
   tic = time.perf_counter()
   s=''
   for x in productdistribution:
      productid=x[0]
      cnt=x[1]
      q=" SELECT p.* FROM product as p where p.id="+str(productid)
      mycursor.execute(q)
      #myresult = mycursor.fetchall()]
      myresult2 = mycursor.fetchone()
      #print(productid,myresult2[1],myresult2[2],cnt)
      s+=f'{productid:9}, {myresult2[1]:10}, {myresult2[2]:6.1f}, {cnt:3}\n'
      #print(f'{productid:9}, {myresult2[1]:10}, {myresult2[2]:6.1f}, {cnt:3}')
      i+=1
      if i>antal: break

   tic2 = time.perf_counter()
   print('Dperf_counter():',tic2-tic)  #0.03s for 100 products
   print(s)
```

most sold products
productID**, name, price , count**
5331, name-5331 , 5331.0, 9072
5606, name-5606 , 5606.0, 7561
4929, name-4929 , 4929.0, 7561
5207, name-5207 , 5207.0, 7561
5018, name-5018 , 5018.0, 7560

5312, name-5312 , 5312.0, 7560
4790, name-4790 , 4790.0, 6806
5328, name-5328 , 5328.0, 6806
4745, name-4745 , 4745.0, 6806
4974, name-4974 , 4974.0, 6806
4462, name-4462 , 4462.0, 6806

Product price is not used in fact2=sales table; sale-value is generated using date information.
I Could have used product-distribution with product price to generate Sales-value

monthly sales for most sold product during hole period, 1.4s

```
productID=5331
q =" SELECT MONTH(f.date) as m, YEAR(f.date) as y, COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg
FROM fact2 as f "
q+=" where productid="+str(productID)+" GROUP BY y,m;"   #fastest  1.34s
tic = time.perf_counter()
mycursor.execute(q)
myresult = mycursor.fetchall()
tic2 = time.perf_counter()
print('Dperf_counter():',tic2-tic)
print('antal:',len(myresult) )
for x in myresult:
  print(x)
antal: 180, =15*12
```

(1, 2010, 51, 5296.0, 103.84313725490196)
(2, 2010, 47, 9532.0, 202.80851063829786)
(3, 2010, 52, 15588.0, 299.7692307692308)
(4, 2010, 48, 18728.0, 390.1666666666667)

monthly sales for named product during hole period, approx. 1s

```
product_name='name-5606'
q =" SELECT MONTH(f.date) as m, YEAR(f.date) as y, COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg
FROM fact2 as f "
q+=" INNER JOIN product p ON f.productid=p.id where p.name='"+product_name+"' GROUP BY y,m;"

tic = time.perf_counter()
mycursor.execute(q)
myresult = mycursor.fetchall()
tic2 = time.perf_counter()
print('Dperf_counter():',tic2-tic)
print('antal:',len(myresult) )

for x in myresult:
  print(x)
total=0
for x in myresult:
   print(x)
   total+=x[2]
print('total sale in hole period:', total )
```

..end of printout
(12, 2024, 42, 1522488.21875, 36249.71949404762)
Total sale in hole period for name-5606, sales value:108139275.72607422, no of sales:7561

Test of fact -dates, finds all 480 rows(for fact-table with 2.6 mio. rows)

for some dates, ca. 0.01s

```
# opslag efter datoer
testdates=[ '2010-01-01'  , '2012-08-20', '2012-10-10' ,'2013-04-01', '2013-07-10','2014-05-05','2014-01-10','2015-01-08','2015-09-07'
        ,'2018-06-25','2019-10-25'  ,'2020-07-20' ,'2021-04-15' ,'2022-01-05' ,'2023-08-15' ,'2024-05-10'
]
#finds all 480 records
perf_cnt=0
for i in range(0,len(testdates)):
   tic = time.perf_counter()
   #q="SELECT f.*,t.date FROM fact as f INNER JOIN timedim as t ON f.date=t.date and t.date='"+testdates[i]+"'";  # same time for both sql, 0.01s
   q="SELECT f.* FROM fact as f where f.date='"+testdates[i]+"'";
   mycursor.execute(q)
   myresult = mycursor.fetchall()
   tic2 = time.perf_counter()
   print('Dperf_counter():',tic2-tic) # 840
   perf_cnt+=tic2-tic
   print('date:',testdates[i],' antal:',len(myresult))
print('gennemsnitstid, avg time:', perf_cnt/i) #   480 per entry , passer med 24*20
#0.01  med index på begge , samme
```

0.03s with 7.5 mio, 3x longer time than 2.5 mio row fact table, linear scaling with fact table size

```
explain SELECT f.* FROM fact2 as f where f.date='2014-10-05';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	f	ref	fact2_date_indx	fact2_date_indx	3	const	1380	

No reason to use TimeDim

```
EXPLAIN SELECT t.year,t.month,avg(f.salg),count(*) FROM fact as f INNER JOIN timedim as t ON f.date=t.date group by t.year,t.month;
```

[Redigér indlejret] [Ret] [Spring over Forklar SQL] [Fremstil PHP-kode]

Extra options

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t	ALL	date_indx	*NULL*	*NULL*	*NULL*	3652	Using temporary; Using filesort
1	SIMPLE	f	ref	date_indx	date_indx	3	db.t.date	249	

Plan based on estimates of fact_date_indx distribution

sum & avg over random date, each date group: ca 0.0025s

```
# sum og avg over hver en tilfældig dato, hver dato har 480/1380 entries, ca. 0.0015s
   perf_cnt=0
   for i in range(0,len(testdates)):
      tic = time.perf_counter()
      #q="SELECT f.*,t.date FROM fact as f INNER JOIN timedim as t ON f.date=t.date and
t.date='"+testdates[i]+"'";  # samme tid for begge 0.01s
      #q=" SELECT sum(f.sale), avg(f.sale), count(*) FROM fact2 as f where f.date='"+testdates[i]+"'"  # 0.005s
, with 1380 rwos
      q=" SELECT sum(f.sale), avg(f.sale), count(*) FROM fact as f where f.date='"+testdates[i]+"'"  # 0.0023s  ,
with 480 rwos
      print(q)
      mycursor.execute(q)
      myresult = mycursor.fetchall()
      tic2 = time.perf_counter()
      print('Dperf_counter():',tic2-tic) # 840
      perf_cnt+=tic2-tic
      print('date:',testdates[i],', antal:',len(myresult))
      print(myresult)
      #Dperf_counter(): 0.37700580000091577print(q)
   print('avg time:', perf_cnt/i)
```

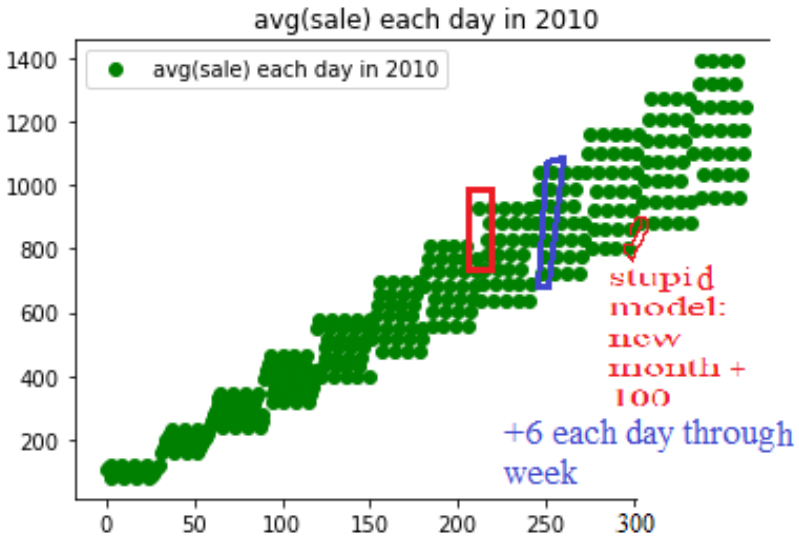
doble time for fact2-table with 7.5 mio the 2.6 mio fact table

```
explain SELECT sum(f.sale), avg(f.sale), count(*) FROM fact as f where f.date='2014-10-05';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	f	ref	fact_date_indx	fact_date_indx	3	const	480	

avg(sale) for each day in one year, 3.6s

```
   year=2010
   tic = time.perf_counter()
   q=" SELECT DAYOFYEAR(f.date) as d, COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg FROM fact2 as f
where YEAR(f.date)="+str(year)+" GROUP BY d;"   # 3.6s for all 365 days
   mycursor.execute(q)
   myresult = mycursor.fetchall()
   tic2 = time.perf_counter()
   print('Dperf_counter():',tic2-tic) # 2.5 (uden fact_date_indx ?!)   , efter index på begge går der 9-10s!
   print('antal:',len(myresult) )
   graf(myresult,3,'avg(sale) each day in 2010')
```



Average-sale over each month, via TimeDim, do not use fact.date_indx

```
tic = time.perf_counter()
q="SELECT t.year,t.month,avg(f.salg),count(*) FROM fact as f IGNORE INDEX(date_indx) INNER JOIN timedim as t ON f.date=t.date group by t.year,t.month"
mycursor.execute(q)
myresult = mycursor.fetchall()
tic2 = time.perf_counter()
print('Dperf_counter():',tic2-tic)
print('antal:',len(myresult) )
for x in myresult:
  print(x)
```

without fact_date_indx: 4.5s

```
explain SELECT t.year,t.month,avg(f.sale),count(*) FROM fact as f IGNORE INDEX(fact_date_indx) INNER JOIN timedim as t ON f.date=t.date group by t.year,t.month;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	f	ALL	*NULL*	*NULL*	*NULL*	*NULL*	2621837	Using temporary; Using filesort
1	SIMPLE	t	eq_ref	timedim_date_indx	timedim_date_indx	3	db.f.date	1	

With fact_date_indx: 15.3s , the index is not used for f

```
explain SELECT t.year,t.month,avg(f.sale),count(*) FROM fact as f INNER JOIN timedim as t ON f.date=t.date group by t.year,t.mont
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t	ALL	timedim_date_indx	*NULL*	*NULL*	*NULL*	5479	Using temporary; Using filesort
1	SIMPLE	f	ref	fact_date_indx	fact_date_indx	3	db.t.Date	239	

fastest method is not to use TimeDIM

avg sale over each month, without TimeDim , use simple filesort through fact, because we have to go through all rows in the table, for the hole period

q=" SELECT MONTH(f.date) as m, YEAR(f.date) as y, COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg FROM fact as f GROUP BY y,m;" **#fastest, 1.8s, for 2.6 mio fact-table**

#q=" SELECT MONTH(f.date) as m, YEAR(f.date) as y, COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg FROM fact2 as f GROUP BY y,m;" #fastest, 5s, 3*longer time ,

The Process is using filesort through all rows, time must be proportional to number of rows

```
explain SELECT MONTH(fact.date) as m, YEAR(fact.date) as y, COUNT(*) ,Avg(fact.sale) FROM fact GROUP BY y,m;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	fact	ALL	*NULL*	*NULL*	*NULL*	*NULL*	2621837	Using temporary; Using filesort

Fact2, 7.5 mio
1, 2010, 42780, 4242120.0, 99.16129032258064) 1380 entries/day, month: ca 31*1380=42780
(2, 2010, 38640, 7573440.0, 196.0)
(3, 2010, 42780, 12428280.0, 290.51612903225805)

Fact, 2.6mio
(1, 2010, 14880, 1505280.0, 101.16129032258064)
(2, 2010, 13440, 2688000.0, 200.0)
(3, 2010, 14880, 4412160.0, 296.51612903225805)

sum over specific (month and year), clear advantage to use Timedim, 0.1s

```
explain SELECT COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg FROM fact2 as f INNER JOIN timedim as t ON f.date=t.date where t.MONTH=1 and t.YEAR=2011;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t	index_merge	timedim_date_indx,timedim_month_indx,timedim_year_indx	timedim_year_indx,timedim_month_indx	4,4	*NULL*	30	Using intersect(timedim_year_in Using where
1	SIMPLE	f	ref	fact2_date_indx	fact2_date_indx	3	db.t.Date	618	

# sum over selected month and year, clear advantage to use Timedim
   year=2013
   month=1
   q=" SELECT COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg "
   q+=" FROM fact2 as f where MONTH(date)="+str(month)+" and YEAR(date)="+str(year)   **# 2.9s!**

   q=" SELECT COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg "
   q+=" FROM fact2 as f INNER JOIN timedim as t ON f.date=t.date where t.MONTH="+str(month)+" and t.YEAR="+str(year)   **# 0,1s!**

avg -sale per week, over the hole period, 1.7s

without TimeDim , process uses simple filesort of fact, because we have to go through all records
q=" SELECT WEEKOFYEAR(fact.date) as Q, YEAR(fact.date) as y, COUNT(*), sum(fact.sale), Avg(fact.sale) FROM fact GROUP BY y,Q " # 1.7s
tic = time.perf_counter()
mycursor.execute(q)
myresult = mycursor.fetchall()
tic2 = time.perf_counter()
print('Dperf_counter():',tic2-tic)
print('antal:',len(myresult)) #785, 15*52=780, count=480*7=3360 som udtrækket viser

```
w=0
for r in myresult:
  print(r)
  w+=1
  if w>4: break
(1, 2010, 3360, 336000.0, 100.0)
(2, 2010, 3360, 336000.0, 100.0)
(3, 2010, 3360, 336000.0, 100.0)
(4, 2010, 3360, 336000.0, 100.0)
(5, 2010, 3360, 672000.0, 200.0)
```

Avg(Sale) over specific (week and year) fastest via TimeDim, 0.05s with fact2(with 7.5mio rows)

year=2020
month=1
week_in_year=5
per day 1380, per week 7*1380=9660 entries
q=" SELECT WEEKOFYEAR(f.date) as w, YEAR(f.date) as y, COUNT(*), sum(f.sale), Avg(f.sale) FROM fact2 as f where w=50 and y=2011 " # kan man ikke
q=" SELECT WEEKOFYEAR(f.date) as w, YEAR(f.date) as y, COUNT(*), sum(f.sale), Avg(f.sale) FROM fact2 as f where WEEKOFYEAR(f.date)=50 and YEAR(f.date)=2011 " # kan man, **2.8s without timedim**
q=" SELECT WEEKOFYEAR(f.date) as w, YEAR(f.date) as y,COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg "
q+=" FROM fact2 as f INNER JOIN timedim as t ON f.date=t.date where t.week_in_year="+str(week_in_year)+" and t.YEAR="+str(year) **# 0,05s with timedim**

print(q)
antal: 1
(5, 2020, 9660, 187095097.82226562, 19368.02254888878), process use all indices on TimeDim

```
explain SELECT WEEKOFYEAR(f.date) as w, YEAR(f.date) as y,COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg FROM fact2 as f INNER JOIN timedim as t ON f.date=t.da
where t.week_in_year=5 and t.YEAR=2020;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t	index_merge	timedim_date_indx,timedim_year_indx,timedim_week_in_year_indx	timedim_week_in_year_indx,timedim_year_indx	5,4	NULL	7	Using intersect(timedim_we; Using where
1	SIMPLE	f	ref	fact2_date_indx	fact2_date_indx	3	db.t.Date	618	

Quarterly sale, no use of TimeDim is fastest, 2.2s, using SQL-Quarter(fact.date)

```
explain SELECT QUARTER(fact.date) as Q, YEAR(fact.date) as y, COUNT(*), sum(fact.sale), Avg(fact.sale) FROM fact GROUP BY y,Q;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	fact	ALL	NULL	NULL	NULL	NULL	2622074	Using temporary; Using filesort

Quarterly sale via TimeDim, 12-13s, it does not matter if fact.date or fact.idTid is used in the join of the two tables

One can do request static record the numbers in table Fact_quarterly_Sale, with keys={year,quarter}

```
# avg -sale over each quarter, via TimeDim, use  fact.date_indx ->TimeDim.date
tic = time.perf_counter()
q=" SELECT t.year,t.quarter,sum(f.sale),avg(f.sale),count(*) FROM fact as f INNER JOIN timedim as t ON
f.date=t.date group by t.year,t.quarter "   # 13s
mycursor.execute(q)
myresult = mycursor.fetchall()
tic2 = time.perf_counter()
print('Dperf_counter():',tic2-tic) # 4.5s (uden index?!)   , efter index på begge går der 15.4
for x in myresult :
  print(x)
quarterly_sale=myresult
```

```
explain SELECT t.year,t.quarter,sum(f.sale),avg(f.sale),count(*) FROM fact as f INNER JOIN timedim as t ON f.date=t.date group by t.year,t.quarter;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t	ALL	timedim_date_indx	*NULL*	*NULL*	*NULL*	5479	Using temporary; Using filesort
1	SIMPLE	f	ref	fact_date_indx	fact_date_indx	3	db.t.Date	239	

Better using fact.date than fact.idTid <->TimeDim.id, simpler, and with shorter index-key_len, no use to introduce idTid key in fact table

```
# avg-sale over each quarter, via TimeDim, use idTid <-> TimeDim.id
tic = time.perf_counter()
#q="SELECT t.year,t.month,avg(f.sale),count(*) FROM fact as f IGNORE INDEX(fact_date_indx) INNER JOIN
timedim as t ON f.date=t.date group by t.year,t.month"
q=" SELECT t.year,t.quarter,sum(f.sale),avg(f.sale),count(*) FROM fact as f INNER JOIN timedim as t ON
f.idTid=t.id group by t.year,t.quarter  "   # 12s
mycursor.execute(q)
myresult = mycursor.fetchall()
tic2 = time.perf_counter()
print('Dperf_counter():',tic2-tic
for x in myresult :
  print(x)
```

```
explain SELECT t.year,t.quarter,sum(f.sale),avg(f.sale),count(*) FROM fact as f INNER JOIN timedim as t ON f.idTid=t.id group by t.year,t.quarter;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t	ALL	PRIMARY	*NULL*	*NULL*	*NULL*	5479	Using temporary; Using filesort
1	SIMPLE	f	ref	fact_idTid_indx	fact_idTid_indx	5	db.t.id	279	

avg(sale) over specific (quarter and year), 0.26s via TimeDim

```
year=2020
Quarter=1
week_in_year=5
q=" SELECT WEEKOFYEAR(f.date) as w, YEAR(f.date) as y,COUNT(*), sum(f.sale) as sum, Avg(f.sale) as avg "
```

```
q+=" FROM fact2 as f INNER JOIN timedim as t ON f.date=t.date where t.Quarter="+str(Quarter)+" and
t.YEAR="+str(year)   # 0,05s !
print(q)
Dperf_counter(): 0.2572491999999329 s
antal: 1
(1, 2020, 125580, 2445651588.046875, 19474.84940314441)
```

Conclusion: no reason to use TimeDim, if we want collection of values over time values as year, month, quarter, date -all these can be extracted directly from fact.date -field with SQL(date) commands.

If we want a ‘point’value-sum, for one specific week,month,quarter, i.e. avg(sale) for week=X in year=Y when TimeDim is must faster