Lab Workbook

System 10 Performance & Tuning

Version 2.2

SYBASE®

The Enterprise Client/Server Company™

58165-01-0220-00

Notice

Sybase Trademarks

SYBASE, the SYBASE logo, APT-FORMS, Data Workbench, DBA Companion, Deft, Gain*Momentum*, SA Companion, SQL Debug, SQL Solutions, SQR, Transact-SQL, and VQL are registered trademarks of Sybase, Inc. ADA Workbench, Adaptable Windowing Environment, Application Manager, Applications from Models, APT Workbench, APT-build, APT-Edit, APT-Execute, APT-Translator, APT-Library, Build *Momentum*, Camelot, Client/Server Architecture for the Online Enterprise, Client/Server for the Real World, Configurator, DataServer, Data Workbench, Database Analyzer, DB-Library, Deft Analyst, Deft Designer, Deft Educational, Deft Professional, Deft Trial, Developers Workbench, Easy SQR, Embedded SQL, Enterprise Builder, Enterprise Client/Server, Enterprise Meta Server, Enterprise Modeler, Enterprise *Momentum*, Gain, Gain*Exposure*, Insight, Mainframe Access Products (MAP), *Momentum*, Movedb, Navigation Server, Net-Gateway, Net Library, Object *Momentum*, OmniSQL Gateway, Omni SQL *Server*, OmniSQL Access Module, Open Client, Open Gateway, Open Server, Open Solutions, PC APT-Execute, PC DB-Net, PC Net Library, Report Workbench, Report Execute, Open Server, Partnerships that Work, Replication Server, Resource Manager, RW-Display Lib, RW-Library, SA Monitor, Secure SQL Server, Secure SQL Toolset, SQL Code Checker, SQL Edit/TPU, SQL Edit, SQL Monitor, SQL Server, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Station, SQR Toolset, SQR Developers Kit, SQR Workbench, SYBASE Client/Server Interfaces, SYBASE Gateways, SYBASE SQL Lifecycle, Sybase Synergy Program, SYBASE Virtual Server Architecture, SYBASE User Workbench System 10, Tabular Data Stream, and The Enterprise Client/Server Company are trademarks of Sybase, Inc.

All other company and product names used herein may be the trademarks or registered trademarks of their respective companies.

Restricted Rights Legend

Use, duplication or disclosure by the Government is subject to restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608

How to Use This Guide

Exercises

- Each lab exercise includes Lab Goals, and a set of high-level tasks.
- The high-level tasks are followed by detailed instructions
- Following the instructions, you will find a set of solutions for every question.

Syntax You may find a syntax sheet following the high level set of instructions. This syntax sheet is designed to help you complete the lab.

Lab Worksheet The Lab Worksheet is designed to provide you with a set of instructions to set up your environment prior to completing your lab. Be sure to complete this worksheet before every lab.

Icons

? Occasionaly, you may be asked a question during the course of a lab. This question will be marked with the Question icon.

To find the answer to the question, look in the solutions section for that part of the lab and find the corresponding solution box.

➡ Continued: The code is continued on the following page

📖 Reference: Additional references are available for the current topic

⚡ Warning: This piece of information is important, and not adhering to it may cause you to run into some problems

Background

All of the labs in this course are based on a set of database objects and applications that were designed, developed and used by a fictitious book company called Books Unlimited. Assume Books Unlimited was founded in 1985 as a seller of technical and leisure books. Say there are currently over 300 stores in the Books Unlimited chain, with gross revenues last year reaching approximately $235 million.

Books Unlimited has contracts with a number of different authors and publishers, who provide a large number of books. The Books Unlimited database system keeps track of the inventory of books sold in all of its stores and keeps track of all individual orders, from the point of sale, through shipping, up to the point at which the bill is paid. In addition, Books Unlimited serves as an agent between the authors and publishers, providing them with sales demographics and a breakdown of earnings and royalties. Books Unlimited is also proud of its capability that allows anyone with a modem to access their system free of charge to browse through the database, including up-to-the-minute pricing.

The following types of users have access to the database:

- Customer
- Store Clerk
- Store Manager
- Shipping Clerk
- Billing Clerk
- Sales Manager
- CEO
- DBA
- Author
- Publisher

The Books Unlimited database system supports the following functional processing requirements with one or more applications each:

- Customer Assistance
  - 24 hrs/day, 7 days/week
  - Anonymous access via dial-up modem
  - Adhoc queries allowed on book database
  - No access to any other resources is allowed

- Order Entry
  - 8am-8pm, 7 days/week

- Allows only valid entries in order forms
- Allows clerks to submit sales orders
- Allows clerks to check on appropriate discounts
- Updated and cancelled sales orders require store manager approval

- Shipping
  - 9am-5pm, Mon-Fri, except holidays
  - Generate packing slips, invoice and box label for each order
  - Update shipping status when sent

- Billing
  - 9am-5pm, Mon-Fri, except holidays
  - Generate bill when order has been shipped
  - Mark order as paid when money is received
  - Generate weekly report of overdue bills

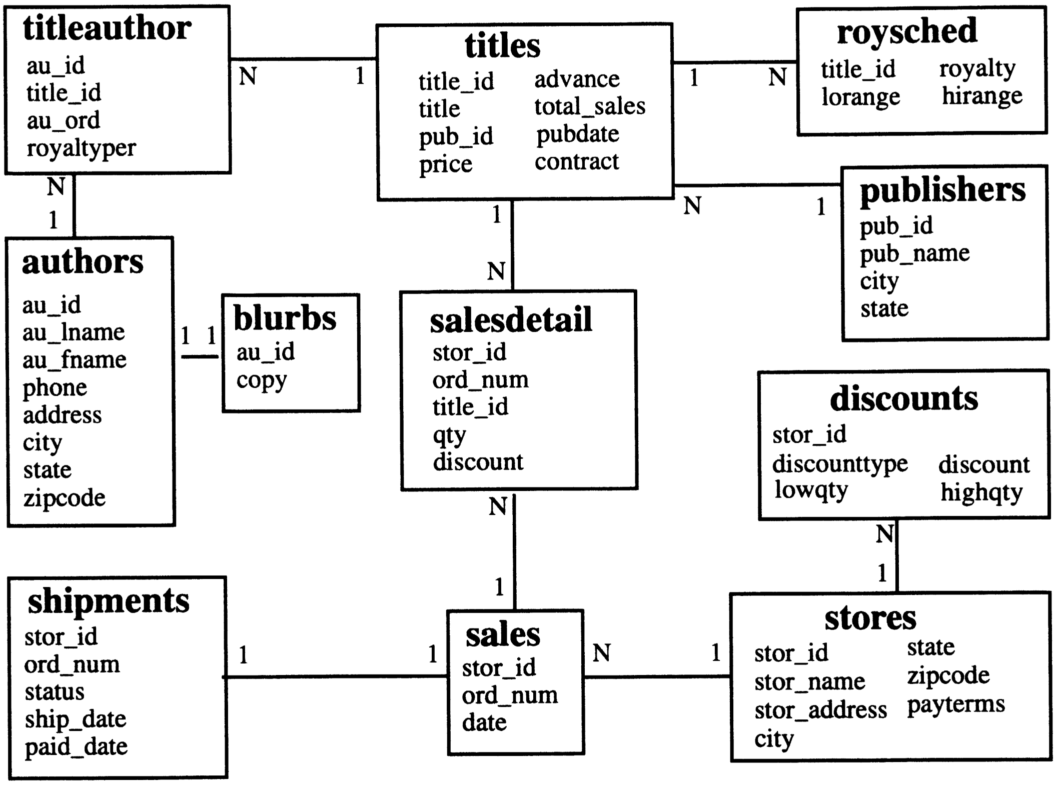
- Sales
  - 10am-3pm, Mon-Fri, except holidays
  - Generate monthly report of sales demographics
  - Compare monthly sales to same month last year

- Author Services
  - 24 hrs/day, 7 days/week
  - Authenticated access via dial-up modem
  - Update personal information
  - Verify royalties earned by book

- Publisher Services
  - 8am-8pm, Mon-Fri
  - Authenticated access via dial-up modem
  - Update address information
  - Analyze sales demographics
  - Analyze distribution of royalties

The Books Unlimited database is called *pubtune*. This database is illustrated on the next page.

pubtune Database



There are also variations on these tables, as shown on the next page.

Variations on the authors table

Table Name	Index(es)
authors	No index
authors_id	Unique clustered index on au_id
authors_idstate	Unique nonclustered index on au_id Nonclustered index on state
authors_idnames	Unique nonclustered index on au_id, au_lname, au_fname
authors_idid	Unique clustered index on au_id Nonclustered index on au_id, au_lname

Variations on the titleauthor table

Table Name	Index(es)
titleauthor	No index
titleauthor_idid	Unique clustered index on au_id, title_id
titleauthor_ididtid	Unique clustered index on au_id, title_id Nonclustered index on title_id

Variations on the titles table

Table Name	**Index(es)**
titles	No index
titles_idpr	Unique clustered index on title_id Non-clustered index on price
titles_titlid	Clustered index on title Non-clustered index on title_id Non-clustered index on pub_id
titles_pridtitl	Clustered on price, unique non-clustered on title_id, non-clustered on title

Lab 1 Performance Overview

(Student Guide, page 1-18)

Exercise Overview

Goals

- Review the definitions and concepts concerning performance and tuning

General Tasks

- Match various P&T terms with definitions
- Identify at least one performance issue for each system layers

Optional Lab

- Start the SQL Monitor Client
- Observe page IO using SQL Monitor cache window

Lab Setup

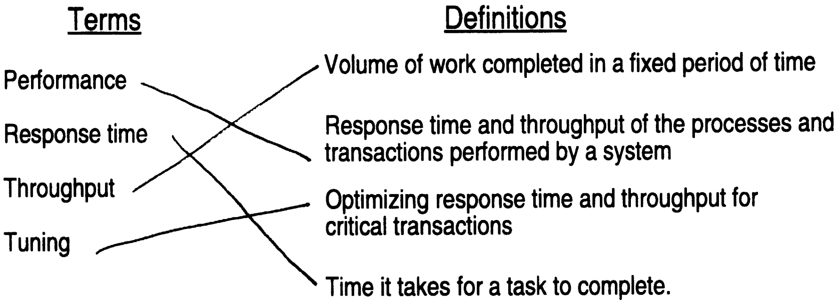
- This is a pencil and paper lab

continued on next page...

Exercises

The purpose of this lab is to review the definitions and concepts concerning database performance and tuning. This is a pencil and paper lab.

1. Match the terms on the left to the appropriate definitions on the right:



Terms

Performance

Response time

Throughput

Tuning

Definitions

Volume of work completed in a fixed period of time

Response time and throughput of the processes and transactions performed by a system

Optimizing response time and throughput for critical transactions

Time it takes for a task to complete.

2. Identify at least one performance issue for each of the following layers of the System Model:

a. Network

bandbreedte / bezetting / beschikbaarheid

(i) Hardware

disk-space , CPU - cap. , cache - grootte

(ii) Operating System

load - balancing , synchr/asynchr. i/o

(iii) SQL Server

concurrent users ; monitoring ; replication

(iv) Devices

raw - devices vs file-system ; high av.

(v) Database

design / distributed / volatility

(vi) Application

design / queries

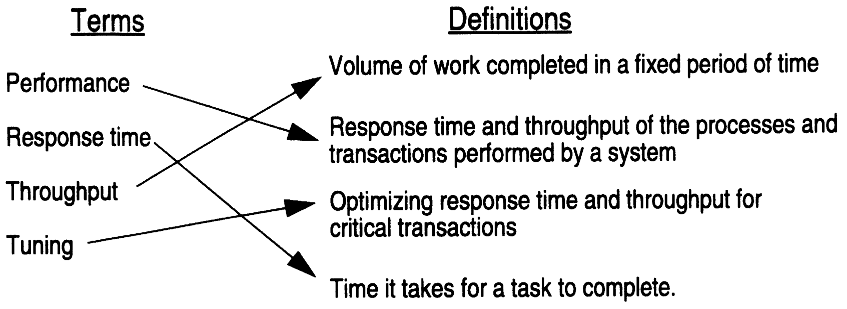
Optional Lab: SQL Monitor

1. Start the SQL Monitor Client
   a. Get the monitor server name from the instructor
   b. Start the **sqlmon** executable with the correct parameters

2. Observe page IO using SQL Monitor cache window
   a. Double click on the **Cache** window option

Solutions

1. *Match the terms on the left to the appropriate definitions on the right:*



Terms

Performance

Response time

Throughput

Tuning

Definitions

Volume of work completed in a fixed period of time

Response time and throughput of the processes and transactions performed by a system

Optimizing response time and throughput for critical transactions

Time it takes for a task to complete.

2. *Identify at least one performance issue for each of the following layers of the System Model:*

   a. Network Layer ____

   network speed, network bottleneck, network speed

   (vii) Hardware Layer

   CPU throughput, disk access, disk backup, memory usage

   (viii) Operating System Layer

   file systems, memory options, threads, priorities, network connection

   (ix) SQL Server Layer

   processing type, number of users, network loads, auditing, replication

   (x) Devices Layer

   file system, fault tolerance, controller/disks, disk array usage

   (xi) Database Layer

   backup/recovery scheme, distributed data, replicated data, auditing, volatility

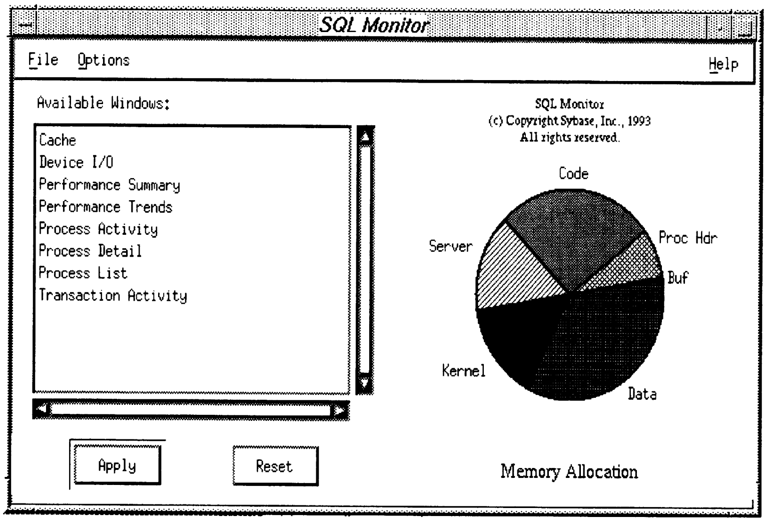
   (xii) Application Layer

   decision support vs. OLTP, transaction design, client/server design, referential integrity, auditing, security

Optional Lab: SQL Monitor

1. *Start the SQL Monitor Client*
   a. Get the monitor server name from the instructor
   b. Start the **sqlmon** executable with the correct parameters

```
gamera% sqlmon -U user_name -P passw -M monitor_server &
```



2. *Observe page IO using SQL Monitor cache window*
   a. Double click on the **Cache** window option

Lab 2 - Physical Database Design

(Student Guide, page 2-32)

Exercise Overview

Goals

- Map the I/O access from several stored procedures to database tables
- Observe the effects of denormalization on SQL Server I/O

General Tasks

- Examine *UpdateQty*, *CheckForTitle*, *PlaceOrder*, *SalesByZip* stored procedures
- Display the procedures using *sp_helptext*
- Which table is the most heavily accessed by the above mention sp's?
- Execute *SalesByZip9* and *SalesByZip9D* with statistics IO feature set
- Document the I/O with reference to the *sales, salesdetail, stores* tables
- What effect did denormalization have in this case?

Lab Setup

Consult the *pubs2* entitiy relationship diagram

Exercise Instructions...

Detailed Instructions

In this lab, you will analyze the database design relative to the OrderEntry and Sales Scenarios and examine the performance improvement achieved through denormalization.

1. Examine **UpdateQty**, **CheckForTitle**, **PlaceOrder**, **SalesByZip** stored procedures

   Note: *Do not create these procedures*

   a. Highlight the reads, writes and the tables accessed by the **UpdateQty** procedure

```
create procedure UpdateQty (@StorId char(4),@ordnum varchar(20),
@titleID tid, @qty smallint)
as
update  salesdetail
set     qty = @qty
where   stor_id =@StorId
and     ord_num = @ordnum
and     title_id = @titleID
```

   b. Highlight the reads, writes and the tables accessed by the **CheckForTitle** procedure

```
create procedure CheckForTitle (@TitleString varchar(80)='%',
@Type char(12)='business', @PubDate datetime='10/10/93',
@LowPrice money=0.00, @HighPrice money=1000.00) AS


read{   select  "TitleID"     = title_id,
                "Title"       = substring(title, 1, 30),
                "Publisher"   = substring(pub_name, 1, 15),
                "Price"       = str(price, 8, 2),
                "Available"   = contract
        from    titles, publishers
        where   titles.pub_id = publishers.pub_id
        and     title like @TitleString
        and     type like @Type
        and     pubdate >= @PubDate
        and     price between @LowPrice and @HighPrice
        order by title
```

   c. Highlight the reads, writes and the tables accessed by the **PlaceOrder** procedure

```
create procedure PlaceOrder (@StorId char(4)='S1')
```

```
        AS
        declare @ordnum varchar(20)
        BEGIN TRANSACTION
                exec InsertSalesOrder  @StorId, @ordnum OUTPUT
read sp         exec InsertSalesDetail @StorId, @ordnum,'new title one', 100
                exec InsertSalesDetail @StorId, @ordnum,'new title two', 24
        COMMIT TRANSACTION
```

InsertSalesOrder:

```
        create procedure InsertSalesOrder
        (
                @stor_id                char(4),
                @ord_num                char(20) OUTPUT
        )
        as

                declare         @stor_number            char(4)
                declare         @year                   char(4)
                declare         @max_num                int
                declare         @one_up                 char(6)
                declare         @char_year              char(5)
                declare         @formatted_stor_number  char(4)

        BEGIN TRANSACTION

nict lezen !    select @year = convert(char, datepart(year, getdate()))
                select @char_year =  @year + "%"

Read            select @stor_number = substring(stor_name, 20, 4)
                from stores
                where stor_id = @stor_id

                select @formatted_stor_number = right("0000" +
        rtrim(@stor_number), 4)

                select  @max_num = isnull(max(convert(int, right(ord_num,
        6))), 0)
Read            from    sales
                where   ord_num like @char_year
                and     stor_id = @stor_id

                select  @one_up = right("000000" + convert(char, 1 +
        @max_num), 6)

                select  @ord_num = @year + "-" + @formatted_stor_number +
        "-" + @one_up

write           insert into sales (stor_id, ord_num, date)
                values (@stor_id, @ord_num, getdate())
```

```
write          insert into shipments (stor_id, ord_num, status)
               values (@stor_id, @ord_num, 'pending')
        COMMIT TRANSACTION
```

InsertSalesDetail:

```
        create procedure InsertSalesDetail
        (
                @stor_id                char(4),
                @ord_num                varchar(20),
                @title_id               tid,
                @qty                    int
        )
        as

                declare         @vol_discount           float
                declare         @store_discount         float
                declare         @discount               float
                declare         @msg                    char(40)

                select @vol_discount = 0
                select @store_discount = 0

                select @vol_discount =  discount from discounts
                                      where @qty between lowqty and highqty
reads                                   and stor_id is null
                                        and discount is not null

                select @store_discount =        discount from discounts
                                                where stor_id = @stor_id
                                                and discount is not null

                select @discount = @vol_discount + @store_discount

write           insert into salesdetail (stor_id, ord_num, title_id, qty,
        discount)
                values (@stor_id, @ord_num, @title_id, @qty, @discount)
```

d. Highlight the reads, writes and the tables accessed by the **SalesByZip** procedure

```
        create procedure SalesByZip
        AS
Read    select  stores.zipcode,
                qty = sum(qty)
write   into    #zip_quan
        from    stores, salesdetail, sales
        where   stores.stor_id = salesdetail.stor_id
        and     salesdetail.ord_num = sales.ord_num
and      salesdetail.stor_id = sales.stor_id
and      sales.date > dateadd(month, -1, getdate())
group by stores.zipcode
```

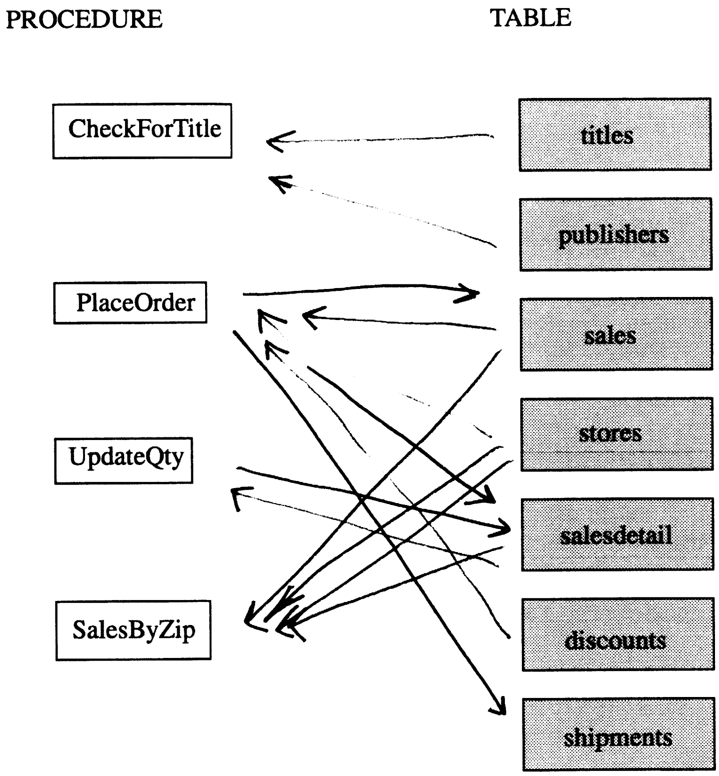
read

```
select distinct
         "Zip Code" = stores.zipcode,
         "City" = stores.city,
         "State" = stores.state,
         "Total Books Sold" = isnull(qty, 0)
from     #zip_quan, stores
where    stores.zipcode *= #zip_quan.zipcode
order by stores.zipcode

drop table #zip_quan
```

2. Display the procedures using **sp_helptext**
   a. Login to your database
   b. Execute sp_helptext

3. Fill in the composite "**Procedure-to-Table Map**"

   The "**Procedure-to-Table Map**" shows the tables involved and the reads and inserts to those tables. Express reads with an arrow leading from table to procedure, and express inserts with an arrow leading from procedure to table. For present purposes, do not try to diagram joins.



4. Which table is the most heavily accessed?

5. Examine the **SalesByZip** stored procedure

SalesByZip computes sum(qty) from stores

```
create procedure SalesByZip
AS
select  stores.zipcode,
        qty = sum(qty)
into    #zip_quan
from    stores, salesdetail, sales
where   stores.stor_id = salesdetail.stor_id
and     salesdetail.ord_num = sales.ord_num
and     salesdetail.stor_id = sales.stor_id
and     sales.date > dateadd(month, -1, getdate())
group by stores.zipcode

select distinct
        "Zip Code" = stores.zipcode,
        "City" = stores.city,
        "State" = stores.state,
        "Total Books Sold" = isnull(qty, 0)
from    #zip_quan, stores
where   stores.zipcode *= #zip_quan.zipcode
order by stores.zipcode

drop table #zip_quan
```

SalesByZip computes sum(qty) from stores.

6. Examine **SalesByZip9** stored procedure

We have created a procedure called **SalesByZip9**. SalesByZip9 is similar to SalesByZip but returns rows with a zipcode greater than '98816'.

```
create procedure SalesByZip9
AS
select  stores.zipcode,
        qty = sum(qty)
into    #zip_quan
from    stores, salesdetail, sales
where   stores.stor_id = salesdetail.stor_id
and     salesdetail.ord_num = sales.ord_num
and     salesdetail.stor_id = sales.stor_id
and     sales.date > dateadd(month, -1, getdate())
group by stores.zipcode

select distinct
```

```
        "Zip Code" = stores.zipcode,
        "City" = stores.city,
        "State" = stores.state,
        "Total Books Sold" = isnull(qty, 0)
from    #zip_quan, stores
where   stores.zipcode *= #zip_quan.zipcode
  and   stores.zipcode > '98116'
order by stores.zipcode

drop table #zip_quan
```

7. Examine the **storesD** table

We have denormalized our data by creating a table called **storesD** which contains a computed column called: sum_sales_qty. (A computation of sum(qty) from salesdetail table by store_id, the column could be maintained by a trigger.):

```
create table storesD (
         stor_id      char(4),
         stor_name    varchar(40) null,
         stor_address varchar(40) null,
         city         varchar(20) null,
         state        char(2) null,
         zipcode      char(5) null,
         payterms     varchar(12),
         sum_sales_qty int null)
```

8. Examine the **SalesByZip9D** stored procedure

We have also created a procedure called **SalesByZip9D** which uses the storesD table instead of stores:

```
create procedure SalesByZip9D
AS
select distinct
        "Zip Code" = zipcode,
        "City" = city,
        "State" = state,
        "Total Books Sold" = isnull(sum_sales_qty, 0)
from    storesD
where zipcode > '98116'
order by zipcode
```

9. Execute **SalesByZip9** and **SalesByZip9D** with statistics IO feature set to on

10. Document the I/O with reference to the **sales, salesdetail, stores** tables
    a. Use the table below to fill in the details

Effects of Denormalization

Procedure	Table	Logical Reads	Physical Reads
SalesByZip9	sales		
	salesdetail		
	stores		
	worktable		
SalesByZip9D	sales		
	salesdetail		
	storesD		
	worktable		

11. What effect did denormalization have in this case?

Solutions

*1. Examine **UpdateQty**, **CheckForTitle**, **PlaceOrder**, **SalesByZip** stored procedures*

Note: *Do not create these procedures*

a. Highlight the read and writes and tables accessed of the **UpdateQty** procedure

```
create procedure UpdateQty (@StorId char(4),@ordnum varchar(20),
@titleID tid, @qty smallint)
as
update  salesdetail
set     qty = @qty
where   stor_id =@StorId
and     ord_num = @ordnum
and     title_id = @titleID
```

b. Highlight the reads, writes and the tables accessed by the **CheckForTitle** procedure

```
create procedure CheckForTitle (@TitleString varchar(80)='%',
@Type char(12)='business', @PubDate datetime='10/10/93',
@LowPrice money=0.00, @HighPrice money=1000.00) AS


select  "TitleID"   = title_id,
        "Title"     = substring(title, 1, 30),
        "Publisher" = substring(pub_name, 1, 15),
        "Price"     = str(price, 8, 2),
        "Available" = contract
from    titles, publishers
where   titles.pub_id = publishers.pub_id
and     title like @TitleString
and     type like @Type
and     pubdate >= @PubDate
and     price between @LowPrice and @HighPrice
order by title
```

c. Highlight the reads, writes and the tables accessed by the **PlaceOrder** procedure

```
create procedure PlaceOrder (@StorId char(4)='S1')
AS
declare @ordnum varchar(20)
BEGIN TRANSACTION
        exec InsertSalesOrder  @StorId, @ordnum OUTPUT
        exec InsertSalesDetail @StorId, @ordnum,'new title one', 100
        exec InsertSalesDetail @StorId, @ordnum,'new title two', 24
COMMIT TRANSACTION
```

InsertSalesOrder:

```
create procedure InsertSalesOrder
(
        @stor_id                char(4),
        @ord_num                char(20) OUTPUT
)
as

        declare         @stor_number            char(4)
        declare         @year                   char(4)
        declare         @max_num                int
        declare         @one_up                 char(6)
        declare         @char_year              char(5)
        declare         @formatted_stor_number  char(4)

BEGIN TRANSACTION

        select @year = convert(char, datepart(year, getdate()))
        select @char_year =  @year + "%"

        select @stor_number = substring(stor_name, 20, 4)
        from stores
        where stor_id = @stor_id

        select @formatted_stor_number = right("0000" +
rtrim(@stor_number), 4)

        select  @max_num = isnull(max(convert(int, right(ord_num,
6))), 0)
        from    sales
        where   ord_num like @char_year
        and     stor_id = @stor_id

        select  @one_up = right("000000" + convert(char, 1 +
@max_num), 6)

        select  @ord_num = @year + "-" + @formatted_stor_number +
"-" + @one_up

        insert into sales (stor_id, ord_num, date)
        values (@stor_id, @ord_num, getdate())

        insert into shipments (stor_id, ord_num, status)
        values (@stor_id, @ord_num, 'pending')
COMMIT TRANSACTION
```

InsertSalesDetail:

```
create procedure InsertSalesDetail
(
        @stor_id                char(4),
        @ord_num                varchar(20),
        @title_id               tid,
        @qty                    int
)
as

        declare         @vol_discount           float
        declare         @store_discount         float
        declare         @discount               float
        declare         @msg                    char(40)

        select @vol_discount = 0
        select @store_discount = 0

        select @vol_discount =  discount from discounts
                               where @qty between lowqty and highqty
                                 and stor_id is null
                                 and discount is not null

        select @store_discount =        discount from discounts
                                        where stor_id = @stor_id
                                        and discount is not null

        select @discount = @vol_discount + @store_discount

        insert into salesdetail (stor_id, ord_num, title_id, qty,
discount)
        values (@stor_id, @ord_num, @title_id, @qty, @discount)
```

d. Highlight the reads, writes and the tables accessed by the **SalesByZip** procedure

```
create procedure SalesByZip
AS
select  stores.zipcode,
        qty = sum(qty)
into    #zip_quan
from    stores, salesdetail, sales
where   stores.stor_id = salesdetail.stor_id
and     salesdetail.ord_num = sales.ord_num
and     salesdetail.stor_id = sales.stor_id
and     sales.date > dateadd(month, -1, getdate())
```

```
group by stores.zipcode

select distinct
        "Zip Code" = stores.zipcode,
        "City" = stores.city,
        "State" = stores.state,
        "Total Books Sold" = isnull(qty, 0)
from    #zip_quan, stores
where   stores.zipcode *= #zip_quan.zipcode
order by stores.zipcode

drop table #zip_quan
```

2. *Display the procedures using **sp_helptext***

   a. Use sp_helptext to view **UpdateQty** procedure

```
edeme1% isql -Uuser -Ppassw
1> use pubtune
2> go
1> sp_helptext UpdateQty
2> go
 # Lines of Text
 ---------------
               3

(1 row affected)

         text


-----------------------------------------------------------------
------


/* Create UpdateQty procedure */

create procedure UpdateQty (@StorId char(4),@ordnum varchar(20),
@titleID tid, @qty s

         mallint)
as
update  salesdetail
set     qty = @qty
where   stor_id =@StorId
and     ord_num = @ordnum
and     title_id = @titleID
```

```
(3 rows affected, return status = 0)
1>
```

b. Use sp_helptext to view **CheckForTitle** procedure

```
1> sp_helptext CheckForTitle
2> go
 # Lines of Text
 ---------------
               3

(1 row affected)

         text

-----------------------------------------------------------------
-------------------
-----------------------


/* Create CheckForTitle procedure */

create procedure CheckForTitle (@TitleString varchar(80)='%',
@Type char(12)='business', @PubDate datetime='10/10/93',
@LowPrice money=0.00, @HighPrice money=1000.00) AS


select   "TitleID"   = title_id,
         "Title"

            = substring(title, 1, 30),
         "Publisher" = substring(pub_name, 1, 15),
         "Price"     = str(price, 8, 2),
         "Available" = contract
from     titles, publishers
where    titles.pub_id = publishers.pub_id
and      title like @TitleString
and      type like @Type
and      pubdate >=

           @PubDate
and      price between @LowPrice and @HighPrice
order by title


(3 rows affected, return status = 0)
1>
```

c. Use sp_helptext to view **PlaceOrder** procedure

```
1> sp_helptext PlaceOrder
2> go
```

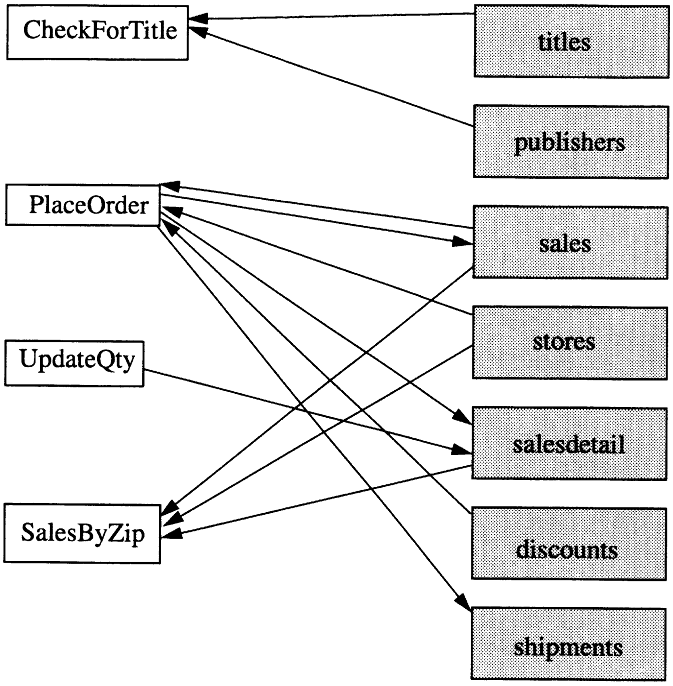
```
 # Lines of Text
 ---------------
               2

(1 row affected)

        text

-----------------------------------------------------------------
--------------
----------------------


/* Create PlaceOrder procedure */

create procedure PlaceOrder (@StorId char(4)='S1')
AS
declare @ordnum varchar(20)
BEGIN TRANSACTION
        exec InsertSalesOrder  @StorId, @ordnum OUTPUT
       exec InsertSalesDetail @StorId, @ordnum,'new title one', 100
        exec

         InsertSalesDetail @StorId, @ordnum,'new title two', 24
COMMIT TRANSACTION


(2 rows affected, return status = 0)
1>
```

d. Use sp_helptext to view **SalesByZip** procedure

```
1> sp_helptext SalesByZip
2> go
 # Lines of Text
 ---------------
               4

(1 row affected)

        text


-----------------------------------------------------------------
------


/* This script prints a report that lists the sales for each zip
code.
# It also lists the city and state for each zip code.  It does an
# outer join to ensure that all zip codes are listed, not just the
# ones that had sales.  The sort is by zip code.


         */
create procedure SalesByZip
AS
select  stores.zipcode,
        qty = sum(qty)
into    #zip_quan
from    stores, salesdetail, sales
where   stores.stor_id = salesdetail.stor_id
and     salesdetail.ord_num = sales.ord_num
and     salesdetail.stor_id = sales.stor_id
and     sales.da

         te > dateadd(month, -1, getdate())
group by stores.zipcode


select distinct
        "Zip Code" = stores.zipcode,
        "City" = stores.city,
        "State" = stores.state,
        "Total Books Sold" = isnull(qty, 0)
from    #zip_quan, stores
where   stores.zipcode *= #zip_quan.zipco

         de
order by stores.zipcode

drop table #zip_quan



(4 rows affected, return status = 0)
1>
```

3. *Fill in the composite "**Procedure-to-Table Map**"*

*The "**Procedure-to-Table Map**" shows the tables involved and the reads and inserts to those tables. Express reads with an arrow leading from table to procedure, and express inserts with an arrow leading from procedure to table. For present purposes, do not try to diagram joins.*



4. *Which table is the most heavily accessed?*

The ***salesdetail*** table and the ***sales*** table are both used by three procedures. ***salesdetail*** is updated by two of them, ***sales*** is updated by one of them.

5. *Examine the **SalesByZip** stored procedure*

SalesByZip computes sum(qty) from stores.

```
create procedure SalesByZip
AS
select  stores.zipcode,
        qty = sum(qty)
into    #zip_quan
from    stores, salesdetail, sales
where   stores.stor_id = salesdetail.stor_id
and     salesdetail.ord_num = sales.ord_num
and     salesdetail.stor_id = sales.stor_id
and     sales.date > dateadd(month, -1, getdate())
group by stores.zipcode

select distinct
        "Zip Code" = stores.zipcode,
        "City" = stores.city,
        "State" = stores.state,
        "Total Books Sold" = isnull(qty, 0)
from    #zip_quan, stores
where   stores.zipcode *= #zip_quan.zipcode
order by stores.zipcode

drop table #zip_quan
```

6. *Examine **SalesByZip9** stored procedure*

We have created a procedure called **SalesByZip9**. SalesByZip9 is similar to SalesByZip but returns rows with a zipcode greater than '98816'.

```
create procedure SalesByZip9
AS
select  stores.zipcode,
        qty = sum(qty)
into    #zip_quan
from    stores, salesdetail, sales
where   stores.stor_id = salesdetail.stor_id
and     salesdetail.ord_num = sales.ord_num
and     salesdetail.stor_id = sales.stor_id
and     sales.date > dateadd(month, -1, getdate())
group by stores.zipcode

select distinct
        "Zip Code" = stores.zipcode,
        "City" = stores.city,
```

continued on next page...

```
                "State" = stores.state,
                "Total Books Sold" = isnull(qty, 0)
        from    #zip_quan, stores
        where   stores.zipcode *= #zip_quan.zipcode
          and   stores.zipcode > '98116'
        order by stores.zipcode

        drop table #zip_quan
```

7. *Examine the* ***storesD*** *table*

   We have denormalized our data by creating a table called **storesD** which contains a computed column called: sum_sales_qty. (A computation of sum(qty) from salesdetail table by store_id, the column could be maintained by a trigger.):

```
create table storesD (
         stor_id        char(4),
         stor_name      varchar(40) null,
         stor_address   varchar(40) null,
         city           varchar(20) null,
         state          char(2) null,
         zipcode        char(5) null,
         payterms       varchar(12),
         sum_sales_qty  int null)
```

8. *Examine the* ***SalesByZip9D*** *stored procedure*

   We have also created a procedure called **SalesByZip9D** which uses the storesD table instead of stores:

```
create procedure SalesByZip9D
AS
select distinct
        "Zip Code" = zipcode,
        "City" = city,
        "State" = state,
        "Total Books Sold" = isnull(sum_sales_qty, 0)
from    storesD
where zipcode > '98116'
order by zipcode
```

9. Execute **SalesByZip9** and **SalesByZip9D** with statistics IO feature set to *on*

```
edemel% isql -Uuser -Ppassw
1> set statistics io on
2> go
```

```
Total writes for this command: 0
1> SalesByZip9
2> go
Table: stores  scan count 254,  logical reads: 5842,  physical
reads: 0
Table: salesdetail  scan count 127,  logical reads: 762,  physical
reads: 0
Table: sales  scan count 1,  logical reads: 3,  physical reads: 0
Table: Worktable  scan count 1,  logical reads: 243,  physical
reads: 0
Table: #zip_quan____01000050012395793  scan count 0,  logical
reads: 10,
physical reads: 0
Total writes for this command: 7
 Zip Code City                 State Total Books Sold
 -------- -------------------- ----- ----------------
 98288    Skykomish            WA                   0
 98324    Carlsborg            WA                   0
 98352    Mc Millin            WA                   0
 98354    Milton               WA                   0
 98455    Tacoma               WA                   0
 98816    Chelan               WA                   0
 98930    Grandview            WA                   0
 99029    Reardan              WA                   0
 99138    Inchelium            WA                   0
 99149    Malden               WA                   0
 99356    Roosevelt            WA                   0
 99625    Levelock             AK                   0
Table: #zip_quan____01000050012395793  scan count 12,  logical
reads: 12,
physical reads: 0
Table: stores  scan count 1,  logical reads: 23,  physical reads: 0
Table: Worktable  scan count 0,  logical reads: 18,  physical
reads: 0
Total writes for this command: 0
Total writes for this command: 1
Total writes for this command: 1

 (return status = 0)

 (return status = 0)

 1> SalesByZip9D
 2> go
  Zip Code City                 State Total Books Sold
  -------- -------------------- ----- ----------------
  98288    Skykomish            WA                   0
  98324    Carlsborg            WA                   0
  98352    Mc Millin            WA                   0
  98354    Milton               WA                   0
  98455    Tacoma               WA                   0
  98816    Chelan               WA                   0
  98930    Grandview            WA                  0
  99029    Reardan              WA                  0
  99138    Inchelium            WA                  0
  99149    Malden               WA                  0
  99356    Roosevelt            WA                  0
  99625    Levelock             AK                  0
Table: storesD  scan count 1,  logical reads: 23,  physical reads:
23
Table: Worktable  scan count 0,  logical reads: 18,  physical
reads: 0
Total writes for this command: 0
Total writes for this command: 0

(return status = 0)
1>
```

10. *Document the I/O with reference to the* ***sales, salesdetail, stores*** *tables*

a. Use the table below to fill in the details

Effects of Denormalization

Procedure	Table	Logical Reads	Physical Reads
SalesByZip9	sales	3	0
	salesdetail	762	0
	stores	5842+23=5865	0
	worktable	243+18=261	0
SalesByZip9D	sales	--	--
	salesdetail	--	--
	storesD	23	23
	worktable	18	0

Note that in this as in all measurements of I/O and response time, your values may be quite different from those we recorded during our test run.

11. *What effect did denormalization have in this case?*

    We were able to eliminate joins to sales and salesdetail and as a result greatly reduce I/O.

Lab 3 – Table Storage: Size

3

(Student Guide, page 3-19)

Exercise Overview

Goals

- Predict the size of database tables through different calculation methods

General Tasks

- For both *authors* and *publishers* table: (given # of datarows)
  - Calculate table size by hand
  - Calculate table size using *sp_estspace*
  - Determine actual size of table using *sp_spaceused*

Lab Setup

Consult the calculation algorithms from the student guide as reference

Exercise Instructions... ➡

Lab 3 – Table Storage: Size: Exercise Overview

Detailed Instructions

This lab concentrates on table and index storage analysis, using stored procedures to predict and measure table size.

For the authors and publishers table

1. Calculate **row size, row density** and **table size** by hand
2. Use **sp_estspace** to estimate table size
3. Use **sp_spaceused** to determine actual table size

Using the above calculations fill in worksheet:

	authors	**publishers**
Row Size (bytes)	4+150+1+1+5+1+2 = 164	4+66+1+1+2+1+2 = 77
Row Density (rows per page)	1962/164	1962/77
# of Rows	5,000	30
Table Size (hand calc)	5000 * 164 820000	30 * 77 2310
Table Size (sp_estspace)	0,46 Mb	0,02 Mb
Table Size (sp_spaceused)	448 Kb Reserved	Reserved 16 Kb

Solutions

1. *Calculate **row size, row density** and **table size** by hand*

Algorithm:

Variable-length columns:
total size of all columns (use maximum)
+ 4 bytes overhead = subtotal

[subtotal] +
+ (floor([subtotal]/256) + 1)
+ (number of variable length columns + 1)
+ 2 bytes overhead
= total

a. Hand calculations for *authors*:

```
[150]+4=154
154+(floor([154]/256)+1)=155
+(6+1)+2 = 164

Row size + O/H = 164 bytes
Density = INT(2016/row size) = INT(2016/165) = 12 rows/page
Table Size = 5000/(12 * 0.75) = 555.56 -> 556 pages = 1.14 MB
```

b. Hand calculations for *publishers*:

```
66+4=70
[70]+(floor([70]/256)+1)=71
+(3+1)+2 = 77

Row size & O/H = 77 bytes
Density = INT(2016/row size) = INT(2016/77) = 26 rows/page
Table Size = 30/(26 * 0.75) = 1.54 -> 2 pages = 4 KB
```

2. *Use **sp_estspace** to estimate the authors table size*

```
1> sp_estspace authors, 5000
2> go
authors has no indexes
 name                 type      idx_level Pages      Kbytes

 -------------------- --------- --------- ---------- ------------

 authors              data              0        237          473
 Total_Mbytes
-----------------
             0.46

(return status = 0)
1>
```

3. *Use **sp_estspace** to estimate the publishers table size*

```
1> sp_estspace publishers, 30
2> go
publishers has no indexes
 name                 type      idx_level Pages      Kbytes

 -------------------- --------- --------- ---------- ------------

 publishers           data              0          8           16

 Total_Mbytes
 -----------------
             0.02

(return status = 0)
```

4. *Use **sp_spaceused** to determine actual authors table size*

```
1> sp_spaceused authors
2> go
name       rowtotal    reserved   data       index_size unused
---------- ----------- ---------- ---------- ---------- ----------
authors           5000     464 KB     442 KB   0 KB          22 KB

(return status = 0)
```

5. *Use **sp_spaceused** to determine actual publishers table size*

```
1> sp_spaceused publishers
2> go
name       rowtotal    reserved   data       index_size unused
---------- ----------- ---------- ---------- ---------- ----------
publishers          30      16 KB       4 KB       0 KB      12 KB

(return status = 0)
```

Using the above calculations fill in worksheet::

	authors	publishers
Row Size (bytes)	164	77
Row Density (rows per page)	12	26
# of Rows	5,000	30
Table Size (hand calc)	556 pages	2 pages
Table Size (sp_estspace)	237 pages total	8 pages
Table Size (sp_spaceused)	232 actual, 11 unused	2 actual, 6 unused

Lab 4a – Indexes and I/O Activity

(Student Guide, page 4-40)

Exercise Overview

Goals

- Observe performance impact on deletes and inserts when adding indexes to a table

General Tasks

- Locate a table called **user*N*tab** (where *N* is your user number)
- Display the contents of the table
- Document the size of the table
- Build a unique clustered index on the table on column *a*
- Document the size of the table
- Write two stored procedures:
    - one that deletes 6 rows
    - another that adds 6 rows
- Document the IO statistics on the deletes and inserts
- Add a non-clustered index
- Document the size of the table
- Document the IO statistics on the deletes and inserts
- Add another non-clustered index (second non-clustered)
- Document the size of the table
- Document the IO statistics on the deletes and inserts

Lab Setup

- Use your *pubtune* database
- Use the T-SQL reference manual to help you create stored procedures

Exercise Instructions...

Lab Tables Worksheet

The following worktables are referred to in the exercise section, please use this page to document statistics.

Table1:

Table Sizes: sp_spaceused

	With No Index	Clustered	+ 1 Non-Cl	+ 2 Non-Cl
Data	12 kb	12 kb		
Index	Ø kb	4 kb		

Table2:

Logical Reads

	Clustered Index	+ 1 Non-Cl Index	+ 2 Non-Cl Indexes
Deletes	15		
	5		
Inserts	3		
	8		
	3		
	8		
	3		
	8		

Detailed Instructions

In this lab, you display the contents and size of a table. Then you create a clustered index for that table, insert and delete rows, display its size, and record I/O activity. Lastly, you add non-clustered indexes and compare sizes and I/O activity.

1. Locate a table called **user*N*tab** (where N is your user number) in your database.

   Example:
   ```
   create table userNtab
   (a int, b char(255), c char(255), d char (255))
   ```

2. Display the contents of the table. It should have 11 rows in it.

3. Document the size of the table
   a. Execute **sp_spaceused** to display the size of the table
   b. Enter the results in the *first* column of table 1 on the *Lab Tables Worksheet* page

4. Build a unique clustered index on the table on column *a*

5. Document the size of the table
   a. Execute **sp_spaceused** to display the size of the table
   b. Enter the results in the *second* column of table1 on the *Lab Tables Worksheet* page

6. Write two stored procedures: one that deletes 6 rows, another that adds 6 rows

   Example:
   ```
   create procedure delrows as
      delete userNtab where a > 5
   go
   create procedure inrows as
   declare @newrow int
   select @newrow = 6
   while @newrow < 12
      begin
         insert into userNtab values (@newrow, "a","b","c")
         select @newrow = @newrow + 1
      end
   go
   ```

7. Document the IO statistics on the deletes and inserts
   a. Set statistics i/o on,
   b. Run the delete and insert procedures.
   c. Record logical reads in the *first* column of table2 on the *Lab Tables Worksheet* page

   Note: There will be two I/O numbers for the deletes: Top one is the Transaction Log I/O and the bottom one is the data page I/O (this is a deferred delete).

8. Add a non-clustered index (first non-clustered)
   a. Set statistics io off
   b. Create a non-clustered index on column *b*

9. Document the size of the table
   a. Execute **sp_spaceused** to display the size of the table
   b. Enter the results in the *third* column of table 1 on the *Lab Tables Worksheet* page

10. Document the IO statistics on the deletes and inserts
    a. Set statistics i/o on,
    b. Run the delete and insert procedures.
    c. Record logical reads in the *second* column of table2 on the *Lab Tables Worksheet* page

11. Add another non-clustered index (second non-clustered)
    a. Set statistics io off
    b. create a non-clustered index on column *c*

12. Document the size of the table
    a. Execute **sp_spaceused** to display the size of the table
    b. Enter the results in the *fourth* column of table 1 on the *Lab Tables Worksheet* page

13. Document the IO statistics on the deletes and inserts
    a. Set statistics i/o on,
    b. Run the delete and insert procedures.
    c. Record logical reads in the *third* column of table2 on the *Lab Tables Worksheet* page
    d. Set statistics io off.

Optional Exercises

1. Determine the index id of each of your indexes with the following command:

   `select name, indid from sysindexes where id = object_id("`**userNtab**`")`

   Then use `dbcc indexalloc ("`*`table_name`*`", indid)` to display sizes of your data and indexes. Compare these results to the results you got from `sp_spaceused` earlier in this lab.

Table Sizes: indexalloc

	Table + Clustered	First Non-Clustered	Second Non-Clustered
Data			
Index			

Solutions

1. *Locate a table called* **user*N*tab** *(where N is your user number) in your database.*

   Sample:

```
create table userNtab
(a int, b char(255), c char(255), d char (255))
```

2. *Display the contents of the table. It should have 11 rows in it.*

```
1> select a, b=substring(b,1,15), c=substring(c,1,15)
2> from usertab
3> go
 a           b                c
 ----------- ---------------  ---------------
           1 a                b
           2 a                b
           3 a                b
           4 a                b
           5 a                b
           6 a                b
           7 a                b
           8 a                b
           9 a                b
          10 a                b
          11 a                b

(11 rows affected)
1>
```

3. *Document the size of the table*

   a. Execute **sp_spaceused** to display the size of the table

```
1> sp_spaceused usertab
2> go
 name                rowtotal   reserved    data     index_size unused
 ----------------    ---------  ---------   -------  ---------- -------
 usertab             11         32 KB       12 KB    0 KB       20 KB

(return status = 0)
1>
```

   b. Enter the results in the *first* column of table 1 on the *Lab Tables Worksheet* page

Table Sizes: sp_spaceused

	With No Index	Clustered	+ 1 Non-Cl	+ 2 Non-Cl
Data	12 Kb			
Index	0 Kb			

4. *Build a unique clustered index on the table on column a*

```
1> create unique clustered index indx1 on usertab(a)
2> go
1>
```

5. *Document the size of the table*

   a. Execute **sp_spaceused** again

```
1> sp_spaceused usertab
2> go
 name                 rowtotal  reserved   data    index_size unused
 ---------------- --------- --------- ------- ---------- -------
 usertab              11        64 KB      12 KB   4 KB       48 KB
(return status = 0)
1>
```

   b. Enter the results in the *second* column of table1 on the *Lab Tables Worksheet* page

Table Sizes: sp_spaceused

	With No Index	Clustered	+ 1 Non-Cl	+ 2 Non-Cl
Data	12 Kb	12 Kb		
Index	0 Kb	4 Kb		

6. *Write two stored procedures: one that deletes 6 rows, another than adds 6 rows*

   Example:

```
1> create procedure delrows as
2> delete userNtab where a > 5
3> go
1> create procedure inrows as
2> declare @newrow int
3> select @newrow = 6
4> while @newrow < 12
5>    begin
6>    insert into userNtab values (@newrow, "a","b","c")
7>    select @newrow = @newrow + 1
8>    end
9> go
```

7. *Document the IO statistics on the deletes and inserts*

   a. Set statistics i/o on

```
1> set statistics io on
2> go
Total writes for this command: 0
1>
```

   b. Run the delete and insert procedures

```
1> delrows
2> go
Table: usertab  scan count 0,  logical reads: 18, physical reads: 0
Table: usertab  scan count 1,  logical reads: 6,  physical reads: 0
Total writes for this command: 5
Total writes for this command: 5
(return status = 0)
1>

1> inrows
2> go
Table: usertab  scan count 0,  logical reads: 3,  physical reads: 0
Total writes for this command: 1
Table: usertab  scan count 0,  logical reads: 6,  physical reads: 0
Total writes for this command: 2
Table: usertab  scan count 0,  logical reads: 3,  physical reads: 0
Total writes for this command: 1
Table: usertab  scan count 0,  logical reads: 6,  physical reads: 0
Total writes for this command: 2
Table: usertab  scan count 0,  logical reads: 3,  physical reads: 0
Total writes for this command: 2
Table: usertab  scan count 0,  logical reads: 6,  physical reads: 0
Total writes for this command: 1
Total writes for this command: 1
(return status = 0)
1>
```

c. Record logical reads in the *first* column of table2 on the *Lab Tables Worksheet* page

Logical Reads

	Clustered Index	+ 1 Non-CI Index	+ 2 Non-CI Indexes
Deletes	18		
	6		
Inserts	3		
	6		
	3		
	6		
	3		
	6		

Note: There will be two I/O numbers for the deletes: Top one is the Transaction Log I/O and the bottom one is the data page I/O (this is a deferred delete).

8. *Add a non-clustered index (first non-clustered)*

a. Set statistics io off

```
1> set statistics io off
2> go
1>
```

b. create a non-clustered index on column *b*

```
1> create index indx2 on usertab(b)
2> go
1>
```

9. *Document the size of the table*

   a. Execute **sp_spaceused** to display the size of the table

```
1> sp_spaceused usertab
2> go
 name                rowtotal  reserved   data    index_size unused
 ----------------    --------- ---------  ------- ---------- -------
 usertab             11        96 KB      12 KB   12 KB      72 KB
(return status = 0)
1>
```

   b. Enter the results in the *third* column of table 1 on the *Lab Tables Worksheet* page

Table Sizes: sp_spaceused

	With No Index	Clustered	+ 1 Non-Cl	+ 2 Non-Cl
Data	12 Kb	12 Kb	12 Kb	
Index	0 Kb	4 Kb	12 Kb	

10. *Document the IO statistics on the deletes and inserts*

   a. Set statistics i/o on

```
1> set statistics io on
2> go
Total writes for this command: 0
1>
```

   b. Run the delete and insert procedures.

```
1> delrows
2> go
Table: usertab  scan count 0,  logical reads: 31, physical reads: 3
Table: usertab  scan count 1,  logical reads: 6,  physical reads: 0
Total writes for this command: 6
Total writes for this command: 6
(return status = 0)


1> inrows
2> go
Table: usertab  scan count 0,  logical reads: 5,  physical reads: 0
Total writes for this command: 2
```

```
Table: usertab  scan count 0,  logical reads: 11, physical reads: 0
Total writes for this command: 4
Table: usertab  scan count 0,  logical reads: 5,  physical reads: 0
Total writes for this command: 1
Table: usertab  scan count 0,  logical reads: 8,  physical reads: 0
Total writes for this command: 2
Table: usertab  scan count 0,  logical reads: 8,  physical reads: 0
Total writes for this command: 4
Table: usertab  scan count 0,  logical reads: 8,  physical reads: 0
Total writes for this command: 2
Total writes for this command: 2
(return status = 0)
1>
```

c. Record logical reads in the *second* column of table2 on the *Lab Tables Worksheet* page

Logical Reads

	Clustered Index	+ 1 Non-Cl Index	+ 2 Non-Cl Indexes
Deletes	15	31	
	5	6	
Inserts	3	5	
	8	11	
	3	5	
	8	8	
	3	8	
	8	8	

11. Add another non-clustered index (second non-clustered)

a. Set statistics io off

```
1> set statistics io off
2> go
```

b. create a non-clustered index on column *c*

```
1> create index indx3 on usertab(c)
2> go
```

12. *Document the size of the table*

   a. Execute **sp_spaceused** to display the size of the table

```
1> sp_spaceused usertab
2> go
 name              rowtotal  reserved   data    index_size unused
 ----------------  --------- ---------  ------- ---------- -------
 usertab           11        128 KB     12 KB   22 KB      94 KB
(return status = 0)
1>
```

   b. Enter the results in the *fourth* column of table 1 on the *Lab Tables Worksheet* page

Table Sizes: sp_spaceused

	With No Index	Clustered	+ 1 Non-Cl	+ 2 Non-Cl
Data	12 Kb	12 Kb	12 Kb	12 Kb
Index	0 Kb	4 Kb	12 Kb	22 Kb

13. *Document the IO statistics on the deletes and inserts*

   a. Set statistics i/o on

```
1> set statistics io on
2> go
Total writes for this command: 0
```

   b. Run the delete and insert procedures.

```
1> delrows
2> go
Table: usertab  scan count 0,  logical reads: 44, physical reads: 3
Table: usertab  scan count 1,  logical reads: 6,  physical reads: 0
Total writes for this command: 8
Total writes for this command: 8
(return status = 0)
```

```
1> inrows
2> go
Table: usertab  scan count 0,  logical reads: 7,  physical reads: 0
Total writes for this command: 2
Table: usertab  scan count 0,  logical reads: 13, physical reads: 0
Total writes for this command: 4
Table: usertab  scan count 0,  logical reads: 7,  physical reads: 0
Total writes for this command: 2
Table: usertab  scan count 0,  logical reads: 10, physical reads: 0
Total writes for this command: 2
Table: usertab  scan count 0,  logical reads: 13, physical reads: 0
Total writes for this command: 6
Table: usertab  scan count 0,  logical reads: 10, physical reads: 0
Total writes for this command: 2
Total writes for this command: 2
(return status = 0)
1>
```

c. Record logical reads in the *third* column of table2 on the *Lab Tables Worksheet* page

Logical Reads

	Clustered Index	+ 1 Non-Cl Index	+ 2 Non-Cl Indexes
Deletes	15	28	44
	5	5	6
Inserts	3	5	7
	8	10	13
	3	5	7
	8	10	10
	3	8	13
	8	10	10

Note that in this as in all measurements of I/O and response time, your values may be quite different from those we recorded during our test run.

d. Set statistics io off

```
1> set statistics io off
2> go
```

Optional Exercises: Solutions

1. *Determine the index id of each of your indexes with the following command:*

```
select name, indid from sysindexes where id = object_id("userNtab")
```

Then use `dbcc indexalloc ("table_name", indid)` *to display sizes of your data and indexes. Compare these results to the results you got from* `sp_spaceused` *earlier in this lab.*

Table Sizes: indexalloc

	Table + Clustered	First Non-Clustered	Second Non-Clustered
Data	6 pages	-	-
Index	2 pages	5 pages	5 pages

Note that results from dbcc indexalloc are in pages, while results from sp_spaceused are in Kilobytes.

Lab 4b – Fill Factor and I/O Statistics

(Student Guide, page 4-50)

Exercise Overview

Goals

- Measure table size with and without indexes with varying fill factors
- Measure the effect fill factor has on I/O

General Tasks

- Drop existing indexes on the *authors* table, if any
- Create a clustered index on *authors*
- Document the size of the table
- Drop the clustered index on *authors*
- Document the size of the table
- Create a clustered index with a fill factor of 30%
- Document the size of the table
- Turn *statistics io* on
- determine the number of I/O's needed to select from the table
- Drop this index
- determine the number of I/O's needed to select from the table
- What effect, if any, did fill factor have on logical and physical reads?

Lab Setup

- Use your *pubtune* database
- Use command reference to lookup commands to drop/create indexes

Exercise Instructions...

Lab Tables Worksheet

Table1:

Table Size With Different Indexes & Fill Factor

characteristics	# of rows	reserved	data	index_size	unused
clustered/default	5000	492 Kb	442 Kb	10 Kb	40 Kb
no index	5000	462	442	0	20
clustered/30%	5000	1376	1306	18	52

Table2:

Logical and Physical I/O With Different Fill Factors

	Logical Reads	Physical Reads
Fill factor = 30		
Fill factor = default		

Detailed Instructions

Measure table size with and without indexes with varying fill factors.

1. Drop existing indexes on the *authors* table, if any
   a) Run **sp_help** to locate index names
   b) Drop existing indexes

2. Create a clustered index on *authors* using the following command:

```
create unique clustered index indx1 on authors(au_id)
```

3. Document the size of the table
   a) Execute **sp_spaceused** to display the size of the table
   b) Enter the results in the *first* row of table 1 on the *Lab Tables Worksheet* page

4. Drop the clustered index on authors

5. Document the size of the table
   a) Execute **sp_spaceused** to display the size of the table
   b) Enter the results in the *second* row of table 1 on the *Lab Tables Worksheet* page

6. Does the existence or lack of an index affect the space reserved? The number of pages used for data? The number of pages used for the index?

7. Create a clustered index with a fill factor of 30% so that pages are only 30% full.

8. Document the size of the table
   a) Execute **sp_spaceused** to display the size of the table
   b) Enter the results in the *third* row of table 1 on the *Lab Tables Worksheet* page

9. Compare the third row to the other two. What effect does fill factor have on overall size of the table? What effect does it have on the number of pages used for data? For indexes?

10. Measure the effect fill factor has on I/O
    a) Drop the clustered index
    (The data pages will still be in the same state, that is, 30% full.)

    b) Turn statistics io on

    c) Determine the number of IO's needed to select from the table

    d) Write these in the *first* row of table 2 on the *Lab Tables Worksheet* page

    e) Set statistics io off

    f) Recreate the clustered index on *authors* with default fill factor

    g) Drop this index

    h) Determine the number of IO's needed to select from the table

    i) Write these in the *second* row of table 2 on the *Lab Tables Worksheet* page

    j) What effect, if any, did fill factor have on logical and physical reads?

Solutions

1. *Drop existing indexes on the authors table, if any*

   a. Run sp_help to locate index names

```
1> sp_help authors
2> go
.
.
.
 zipcode          char                      5 NULL   NULL      1 NULL
          ziprule                0

 index_name            index_description
          index_keys

 -------------------- ---------------
-----------------------
 indx1                non-clustered located on default
           au_id

No defined keys for this object.

(1 row affected, return status = 0)
1>
```

   b. Drop existing indexes

```
1> drop index authors.indx1
2> go
1>
```

2. *Create a clustered index on authors using the following command:*

```
create unique clustered index indx1 on authors(au_id)
```

```
1> create unique clustered index indx1 on authors(au_id)
2> go
1>
```

3. *Document the size of the table*

   a. Execute **sp_spaceused** to display the size of the table

```
1> sp_spaceused authors
2> go
 name              rowtotal  reserved   data    index_size unused
 ----------------- --------- ---------  ------- ---------- -------
 authors           5000      494 KB     442 KB  10 KB      42 KB
```

```
(return status = 0)
1>
```

b. Enter the results in the *first* row of table 1 on the *Lab Tables Worksheet* page

Table Size With Different Indexes & Fill Factor

characteristics	# of rows	reserved	data	index_size	unused
clustered/default	5000	494	442	10	42
no index					
clustered/30%					

4. *Drop the clustered index on authors*

```
1> drop index authors.indx1
2> go
1>
```

5. *Document the size of the table*

a. Execute **sp_spaceused** to display the size of the table

```
1> sp_spaceused authors
2> go
 name              rowtotal  reserved   data    index_size unused
 ----------------  --------- ---------  ------- ---------- -------
 authors           5000      464 KB     442 KB  0 KB       22 KB
(return status = 0)
```

b. Enter the results in the *second* row of table 1 on the *Lab Tables Worksheet* page

Table Size With Different Indexes & Fill Factor

characteristics	# of rows	reserved	data	index_size	unused
clustered/default	5000	494	442	10	42
no index	5000	464	442	0	22

Table Size With Different Indexes & Fill Factor

clustered/30%					

6. *Does the existence or lack of an index affect the space reserved? The number of pages used for data? The number of pages used for the index?*

   There is more space reserved when the clustered index exists. The number of pages used for data does not change (unless you change fill factor--see below), but some additional space is used for the index itself.

7. *Create a clustered index with a fill factor of 30% so that pages are only 30% full.*

```
1> create unique clustered index indx1 on authors(au_id)
2> with fillfactor = 30
2> go
1>
```

8. *Document the size of the table*

   a. Execute **sp_spaceused** to display the size of the table

```
1> sp_spaceused authors
2> go
 name                rowtotal  reserved   data    index_size unused
 ----------------    --------- ---------  ------- ---------- -------
 authors             5000      1374 KB    1306 KB 18 KB      50 KB
(return status = 0)
```

   b. Enter the results in the *third* row of table 1 on the *Lab Tables Worksheet* page

Table Size With Different Indexes & Fill Factor

characteristics	# of rows	reserved	data	index_size	unused
clustered/default	5000	494	442	10	42
no index	5000	464	442	0	22
clustered/30%	5000	1376	1306	18	52

9. *Compare the third row to the other two. What effect does fill factor have on overall size of the table? What effect does it have on the number of pages used for data? For indexes?*

   Creating a clustered index with a low fill factor tripled the size of the table. Number of pages went up, number of data pages went up, and index size increased.

10. *Measure the effect fill factor has on I/O*

    a) Drop the clustered index
    (The data pages will still be in the same state, that is, 30% full.)

```
1> drop index authors.indx1
2> go
1>
```

    b) Turn statistics io on

```
1> set statistics io on
2> go
Total writes for this command: 0
1>
```

    c) Determine the number of IO's needed to select from the table

```
1> select count(*) from authors
2> go

 -----------
        5000
Table: authors  scan count 1,logical reads: 653,physical reads: 652
Total writes for this command: 13

(1 row affected)
1>
```

    d) Write these in the *first* row of table 2 on the *Lab Tables Worksheet* page

Logical and Physical I/O With Different Fill Factors

	Logical Reads	Physical Reads
Fill factor = 30	653	652

Logical and Physical I/O With Different Fill Factors

	Logical Reads	Physical Reads
Fill factor = default		

e) Set statistics io off

```
1> set statistics io off
2> go
```

f) Recreate the clustered index on *authors* with default fill factor

```
1> create unique clustered index indx1 on authors(au_id)
2> go
```

g) Drop this index

```
1> drop index authors.indx1
2> go
```

h) Determine the number of IO's needed to select from the table

```
1> select count(*) from authors
2> go

 -----------
       5000
Table: authors  scan count 1,logical reads: 221,physical reads: 220
Total writes for this command: 0

(1 row affected)
1>
```

i) Write these in the *second* row of table 2 on the *Lab Tables Worksheet* page

Logical and Physical I/O With Different Fill Factors

	Logical Reads	Physical Reads
Fill factor = 30	653	652
Fill factor = default	221	220

j) What effect, if any, did fill factor have on logical and physical reads?

A higher fill factor reduces i/o dramatically. The data and indexes are on fewer pages, so the server has fewer pages to read.

Lab 4c – Where do Updated Rows Go?

(Student Guide, page 4-58)

Exercise Overview

Goals

- Perform updates and determine where the updates occurred

General Tasks

- Determine update mode (Scenario #1)
  - Describe the physical order of the data in *update_table* briefly
  - Then update *update_table* colB with a new value
  - What update mode was chosen?
- Determine update mode (Scenario #2)
  - Update colC with a new value
  - What update mode was chosen? Explain
- Determine update mode (Scenario #3)
  - Create an update trigger in your table
  - Update colC with a new value
  - What update mode was chosen?

Lab Setup

- Use your *pubtune* database
- Use the T-SQL manual for help in creating triggers

continued on next page...

Detailed Instructions

In this lab, you perform updates and determine where the updates occurred.

1. Determine update mode (Scenario #1)

   a. Examine the table *update_table* using sp_help

   b. Display the procedure *showroworder* using sp_helptext

   c. Display the current physical order of rows in table *update_table* using the stored procedure *showroworder* Result 12

   d. Describe the physical order of the data in *update_table* briefly:

   e. Turn showplan on

   f. Then update *update_table colB* with a new value, your user number, for a single row:

```
update update_table
   set colB = "user11"
   where colA = 11
go
```

   g. What update mode was chosen? deferred

   h. Turn showplan off

   i. Use *showroworder* to display the current physical order of rows in the table

   j. Has it changed from its original order? Explain.

2. Determine update mode (Scenario #2)

   a) Turn showplan on

b) Set statistics io on.

c) Update *colC* with a new value, your user number, for that same row:

```
update update_table
   set colC = "user10"
   where colA = 10
go
```

d) What update mode was chosen? Explain direct

e) Record the i/o for the update. scan count 1, logical reads 2

f) Turn showplan and statistics io off

g) Use *showroworder* to display the current physical order of rows in the table.

h) Has it changed? Explain.

3. Determine update mode (Scenario #3)

a) Create an update trigger in your table

```
create trigger upd_trigger
on update_table
for update
as
print "update trigger fired"
go
```

b) Turn showplan and statistics io on

c) Update colC with a new value:

```
update update_table
   set colC = "user7"
   where colA = 7
go
```

d) What update mode was chosen? direct

e) Record the i/o for the update

f) Compare with 2, explain:

Solutions

1. *Determine update mode (Scenario #1)*

   a. Examine the table *update_table* using sp_help

```
1> sp_help update_table
2> go
 Name                           Owner
         Type
 ------------------------------ ------------------------------
        ----------------------
 update_table                   dbo
         user table

 Data_located_on_segment        When_created
 ------------------------------ --------------------------
 default                               May 25 1994  1:38PM

 Column_name     Type            Length Prec Scale Nulls
Default_name
         Rule_name       Identity
 --------------- --------------- ------ ---- ----- -----
---------------
 colA            int          4   NULL  NULL 0   NULL  NULL      0
 colB            varchar  200   NULL  NULL 0   NULL  NULL      0
 colC            char     200   NULL  NULL 0   NULL  NULL      0

 index_name              index_description
         index_keys


 --------------------
-------------------------------------------------------
 upd_idx        nonclustered located on default     colA


No defined keys for this object.

(1 row affected, return status = 0)
1>
```

   b. Display the procedure *showroworder* using sp_helptext

```
1> sp_helptext showroworder
2> go
 # Lines of Text
 ---------------
               1
```

```
(1 row affected)

         text

-------------------------------------------------------------------

create procedure showroworder
as
select colA, colB=substring(colB,1,20), colC=substring(colC,1,15)
from update_table


(1 row affected, return status = 0)
1>
```

c. Display the current physical order of rows in table *update_table* using the stored procedure *showroworder*

```
1> showroworder
2> go
 colA        colB                 colC
 ----------- -------------------- ---------------
           1 not updated yet      row 1
           2 not updated yet      row 2
           3 not updated yet      row 3
           4 not updated yet      row 4
           5 not updated yet      row 5
           6 not updated yet      row 6
           7 not updated yet      row 7
           8 not updated yet      row 8
           9 not updated yet      row 9
          10 not updated yet      row 10
          11 not updated yet      row 11
          12 not updated yet      row 12
          13 not updated yet      row 13
          14 not updated yet      row 14
          15 not updated yet      row 15

(15 rows affected, return status = 0)
1>
```

d. Describe the physical order of the data in *update_table* briefly:

The data is physically ordered in ascending order (1 to 15)

e. Turn showplan on

```
1> set showplan on
2> go
1>
```

f. Update *update_table colB* with a new value, your user number, for a single row

```
1> update update_table
2>              set colB = "user11"
3>              where colA = 11
4> go
STEP 1
The type of query is UPDATE.
The update mode is deferred.
FROM TABLE
update_table
Nested iteration
Table Scan
TO TABLE
update_table
(1 row affected)
1>
```

g. What update mode was chosen?

Deferred

h. Turn showplan off

```
1> set showplan off
2> go
STEP 1
The type of query is SET OPTION OFF.
1>
```

i. Use *showroworder* to display the current physical order of rows in the table

```
1> showroworder
2> go
 colA        colb                 colc
 ----------- -------------------- ---------------
           1 not updated yet      row 1
           2 not updated yet      row 2
           3 not updated yet      row 3
           4 not updated yet      row 4
           5 not updated yet      row 5
           6 not updated yet      row 6
           7 not updated yet      row 7
           8 not updated yet      row 8
```

```
          9 not updated yet      row 9
         10 not updated yet      row 10
         12 not updated yet      row 12
         13 not updated yet      row 13
         14 not updated yet      row 14
         15 not updated yet      row 15
         11 user11               row 11

(15 rows affected, return status = 0)
1>
```

j. Has it changed from its original order? Explain.

Yes, 11 has moved to the end of the table. A variable length column was updated causing a deffered update operation. (A *delete* followed by an entry to the *log* and *Insert* back into the table)

2. *Determine update mode (Scenario #2)*

a. Turn showplan on

```
1> set showplan on
2> go
1>
```

b. Set statistics io on.

```
1> set statistics io on
2> go
STEP 1
The type of query is SET STATUS ON.
Total writes for this command: 0
1>
```

c. Update *colC* with a new value, your user number, for that same row:

```
1> update update_table
2> set colC = "user10"
3> where colA = 10
4> go
STEP 1
The type of query is UPDATE.
The update mode is direct.
FROM TABLE
update_table
Nested iteration
```

```
Table Scan
TO TABLE
update_table
Table: update_table  scan count 0,  logical reads: 0,  physical
reads: 0
Table: update_table  scan count 1,  logical reads: 2,  physical
reads: 0
Total writes for this command: 1
(1 row affected)
1>
```

d. What update mode was chosen? Explain

Direct. The update qualifies for an update in place: one row, not key change, no length change, no update trigger present.

e. Record the i/o for the update.

```
Total writes for this command: 1
```

f. Turn showplan and statistics io off

```
1> set showplan off
2> go
STEP 1
The type of query is SET OPTION OFF.
Total writes for this command: 0
1> set statistics io off
2> go
1>
```

g. Use *showroworder* to display the current physical order of rows in the table

```
1> showroworder
2> go
 colA        colb                 colc
 ----------- -------------------- ---------------
           1 not updated yet      row 1
           2 not updated yet      row 2
           3 not updated yet      row 3
           4 not updated yet      row 4
           5 not updated yet      row 5
           6 not updated yet      row 6
           7 not updated yet      row 7
           8 not updated yet      row 8
           9 not updated yet      row 9
          10 not updated yet      user10
          12 not updated yet      row 12
          13 not updated yet      row 13
          14 not updated yet      row 14
```

```
             15 not updated yet      row 15
             11 user11               row 11

    (15 rows affected, return status = 0)
    1>
```

h. Has it changed? Explain.

No, the physical order of the data has stayed the same because an update in place occurred.

3. *Determine update mode (Scenario #3)*

i. Create an update trigger in your table

```
1> create trigger upd_trigger
2> on update_table
3> for update
4> as
5> print "update trigger fired"
6> go
1>
```

j. Turn showplan and statistics io on

```
1> set showplan on
2> go
1> set statistics io on
2> go
STEP 1
The type of query is SET STATUS ON.
Total writes for this command: 0
1>
```

k. Update colC with a new value

```
1> update update_table
2> set colC = "user7"
3> where colA = 7
4> go
STEP 1
The type of query is UPDATE.
The update mode is direct.
FROM TABLE
update_table
Nested iteration
Table Scan
TO TABLE
update_table
STEP 1
```

```
The type of query is PRINT.
update trigger fired
Total writes for this command: 0
Total writes for this command: 2
(1 row affected)
1>
```

l. What update mode was chosen?

we just did an update not in place.

m. Record the i/o for the update

Total writes for this command: 2

n. Use *showroworder* to display the current physical order of rows in the table

```
colA        colb                 colc
----------- -------------------- ---------------
          1 not updated yet      row 1
          2 not updated yet      row 2
          3 not updated yet      row 3
          4 not updated yet      row 4
          5 not updated yet      row 5
          6 not updated yet      row 6
          7 not updated yet      user7
          8 not updated yet      row 8
          9 not updated yet      row 9
         10 not updated yet      row 10
         11 not updated yet      row 11
         12 not updated yet      row 12
         13 not updated yet      row 13
         14 not updated yet      row 14
         15 not updated yet      row 15
```

o. Compare with 2, explain:

Direct update.

Total writes = 2

Conclusion: update "not in place"

5

Lab 5 – Indexes and Performance

(Student Guide, page 5-26)

Exercise Overview

Goals

- Compare the performance of various queries for different types of indexes
- Collect statistics on elapsed time, logical I/O, and physical I/O

General Tasks

- Compare the performance of queries using various indexes
- Define and use indexes to run queries most efficiently
- Use clustered and non-clustered indexes

Lab Setup

- Use your *pubtune* database

Exercise Instructions...

Detailed Instructions

In this lab, you will compare the performance of various queries for different types of indexes.

1. For the following queries, use both *titles_pridtitl* and *titles_idpr* tables where *table_name* has been specified. The first has a clustered index on *price*, the second a non-clustered index on *price.* Collect statistics on elapsed time, logical i/o, and physical i/o for both sets, and fill in the supplied tables. **Execute each query twice** and record both sets of statistics. (In the first case, the pages are not likely to be in cache; in the second, they are.)

Query A

```
SELECT title
FROM table_name
WHERE price BETWEEN 200 AND 300
```

Query	Index	Response Time		Logical I/O		Physical I/O	
Query A	Clustered	396ms	6ms	18	18	17	Ø
	Non-cl	466ms	6ms	56	56	54	Ø

Query B

```
SELECT title
FROM table_name
WHERE price BETWEEN 20 AND 30
```

Query	Index	Response Time		Logical I/O		Physical I/O	
Query B	Clustered	313	53	85	85	84	Ø
	Non-cl	736	100	621	621	568	Ø

Query C

```
SELECT price
FROM table_name
WHERE price BETWEEN 200 and 300
```

Query	Index	Response Time		Logical I/O		Physical I/O	
Query C	Clustered	6	1Ø	18	18	Ø	Ø
	Non-cl	3	6	2	2	1	Ø

2. Assume a table called employee with the following columns:

   number int
   name char(20)
   salary money
   dept char(10)
   commission money

   What indexes could be used to "cover" each the following queries most efficiently? (Ignore update considerations.)

```
a. select dept, sum(salary) from employee
b. select name, salary+commission from employee where salary > 2000
c. select number, dept from employee where number between 10 and 20
d. select dept, avg(salary) from employee
   where dept in ('research', 'accounting')
   group by dept
```

a. op dept + salary
b.

Solutions

1. *For the following queries, use both titles_pridtitl and titles_idpr tables where table_name has been specified. The first has a clustered index on price, the second a non-clustered index on price.*
   Collect statistics on elapsed time, logical i/o, and physical i/o for both sets, and fill in the table below. ***Execute each query twice*** *and record both sets of statistics. (In the first case, the pages are not likely to be in cache; in the second, they are.)*

Query A

```
1> set statistics io on
2> go
Total writes for this command: 0
1> set statistics time on
2> go
Total writes for this command: 0
1> SELECT count(title)
2> FROM titles_pridtitl
3> WHERE price BETWEEN 200 AND 300
4> go
Parse and Compile Time 0.
SQL Server cpu time: 0 ms.

 -----------
          54
Table: titles_pridtitl  scan count 1,  logical reads: 18,  physical
reads: 18
Total writes for this command: 0

Execution Time 0.
SQL Server cpu time: 0 ms.  SQL Server elapsed time: 203 ms.

(1 row affected)
1>

1> SELECT count(title)
2> FROM titles_pridtitl
3> WHERE price BETWEEN 200 AND 300
4> go
Parse and Compile Time 0.
SQL Server cpu time: 0 ms.

 -----------
          54
Table: titles_pridtitl  scan count 1,  logical reads: 18,  physical
reads: 0
Total writes for this command: 0
```

```
Execution Time 0.
SQL Server cpu time: 0 ms.  SQL Server elapsed time: 13 ms.

(1 row affected)
1>
```

Query	Index	Times		Logical I/O		Physical I/O	
Query A	Clustered	203	13	18	18	17	0
	Non-cl	1936	10	56	56	54	0

Query B

```
SELECT title
FROM table_name
WHERE price BETWEEN 20 AND 30
```

Query	Index	Times		Logical I/O		Physical I/O	
Query B	Clustered	1106	1026	85	85	84	0
	Non-cl	1510	826	608	608	377	0

Query C

```
SELECT price
FROM table_name
WHERE price BETWEEN 200 AND 300
```

Query	Index	Times		Logical I/O		Physical I/O	
Query C	Clustered	3	3	18	18	0	0
	Non-cl	3	3	2	2	0	0

Note that in this as in all measurements of I/O and response time, your values may be quite different from those we recorded during our test run.

Please comment on any differences.

In general, the use of a clustered index on price for the range queries reduced I/O. Query C was covered.

2. *Assume a table called employee with the following columns:*

   number int
   name char(20)
   salary money
   dept char(10)
   commission money

   What indexes could be used to "cover" each the following queries most efficiently? (Ignore update considerations.)

```
a. select dept, sum(salary) from employee
```

Non-clustered on dept, salary.

```
b. select name, salary+commission from employee where salary > 2000
```

Non-clustered on salary, name, commission.

```
c. select number, dept from employee where number between 10 and 20
```

Non-clustered on number, dept.

```
d. select dept, avg(salary) from employee
   where dept in ('research', 'accounting')
   group by dept
```

Non clustered on dept, salary.

Lab 6a – Search Arguments

(Student Guide, page 6-16)

Exercise Overview

Goals

- Identify SARGs and determine if the SARG is supported by an index using the different variations on the titles table

General Tasks

- Describe and construct table query plans
- Choose query options using table references
- Compare run results of various queries by using *showplan*

Lab Setup

Consult the Lab Worksheet on the following pages as a reference to the table names and their indexes

Exercise Instructions...

Lab Worksheet

Variations on *authors*

#	Table Name	Index(es)/Notes
1	authors	• No index • Original table
2	authors_id	• Unique clustered index on **au_id** **(idxl)** • Used to illustrate primary key approach. • Along with **authors_idstate**, used to contrast JOINS with au_id being clustered or non-clustered.
3	authors_idstate	• Unique non-clustered index on **au_id** **(idxl)** • Non-clustered index on **state** **(idx2)** • Used in SARG work. • Along with **authors_id**, used to contrast JOINS with **au_id** being clustered or non-clustered.
4	authors_idnames	• Unique non-clustered index on **au_id**, **au_lname**, **au_fname** **(idxl)** • Used to illustrate index covering.
5	authors_idid	• Unique clustered index on **au_id** **(idxl)** • Non-clustered index on **au_id**, **au_lname** **(idx2)** • Used to illustrate index covering.

Variations on *titleauthors*

#	Table Name	Index(es)
1	titleauthor	• No index
2	titleauthor_idid	• Unique clustered index on **au_id**, **title_id** **(idxl)**
3	titleauthor_ididtid	• Unique clustered index on **au_id**, **title_id** **(idxl)** • Non-clustered index on **title_id** **(idx2)** • When compared with **titleauthor_idid**, used to illustrate join to 2nd column of two-column index.

Variations on *titles*

#	Table Name	Index(es)
1	titles	• No index
2	titles_idpr	• Unique clustered index on **title_id** **(idxl)** • Non-clustered index on **price** **(idx2)**
3	titles_titlid	• Clustered index on **title** **(idxl)** • Non-clustered index on **title_id** **(idx2)** • Non-clustered index on **pub_id** **(idx3)**
4	titles_pridtitl	• Clustered on **price** **(idxl)** • Unique non-clustered on **title_id** **(idx2)** • Non-clustered on **title** **(idx3)**

Detailed Instructions

In this lab you will analyze queries to identify SARGs and determine if a SARG is supported by an index. You will be using variations on the titles table, so consult the table descriptions at the beginning of this book or in the preceding two pages, if you need to.

1. Set both showplan and noexec on, and display the query plans for each of the following queries. Circle the options in tables that describe how query is processed.

   a) **Query A**

```
SELECT title, price
FROM   titles_idpr
WHERE  price = 10
```

Access Method	Index	Why?
Table Scan	Clustered	[✓ SARG]
[✓ Index]	[✓ Non Clustered]	No SARG

   b) **Query B**

```
SELECT title, price
FROM   titles_idpr
WHERE  price + 0 = 1
```

Access Method	Index	Why?
[✓ Table Scan]	Clustered	SARG
Index	Non Clustered	[✓ No SARG]

   c) Describe the plans, and explain any differences between processing in (a) and (b).

   d) **Query C**

```
SELECT title, price
FROM   titles_titlid
WHERE  title like 'B%'
```

Access Method	Index	Why?
Table Scan	[✓ Clustered]	[✓ SARG]
[✓ Index]	Non Clustered	No SARG

e) **Query D**

```
SELECT title, price
FROM   titles_titlid
WHERE  substring(title,1,1) = 'B'
```

Access Method	Index	Why?
[✓ Table Scan]	Clustered	SARG
Index	Non Clustered	[✓ No SARG]

f) **Query E** (note that we select *title* only, not *title* and *price*)

```
SELECT title
FROM   titles_titlid
WHERE  title like '%B'
```

Access Method	Index	Why?
[✓ Table Scan]	Clustered	SARG
Index	Non Clustered	[✓ No SARG]

(% vooraf)

Optional Exercises

2. Answer the following questions regarding the results from the previous exercise:
   a. Why does the optimizer not use an index for Query D?

   b. Why does the optimizer not use an index for Query E?

3. Display the query plan for the following query, and explain.

Query F

```
SELECT title, price
FROM   titles_pridtitl
WHERE  price > 30
AND    title like 'As%'
```

Access Method	Index	Why?
Table Scan	Clustered	SARG
Index	Non Clustered	No SARG

Solutions

1. Set both showplan and noexec on, and display the query plans for each of the following queries. (Describe the plans briefly--just whether a table scan or index is used, and if an index, which one.)

a) Set noexec on and execute **Query A**

```
1> set noexec on
2> go
1> SELECT title, price
2> FROM   titles_idpr
3> WHERE  price = 10
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_idpr
Nested iteration
Index : idx2
```

Access Method	Index	Why?
Index	Non Clustered	SARG

b) Execute **Query B**

```
1> SELECT title, price
2> FROM   titles_idpr
3> WHERE  price + 0 = 1
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_idpr
Nested iteration
Table Scan
```

Access Method	Index	Why?
Table Scan	None	No SARG

c) Describe the plans, and explain any differences.

Query A uses index idx2 (non-clustered index on price) and Query B uses a table scan. "price = 10" qualifies as a SARG, but "price + 0 = 1" does not.

d) Execute **Query C**

```
1> SELECT title, price
2> FROM   titles_titlid
3> WHERE  title like 'B%'
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_titlid
Nested iteration
Using Clustered Index
```

Access Method	Index	Why?
Index	Clustered	SARG

e) Execute **Query D**

```
1> SELECT title, price
2> FROM   titles_titlid
3> WHERE  substring(title,1,1) = 'B'
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_titlid
Nested iteration
Table Scan
```

Access Method	Index	Why?
Table Scan	None	No SARG

f) Execute **Query E** (note that we select *title* only, not *title* and *price*)

```
1> SELECT title
2> FROM   titles_titlid
3> WHERE  title like '%B'
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_titlid
Nested iteration
Table Scan
```

Access Method	Index	Why?
Table Scan	None	No SARG

Optional Exercises: Solutions

2. *Answer the following questions regarding the results from the previous exercise:*

   a) *Why does the optimizer not use an index for Query D?*

   The *where* clause contains the *substring* function. Functions cannot be used in SARGs.

   b) *Why does the optimizer not use an index for Query E?*

   If there is no *leading* value, an index is generally useless, as there is no starting point for a range search.

3. *Display the query plan for the following query, and explain.*

Query F

```
1> SELECT title, price
2> FROM   titles_pridtitl
3> WHERE  price > 30
4> AND    title like 'As%'
5> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_pridtitl
Nested iteration
Index : idx3
```

Access Method	Index	Why?
Index	Non-clustered	SARG

Although *price>30* is a SARG for the clustered index on *price,* the non-clustered index on *title (idx3)* has better selectivity.

Lab 6b – OR Strategies

(Student Guide, page 6-21)

Exercise Overview

Goals

- Examine and explain the query plans of certain OR queries.

General Tasks

- Display query plans by executing each query with the *showplan* option
- Explain the chosen query plan for each query

Lab Setup

Consult the Lab Worksheet as a reference to the table names and their indexes

Exercise Instructions...

Detailed Instructions

In this lab, you will examine and explain the query plans of certain OR queries.

1. Set showplan and noexec on and execute the following queries. Describe their query plans briefly.

   a) **Query A**

```
SELECT title
FROM   titles_titlid
WHERE  title_id = 'T81002'
OR     title = 'Assigned Numbers'
```

Access Method	Index	Why?
Table Scan	None	[✓ SARG]
[✓ Index]	Clustered & Clustered	No SARG
	[✓ Clustered & Non Clustered]	[✓ Covered]
	Non Clustered & Non Clustered	[✓ Or Clause]
	Dynamic Index	> Range

   b) **Query B**

```
SELECT title
FROM   titles_titlid
WHERE  title_id = 'T81002'
OR     title_id = 'T69002'
```

Access Method	Index	Why?
Table Scan [✓ Index]	None	[✓ SARG]
	Clustered & Clustered	No SARG
	Clustered & Non Clustered	[✓ Covered]
	[✓ Non Clustered & Non Clustered]	[✓ Or Clause]
	Dynamic Index	> Range

c) **Query C**

```
SELECT title
FROM   titles_idpr
WHERE  title_id in ('T81002', 'T69002')
```

Access Method	Index	Why?
Table Scan [✓ Index]	None	[✓ SARG]
	[✓ Clustered & Clustered]	No SARG
	Clustered & Non Clustered	[✓ Covered]
	Non Clustered & Non Clustered	[✓ Or Clause]
	Dynamic Index	> Range

d) **Query D**

```
SELECT title
FROM   titles_idpr
WHERE  title_id = 'T81002'
OR price > 5.00
```

Access Method	Index	Why?
[✓ Table Scan]	[✓ None]	SARG
Index	Clustered & Clustered	No SARG
	Clustered & Non Clustered	Covered
	Non Clustered & Non Clustered	Or Clause
	Dynamic Index	[✓ > Range]

Solutions

1. *Set showplan and noexec on and execute the following queries. Describe their query plans briefly.*

   a) **Query A**

```
1> SELECT title
2> FROM   titles_titlid
3> WHERE  title_id = 'T81002'
4> OR     title = 'Assigned Numbers'
5> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_titlid
Nested iteration
Index : idx2
FROM TABLE
titles_titlid
Nested iteration
Using Clustered Index
FROM TABLE
titles_titlid
Nested iteration
Using Dynamic Index
```

Access Method	Index	Why?
Index	Clustered & Non Clustered Dynamic Index	SARG & Or Clause

   b) **Query B**

```
1> SELECT title
2> FROM   titles_titlid
3> WHERE  title_id = 'T81002'
4> OR     title_id = 'T69002'
5> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_titlid
Nested iteration
Index : idx2
FROM TABLE
titles_titlid
Nested iteration
Index : idx2
```

```
FROM TABLE
titles_titlid
Nested iteration
Using Dynamic Index
```

Access Method	Index	Why?
Index	Non Clustered & Non Clustered Dynamic Index	SARG & Or Clause

c) **Query C**

```
1> SELECT title
2> FROM   titles_idpr
3> WHERE  title_id in ('T81002', 'T69002')
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
FROM TABLE
titles_idpr
Nested iteration
Using Dynamic Index
```

Access Method	Index	Why?
Index	Clustered & Clustered Dynamic Index	SARG & Or Clause

d) **Query D**

```
1> SELECT title
2> FROM   titles_idpr
3> WHERE  title_id = 'T81002'
4> OR price > 5.00
5> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_idpr
Nested iteration
Table Scan
```

Access Method	Index	Why?
Table Scan	None	>Range

(Table scan. The optimizer will choose a table scan for this query because the resulting cost of creating a dynamic index is greater than scanning the entire table once.)

Lab 6c – Index Covering and Aggregates

(Student Guide, page 6-27)

Exercise Overview

Goals

- Gain understanding of choosing correct indexes for queries

General Tasks

- Experiment and execute several queries using a variety of *titles* and *authors* tables using clustered and non-clustered composite indexes
- Determine which indexes were used and explain
- Explain the optimizer's decision to use unique non-clustered indexes

Lab Setup

Consult the Lab Worksheet as a reference to the table names and their indexes

Exercise Instructions...

Detailed Instructions

In this lab, you will execute several queries and explain why certain indexes are chosen. You will be using variations on the titles and authors tables, so consult the table descriptions at the beginning of this book if you need to.

1. Execute the following queries against the *authors_idnames* table, which has a non-clustered composite index on *au_id*, *au_lname*, and *au_fname*. Determine whether each query uses the index, and explain.

   a) **Query A**

   ```
   SELECT *
   FROM   authors_idnames
   WHERE  au_id = 'A40650655'
   ```

   Does it use the composite index? Why or why not?

   Ja : covers au_id (1e veld)

   b) **Query B**

   ```
   SELECT *
   FROM   authors_idnames
   WHERE  au_lname = "Carton"
   ```

   Does it use the composite index? Why or why not?

   Nee (niet 1e veld)

   c) **Query C**

   ```
   SELECT *
   FROM   authors_idnames
   WHERE  au_id = 'A40650655'
   AND    au_fname = 'Nita'
   ```

   Does it use the composite index? Why or why not?

   Ja (bevat 1e veld) (plus 2e natuurlijk)

2. Execute the following queries against the *titles_idpr* table, which has a clustered index on *title_id*. Determine whether each query uses the index, and explain.

   a) **Query D**

```
SELECT count(title)
FROM   titles_idpr
WHERE  title_id between 'T10' and 'T99584'
```

   Does it use the clustered index? Why or why not? Om covered Range te selecteren

Access Method	Index	Why?
Table Scan	Clustered	SARG
Index	Non Clustered	No SARG
		Covered

table scan op scalar aggregate

   b) **Query E**

```
SELECT count(title)
FROM   titles_idpr
WHERE  title_id between 'T10' and 'T11'
```

   Does it use the clustered index? Why or why not? Idem

Access Method	Index	Why?
Table Scan	Clustered	SARG
Index	Non Clustered	No SARG
		Covered

3. Execute queries D and E against the *titles_pridtitl* table, which has a unique non-clustered index. Explain the optimizer's decision.

   a) Run query D against the *titles_pridtitl* table
   b) Determine the number of IO's required for the query.
   c) Do the number of IO's confirm the plan from step (a)?
   d) Run query E against the *titles_pridtitl* table
   e) Determine the number of IO's required for the query.
   f) Do the number of IO's confirm the plan from step (d)?

4. Execute the following query against the *titles_pridtitl* table. Explain the optimizer's decision.

Query F (same comment as above)

```
SELECT title
FROM   titles_pridtitl
WHERE  title like '%B'
```

Solutions

1. *Execute the following queries against the authors_idnames table, which has a non-clustered composite index on au_id, au_lname, and au_fname. Determine whether each query uses the index, and explain.*

   a) **Query A**

```
1> SELECT *
2> FROM   authors_idnames
3> WHERE  au_id = 'A40650655'
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
authors_idnames
Nested iteration
Index : idx1
```

Does it use the composite index? Why or why not?

Yes, the leading column of the composite index is *au_id*, so this query can use that index.

   b) **Query B**

```
1> SELECT *
2> FROM   authors_idnames
3> WHERE  au_lname = "Carton"
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
authors_idnames
Nested iteration
Table Scan
```

Does it use the composite index? Why or why not?

The composite index cannot be used because *au_lname* is not the leading column.

   c) **Query C**

```
1> SELECT *
2> FROM   authors_idnames
3> WHERE  au_id = 'A40650655'
4> AND    au_fname = 'Nita'
5> go
STEP 1
The type of query is SELECT.
FROM TABLE
```

```
authors_idnames
Nested iteration
Index : idx1
```

Does it use the composite index? Why or why not?

Yes, because the leading column is supplied.

2. *Execute the following queries against the titles_idpr table, which has a clustered index on title_id. Determine whether each query uses the index, and explain.*

a) **Query D**

```
1> SELECT count(title)
2> FROM   titles_idpr
3> WHERE  title_id between 'T10' and 'T99584'
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
STEP 2
The type of query is SELECT.
Table Scan
```

Does it use the clustered index? Why or why not?

Yes, it is the starting point for a scan.

b) **Query E**

```
1> SELECT count(title)
2> FROM   titles_idpr
3> WHERE  title_id between 'T10' and 'T11'
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
STEP 2
The type of query is SELECT.
Table Scan
```

Does it use the clustered index? Why or why not?

Yes, as above.

3. *Execute queries D and E against the titles_pridtitl table, which has a unique non-clustered index. Explain the optimizer's decisions.*

   a. Run query D against the *titles_pridtitl* table

```
1> SELECT count(title)
2> FROM   titles_pridtitl
3> WHERE  title_id between 'T10' and 'T99584'
4> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles_pridtitl
Nested iteration
Using Clustered Index
STEP 2
The type of query is SELECT.
Table Scan
```

   (The optimizer appears to choose the clustered index on *price.)*

   b. Determine the number of IO's required for the query

```
1> SELECT count(title)
2> FROM   titles_pridtitl
3> WHERE  title_id between 'T10' and 'T99584'
4> go

 -----------
        5000
Table: titles_pridtitl scans 1,logical reads: 652,phys reads: 634
Total writes for this command: 2
```

   c. Conclusion for query D

   Although showplan indicates that an index was used (clustered index on price), statistics io reports 652 total io's for the query. 652 io's indicates that a Table Scan took place as this number is the total data pages for the table (652)

   d. Run query E against the *titles_pridtitl* table

```
1> SELECT count(title)
2> FROM   titles_pridtitl
3> WHERE  title_id between 'T10' and 'T11'
4> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles_pridtitl
```

```
Nested iteration
Index : idx2
STEP 2
The type of query is SELECT.
Table Scan
```

e. Determine the number of IO's required for the query

```
1> set statistics io on
2> go
1> set noexec off
2> go
STEP 1
The type of query is SET OPTION OFF.
1> set showplan off
2> go
STEP 1
The type of query is SET OPTION OFF.

Total writes for this command: 0
1> SELECT count(title)
2> FROM   titles_pridtitl
3> WHERE  title_id between 'T10' and 'T11'
4> go

 -----------
          58
Table: titles_pridtitl  scans 1,logical reads: 60,phys reads: 0
Total writes for this command: 0
```

f. Conclusion query E

Showplan on query H indicates that the optimizer has decided to use the non-clustered index. As we run the query with statistics io on we notice that the io's used to satisfy the query is well below the total pages for the table (resp. 60 and 652). We can conclude that the non-clustered index has indeed been used

Overall conclusion

Although there was a relevant index available on query D, the optimizer choose to scan the table instead. When making page estimates based on row estimates, the optimizer always assumes the worst case of one data page to be read for each qualifying row in the non-clustered index. This sometimes results in a higher cost to use the non-clustered index (in terms of total index pages and data pages to be read) than to perform a table scan.

If the search range is reduced (between 'T10' and 'T11') the optimizer finds that using the index is useful.

4. *Execute the following query against the titles_pridtitl table. Explain the optimizer's decision.*

Query F (same comment as above)

```
1> SELECT title
2> FROM   titles_pridtitl
3> WHERE  title like '%B'
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_pridtitl
Nested iteration
Index : idx3
```

Access Method	Index	Why?
Index	Non Clustered	Covered

In Lab 6a, Query E is similar but against a table that has a clustered index instead of the non-clustered index there is here. The clustered index could not be used because the leading wild-card does not provide a starting value for a range search.

Here in Query F there is a non-clustered index. The optimizer is actually scanning the non-clustered index--the entire leaf level--to see if there are any titles with an upper case B in them.

Lab 6d – Predicting Index Usage

(Student Guide, page 6-34)

Exercise Overview

Goals

- Identify different optimizer decisions when using aggregates.
- Compare the I/O required to execute queries that are "covered" by the index vs. queries that are not.

General Tasks

- Predict which index, if any, will be used for each aggregate lab query
- Note how clustered indexes and non-clustered indexes are used by the aggregate queries
- Use the option, *set statistics io* to determine the savings on I/O of queries that are covered versus those that are not.

Lab Setup

Consult the Lab Worksheet as a reference to the table names and their indexes

Exercise Instructions...

Detailed Instructions

In this lab, you compare I/O required to execute queries that are "covered" by the index vs. queries that are not. Also, you will identify different optimizer decisions when using aggregates.

1. For each of the following queries, predict which index, if any, will be used. (set showplan and noexec on). Note that the *authors_idid* table has a clustered index on *au_id* and a non-clustered index on *au_id, au_lname.*

Query A

```
SELECT count(*)
FROM   authors_idid
```

Access Method	Index	Why?
Table Scan [✓ Index]	None ~~Clustered~~ [✓ Non Clustered] Clustered & Non Clustered	SARG No SARG [✓ Covered] > Range

Query B

```
SELECT count(*)
FROM   authors_idid
WHERE  au_id > 'A1'
```

Access Method	Index	Why?
Table Scan [✓ Index]	None Clustered [✓ Non Clustered] Clustered & Non Clustered	SARG No SARG [✓ Covered] ~~> Range~~

Query C

```
SELECT count(phone)
FROM   authors_idid
WHERE  au_id > 'A1'
```

Access Method	Index	Why?
Table Scan	None	SARG
[✓ Index]	[✓ Clustered]	No SARG
	Non Clustered	Covered
	Clustered & Non Clustered	> Range

Query D

```
SELECT au_fname
FROM   authors_idid
WHERE  au_id > 'A1'
```

Access Method	Index	Why?
Table Scan	None	[✓ SARG]
[✓ Index]	[✓ Clustered]	No SARG
	Non Clustered	Covered
	Clustered & Non Clustered	> Range

2. What index could you create to cover the previous query? Try the query against *authors_idnames*. Check what index(es) are on that table. Which index is used?

3. What is the savings in I/O when the query is covered versus when it is not? To determine this, set statistics io on, set no exec off, and execute the following queries:

Query E

```
SELECT count(au_fname)
FROM   authors_idid
WHERE  au_id > 'A'
```

22[illegible]3 Reads

Query F

```
SELECT count(au_fname)
FROM   authors_idnames
WHERE  au_id > 'A'
```

93 reads

Covered vs. Not Covered

	Logical Reads	Physical Reads
Query E		
Query F		

Solutions

1. *For each of the following queries, predict which index, if any, will be used. Then run the query (set showplan and noexec on) to confirm your hypothesis. (Note that the authors_idid table has a clustered index on au_id and a non-clustered index on au_id, au_lname.)*

Query A

```
1> SELECT count(*)
2> FROM   authors_idid
3> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
authors_idid
Nested iteration
Index : idx2
STEP 2
The type of query is SELECT.
Table Scan

 -----------
        5000

(1 row affected)
1>
```

Access Method	Index	Why?
Index	Non Clustered	Covered

It uses the non-clustered index on au_id and scans the index leaf pages. Even without a where clause, the optimizer will use any available non-clustered index to cover a "select count(*)" query.

Query B

```
1> SELECT count(*)
2> FROM   authors_idid
3> WHERE  au_id > 'A1'
4> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
authors_idid
```

```
Nested iteration
Index : idx2
STEP 2
The type of query is SELECT.
Table Scan

 -----------
        5000

(1 row affected)
1>
```

Access Method	Index	Why?
Index	Non Clustered	Covered

It uses the non-clustered index on au_id and scans the index leaf pages.

Query C

```
1> SELECT count(phone)
2> FROM   authors_idid
3> WHERE  au_id > 'A1'
4> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
authors_idid
Nested iteration
Using Clustered Index
STEP 2
The type of query is SELECT.
Table Scan

 -----------
        5000

(1 row affected)
1>
```

Access Method	Index	Why?
Index	Clustered	SARG Column phone is not covered by the Non Clustered index

It uses the clustered index to find the first candidate row and then scans the data pages.

Query D

```
1> SELECT au_fname
2> FROM   authors_idid
3> WHERE  au_id > 'A1'
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
authors_idid
Nested iteration
Using Clustered Index
```

Access Method	Index	Why?
Index	Clustered	SARG Column au_fname is not covered by the Non Clustered index

It uses the clustered index to find the first candidate row and then scans the data pages.

2. *What index could you create to cover the previous query? Try the query against authors_idnames. Check what index(es) are on that table. Which index is used?*

   An index on *au_id*, *au_lname*, and *au_fname* would help, and *authors_idnames* has an index just like that! It gets used for the query.

```
1> SELECT au_fname
2> FROM   authors_idnames
3> WHERE  au_id > 'A1'
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
authors_idnames
Nested iteration
Index : idx1
```

3. *What is the savings in I/O when the query is covered versus when it is not? To determine this, set statistics io on, set no exec off, and execute the following queries:*

Query E

```
1> SELECT count(au_fname)
2> FROM   authors_idid
3> WHERE  au_id > 'A'
```

```
4> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
authors_idid
Nested iteration
Using Clustered Index
STEP 2
The type of query is SELECT.
Table Scan

 -----------
       5000
Table: authors_idid  scan count 1,  logical reads: 223,  physical
reads: 0
Total writes for this command: 0
```

Query F

```
1> SELECT count(au_fname)
2> FROM   authors_idnames
3> WHERE  au_id > 'A'
4> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
authors_idnames
Nested iteration
Index : idx1
STEP 2
The type of query is SELECT.
Table Scan

 -----------
       5000
Table: authors_idnames  scan count 1,  logical reads: 93,  physical
reads: 0
Total writes for this command: 0

(1 row affected)
```

Covered vs. Not Covered

	Logical Reads	Physical Reads
Query E	223	0
Query F	93	0

Lab 6e – Join Strategies

(Student Guide, page 6-57)

Exercise Overview

Goals

- Determine the join strategy the optimizer will choose for a given query
- Evaluate the effect of search arguments and variety of possible indexes

General Tasks

- Determine the optimizer decisions for join order and table access.
- Run the query (*set noexec on*) to confirm your hypothesis.
- Determine the differences between the queries which run on the same tables but have different indexes
- Determine the advantage of having indexes on all join columns

Lab Setup

Consult the Lab Worksheet as a reference to the table names and their indexes

Exercise Instructions... ➡

Detailed Instructions

In this lab, you will determine the join strategy the optimizer will choose for a given query. You will also evaluate the effect of search arguments and indexes on this strategy. Consult the table descriptions at the beginning of this book if you need to.

1. For each of the queries, determine the optimizer decisions for join order and access. Run the query (set no exec on) to confirm your hypothesis. (Note that in each query the two tables involved are the same size. One has an index (titles_idpr), one does not (titles).)

Query A

a. Determine the order of the join selection and the access method used

```
SELECT t1.title, t2.price
FROM   titles t1, titles_idpr t2
WHERE  t1.title_id = t2.title_id
```

	Table Name	Access Method
Outer Table	titles	scan
Inner Table	t2	index non-cl

Query B (same as Query A except for first table in FROM clause)

b. Determine the order of the join selection and the access method used

```
SELECT t1.title, t2.price
FROM   titles_titlid t1, titles_idpr t2
WHERE  t1.title_id = t2.title_id
```

	Table Name	Access Method
Outer Table	T1	scan
Inner Table	T2	clustered

Query C

c. Determine the order of the join selection and the access method used

```
SELECT a1.au_lname, a2.state
FROM   authors_idstate a1, authors a2
WHERE  a1.au_id = a2.au_id
```

	Table Name	Access Method
Outer Table		
Inner Table		

2. For each of the following queries, determine the optimizer decisions for the join order and access.

Query D (same as Query B but with additional AND clause)

a. Determine the order of the join selection and the access method used

```
SELECT t1.title, t2.price
FROM   titles_idpr t1, titles_titlid t2
WHERE  t1.title_id = t2.title_id
AND    t1.title_id like "T69%"
```

	Table Name	Access Method
Outer Table		
Inner Table		

Query E

b. Determine the order of the join selection and the access method used

```
SELECT t1.title, t2.price
FROM   titles t1, titles_titlid t2
WHERE  t1.title_id = t2.title_id
AND    t2.title_id like "T691%"
```

	Table Name	Access Method
Outer Table		
Inner Table		

Query F

c. Determine the order of the join selection and the access method used

```
SELECT a1.au_lname, a2.au_ord
FROM   authors_id a1, titleauthor_ididtid a2
WHERE  a2.au_id = 'A1374065371'
AND    a2.au_id = a1.au_id
```

	Table Name	Access Method
Outer Table		
Inner Table		

d. What is the advantage of having indexes on both join columns?

Optional Exercises

3. For each of the queries, determine the optimizer decisions for join order and access. Run the query (set no exec on) to confirm your hypothesis.

Query G

a. Determine the order of the join selection and the access method used

```
SELECT a.au_lname, t.title, a.state
FROM   authors_idstate a, titleauthor_ididtid ta,
       titles_titlid t
WHERE  a.au_id = ta.au_id
AND    ta.title_id = t.title_id
AND    t.title like "B%"
```

	Table Name	Access Method
Outer Table		
First Inner Table		
Second Inner Table		

Query H (same as Query G except for last line)

b. Determine the order of the join selection and the access method used

```
SELECT a.au_lname, t.title, a.state
FROM   authors_idstate a, titleauthor_ididtid ta,
       titles_titlid t
WHERE  a.au_id = ta.au_id
AND    ta.title_id = t.title_id
AND    a.state = 'CA'
```

	Table Name	Access Method
Outer Table		
First Inner Table		
Second Inner Table		

Query I

c. Determine the order of the join selection and the access method used

```
SELECT au_lname
FROM   authors_id
WHERE  au_id =
   (SELECT au_id
    FROM   titleauthor_idid
    WHERE  au_ord = 1
    AND    title_id = 'T61159')
```

	Table Name	Access Method
Outer Table		
Inner Table		

d. Where is the join clause?

Solutions

1. *For each of the following queries, predict the order in which the optimizer will approach the tables and which indexes it will use for each one. Then run the query (set no exec on) to confirm your hypothesis. (Note that in each query the two tables involved are the same size. In most cases one has an index, one does not.)*

Query A

a. Determine the order of the join selection and the access method used

```
1> SELECT t1.title, t2.price
2> FROM   titles t1, titles_idpr t2
3> WHERE  t1.title_id = t2.title_id
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles
Nested iteration
Table Scan
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
```

	Table Name	Access Method
Outer Table	titles	scan
Inner Table	titles_idpr	clustered index

Because there is no index on titles, for each matching row in titles_idpr it would have to perform a table scan of the titles table to find the matching row. It is cheaper to scan titles once and use the clustered index to find the matching rows from the titles_idpr table.

Query B (same as Query A except for first table in FROM clause)

b. Determine the order of the join selection and the access method used

```
1> SELECT t1.title, t2.price
2> FROM   titles_titlid t1, titles_idpr t2
3> WHERE  t1.title_id = t2.title_id
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_titlid
Nested iteration
Table Scan
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
```

	Table Name	Access Method
Outer Table	titles_titlid	scan
Inner Table	titles_idpr	clustered index

The index on title_id in titles_titlid is non-clustered. The query optimizer uses the worst case estimate of one data page read for each qualifying index row. Because all rows qualify (there is no SARG) the resulting i/o cost of reading the index pages and reading each data page would be higher than performing a table scan on the titles_titlid table.

For example, if there were 3 levels in the index and the data page to be read for each row in the index, that would require 4 reads per qualifying row, or 4 * 5000 (20,000) reads. This is much more than scanning all the data pages once with a table scan.

Query C

c. Determine the order of the join selection and the access method used

```
1> SELECT a1.au_lname, a2.state
2> FROM   authors_idstate a1, authors a2
3> WHERE  a1.au_id = a2.au_id
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
authors
Nested iteration
Table Scan
FROM TABLE
authors_idstate
Nested iteration
Index : idx1
```

	Table Name	**Access Method**
Outer Table	authors	scan
Inner Table	authors_idstate	non-clustered index on au_id

Because there is no index on authors, for each matching row in authors_idstate it would have to perform a table scan of the authors table to find the matching row. It is cheaper to scan authors once and use the non-clustered index on au_id to find the matching rows from the authors_idstate table.

2. *For each of the following queries, predict the order in which the optimizer will approach the tables and which indexes it will use for each one. Then run the query to confirm your hypothesis. (Note that both tables have indexes on join columns.)*

Query D (same as Query B but with additional AND clause)

a. Determine the order of the join selection and the access method used

```
1> SELECT t1.title, t2.price
2> FROM   titles_idpr t1, titles_titlid t2
3> WHERE  t1.title_id = t2.title_id
4> AND    t1.title_id like "T69%"
5> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
FROM TABLE
titles_titlid
Nested iteration
Index : idx2
```

	Table Name	**Access Method**
Outer Table	titiles_idpr	clustered index
Inner Table	titles_titlid	non-clustered on title_id

The optimizer chooses the clustered index on titles_idpr to satisfy the range query and uses the non-clustered index on titles_titlid to find the matching title_id.

Query E

b. Determine the order of the join selection and the access method used

```
1> SELECT t1.title, t2.price
2> FROM   titles t1, titles_titlid t2
3> WHERE  t1.title_id = t2.title_id
4> AND    t2.title_id like "T691%"
5> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_titlid
Nested iteration
Index : idx2
FROM TABLE
titles
Nested iteration
Table Scan
```

	Table Name	Access Method
Outer Table	titles_titlid	non-clustered index on title_id
Inner Table	titles	scan

In this query, the optimizer uses the non-clustered index on titles_id for the titles_titlid table because the search clause is very selective, estimating that only one row will satisfy the query. If there were 3 levels in the non-clustered index in addition to the data page, that would result in 4 total reads to access the one row in the titles_titlid table. Because there is no index on titles, a table scan would have to be performed. If the titles table were accessed first, the server would have to perform 4 reads to find a potential match for each row in the titles table. Since there are 5000 rows in the titles table, this would result in 20,000 total reads. In comparison, the cost in i/o by first accessing the qualifying row(s) in the titles_titlid table and then performing a table scan of titles is 4 reads plus the number of data pages in titles * the number of qualifying rows in titles_titlid, in this case estimated to be just one qualifying row.

Query F

c. Determine the order of the join selection and the access method used

```
1> SELECT a1.au_lname, a2.au_ord
2> FROM   authors_id a1, titleauthor_ididtid a2
3> WHERE  a2.au_id = 'A1374065371'
4> AND    a2.au_id = a1.au_id
5> go
```

```
STEP 1
The type of query is SELECT.
FROM TABLE
titleauthor_ididtid
Nested iteration
Using Clustered Index
FROM TABLE
authors_id
Nested iteration
Using Clustered Index
```

	Table Name	Access Method
Outer Table	titleauthor_ididtid	clustered index,
Inner Table	authors_id	clustered index

The optimizer uses the clustered index to resolve the SARG on titleauthor_ididtid.au_id which limits the number of au_ids which need to be searched in the authors_id table. It uses the clustered index on authors_id.au_id to find the matching au_ids.

d. What is the advantage of having indexes on both join columns?

The optimizer can take advantage of one for SARG work, the other for a join.

Optional Exercises: Solutions

3. *For each of the following queries, predict the order in which the optimizer will approach the tables and which indexes it will use for each one. Then run the query to confirm your hypothesis.*

Query G

a. Determine the order of the join selection and the access method used

```
1> SELECT a.au_lname, t.title, a.state
2> FROM   authors_idstate a, titleauthor_ididtid ta,
3>        titles_titlid t
4> WHERE  a.au_id = ta.au_id
5> AND    ta.title_id = t.title_id
6> AND    t.title like "B%"
7> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_titlid
Nested iteration
Using Clustered Index
FROM TABLE
titleauthor_ididtid
Nested iteration
Index : idx2
FROM TABLE
authors_idstate
Nested iteration
Index : idx1
```

	Table Name	Access Method
Outer Table	titles_titlid	clustered
First Inner Table	titleauthor_ididtid	idx2
Second Inner Table	authors_idstate	idx1

In this case, the optimizer is using the SARG on the titles_titlid table to limit the number of qualifying rows from the titles table. Remember, the LIKE "B%" gets converted to a " >= "B" and < "C" and clustered indexes work well with range searches. Also in this case, using the non-clustered indexes on the titleauthor_ididtid (title_id) and authors_idstate (au_id) are less costly in terms of i/o than performing a table scan for each iteration through each table.

Query H (same as Query G except for last line)

b. Determine the order of the join selection and the access method used

```
1> SELECT a.au_lname, t.title, a.state
2> FROM   authors_idstate a, titleauthor_ididtid ta,
3>        titles_titlid t
4> WHERE  a.au_id = ta.au_id
5> AND    ta.title_id = t.title_id
6> AND    a.state = 'CA'
7> go
STEP 1
The type of query is SELECT.
FROM TABLE
authors_idstate
Nested iteration
Table Scan
FROM TABLE
titleauthor_ididtid
Nested iteration
Using Clustered Index
FROM TABLE
titles_titlid
Nested iteration
Index : idx2
```

	Table Name	Access Method
Outer Table	authors_idstate	scan
First Inner Table	titleauthor_ididtid	clustered
Second Inner Table	titles_titlid	idx2

In this case, the optimizer is using the SARG on the authors_idstate table to limit the number of qualifying rows from the titles table. Because the SARG on the state column is not very selective, a table scan of the authors_idstate table is more efficient in terms of total i/o on the authors_idstate table versus using the index to find all the matching rows.

Followup question: What is the advantage of using the table with the restrictive SARG as the outer table in the join?

Using the table with the SARG as the outer table reduces the total amount of I/O for the query be reducing the number of iterations required of the inner tables.

Query I

c. Determine the order of the join selection and the access method used

```
1> SELECT au_lname
2> FROM   authors_id
3> WHERE  au_id =
4>              (SELECT au_id
5>               FROM   titleauthor_idid
6>               WHERE  au_ord = 1
7>               AND    title_id = 'T61159')
8> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titleauthor_idid
Nested iteration
Table Scan
STEP 2
The type of query is SELECT.
FROM TABLE
authors_id
Nested iteration
Using Clustered Index
```

	Table Name	Access Method
Outer Table	titleauthor_idid	scan
Inner Table	authors_id	clustered

The optimizer performs a table scan on the titleauthor_idid table because the only available index for this table is based on au_id and title_id, with au_id as the first column in the index. The au_id column is not included in the where clause for the subquery, therefore the optimizer cannot make use of this index and a table scan is required to find the resulting rows. The results of the subquery are then joined with the authors_id table.

d. Where is the join clause?

The optimizer "flattens" the nested equality into a join.

Lab 6f – Work Tables and Sorts

(Student Guide, page 6-66)

Exercise Overview

Goals

- Identify the queries that require work tables, sorts, and reformatting
- Rewrite the queries to change query plans, and improve the performance of queries by using different tables (using different indexes)

General Tasks

- Run the query (set showplan and noexec on)
- Predict what query plans the optimizer will choose
- Run queries without using the final (order by*)* and note differences
- Determine if time or I/O would improve if a table scan were used (as opposed to reformatting)

Lab Setup

Consult the Lab Worksheet as a reference to the table names and their indexes

Exercise Instructions...

Detailed Instructions

In this lab, you will identify queries requiring work tables, sorts, and reformatting. You will rewrite queries to change query plans, and you will improve the performance of queries by adding indexes.

1. Run the query (set showplan and noexec on).

Query A

```
SELECT state, city, sum(advance)
FROM   authors a, titleauthor ta, titles t
WHERE  a.au_id = ta.au_id
AND    ta.title_id = t.title_id
GROUP BY  state, city
ORDER BY  state, city
```

a. What query plan do you predict the optimizer will choose?

	Table Name	Access Method
Outer Table		
First Inner Table		
Second Inner Table		

b. Run this query again without the final *order by.* What difference does this make?

	Table Name	Access Method
Outer Table		
First Inner Table		
Second Inner Table		

2. Predict what plan the optimizer will choose, and then run the query to test your hypothesis.

Query B

```
SELECT count(*)
FROM   titles_idpr
WHERE  price > 30
AND    title_id in ('T18159', 'T61159', 'T44078')
```

a. What query plan do you predict the optimizer will choose? Are you correct?

3. Would time or I/O improve if a table scan were used?

a. Set noexec off and statistics io on

b. Run **Query B** again

c. Substitute titles_idpr with titles (no indexes) and run the query again

d. Document the difference in IO between the queries

Index vs. Table Scan for OR's

Query	Logical Reads	Physical Reads
With Index		
Table Scan		

4. Predict what plan the optimizer will choose, and then run the query to test your hypothesis.

Query E

```
SELECT pub_name, title
FROM   publishers p, titles_titlid t
WHERE  p.pub_id *= t.pub_id
```

a. What query plan do you predict the optimizer will choose? Are you correct?

b. What is the effect of the asterisk on the logic? Remove it to see the effect on the query plan.

```
SELECT pub_name, title
FROM   publishers p, titles_titlid t
WHERE  p.pub_id = t.pub_id
```

5. Predict what plan the optimizer will choose, and then run the query to test your hypothesis.

Query F

```
SELECT city, state
FROM   authors_idstate
WHERE  state = 'CA'
UNION
SELECT city, state
FROM   publishers
UNION
SELECT city, state
FROM   stores
```

a. What query plan do you predict the optimizer will choose? Are you correct?

b. In what way is this query similar to one with an OR clause?

Optional Exercises

1. Predict what plan the optimizer will choose, and then run the query (with showplan and noexec on) to test your hypothesis.

Query A

```
SELECT a.au_lname, a.au_fname, t.title, p.pub_id
FROM   authors a, titleauthor ta, titles t, publishers p
WHERE  a.au_id = ta.au_id
AND    ta.title_id = t.title_id
AND    t.pub_id = p.pub_id
AND    t.price > 1000
AND    a.au_lname between 'G' and 'Q'
AND    p.state in ('MD', 'PA', 'NY')
```

a. What query plan do you predict the optimizer will choose? Are you correct?

b. Document the total IO's performed

IO Performance

Query	Logical Reads	Physical Reads

2. Choose different variations on the basic tables. How does the plan change? Find the best plan in terms of logical I/O. Here are some possibilities:

a. Use *authors_id* instead of *authors*

IO Performance

Query	Logical Reads	Physical Reads

b. Use *authors_idnames* instead of *authors*

IO Performance

Query	Logical Reads	Physical Reads

c. Use *titles_idpr* instead of *titles*

IO Performance

Query	Logical Reads	Physical Reads

d. Use *titles_pridtitl* instead of *titles*

IO Performance

Query	Logical Reads	Physical Reads

3. What is the best combination of tables/indexes to minimize logical I/O?

Solutions

1. *Predict what plan the optimizer will choose, and then run the query (set showplan and noexec on) to test your hypothesis.*

Query A

```
SELECT state, city, sum(advance)
FROM   authors a, titleauthor ta, titles t
WHERE  a.au_id = ta.au_id
AND    ta.title_id = t.title_id
GROUP BY  state, city
ORDER BY  state, city
```

a. What query plan do you predict the optimizer will choose? Are you correct?

```
STEP 1
The type of query is INSERT.
The update mode is direct.
Worktable created for REFORMATTING.
FROM TABLE
titleauthor
Nested iteration
Table Scan
TO TABLE
Worktable
STEP 2
The type of query is INSERT.
The update mode is direct.
Worktable created for REFORMATTING.
FROM TABLE
titles
Nested iteration
Table Scan
TO TABLE
Worktable
STEP 3
The type of query is SELECT (into a worktable).
GROUP BY
Vector Aggregate
FROM TABLE
authors
Nested iteration
Table Scan
FROM TABLE
Worktable
Nested iteration
Using Clustered Index
FROM TABLE
Worktable
```

```
Nested iteration
Using Clustered Index
TO TABLE
Worktable
STEP 4
The type of query is SELECT.
FROM TABLE
Worktable
Nested iteration
Table Scan
1>
```

	Table Name	Access Method
Outer Table	titleauthor	scan & reformatting
First Inner Table	titles	scan & reformatting
Second Inner Table	authors	scan

The optimizer creates worktables for reformatting for titles and titleauthor as neither table has any indexes defined. Using the reformatting strategy is less costly than iterative table scans of the titles and titleauthor tables. It then uses the authors table as the outer table and joins it with the reformatted work tables for titleauthor and titles using the clustered indexes created on those worktables. These results are selected into a work table to perform the group by and calculate the aggregate then the final ordered results are selected from the work table. In all, the optimizer will require four steps to resolve this query.

b. Run this query again without the final order by. What difference does this make?

	Table Name	Access Method
Outer Table	titleauthor	scan & reformatting
First Inner Table	titles	scan & reformatting
Second Inner Table	authors	scan

Removing the order by has no effect on the query plan as the order by clause uses the same worktable as the group by clause, which has already ordered the data. If the order by specified a different sort order, then an additional step would be required to select the group by results into a worktable for sorting.

If a marketing department regularly ran queries per state/city, what indexing scheme might you consider to support these queries?

Consider creating a clustered index on the state and city columns of authors, or create a non-clustered index on state, city, and advance which could be used to cover the query. In addition, clustered indexes should be created on titles(title_id) and titleauthor(au_id).

2. *Predict what plan the optimizer will choose, and then run the query to test your hypothesis.*

Query B

```
SELECT count(*)
FROM   titles_idpr
WHERE  price > 30
AND    title_id in ('T18159', 'T61159', 'T44078')
```

a. What query plan do you predict the optimizer will choose? Are you correct?

```
The type of query is SELECT.
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
FROM TABLE
titles_idpr
Nested iteration
Using Dynamic Index
```

This query uses the OR Strategy due to the IN clause which gets converted to an OR. It performs three selects for each of the title_ids in the list using the clustered index on title_id and copies the row ids into a worktable and uses it as a Dynamic Index to check each of the rows for the additional where clause conditions (price > 30).

3. *What time or I/O improve if a table scan were used?*

   a. Set noexec off and statistics io on

```
1> set noexec off
2> go
1> set statistics io on
2> go
```

   b. Run **Query B** again

Query C

```
1> SELECT count(*)
2> FROM   titles_idpr
3> WHERE  price > 30
4> AND    title_id in ('T18159', 'T61159', 'T44078')
5> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
FROM TABLE
titles_idpr
Nested iteration
Using Dynamic Index
STEP 2
The type of query is SELECT.
Table Scan

 -----------
           2
Table: titles_idpr  scan count 3,  logical reads: 13,  physical
reads: 0
Table: Worktable  scan count 1,  logical reads: 17,  physical
reads: 1
Total writes for this command: 1
```

   c. Substitute titles_idpr with titles (no indexes) and run the query again

Query D

```
1> SELECT count(*)
2> FROM   titles
3> WHERE  price > 30
4> AND    title_id in ('T18159', 'T61159', 'T44078')
5> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles
Nested iteration
Table Scan
STEP 2
The type of query is SELECT.
Table Scan

 -----------
           2
Table: titles  scan count 1,  logical reads: 624,  physical reads:
624
Total writes for this command: 11

(1 row affected)
```

d. Document the difference in IO between the queries

Index vs. Table Scan for OR's

Query	Logical Reads	Physical Reads
Query C	30	1
Query D	624	624

4. *Predict what plan the optimizer will choose, and then run the query to test your hypothesis.*

Query E

```
SELECT pub_name, title
FROM   publishers p, titles_titlid t
WHERE  p.pub_id *= t.pub_id
```

a. What query plan do you predict the optimizer will choose? Are you correct?

```
STEP 1
The type of query is SELECT.
FROM TABLE
publishers
```

```
Nested iteration
Table Scan
FROM TABLE
titles_titlid
Nested iteration
Table Scan
```

Table scan on both tables. Table scans are performed on both tables as the query contains no SARGs and the using the non-clustered index on titles_titlid(pub_id) to resolve the query would result in more I/O than a table scan.

b. What is the effect of the asterisk on the logic? Remove it to see the effect on the query plan.

```
1> SELECT pub_name, title
2> FROM   publishers p, titles_titlid t
3> WHERE  p.pub_id = t.pub_id
4> go
STEP 1
The type of query is SELECT.
FROM TABLE
titles_titlid
Nested iteration
Table Scan
FROM TABLE
publishers
Nested iteration
Table Scan
```

The asterisk forces the publishers table to be the outer table, selecting all rows from publishers whether or not they successfully join with any rows from the titles_titlid table. Removing the * results in the titles_titlid being the outer table, which would be expected as the publishers table is the smaller of the two and the optimizer will generally make the smaller table as the inner table to reduce overall I/O.

5. *Predict what plan the optimizer will choose, and then run the query to test your hypothesis.*

Query F

```
SELECT city, state
FROM   authors_idstate
WHERE  state = 'CA'
UNION
SELECT city, state
FROM   publishers
UNION
SELECT city, state
FROM   stores
```

a. What query plan do you predict the optimizer will choose? Are you correct?

(Table scan on all three.)

```
STEP 1
The type of query is INSERT.
The update mode is direct.
FROM TABLE
authors_idstate
Nested iteration
Table Scan
TO TABLE
Worktable
STEP 1
The type of query is INSERT.
The update mode is direct.
FROM TABLE
publishers
Nested iteration
Table Scan
TO TABLE
Worktable
STEP 1
The type of query is INSERT.
The update mode is direct.
FROM TABLE
stores
Nested iteration
Table Scan
TO TABLE
Worktable
STEP 1
The type of query is SELECT.
This step involves sorting.
FROM TABLE
Worktable
Using GETSORTED
Table Scan
```

The optimizer performs a table scan for all three queries because their are no SARGS for the queries against publishers and stores and there is no non-clustered index available to cover the queries; the SARG on authors_idstate is not selective enough to result in the optimizer using the index over a table scan.

b. In what way is this query similar to one with an OR clause?

Each select is executed and the results stored in a worktable.

Optional Exercises: Solutions

1. *Predict what plan the optimizer will choose, and then run the query (with showplan and noexec on) to test your hypothesis.*

Query A

```
SELECT a.au_lname, a.au_fname, t.title, p.pub_id
FROM   authors a, titleauthor ta, titles t, publishers p
WHERE  a.au_id = ta.au_id
AND    ta.title_id = t.title_id
AND    t.pub_id = p.pub_id
AND    t.price > 1000
AND    a.au_lname between 'G' and 'Q'
AND    p.state in ('MD', 'PA', 'NY')
```

a. What query plan do you predict the optimizer will choose? Are you correct?

```
STEP 1
The type of query is INSERT.
The update mode is direct.
Worktable created for REFORMATTING.
FROM TABLE
titleauthor
Nested iteration
Table Scan
TO TABLE
Worktable
STEP 2
The type of query is INSERT.
The update mode is direct.
Worktable created for REFORMATTING.
FROM TABLE
authors
Nested iteration
Table Scan
TO TABLE
Worktable
STEP 3
The type of query is SELECT.
FROM TABLE
titles
Nested iteration
Table Scan
FROM TABLE
Worktable
Nested iteration
Using Clustered Index
FROM TABLE
Worktable
```

```
Nested iteration
Using Clustered Index
FROM TABLE
publishers
Nested iteration
Table Scan
 au_lname                  au_fname       title        pub_id
 ---------------------------------------- --------------------
------  --------------- -----------
 Kimbrough                                Tim
 Guidelines for OSI NSAP allocation in the internet  P780
 Gradenigo                                Patricia
 Guidelines for OSI NSAP allocation in the internet  P780
 Loggins                                  Martin
 Host names on-line                                  P095
Table:authors  scan count 1,logical reads: 223,phys. reads: 223
Table:titleauthor  scan count 1,logical reads: 106,phys. reads: 106
Table: titles  scan count 1,logical reads: 624,phys. reads: 0
Table: publishers  scan count 22,logical reads: 44,phys. reads: 2
Table: Worktable  scan count 62,logical reads: 6753,phys. reads: 40
Table: Worktable  scan count 74,logical reads: 2381,phys. reads: 32
Total writes for this command: 241

(3 rows affected)
```

None of the tables in this query have any indexes defined to support this query. The optimizer chooses to use the reformatting strategy for the titleauthor and authors table. It then uses the titles table as the outer table since it has the most selective SARG and joins it with the REFORMATTED titleauthor and authors tables using the generated clustered indexes. The publishers table is the innermost table since it is the smallest table.

b. Document the total IO's performed

IO Performance

Query	Logical Reads	Physical Reads
Query A	10131	403

2. *Choose different variations on the basic tables. How does the plan change? Find the best plan in terms of logical I/O. Here are some possibilities:*

   a. Use *authors_id* instead of *authors*

```
Table: authors_id  scan count 74,logical reads: 228,phys. reads: 71
Table: titleauthor  scan count 1,logical reads: 106,phys. reads: 0
Table: titles  scan count 1,logical reads: 624,phys. reads: 0
Table: publishers  scan count 22,logical reads: 44,phys. reads: 0
Table: Worktable  scan count 62,logical reads: 6752,phys. reads: 40
Total writes for this command: 202
```

IO Performance

Query	Logical Reads	Physical Reads
2 a	7754	111

   b. Use *authors_idnames* instead of *authors*

```
Table: authors_idnames  scan 74,logical reads: 224,phys. reads: 52
Table: titleauthor  scan count 1,logical reads: 106,phys. reads: 0
Table: titles  scan count 1,logical reads: 624,phys. reads: 0
Table: publishers  scan count 22,logical reads: 44,phys. reads: 0
Table: Worktable  scan count 62,logical reads: 6752,phys. reads: 40
Total writes for this command: 202

(3 rows affected)
```

IO Performance

Query	Logical Reads	Physical Reads
2b	7750	92

   c. Use *titles_idpr* instead of *titles*

```
Table: authors  scan count 1,logical reads: 223,phys. reads: 0
Table: titleauthor  scan cnt 62,logical reads: 6572,phys. reads: 0
Table: titles_idpr  scan count 1,logical reads: 65,phys. reads: 35
Table: publishers  scan count 22,logical reads: 44,phys. reads: 0
Table: Worktable  scan count 74,logical reads: 2377,phys. reads: 32
Total writes for this command: 39

(3 rows affected)
```

IO Performance

Query	Logical Reads	Physical Reads
2c	9281	67

d. Use *titles_pridtitl* instead of *titles*

```
Table: authors  scan count 1,logical reads: 223,phys. reads: 0
Table: titleauthor  scan cnt 62,logical reads: 6572,phys. reads: 0
Table: titles_pridtitl  scan 1,logical reads: 19,phys. reads: 18
Table: publishers  scan count 22,logical reads: 44,phys. reads: 0
Table: Worktable  scan count 74,logical reads: 2377,phys. reads: 32
Total writes for this command: 39

(3 rows affected)
```

IO Performance

Query	Logical Reads	Physical Reads
2d	9235	50

3. ***What is the best combination of tables/indexes to minimize logical I/O?***

authors_idnames, titleauthor_ididtid, titles_pridtitl, publishers

Lab 6g – Stored Procedures

(Student Guide, page 6-74)

Exercise Overview

Goals

- Compare the effect of using a current query plan vs. an old query plan.

General Tasks

- Use the table *titles_idpr* which has a non-clustered index on the *price* column. With showplan and statistics io, examine the query plans and logical i/o count for several queries
- Create a stored procedure that queries a range of data (on the index key)
- Explain what is happening when you run your procedures with different ranges of prices
- Record results and explain the outcome of running the stored procedure using the *exec with recompile* option
- Define a permanent solution

Lab Setup

Consult the Lab Worksheet as a reference to the table names and their indexes

Exercise Instructions...

Detailed Instructions

In this lab, you will execute stored procedures and compare the effect of using a current query plan vs. recompiling.

1. Use the table *titles_idpr* which has a non-clustered index on the *price* column. With showplan and statistics io, examine the query plans and logical i/o count for the following:

Query A

```
SELECT count(title)
FROM   titles_idpr
WHERE  price > $1.95
```

Query B

```
SELECT count(title)
FROM   titles_idpr
WHERE  price > $75.95
```

Fill in the first two rows below:

Query	SARG	Query Plan	Logical I/O
A	price >$1.95		621
B	price >$75.95		196
Stored Proc	price > 1.95		621
Stored Proc	price > 75.95		621
Recompile	price >		196
Recompile	price >		621

2. Create a stored procedure as follows, replacing "N" with your name or user number:

```
CREATE PROC userNcount
   @price  money
AS
SELECT count(title) from titles_idpr
WHERE  price > @price
```

3. Run your procedure first with *$1.95* as the price, then with *$75.95* as the price, and fill in rows three and four in the above table. Explain what is happening.

4. Now run the stored procedure with the recompile option using *$75.95* as the price. Then run it without recompile, using *$1.95* as the price. Record your results and explain.

5. What is a permanent solution to the problem?

Solutions

1. *Use the table titles_idpr which has a non-clustered index on the price column. With showplan and statistics io, examine the query plans and logical i/o count for the following queries:*

Query A

```
1> SELECT count(title)
2> FROM   titles_idpr
3> WHERE  price > $1.95
4> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
STEP 2
The type of query is SELECT.
Table Scan

 -----------
        4936
Table: titles_idpr  scan count 1,  logical reads: 621,  physical
reads: 0
Total writes for this command: 0

(1 row affected)
```

Query B

```
1> SELECT count(title)
2> FROM   titles_idpr
3> WHERE  price > $75.95
4> go
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles_idpr
Nested iteration
Index : idx2
STEP 2
The type of query is SELECT.
Table Scan

 -----------
         192
```

```
Table: titles_idpr  scan count 1,  logical reads: 196,  physical
      reads: 3
      Total writes for this command: 0

      (1 row affected)
```

a. Fill in the first two rows in table below:

Query	SARG	Query Plan	Logical I/O
Query A	price >$1.95	Table Scan (621 IO's) Clustered index is used for table access only	621
Query B	price >$75.95	idx2	196

2. *Create a stored procedure as follows, replacing "N" with your name or user number:*

```
1> CREATE PROC userNcount
2>    @price  money
3> AS
4> SELECT count(title) from titles_idpr
5> WHERE  price > @price
6> go
1>
```

3. *Run your procedure first with $1.95 as the price, then with $75.95 as the price, and fill in rows three and four in the above table. Explain what is happening.*

Exec A

```
1> exec userNcount 1.95
2> go
STEP 1
The type of query is EXECUTE.
STEP 1
The type of query is DECLARE.
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
STEP 2
The type of query is SELECT.
```

```
Table Scan

 -----------
        4936
Table: titles_idpr  scan count 1,  logical reads: 621,  physical
reads: 0
Total writes for this command: 0
Total writes for this command: 0

(return status = 0)
```

Exec B

```
1> exec userNcount 75.95
2> go
STEP 1
The type of query is EXECUTE.
STEP 1
The type of query is DECLARE.
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles_idpr
Nested iteration
Using Clustered Index
STEP 2
The type of query is SELECT.
Table Scan

 -----------
        192
Table: titles_idpr  scan count 1,  logical reads: 621,  physical
reads: 0
Total writes for this command: 0
Total writes for this command: 0

(return status = 0)
```

Query	SARG	Query Plan	Logical I/O
Stored Proc	price >$1.95	table scan	621
Stored Proc	price >$75.95	table scan	621

a. Conclusion Exec A & B

In both cases the IO count is the same (621, Table Scan)

The query plan generated at the first invocation is reused by the second invocation of the stored procedure although the value is different. The query plan uses a table scan.

4. *Now run the stored procedure with the recompile option using $75.95 as the price. Then run it without recompile, using $1.95 as the price. Record your results and explain.*

Exec C

```
1> exec userNcount 75.95 with recompile
2> go
STEP 1
The type of query is EXECUTE.
STEP 1
The type of query is DECLARE.
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles_idpr
Nested iteration
Index : idx2
STEP 2
The type of query is SELECT.
Table Scan

 -----------
        192
Table: titles_idpr  scan count 1,  logical reads: 196,  physical
reads: 0
Total writes for this command: 0
Total writes for this command: 0

(return status = 0)
```

Exec D

```
1> exec userNcount 1.95
2> go
STEP 1
The type of query is EXECUTE.
STEP 1
The type of query is DECLARE.
STEP 1
The type of query is SELECT.
Scalar Aggregate
FROM TABLE
titles_idpr
Nested iteration
Index : idx2
STEP 2
The type of query is SELECT.
Table Scan

 -----------
        4936
```

```
Table: titles_idpr  scan count 1,  logical reads: 4987,  physical
reads: 47
Total writes for this command: 0
Total writes for this command: 0

(return status = 0)
```

Query	SARG	Query Plan	Logical I/O
Recompile	price >$75.95	idx2	196
Recompile	price >$1.95	idx2	4987

a. Conclusion Exec C & D

The first invocation generates a query plan using the index. The second invocation uses this same plan and demonstrates the inefficiency of using a non-clustered index to access a large proportion of the rows in a table.

5. *What is a permanent solution to the above problem?*

Always execute this procedure with the *recompile* option.

Lab 6g – Stored Procedures: Solutions

Lab 7 – Distributing Data Across Devices

(Student Guide, page 7-40)

Exercise Overview

Goals

- Examine database placement on devices and plan object placement strategies to improve performance

General Tasks

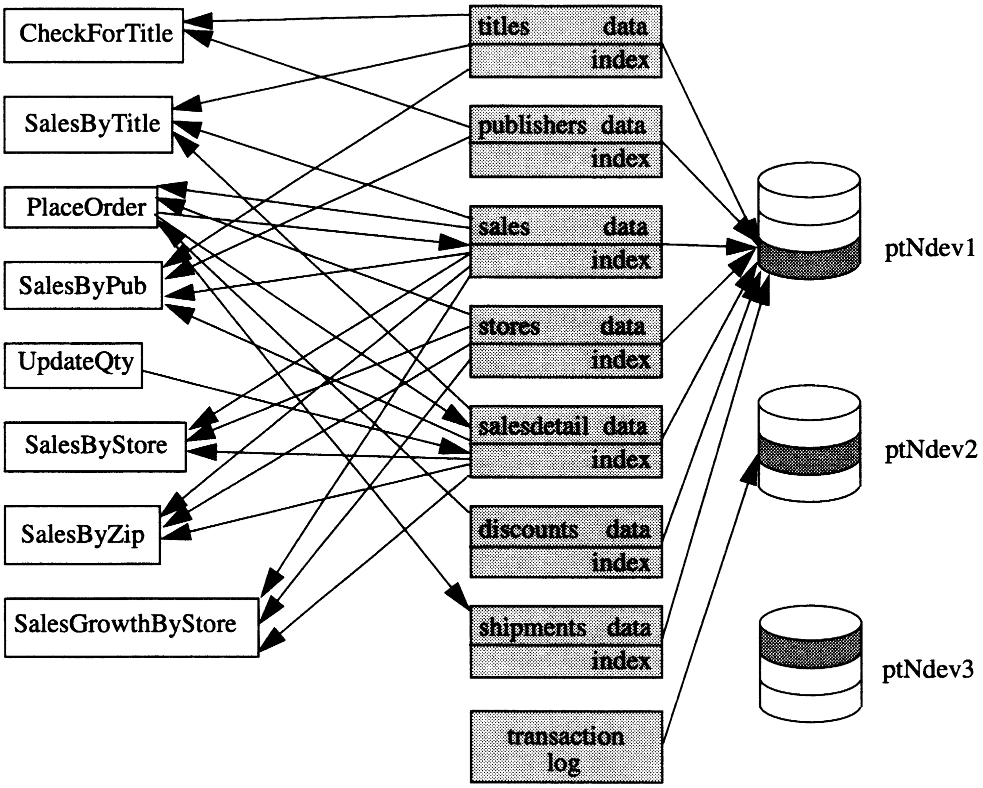
- Discuss current mapping procedures of tables to devices in *pubtune*
- Determine a better object placement strategy that would improve the *"CheckForTitle"* transaction for critical performance.
  - Write the procedural steps involved in your strategy to obtain better performance.
  - Measure baseline (before) and new (after) performance and record findings by filling in the information in the columns provided.
  - Determine if performance has dramatically improved

Exercise Instructions...

Detailed Instructions

In this lab, you will examine database placement on devices and plan object placement strategies to improve performance.

1. The current mapping of procedures to tables to devices in *pubtune* is illustrated below:

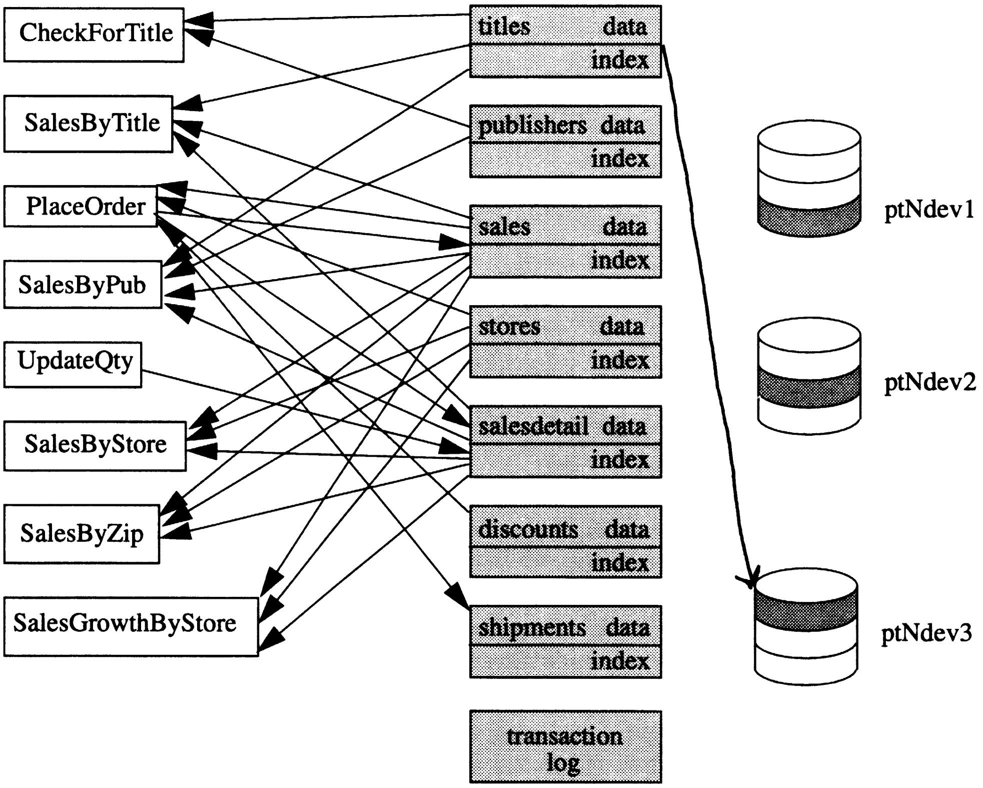


2. The table below includes average frequencies of execution for each procedure. Also, procedures with asterisks have been flagged as critical. Determine a better object placement strategy that would improve the "CheckForTitle" transaction (Assume you have multiple physical disks available.) Write down all the steps you would take to carry this out.

	Procedure	Frequency	Baseline Response Time
**	CheckForTitle	20,000/day	
**	PlaceOrder	1,000/day	
	UpdateQty	1,500/day	
**	SalesByStore	50,000/day	
	SalesGrowthByStore	2/day	
	SalesByTitle	5,000/day	
**	SalesByPub	10/day	
	SalesByZip	2,000/day	

**** indicates procedures where performance is critical**

3. Show the revised mapping in the following diagram:



Optional:

4. There are several other critical procedures we should tune for. What object placement strategies might you use to improve performance?

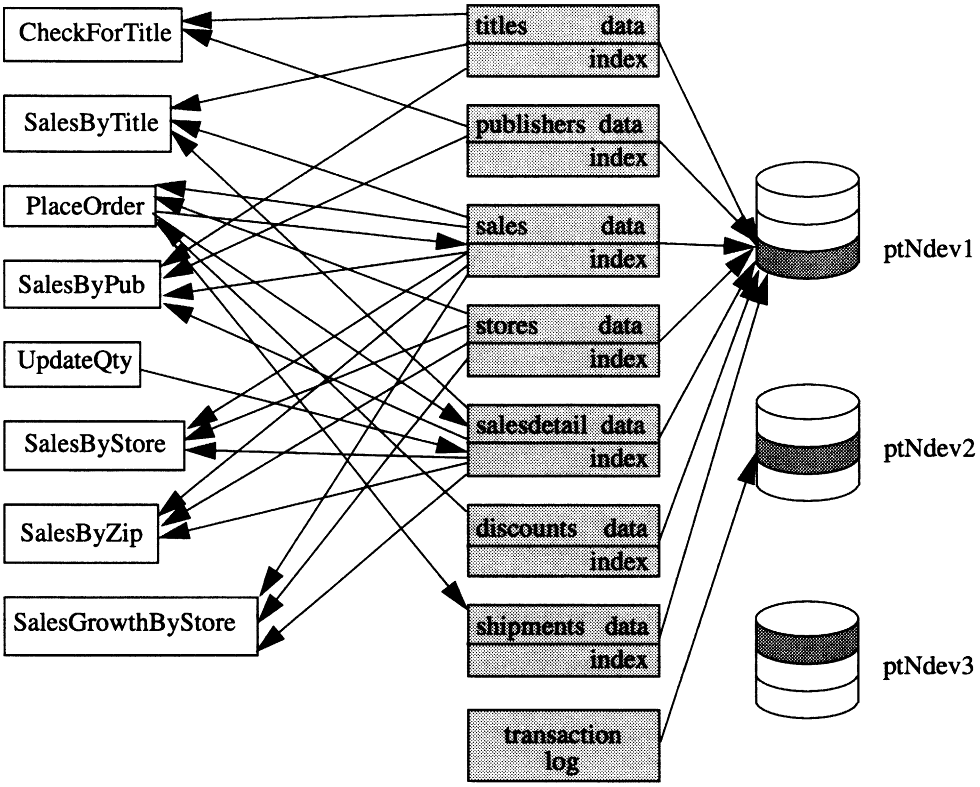
5. If system resources permit, put your strategies to work. Measure baseline (before) and new (after) performance and fill in the columns of the following table. Has performance improved significantly?

	Procedure	Frequency	Baseline Response Time	New Response Time
**	CheckForTitle	20,000/day		
**	PlaceOrder	1,000/day		
	UpdateQty	1,500/day		
**	SalesByStore	50,000/day		
	SalesGrowthByStore	2/day		
	SalesByTitle	5,000/day		
**	SalesByPub	10/day		
	SalesByZip	2,000/day		

**** indicates procedures where performance is critical**

Solutions

1. *The current mapping of procedures to tables to devices in pubtune is illustrated below:*



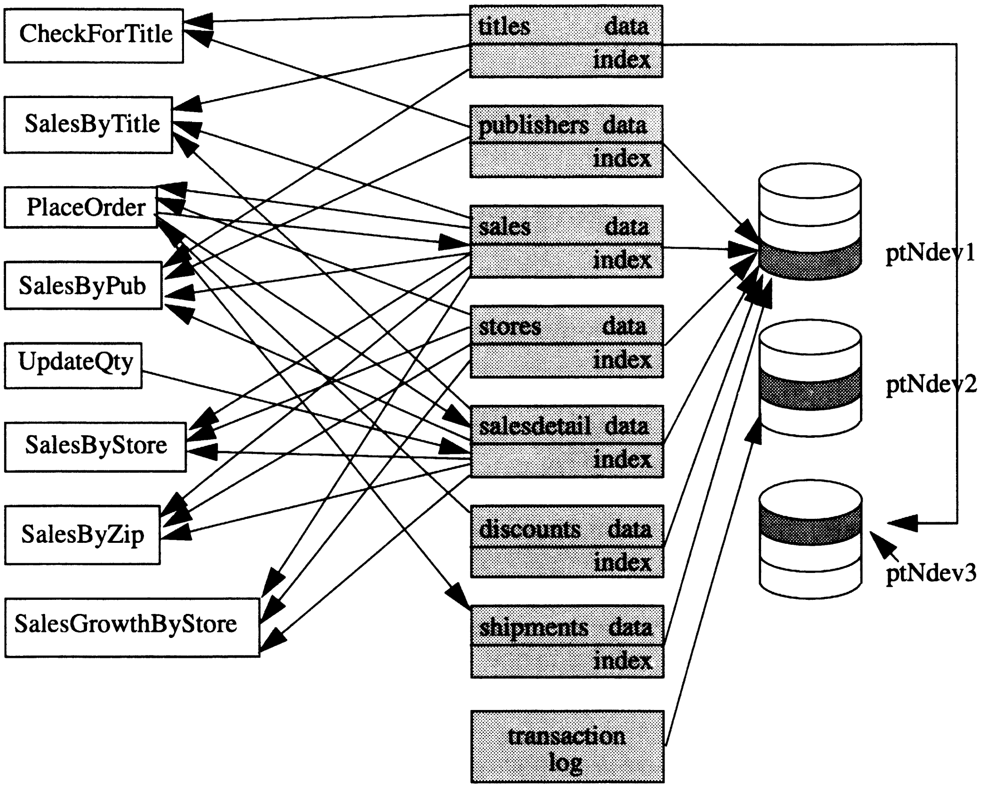
2. *The table includes average frequencies of execution for each procedure. Also, procedures with asterisks have been as critical. Determine a better object placement strategy that would improve the "CheckForTitle" transaction. (Assume you have multiple physical disks available.) Write down all the steps you would take to carry this out.*

	Procedure	Frequency	Baseline Response Time
**	CheckForTitle	20,000/day	
**	PlaceOrder	1,000/day	
	UpdateQty	1,500/day	
**	SalesByStore	50,000/day	
	SalesGrowthByStore	2/day	
	SalesByTitle	5,000/day	
**	SalesByPub	10/day	
	SalesByZip	2,000/day	

***** indicates procedures where performance is critical***

In order to tune the system for *CheckForTitle*, put the *titles* table on a separate physical disk, away from other heavily-accessed tables. Remember to execute `sp_recompile titles` or execute the procedures `with recompile`.

3. *Show the revised mapping in the following diagram:*



Optional:

4. *There are several other critical procedures we should tune for. What object placement strategies might you use to improve performance?*

   Isolate the other heavily-used tables and their non-clustered indexes as well, putting them on separate physical disks.

5. *If system resources permit, put your strategies to work. Measure baseline (before) and new (after) performance and fill in the columns of the following table. Has performance improved significantly?*

	Procedure	Frequency	Baseline Response Time	New Response Time
**	CheckForTitle	20,000/day		
**	PlaceOrder	1,000/day		
	UpdateQty	1,500/day		
**	SalesByStore	50,000/day		
	SalesGrowthByStore	2/day		
	SalesByTitle	5,000/day		
**	SalesByPub	10/day		
	SalesByZip	2,000/day		

***** indicates procedures where performance is critical***

8

Lab 8 – Locking and Performance

(Student Guide, page 8-32)

Exercise Overview

Goals	• Run transactions which causes database locking
General Tasks	• Experiment with several database loacking scenarios • Measure the impact of multiuser access
Lab Setup	Work with a partner or work alone using several window sessions

Exercise Instructions... ➡

Detailed Instructions

1. *Database locking (scenario #1)*

   a. Begin a transaction

   b. Insert a row into titles_idpr

```
insert into titles_idpr
values
('T12345', 'This is a new title', 'cooking','P076',
$99.99, $450.00, 235, 'This is a fake book', getdate(), 1)
```

   c. In another window, start another transaction

   d. From that window, insert a row into titles_idpr (increase title_id by 1)

```
insert into titles_idpr
values
('T12346', 'This is another new title', 'cooking','P076',
$99.99, $450.00, 235, 'This is another fake book', getdate(), 1)
```

   e. Does locking occur?

   f. Identify the locking resource (using *sp_lock*)

   g. Roll back the transaction in the first window

   h. What happens to the second transaction?

   i. Roll back the second transaction

2. *Database locking (scenario #2)*

    a. In this exercise, start two transactions which will attempt to insert the *same* rows as above into table title_titlid, which has a clustered index on title and a non-clustered index on title_id. Begin a transaction in each window and try the same inserts on the titles_titlid table. Do you think it will lock? Explain your results.

    b. Roll back both transactions

3. *Measure the impact of multiuser access to the same data.*

    a. Execute the OrderEntryBatch procedure 2 times and note response time of last batch

```
declare @start datetime
select @start = getdate()
exec OrderEntryBatch
select Time=datediff(ms, @start, getdate())
```

    b. Record response time

Number of users	Total time (ms)
1 user	

    c. Re-enter the transaction in window #1 (Do *not* execute)

    d. Re-enter the transaction in window #2 (Do *not* execute)

    e. Execute both batches simultaneously

    f. Determine response times

g. Record worst response time

Number of users	Total time (ms)
2 users	

h. If possible, your instructor will ask all those on your Server to execute the OrderEntryBatch procedure simultaneously

i. Record response time

Number of users	Total time (ms)
>2 users	

Optional Exercises

Three students are needed for the following exercise, and they should be accessing the same database

Note:
Please read the whole exercise first., make sure you understand every step of the way!

1. *Determine locking (scenario #3)*
   a. Student 1 executes *OrderEntryBatch_TD* (a variant of OrderEntryBatch)
   b. Student 2 examines the locks that are held
   c. Student 3 executes *OrderEntryBatch* and records response time
   d. How long does it take for Student 3 to execute *OrderEntryBatch*

Number of users	Total time (ms)
2 users	

   e. Examine *OrderEntryBatch_TD* to determine the problem
   f. What is your analysis of the problem?

   OrderEntryBatch_TD demonstrates a poor programming technique--it holds a transaction longer than it should. Performance for other users suffers as a result.

Solutions

1. *Database locking (scenario #1)*

   a. Begin a transaction

```
1> begin tran
2> go
1>
```

   b. Insert a row into titles_idpr

```
1> insert into titles_idpr
2> values
3> ('T12345', 'Here is a new title', 'business','P076', $1200.00,
4> $500.00, 35, 'This is not a real book', getdate(), 0)
5> go
1>
```

   c. In another window, start another transaction

```
1> begin tran
2> go
1>
```

   d. From that window, insert a row into titles_idpr (increase title_id by 1)

```
1> insert into titles_idpr
2> values
3> ('T12346', 'This is another new title', 'cooking','P076',
4> $99.99, $450.00, 235, 'This is another fake book', getdate(), 1)
5> go
```

   e. Does locking occur?

   Yes

   f. Identify the locking resource

```
1> sp_lock
2> go
The class column will display the cursor name for locks associated
with a cursor
for the current user and the cursor id for other users.
 spid   locktype                    table_id    page        dbname
          class
 ------ ---------------------------- ----------- -----------
---------------
        ------------------------------
      1 Sh_intent                     384004399          0 master
         Non Cursor Lock
      1 Ex_intent                   1312007705          0 pubtune
         Non Cursor Lock
      1 Ex_page                     1312007705       1640 pubtune
```

```
          Non Cursor Lock
        1 Ex_page                    1312007705         1715 pubtune
          Non Cursor Lock
        1 Ex_page                    1312007705         1723 pubtune
          Non Cursor Lock
        1 Ex_page                    1312007705         1724 pubtune
          Non Cursor Lock
        1 Ex_page                    1312007705         3000 pubtune
          Non Cursor Lock
        1 Ex_page                    1312007705         3002 pubtune
          Non Cursor Lock
        1 Ex_page                    1312007705         3003 pubtune
          Non Cursor Lock
        1 Ex_page                    1312007705         3007 pubtune
          Non Cursor Lock
        1 Ex_page-blk                1312007705         2825 pubtune
          Non Cursor Lock
        5 Ex_intent                  1312007705            0 pubtune
          Non Cursor Lock

(12 rows affected, return status = 0)
1>
```

g. Roll back the transaction in the first window

```
1> rollback tran
2> go
1>
```

h. What happens to the second transaction?

It completes.

i. Roll back the second transaction

```
1> rollback tran
2> go
```

2. *Database locking (scenario #2)*

a. In this exercise, start two transactions which will attempt to insert the *same* rows as above into table title_titlid, which has a clustered index on title and a non-clustered index on title_id. Begin a transaction in each window and try the same inserts on the titles_titlid table. Do you think it will lock? Explain your results.

It locks--on the *index* pages, not the data page!

b. Roll back both transactions

3. *Measure the impact of multiuser access to the same data.*

   a. Execute the OrderEntryBatch procedure 2 times and record response time of last batch

```
1> declare @start datetime
2> select @start = getdate()
3> exec OrderEntryBatch
4> select Time=datediff(ms, @start, getdate())
5> go
(1 row affected)
 TitleID Title                  Publisher       Price   Available
 ------- ---------------------- --------------- -------- ---------
 T81002  RIP Version 2 Protocol McGraw-Hill        44.95         0

(1 row affected)
 TitleID Title                   Publisher       Price    Available
 ------- ----------------------- --------------- -------- ---------
 T63365  Assigned numbers        Bantam Books       58.95         0
 T62154  Assigned numbers        Howard W. Sams     44.95         1
 T6544   Assigned numbers        Wadsworth Publi    51.95         1
 T69002  Assigned numbers        O'Reilly & Asso     3.95         1

(4 rows affected)
 TitleID Title                   Publisher       Price    Available
 ------- ----------------------- --------------- -------- ---------
 T63002  Explaining the role of  John Wiley & So    74.95         0
 T64412  Explaining the role of  Prentice-Hall       4.95         1

(2 rows affected, return status = 0)
 Time
 -----------
        920

(1 row affected)
1>
```

   b. Record response time

Number of users	Total time (ms)
1 user	920 ms

   c. Re-enter the transaction in window #1 (Do *not* execute)

```
1> declare @start datetime
2> select @start = getdate()
3> exec OrderEntryBatch
4> select Time=datediff(ms, @start, getdate())
5>
```

d. Re-enter the transaction in window #2 (Do *not* execute)

```
1> declare @start datetime
2> select @start = getdate()
3> exec OrderEntryBatch
4> select Time=datediff(ms, @start, getdate())
5>
```

e. Execute both batches simultaneously

```
5> go

5> go
```

f. Determine response times

```
1> declare @start datetime
2> select @start = getdate()
3> exec OrderEntryBatch
4> select Time=datediff(ms, @start, getdate())
5> go
(1 row affected)
 TitleID Title                  Publisher       Price    Available
 ------- ---------------------- --------------- -------- ---------
 T81002  RIP Version 2 Protocol McGraw-Hill        44.95         0

(1 row affected)
 TitleID Title                  Publisher       Price    Available
 ------- ---------------------- --------------- -------- ---------
 T63365  Assigned numbers       Bantam Books       58.95         0
 T62154  Assigned numbers       Howard W. Sams     44.95         1
 T6544   Assigned numbers       Wadsworth Publi    51.95         1
 T69002  Assigned numbers       O'Reilly & Asso     3.95         1

(4 rows affected)
 TitleID Title                  Publisher       Price    Available
 ------- ---------------------- --------------- -------- ---------
 T63002  Explaining the role of John Wiley & So    74.95         0
 T64412  Explaining the role of Prentice-Hall       4.95         1

(2 rows affected, return status = 0)
 Time
 -----------
        1087

(1 row affected)
1>
1> declare @start datetime
2> select @start = getdate()
3> exec OrderEntryBatch
4> select Time=datediff(ms, @start, getdate())
5> go
(1 row affected)
 TitleID Title                  Publisher       Price    Available
 ------- ---------------------- --------------- -------- ---------
 T81002  RIP Version 2 Protocol McGraw-Hill       44.95         0

(1 row affected)
 TitleID Title                  Publisher       Price    Available
 ------- ---------------------- --------------- -------- ---------
 T63365  Assigned numbers       Bantam Books       58.95         0
 T62154  Assigned numbers       Howard W. Sams     44.95         1
 T6544   Assigned numbers       Wadsworth Publi    51.95         1
 T69002  Assigned numbers       O'Reilly & Asso     3.95         1

(4 rows affected)
 TitleID Title                  Publisher       Price    Available
 ------- ---------------------- --------------- -------- ---------
 T63002  Explaining the role of John Wiley & So    74.95         0
 T64412  Explaining the role of Prentice-Hall       4.95         1

(2 rows affected, return status = 0)
 Time
 -----------
        1250

(1 row affected)
1>
```

g. Record worst response time

Number of users	Total time (ms)
2 users	1250

h. If possible, your instructor will ask all those on your Server to execute the OrderEntryBatch procedure simultaneously

i. Record response time

Number of users	Total time (ms)
>2 users	

Optional Exercises: Solutions

Three students are needed for the following exercise, and they should be accessing the same database

Note:

Please read the whole exercise first., make sure you understand every step of the way!

1. *Determine locking (scenario #3)*

   a. Student 1 executes *OrderEntryBatch_TD* (a variant of OrderEntryBatch)

```
1> OrderEntryBatch_TD
2> go
(1 row affected)
 TitleID Title                     Publisher       Price   Available
 ------- --------------------- --------------- -------- ---------
 T81002  RIP Version 2 Protocol McGraw-Hill        44.95         0

(1 row affected)
 TitleID Title                     Publisher       Price    Available
 ------- ----------------------- --------------- -------- ---------
 T63365  Assigned numbers          Bantam Books       58.95         0
 T62154  Assigned numbers          Howard W. Sams     44.95         1
 T6544   Assigned numbers          Wadsworth Publi    51.95         1
 T69002  Assigned numbers          O'Reilly & Asso     3.95         1
```

   b. Student 2 examines the locks that are held

```
1> sp_lock
2> go
The class column will display the cursor name for locks associated
with a cursor for the current user and the cursor id for other
users.
 spid   locktype                    table_id    page        dbname
          class
 ------ --------------------------- ----------- -----------
---------------
        ------------------------------
      1 Ex_intent                     112003430           0 pubtune
          Non Cursor Lock
      1 Ex_page                       112003430         378 pubtune
          Non Cursor Lock
      1 Ex_intent                     144003544           0 pubtune
          Non Cursor Lock
      1 Ex_page                       144003544         389 pubtune
          Non Cursor Lock
      1 Ex_page                       144003544         390 pubtune
          Non Cursor Lock
      1 Ex_intent                     304004114           0 pubtune
```

```
         Non Cursor Lock
      1 Ex_page                           304004114        435 pubtune
         Non Cursor Lock
      5 Sh_intent                          384004399          0 master
         Non Cursor Lock

(8 rows affected, return status = 0)
```

c. Student 3 executes *OrderEntryBatch* and records response time

```
1> declare @start datetime
2> select @start = getdate()
3> exec OrderEntryBatch
4> select Time=datediff(ms, @start, getdate())
5> go
(1 row affected)
 TitleID Title                  Publisher       Price    Available
 ------- ---------------------- --------------- -------- ---------
 T81002  RIP Version 2 Protocol McGraw-Hill        44.95         0

(1 row affected)
 TitleID Title                   Publisher       Price    Available
 ------- ----------------------- --------------- -------- ---------
 T63365  Assigned numbers        Bantam Books       58.95          0
 T62154  Assigned numbers        Howard W. Sams     44.95          1
 T6544   Assigned numbers        Wadsworth Publi    51.95          1
 T69002  Assigned numbers        O'Reilly & Asso     3.95          1

(4 rows affected)
 TitleID Title                   Publisher       Price    Available
 ------- ----------------------- --------------- -------- ---------
 T63002  Explaining the role of  John Wiley & So    74.95          0
 T64412  Explaining the role of  Prentice-Hall       4.95          1

(2 rows affected, return status = 0)
 Time
 -----------
       28910

(1 row affected)
```

d. How long does it take for Student 3 to execute *OrderEntryBatch*

Number of users	Total time (ms)
2 users	28910 ms

e. Examine *OrderEntryBatch_TD* to determine the problem

```
1> sp_helptext OrderEntryBatch_TD
2> go
          text
----------------------------------------------------- ------------

create procedure OrderEntryBatch_TD
as
exec PlaceOrder 'S1'
exec CheckForTitle 'RIP%','adventure', '10/10/1979',0.00,100.00
exec PlaceOrder 'S7'
exec CheckForTitle 'Assigned%','computer','10/10/1979',0.0,200.00
exec PlaceOrder 'S30'
exec PlaceOrder 'S60'
exec PlaceOrder 'S904'
exec PlaceOrder 'S802'
exec PlaceOrder 'S407'
exec CheckForTitle
'%Explaining%','computer','10/10/1979',0.0,1000.00
begin transaction
exec PlaceOrder 'S304'
waitfor delay "00:00:30" /* wait for 30 seconds to hold resource */
commit transaction
exec PlaceOrder 'S103'

(3 rows affected, return status = 0)
1>
```

f. What is your analysis of the problem?

OrderEntryBatch_TD demonstrates a poor programming technique--it holds a transaction longer than it should. Performance for other users suffers as a result.

Lab 9 – Memory and Performance

(Student Guide, page 9-35)

Exercise Overview

Goals

- Evaluate the effectiveness of memory allocation
- Calculate memory requirements

General Tasks

- Determine cache hit ratio on *OrderEntryBatch* procedure
- Determine data cache requirements given the table and andex sizes
- Determine memory allocation for SQL Server

Exercise Instructions...

Detailed Instructions

In this lab you will evaluate the effectiveness of the initial memory allocation and experiment with changing memory allocation.

1. Determine cache hit ratio on OrderEntryBatch procedure
   a. Execute the OrderEntryBatch procedure.
   b. Execute dbcc traceon
   c. Execute dbcc tablealloc() on publishers
   d. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
publishers	100 %

   e. Execute dbcc tablealloc() on *titles*
   f. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
titles	43 of 625

   g. Execute dbcc tablealloc() on *salesdetail*
   h. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
salesdetail	100 %

i. Execute dbcc tablealloc() on *discounts*

j. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
discounts	100%

k. Execute dbcc tablealloc() on *sales*

l. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
sales	

m. Execute dbcc tablealloc() on *stores*

n. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
stores	

o. Execute dbcc tablealloc() on *shipments*

p. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
shipments	

q. Calculate the overall cache hit ratio on all tables

OrderEntryBatch Cache Hit Ratios

Table	Overall Cache Hit Ratio
ALL TABLES	

r. Comment, Is the cache hit ratio acceptable? If not, what could you do?

2. Determine data cache requirements given the following table and index sizes.

Size of Tables & Indexes

Tables & Indexes	Size (Mb)
authors	1.3
titles	1.2
stores	.46
publishers	.04
salesdetail	1.14
discounts	.06

Size of Tables & Indexes

Tables & Indexes	Size (Mb)
sales	.40
shipments	.48

a. How much space will you need in data cache to put all objects in cache?

3. Determine memory allocation for SQL Server

   a. Create sql script at the OS level

   Contents of script:

```
dbcc traceon(3604)
go
dbcc memusage
go
dbcc traceoff(3604)
go
```

   b. Run it, redirecting output to a file

   c. Examine memory allocation

   d. fill in the table below

Memusage Results

	Size (Mb)
Configured Memory	10
SQL Server Executable	2,7
Kernel Structures	1,6
Server Structures	2,9
Data Cache	2,2
Procedure Cache	

Optional Execises

4. The executable is unchangeable, and for the purpose of this exercise, let us assume the kernel and server structures fulfill our requirements nicely and do not need to be altered. If we assume, in addition, that we need 3.0 Mb for procedure cache and use our data cache requirement derived in an earlier exercise, how much memory do we need altogether? Fill in the appropriate boxes in the table below.

Memory Requirements

	Size (Mb)
SQL Server Executable	
Kernel Structures	
Server Structures	
Data Cache	
Procedure Cache	3.0
MEMORY REQUIREMENTS	

Solutions

1. *Determine cache hit ratio on OrderEntryBatch procedure*

a. Execute the OrderEntryBatch procedure.

```
1> exec OrderEntryBatch
2> go
 TitleID Title                  Publisher        Price    Available
 ------- ---------------------- --------------- -------- ---------
 T81002  RIP Version 2 Protocol McGraw-Hill        44.95         0

(1 row affected)
 TitleID Title                  Publisher        Price    Available
 ------- ---------------------- --------------- -------- ---------
 T63365  Assigned numbers       Bantam Books       58.95         0
 T62154  Assigned numbers       Howard W. Sams     44.95         1
 T6544   Assigned numbers       Wadsworth Publi    51.95         1
 T69002  Assigned numbers       O'Reilly & Asso     3.95         1

(4 rows affected)
 TitleID Title                  Publisher        Price    Available
 ------- ---------------------- --------------- -------- ---------
 T63002  Explaining the role of John Wiley & So    74.95         0
 T64412  Explaining the role of Prentice-Hall       4.95         1

(2 rows affected, return status = 0)
```

b. Execute dbcc traceon.

```
1> dbcc traceon(3604)
2> go
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

c. Execute dbcc tablealloc() on *publishers*

```
1> dbcc tablealloc(publishers)
2> go
The default report option of OPTIMIZED is used for this run.
The default fix option of FIX   is used for this run.
***************************************************************
TABLE: publishers               OBJID = 48003202
INDID=0  FIRST=313       ROOT=314       SORT=0
        Data level: 0.  2 Data  Pages in 1 extents.
TOTAL # of extents = 1
Alloc page 256 (# of extent=1 used pages=3 ref pages=3)
Total (# of extent=1 used pages=3 ref pages=3) in this database

 Statistical information for this run follows:
Total # of pages read = 3
Total # of pages found cache = 3
Total # of physical reads = 0
Total # of saved I/O = 0
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

d. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
publishers	3/(3+0) =1 = 100%

e. Execute dbcc tablealloc() on *titles*

```
1> dbcc tablealloc(titles)
2> go
The default report option of OPTIMIZED is used for this run.
The default fix option of FIX   is used for this run.
****************************************************************
TABLE: titles          OBJID = 208003772
INDID=0  FIRST=409      ROOT=1424       SORT=0
        Data level: 0.  624 Data  Pages in 79 extents.
TOTAL # of extents = 79
Alloc page 256 (# of extent=1 used pages=8 ref pages=8)
Alloc page 768 (# of extent=29 used pages=232 ref pages=232)
Alloc page 1024 (# of extent=31 used pages=248 ref pages=248)
Alloc page 1280 (# of extent=18 used pages=137 ref pages=137)
Total (# of extent=79 used pages=625 ref pages=625) in this
database

 Statistical information for this run follows:
Total # of pages read = 625
Total # of pages found cache = 411
Total # of physical reads = 214
Total # of saved I/O = 0
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

Note that in this as in all measurements of I/O, your values may be quite different from those we recorded during our test run.

f. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
titles	411/(411+214)=0.66 = 66%

g. Execute dbcc tablealloc() on *salesdetail*

```
1> dbcc tablealloc(salesdetail)
2> go
The default report option of OPTIMIZED is used for this run.
The default fix option of FIX   is used for this run.
***************************************************************
TABLE: salesdetail              OBJID = 144003544
INDID=0  FIRST=385       ROOT=386        SORT=0
        Data level: 0.  2 Data  Pages in 1 extents.
TOTAL # of extents = 1
Alloc page 256 (# of extent=1 used pages=3 ref pages=3)
Total (# of extent=1 used pages=3 ref pages=3) in this database

 Statistical information for this run follows:
Total # of pages read = 3
Total # of pages found cache = 3
Total # of physical reads = 0
Total # of saved I/O = 0
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

h. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
salesdetail	3/(3+0) =1 = 100%

i. Execute dbcc tablealloc() on *discounts*

```
1> dbcc tablealloc(discounts)
2> go
The default report option of OPTIMIZED is used for this run.
The default fix option of FIX   is used for this run.
***************************************************************
TABLE: discounts                OBJID = 272004000
```

```
INDID=0  FIRST=425       ROOT=427        SORT=0
        Data level: 0.  3 Data  Pages in 1 extents.
TOTAL # of extents = 1
Alloc page 256 (# of extent=1 used pages=4 ref pages=4)
Total (# of extent=1 used pages=4 ref pages=4) in this database

 Statistical information for this run follows:
Total # of pages read = 4
Total # of pages found cache = 1
Total # of physical reads = 3
Total # of saved I/O = 0
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

j. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
discounts	1/(1+3)=0.25 = 25%

k. Execute dbcc tablealloc() on *sales*

```
1> dbcc tablealloc(sales)
2> go
The default report option of OPTIMIZED is used for this run.
The default fix option of FIX   is used for this run.
***************************************************************
TABLE: sales             OBJID = 112003430
INDID=0  FIRST=377       ROOT=377        SORT=0
        Data level: 0.  1 Data  Pages in 1 extents.
TOTAL # of extents = 1
Alloc page 256 (# of extent=1 used pages=2 ref pages=2)
Total (# of extent=1 used pages=2 ref pages=2) in this database

 Statistical information for this run follows:
Total # of pages read = 2
Total # of pages found cache = 2
Total # of physical reads = 0
Total # of saved I/O = 0
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

l. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
sales	2/(2+0) =1 = 100%

m. Then execute dbcc tablealloc() on *stores*

```
1> dbcc tablealloc(stores)
2> go
The default report option of OPTIMIZED is used for this run.
The default fix option of FIX   is used for this run.
****************************************************************
TABLE: stores           OBJID = 240003886
INDID=0  FIRST=417       ROOT=1447        SORT=0
        Data level: 0.  23 Data  Pages in 3 extents.
TOTAL # of extents = 3
Alloc page 256 (# of extent=1 used pages=8 ref pages=8)
Alloc page 1280 (# of extent=2 used pages=16 ref pages=16)
Total (# of extent=3 used pages=24 ref pages=24) in this database

 Statistical information for this run follows:
Total # of pages read = 24
Total # of pages found cache = 13
Total # of physical reads = 11
Total # of saved I/O = 0
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

n. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
stores	13/(13+11)=0.54 = 54%

o. Execute dbcc tablealloc() on *shipments*

```
1> dbcc tablealloc(shipments)
2> go
The default report option of OPTIMIZED is used for this run.
The default fix option of FIX   is used for this run.
****************************************************************
```

```
TABLE: shipments                OBJID = 304004114
INDID=0  FIRST=433       ROOT=433        SORT=0
        Data level: 0.  1 Data  Pages in 1 extents.
TOTAL # of extents = 1
Alloc page 256 (# of extent=1 used pages=2 ref pages=2)
Total (# of extent=1 used pages=2 ref pages=2) in this database

 Statistical information for this run follows:
Total # of pages read = 2
Total # of pages found cache = 2
Total # of physical reads = 0
Total # of saved I/O = 0
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

p. Determine cache hit ratio and document results

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
shipments	2/(2+0) =1 = 100%

q. Calculate the overall cache hit ratio on all tables

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
publishers	100%
titles	66%
salesdetail	100%
discounts	25%
sales	100%
stores	54%
shipments	100%

OrderEntryBatch Cache Hit Ratios

Table	Cache Hit Ratio
OVERALL CACHE HIT RATE:	77%

r. Comment on these. Is the cache hit ratio acceptable? If not, what could you do?

(Results will vary depending on the number of users.) A 77% cache hit rate seems reasonable as a general guideline.

2. *Determine data cache requirements given the following table and index sizes.*

Size of Tables & Indexes

Tables & Indexes	Size (Mb)
authors	1.3
titles	1.2
stores	.46
publishers	.04
salesdetail	1.14
discounts	.06
sales	.40
shipments	.48

a. How much space will you need in data cache to put all objects in cache?

5.08 Mb.

3. *Determine memory allocation for SQL Server*

a. Create sql script at the OS level

```
edeme2% vi runmemusage
```

Contents of script:

```
dbcc traceon(3604)
go
dbcc memusage
go
dbcc traceoff(3604)
go

:wq
```

b. Run it, redirecting output to a file

```
edeme2% isql -Usa -Pheydude <runmemusage >memusage.out
```

c. Examine memory allocation

```
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
Memory Usage:


                                 Meg.        2K Blks           Bytes

      Configured Memory: 10.0000            5120        10485760

              Code size:  2.4977            1279         2619068
      Kernel Structures:  1.5459             792         1620986
      Server Structures:  2.8804            1475         3020336
             Page Cache:  2.4359            1248         2554184
           Proc Buffers:  0.0215              12           22572
           Proc Headers:  0.4434             227          464896

Number of page buffers:     1186
Number of proc buffers:      296
```

d. fill in the table below.

Memusage Results

	Size (Mb)
Configured Memory	10
SQL Server Executable	2.4977
Kernel Structures	1.5459
Server Structures	2.8804
Data Cache	2.4359
Procedure Cache	0.4649

Optional Exercises: Solutions

4. *The executable is unchangeable, and for the purpose of this exercise, let us assume the kernel and server structures fulfill our requirements nicely and do not need to be altered. If we assume, in addition, that we need 3.0 Mb for procedure cache and use our data cache requirement derived in an earlier exercise, how much memory do we need altogether? Fill in the appropriate boxes in the table below.*

Memory Requirements

	Size (Mb)
SQL Server Executable	2.48
Kernel & Server Structures	2.36
Data Cache	5.08
Procedure Cache	3.0
MEMORY REQUIREMENTS	12.92

Lab 10 – Maintenance and Performance

(Student Guide, page 10-18)

Exercise Overview

Goals

- Measure the performance of bulk copy
- Measure performance of create index under various conditions
- Compare performance of bulk copy load on indexed vs. non-indexed table

General Tasks

- Copy the *authors* table to an O/S file using bcp
- Record the transfer time as displayed by bcp
- Determine bcp performance on heap table
- Determine index creation time on the *authors* table
- Determine bcp transfer time on indexed table
- Compare Load in heap table + index creation time vs. load on indexed table
- Which is faster?

Lab Setup

Consult the Command Reference Guide (Utilities) on syntax for bcp

continued on next page...

Detailed Instructions

In this lab, you will measure the performance of bulk copy and create index under various conditions.

1. *Copy the authors table to an O/S file using bcp. Record the transfer time as displayed by bcp.*

2. *Determine bcp performance on heap table*
   a. Truncate the authors table
   b. Reload the table using bcp
   c. Record transfer time

BCP Performance

Table Structure	Transfer Time
Authors (heap)	6 sec.

3. *Determine index creation time on the authors table*
   a. Create a clustered index on au_lname and au_fname
   b. Record creation time of clustered index

Index Creation Performance

Index	Creation Time
Clustered	2,3 sec.

   c. Create a non-clustered index on au_id
   d. Record creation time of non-clustered index

Index Creation Performance

Index	Creation Time
Non Clustered	1,6

4. *Determine bcp transfer time on indexed table*

   a. Truncate the authors table

   b. Reload it using the same bcp command as above

   c. Record load time

BCP Performance

Table Structure	Transfer Time
Authors (indexed)	11 sec.

5. *Compare Load in heap table + index creation time vs. load on indexed table*

BCP Performance

Type of load	Transfer Time
Load & Create Indexes	9.9 sec
Load into indexed table	11 sec

6. *Compare the load time. Which is faster?*

   It is much faster to load the table, then create indexes.

Solutions

1. *Copy the authors table to an O/S file using bcp. Record the transfer time as displayed by bcp.*

```
zappa% bcp pubtune..authors out authors.out -c -Usa -Pheydude -b100

Starting copy...
1000 rows successfully bulk-copied to host-file.
1000 rows successfully bulk-copied to host-file.
1000 rows successfully bulk-copied to host-file.
1000 rows successfully bulk-copied to host-file.
1000 rows successfully bulk-copied to host-file.

5000 rows copied.
Clock Time (ms.): total = 6487   Avg = 1     (770.77 rows per sec.)
zappa%
```

2. *Determine bcp performance on heap table*

   a. Truncate the authors table

```
1> use pubtune
2> go
1> truncate table authors
2> go
1>
```

   b. Reload the table using bcp

```
zappa% bcp pubtune..authors in authors.out -c -Usa -Pheydude -b100

Starting copy...
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.

5000 rows copied.
Clock Time (ms.): total = 6804   Avg = 1     (734.86 rows per sec.)
```

c. Record transfer time

BCP Performance

Table Structure	Transfer Time
Authors (heap)	6.804 sec

3. *Determine index creation time on the authors table*

a. Create a clustered index on au_lname and au_fname

Example:

```
1> declare @start datetime
2> select @start = getdate()
3> create clustered index idx1 on authors(au_lname, au_fname)
4> select Time = datediff(ms, @start, getdate())
5> go
(1 row affected)
 Time
 -----------
        4623

(1 row affected)
1>
```

b. Record creation time of clustered index

Index Creation Performance

Index	Creation Time
Clustered	4.623 sec

c. Create a non-clustered index on au_id

Example:

```
1> declare @start datetime
2> select @start = getdate()
3> create index idx2 on authors(au_id)
4> select Time = datediff(ms, @start, getdate())
5> go
(1 row affected)
 Time
 -----------
        1473
```

```
(1 row affected)
1>
```

d. Record creation time of non-clustered index

Index Creation Performance

Index	Creation Time
Non Clustered	1.473 sec

4. *Determine bcp transfer time on indexed table*

a. Truncate the authors table

```
1> truncate table authors
2> go
1>
```

b. Reload it using the same bcp command as above

```
zappa% bcp pubtune..authors in authors.out -c -Usa -Pheydude -b100

Starting copy...
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.
Batch successfully bulk-copied to SQL Server.

5000 rows copied.
Clock Time (ms.): total = 23502  Avg = 4     (212.75 rows per sec.)
```

c. Record load time

BCP Performance

Table Structure	Transfer Time
Authors (indexed)	23.502 sec

5. *Compare Load in heap table + index creation time vs. load on indexed table*

BCP Performance

Type of load	Transfer Time
Load & Create Indexes	6.804+4.623+1.473= **12.9** sec
Load into indexed table	**23.502** sec

6. *Compare the load time. Which is faster?*

   It is much faster to load the table, then create indexes.

Lab 11 – Designing Applications for Performance

11

(Student Guide, page 11-20)

Exercise Overview

Goals

- Determine which system issues would need to be considered for a given application

General Tasks

- Requirements and constraints (you may assume these are the only ones):
  - *CheckForTitle* is executed by sales clerks to print all of the titles that have a given
    keyword or phrase somewhere in the title.
  - *Stores* are open every day of the year, from 9am to 9pm.
  - *CheckForTitle* may be executed 24 hours per day, 365 days per year by customers who
    dial-in using the toll-free modem line.
  - On average, each store places approx. 500 new orders per working day.
  - Approx. 10 percent of all orders are updated at least once prior to being shipped.
  - Approx. 30 percent of all updated orders are updated more than 4 times.
  - Only the store manager can execute the UpdateQty transaction.
  - Approx. 50 users will be running the application at any one time.
  - This application is critical. No data can be lost.

Lab Setup

- There are no "right" answers--choose an answer you can defend

Exercise Instructions... ➡

Lab 11 – Designing Applications for Performance: Exercise Overview

Detailed Instructions

In this lab, you will determine which system issues would need to be considered for a given application.

1. For this exercise, assume that the Order Entry application is the only one running on the Server. Given the following requirements and constraints for that application, decide which of the items that follow are likely to be issues for consideration. (There are no "right" answers--choose an answer you can defend.)
   Requirements and constraints (you may assume these are the only ones):
   a. CheckForTitle is executed by sales clerks to print all of the titles that have a given keyword or phrase somewhere in the title.
   b. Stores are open every day of the year, from 9am to 9pm.
   c. CheckForTitle may be executed 24 hours per day, 365 days per year by customers who dial-in using the toll-free modem line.
   d. On average, each store places approx. 500 new orders per working day.
   e. Approx. 10 percent of all orders are updated at least once prior to being shipped.
   f. Approx. 30 percent of all updated orders are updated more than 4 times.
   g. Only the store manager can execute the UpdateQty transaction.
   h. Approx. 50 users will be running the application at any one time.
   i. This application is critical. No data can be lost.

Extensive user base?	[✓ Y]/N
Heavy network traffic?	Y/[✓ N]
Heavy demands on hardware?	Y/[✓ N]
Server-to-Server communication?	Y/[✓ N]
Big procedure cache needed?	[✓ Y]/[✓ N]
Auditing needed?	Y/[✓ N]
Mirroring required?	[✓ Y]/N
Transaction log separate from data?	[✓ Y]/N
Locking likely?	[✓ Y]/N

Need to isolate tables from indexes?	[✓ Y]/N
Referential integrity implemented using triggers?	[✓ Y]/N
Transaction size an issue?	[✓ Y]/N
A lot of sorting?	Y/[✓ N]
Need for stored procedures?	[✓ Y]/N
Need to consider fillfactor?	[✓ Y]/N
Clustered index on order number?	Y/[✓ N]
Need for different levels of security?	[✓ Y]/N
Validate data for application-specific columns on Server?	Y/N
Should some transactions be done during off-hours?	Y/N

Solutions

1. *For this exercise, assume that the Order Entry application is the only one running on the Server. Given the following requirements and constraints for that application, decide which of the items that follow are likely to be issues for consideration. (There are no "right" answers--choose an answer you can defend.)*
   Requirements and constraints (you may assume these are the only ones):
   a. CheckForTitle is executed by sales clerks to print all of the titles that have a given keyword or phrase somewhere in the title.
   b. Stores are open every day of the year, from 9am to 9pm.
   c. CheckForTitle may be executed 24 hours per day, 365 days per year by customers who dial-in using the toll-free modem line.
   d. On average, each store places approx. 500 new orders per working day.
   e. Approx. 10 percent of all orders are updated at least once prior to being shipped.
   f. Approx. 30 percent of all updated orders are updated more than 4 times.
   g. Only the store manager can execute the UpdateQty transaction.
   h. Approx. 50 users will be running the application at any one time.
   i. This application is critical. No data can be lost.

   Sample answers to questions posed:

   Extensive user base? No. We are told there are 50 users expected at any one time. This is not considered extensive.

   Heavy network traffic? No. This is the only application running on the Server, and 500 new orders per day is not a heavy load.

   Heavy demands on hardware? No, because of the limited user base and the fact that this is the only application running on the Server.

   Server-to-Server communication? No, none indicated. Application appears to access just one Server.

   Big procedure cache needed? Yes, depending on the implementation. (In other words, this needs to be evaluated.) If the processes use stored procedures, you will want to consider enlarging the procedure cache.

   Auditing needed? Yes, possibly, because the application is so critical.

   Mirroring required? Yes, definitely, since the application is critical.

   Transaction log separate from data? Yes, this is a general recommendation for all implementations.

   Locking likely? Yes, because of the nature of order entry: new orders frequently inserted at the same point at the same time. Also, more than one table may be held by a single transaction.

   Need to isolate tables from indexes? Yes, always a good idea for critical tables, to speed access to both tables and indexes.

Referential integrity implemented using triggers? Yes, this is the best approach in the order entry world.

Transaction size an issue? Yes, potentially. The UpdateQuantity transaction is a good candidate--you will want to modularize that to the extent possible.

A lot of sorting? No, none evident. There is no reporting going on.

Need for stored procedures? Yes, this is always a good idea.

Need to consider fillfactor? Yes, especially since inserts are frequent. You want to minimize the need for page splits. However, you will need to consider the fact that CheckForTitle is performed 7days a week, 24 hours a day. You may initially create the index with a low fill factor, but pages will fill up as time goes by. You will need to take the system off-line to rebuild indexes and re-establish low fill factor.

Clustered index on order number? No, probably not, because that will cause a "hot spot" that will slow things down.

Need for different levels of security? Yes. Store managers perform certain tasks that others do not, and they need update access to tables or columns that others do not need access to.

Validate data for application-specific columns on Server? No, this is probably best done by the application to avoid network traffic.

Should some transactions be done during off-hours? No, there are no obvious candidates. But keep your eyes open!

Lab 12 – Networks and Performance

(Student Guide, page 12-34)

Exercise Overview

Goals

- Measure the number of packets sent and received for the script vs. the stored procedure
- Experiment with different packet sizes

General Tasks

- Compare response time of a sql script to that of a stored procedure
  - Execute this script three times
  - Note response time of the last two executions
  - Average the results
  - Execute the stored procedure SalesByZip three times
  - Average the results of the last two executions
- Compare the network packets send of sql script vs. stored procedure
- Compare the performance of a script with varying network packet sizes
- Compare performance of *PrintAuthors* procedure using various packet sizes

Lab Setup

Exercises (2,3,4) should be performed as a demo by the instructor only as *sp_monitor* uses globally updated variables

Exercise Instructions... ➡

Detailed Instructions

1. Compare response time of a sql script to that of a stored procedure, as follows:

   a. Read the SQL script called SalesBZ.sql into the isql buffer. (It includes the getdate() statements that measure elapsed time.)

```
/* This script prints a report that lists the sales for each zip
# It also lists the city and state for each zip code.  It does an
# outer join to ensure that all zip codes are listed, not just the
# ones that had sales.  The sort is by zip code.
*/
declare @start datetime
select @start=getdate()
select  stores.zipcode,
        qty = sum(qty)
into    #zip_quan
from    stores, salesdetail, sales
where   stores.stor_id = salesdetail.stor_id
and     salesdetail.ord_num = sales.ord_num
and     salesdetail.stor_id = sales.stor_id
and     sales.date > dateadd(month, -1, getdate())
group by stores.zipcode


select distinct
        "Zip Code" = stores.zipcode,
        "City" = stores.city,
        "State" = stores.state,
        "Total Books Sold" = isnull(qty, 0)
from    #zip_quan, stores
where   stores.zipcode *= #zip_quan.zipcode
order by stores.zipcode

drop table #zip_quan
select datediff(ms,@start,getdate())
```

   b. Execute this script three times

   c. Note response time of the last two executions

   433 436 426

   d. Average the results

   431

e. Document the results in the table below:

Script vs. Procedure Response Time

	Second	Third	Average
SalesBZ.sql Script			

f. Execute the stored procedure *SalesByZip* three times, (discarding your first results, which include time to compile the procedure and generate a query plan.)

```
1> declare @start datetime
2> select @start=getdate()
3> exec SalesByZip
4> select datediff(ms,@start,getdate())
5> go
```

g. Average the results of the last two executions

543 840 430 483

h. Add the results to the table below: 476 48ø 426 426

Script vs. Procedure Response Time

446

	Second	Third	Average
SalesByZip Procedure			

Goal of following exercises is to measure the number of packets sent and received for the script vs. the stored procedure

***Note*: The following exercises (2,3,4) should be performed as a demo by the instructor only as sp_monitor uses globally updated variables.**

2. Determine baseline network packets

   a. Execute sp_monitor twice, and record the packets sent and received as reported by the second execution of the command.

      (This reports on how many packets were sent and received by the previous sp_monitor command.)

b. Record these in the first row of the following table

This will be your baseline--to be subtracted from future *sp_monitor* result sets

Number of Packets Sent & Received

	Packets Received	Packets Sent
Baseline (sp_monitor)		

3. Determine the number of packets send by the SalesBZ script
   c. Read the *SalesBZ.sql* script into the isql buffer, and execute it
   d. Execute *sp_monitor* to measure the packets sent and received by this script
   e. Subtract the baseline figures
   f. Record your results in the table

Number of Packets Sent & Received

	Packets Received	Packets Sent
Baseline (sp_monitor)		
Script		

4. Determine the number of packets send by SalesByZip procedure
   a. Execute the *SalesByZip* stored procedure
   b. Execute *sp_monitor* to measure packets sent and received by the procedure
   c. Subtract the baseline figures
   d. record your results in the table

Number of Packets Sent & Received

	Packets Received	Packets Sent
Baseline (sp_monitor)		
SalesByZip		

e. What results did you get? Were packets sent different? Were packets received different? Why?

f. Look at the response time results you recorded in the first exercise. Do packets sent or received appear to be related to overall response time?

5. Compare the performance of a script with varying network packet sizes
   a. Run sp_configure
   b. Check the runtime values for "default network packet size"
   c. Determine configuration values for "maximum network packet size"
   d. Determine configuration values for "additional netmem"
   e. note the values below:

Network Configuration Values

Configuration variable	Runtime values
default network packet size	
maximum network packet size	
additional netmem	

These values define the extent to which you can vary network packet size for your session.

6. Execute *PrintAuthors* procedure using various packet sizes

The PrintAuthors stored procedure displays the names of all authors and their blurbs. Adjust network packet size by specifying the desired size as an input parameter to isql. Use the -A flag to vary network packet size, and be sure to stay within the limits of your system as recorded above. Use the -p flag to ensure that statistics are recorded, or use the getdate() statements we have been using to measure response time. Record number of packets sent and response time below. (Some network packet sizes have been provided as suggestions.)

```
isql -Uusername -Ppassword -Apacketsize -p
```

Example:

```
isql -Uuser33 -P -A1024 -p
```

Packet Size & Performance

Packet Size	# of Packets Sent	Response Time
512		
1024		
2048		
4096		
8192		

a. Comment on your results.

Solutions

1. Compare response time of a sql script to that of a stored procedure, as follows:

a. Read the SQL script called SalesBZ.sql into the isql buffer. (It includes the getdate() statements that measure elapsed time.)

```
1> :r SalesBZ.sql
31> vi
/* This script prints a report that lists the sales for each zip
# It also lists the city and state for each zip code.  It does an
# outer join to ensure that all zip codes are listed, not just the
# ones that had sales.  The sort is by zip code.
*/
declare @start datetime
select @start=getdate()
select  stores.zipcode,
        qty = sum(qty)
into    #zip_quan
from    stores, salesdetail, sales
where   stores.stor_id = salesdetail.stor_id
and     salesdetail.ord_num = sales.ord_num
and     salesdetail.stor_id = sales.stor_id
and     sales.date > dateadd(month, -1, getdate())
group by stores.zipcode


select distinct
        "Zip Code" = stores.zipcode,
        "City" = stores.city,
        "State" = stores.state,
        "Total Books Sold" = isnull(qty, 0)
from    #zip_quan, stores
where   stores.zipcode *= #zip_quan.zipcode
order by stores.zipcode

drop table #zip_quan
select datediff(ms,@start,getdate())
```

b. Execute this script three times,

```
1> go
.
.
.
 98352    Mc Millin           WA                    0
 98354    Milton              WA                    0
 98455    Tacoma              WA                    0
 98816    Chelan              WA                    0
 98930    Grandview           WA                    0
 99029    Reardan                    WA                     0
 99138    Inchelium                  WA                     0
 99149    Malden                     WA                     0
 99356    Roosevelt                  WA                     0
 99625    Levelock                   AK                     0

(496 rows affected)

 -----------
        1696
1> :r SalesBZ.sql
31> go
.
.
.
 98930    Grandview                  WA                     0
 99029    Reardan                    WA                     0
 99138    Inchelium                  WA                     0
 99149    Malden                     WA                     0
 99356    Roosevelt                  WA                     0
 99625    Levelock                   AK                     0

(496 rows affected)

 -----------
        1690

(1 row affected)
1>
1> :r SalesBZ.sql
31> go
 98816    Chelan                     WA                     0
 98930    Grandview                  WA                     0
 99029    Reardan                    WA                     0
 99138    Inchelium                  WA                     0
 99149    Malden                     WA                     0
 99356    Roosevelt                  WA                     0
 99625    Levelock                   AK                     0

(496 rows affected)

 -----------
        1510

(1 row affected)
1>
```

c. Note response time of the last two executions

1690+1510 ms

d. Average the results

1690+1510=3200/2=1600 ms

e. Document the results in the table below:

Script vs. Procedure Response Time

	Second	Third	Average
SalesBZ.sql Script	1690	1510	1600

f. Execute the stored procedure *SalesByZip* three times, (discarding your first results, which include time to compile the procedure and generate a query plan.)

```
1> declare @start datetime
2> select @start=getdate()
3> exec SalesByZip
4> select datediff(ms,@start,getdate())
5> go
(1 row affected)
.
.
 99138    Inchelium            WA                  0
 99149    Malden               WA                  0
 99356    Roosevelt            WA                  0
 99625    Levelock             AK                  0

(496 rows affected, return status = 0)

 -----------
        1463
```

```
1> declare @start datetime
2> select @start=getdate()
3> exec SalesByZip
4> select datediff(ms,@start,getdate())
5> go
(1 row affected)

 99138    Inchelium            WA                  0
 99149    Malden               WA                  0
 99356    Roosevelt            WA                  0
 99625    Levelock             AK                  0

(496 rows affected, return status = 0)

 -----------
```

```
        1386

(1 row affected)
```

g. Average the results of the last two executions

   1463+1386=2849

h. Add the results to the table below:

Script vs. Procedure Response Time

	Second	Third	Average
SalesBZ.sql Script	1690	1510	1600 ms
SalesByZip Procedure	1463	1386	1424 ms

Goal of following exercises is to measure the number of packets sent and received for the script vs. the stored procedure

***Note*: The following exercises (2,3,4) should be performed as a demo by the instructor only as sp_monitor uses globally updated variables.**

2. *Determine baseline network packets*

   a. Execute sp_monitor twice, and record the packets sent and received as reported by the second execution of the command.

   (This reports on how many packets were sent and received by the previous sp_monitor command.) You should get the following results: 1 packet received, 2 packets sent.

```
1> sp_monitor
2> go
 last_run                    current_run                    seconds
 -------------------------- -------------------------- -----------
       Sep  2 1994 12:49PM        Sep  2 1994 12:49PM            5

 cpu_busy                   io_busy                    idle
 -------------------------- -------------------------- -------------
 265(0)-0%                  0(0)-0%                    83792(5)-100%

 packets_received           packets_sent               packet_errors
 -------------------------- -------------------------- -------------
 1285(1)                    804(2)                     1(0)
 total_read          total_write         total_errors        conn
 ------------------- ------------------- ------------------- -----
 1479(1)             77324(2)            0(0)                8(0)

(return status = 0)
```

b. Record these in the first row of the following table.

This will be your baseline--to be subtracted from future *sp_monitor* result sets

Number of Packets Sent & Received

	Packets Received	Packets Sent
Baseline (sp_monitor)	1	2

3. *Determine the number of packets send by the SalesBZ script*

a) Read the *SalesBZ.sql* script into the isql buffer, and execute it

```
1> :r SalesBZ.sql
31> go
(1 row affected)
.
.
 99149    Malden              WA                     0
 99356    Roosevelt           WA                     0
 99625    Levelock            AK                     0

(496 rows affected, return status = 0)

 -----------
        1867
```

b) Execute *sp_monitor* to measure the packets sent and received by this script

```
1> sp_monitor
2> go
 last_run                   current_run                 seconds
 -------------------------- --------------------------- -----------
        Sep  2 1994  1:49PM        Sep  2 1994  1:50PM           25

 cpu_busy                   io_busy                     idle
 -------------------------- --------------------------- -------------
 276(1)-4%                  0(0)-0%                     87425(24)-96%

 packets_received           packets_sent                packet_errors
 -------------------------- --------------------------- -------------
 1325(3)                    892(34)                     1(0)
 total_read          total_write         total_errors        conn
 ------------------- ------------------- ------------------- ------
 1479(0)             80582(14)           0(0)                9(0)

(return status = 0)
```

c. Subtract the baseline figures

3-1=2
34-2=32

d. Record your results in the table

Number of Packets Sent & Received

	Packets Received	**Packets Sent**
Baseline (sp_monitor)	1	2
Script	2	32

4. *Determine the number of packets send by SalesByZip procedure*

a. Execute the *SalesByZip* stored procedure

```
1> exec SalesByZip
2> go
(1 row affected)
.
.
 99149    Malden                 WA                    0
 99356    Roosevelt              WA                    0
 99625    Levelock               AK                    0

(496 rows affected, return status = 0)
```

b. Execute *sp_monitor* to measure packets sent and received by the procedure

```
1> sp_monitor
2> go
 last_run                   current_run                seconds
 -------------------------- -------------------------- -----------
        Sep  2 1994  1:50PM        Sep  2 1994  1:52PM         134

 cpu_busy                  io_busy                   idle
 ------------------------- ------------------------- -------------
 278(1)-0%                 0(0)-0%                   87558(133)-99%

 packets_received          packets_sent              packet_errors
```

```
------------------------ ------------------------ -------------
1327(2)                   926(34)                   1(0)

total_read          total_write         total_errors         conn
------------------- ------------------- ------------------- ------
1479(0)             80709(127)          0(0)                 9(0)

(return status = 0)
```

c. Subtract the baseline figures
2-1=1
34-2=32

d. record your results in the table

Number of Packets Sent & Received

	Packets Received	Packets Sent
Baseline (sp_monitor)	1	2
SalesByZip	2	32

e. What results did you get? Were packets sent different? Were packets received different? Why?

Fewer packets were received by the server (1 instead of 2) because the stored procedure required just a single packet.

An equal number of packets were sent by the server to the client

f. Look at the response time results you recorded in the first exercise. Do packets sent or received appear to be related to overall response time?

There is a slight improvement. We would need more data to make a correlation.

5. *Compare the performance of a script with varying network packet sizes*

a. Run sp_configure

```
1> sp_configure
2> go
name                  minimum      maximum    config_value run_value
----------------- ----------- ----------- ------------ -----------
recovery interval           1       32767            0           5
allow updates               0           1            0           0
user connections            5  2147483647            0          25
```

```
memory                        3850  2147483647         0       5120
open databases                   5  2147483647        50         12
locks                         5000  2147483647     10000       5000
open objects                   100  2147483647         0        500
procedure cache                  1          99         0         20
fill factor                      0         100         0          0
time slice                      50        1000         0        100
database size                    2       10000         0          2
tape retention                   0         365         0          0
recovery flags                   0           1         0          0
nested triggers                  0           1         1          1
devices                          4         256       100         10
remote access                    0           1         1          1
remote logins                    0  2147483647         0         20
remote sites                     0  2147483647         0         10
remote connections               0  2147483647         0         20
pre-read packets                 0  2147483647         0          3
upgrade version                  0  2147483647      1001       1001
default sortorder id             0         255        50         50
default language                 0  2147483647         0          0
language in cache                3         100         3          3
max online engines               1          32         1          1
min online engines               1          32         1          1
engine adjust interval           1          32         0          0
cpu flush                        1  2147483647       200        200
i/o flush                        1  2147483647      1000       1000
default character set id         0         255         1          1
stack size                   20480  2147483647         0      28672
password expiration              0       32767         0          0
audit queue size                 1       65535       100        100
additional netmem                0  2147483647    163840          0
default network packet size    512      524288         0        512
maximum network packet size    512      524288      8192        512
extent i/o buffers               0  2147483647         0          0
identity burning set factor      1     9999999      5000       5000

(38 rows affected, return status = 0)
```

b. Check the runtime value for "default network packet size"

512

c. Determine configuration value for "maximum network packet size"

512

d. Determine configuration value for "additional netmem"

0

e. note the values below:

Network Configuration Values

Configuration variable	Runtime values
default network packet size	512
maximum network packet size	512
additional netmem	0

These values define the extent to which you can vary network packet size for your session.

6. *Execute PrintAuthors procedure using various packet sizes*

The PrintAuthors stored procedure displays the names of all authors and their blurbs. Adjust network packet size by specifying the desired size as an input parameter to isql. Use the -A flag to vary network packet size, and be sure to stay within the limits of your system as recorded above. Use the -p flag to ensure that statistics are recorded, or use the getdate() statements we have been using to measure response time. Record number of packets sent and response time below. (Some network packet sizes have been provided as suggestions.)

```
isql -Uusername -Ppassword -Apacketsize -p
```

Example:

```
isql -Uuser33 -P -A1024 -p
```

Packet Size & Performance

Packet Size	# of Packets Sent	Response Time
512	13	976
1024	7	266
2048	4	213
4096	3	223
8192	2	156

a. Comment on your results.

There was a significant reduction in network traffic.

13

Lab 13 – tempdb

(Student Guide, page 13-24)

Exercise Overview

Goals

- Examine the current allocation and placement of tempdb

General Tasks

- Assume we have determined that we require tempdb to be 30 Mb
  - Where would you alter it?
  - What steps would you go through to do this correctly?
  - Would it fit on the master device?

Exercise Instructions...

Detailed Instructions

In this lab, you examine the current allocation and placement of tempdb.

1. What is the current allocation for *tempdb*, and where is it located?

```
sp_helpdb tempdb
go
```

2 Mb op master-device

2. Assume we have determined that we require *tempdb* to be 30 Mb.
   a. Where would you alter it?
   b. What steps would you go through to do this correctly?
   c. Would it fit on the master device? Kan wel !

Solutions

1. *What is the current allocation for tempdb, and where is it located?*

```
sp_helpdb tempdb
go
```

   2.0 Mb, on master device.

2. *Assume we have determined that we require tempdb to be 30 Mb.*
   a. *Where would you alter it?*
   b. *What steps would you go through to do this correctly?*
   c. *Would it fit on the master device?*

   Identify the device, disk init if necessary, alter tempdb to new device, 28 Mb.

Lab 14 – Cursors

(Student Guide, page 14-18)

Exercise Overview

Goals

- Examine the locks that are set up when cursors are used

General Tasks

- Identify cursor database locking (scenario #1)
  - On connection 1, declare a read-only cursor Open the cursor
  - On connection 1, examine the locks using sp_lock
  - On connection 2, examine the locks
  - How are they different?
  - On connection 2, attempt to delete the table inside a transaction.
  - What happens?
  - On connection 1, close the cursor
  - What happens on connection 2, and why?
- Identify cursor database locking (scenario #2)
  - On connection 1, reopen curs_auth and run sp_lock
  - What locks are set up?
  - Fetch a row and write down its au_id; Close the cursor and reopen it
  - On connection 2, attempt to delete a row inside a transaction.
  - Were you successful? Why is this different from the last time? (Hint: examine the locks that are held by both processes.)
- Identify cursor database locking (scenario #3)
  - Declare an update cursor; What cursor locks have been taken?
  - Open the cursor; What locks have been taken now
  - Fetch a row; What locks have been taken?
  - Fetch 100 rows; What locks have been taken?
  - Close the cursor and look at the locks again

Detailed Instructions

In this lab, you will examine the locks that are set up when cursors are used.

1. *Identify cursor database locking (scenario #1)*

   a. On connection 1, declare a read-only cursor

   ```
   declare curs_auth cursor for
   select au_id, au_lname, au_fname
   from authors_id
   where au_id like 'A1%'
   for read only
   go
   ```

   b. Open the cursor

   c. On connection 1, examine the locks using *sp_lock*

   d. On connection 2, examine the locks

   e. How are they different?

   f. On connection 2, attempt to delete the table inside a transaction.

   g. What happens?

   h. On connection 1, close the cursor

   i. What happens on connection 2, and why?

   j. On connection 2, roll back the transaction to undo the delete.

2. *Identify cursor database locking (scenario #2)*

   a. On connection 1, reopen curs_auth

   b. Run sp_lock

   c. What locks are set up?

   d. Fetch a row and write down its au_id:

   e. Close the cursor and reopen it

   f. On connection 2, attempt to delete a row inside a transaction.

   g. Were you successful? Why is this different from the last time? (Hint: examine the locks that are held by both processes.)

   h. On connection 2, roll back the transaction.

   i. On connection 1, close the cursor.

3. *Identify cursor database locking (scenario #3)*

    a. Declare an update cursor

    ```
    declare curs_upd cursor for
    select au_id, au_lname, au_fname
    from authors_id
    where au_id like 'A1%'
    for update
    go
    ```

    b. `Run` *sp_lock*

    c. What cursor locks have been taken?

    d. Open the cursor

    e. Run *sp_lock*

    f. What locks have been taken now?

    g. Fetch a row.

    h. What locks have been taken?

    i. Fetch 100 rows

    j. Run *sp_lock*

    k. What locks have been taken?

    f. Close the cursor and look at the locks again.

Solutions

In this lab, you will examine the locks that are set up when cursors are used.

1. *Identify cursor database locking (scenario #1)*

   a. On connection 1, declare a read-only cursor

```
1> declare curs_auth cursor for
2> select au_id, au_lname, au_fname
3> from authors_id
4> where au_id like 'A1%'
5> for read only
6> go
```

   b. Open the cursor

```
1> open curs_auth
2> go
```

   c. On connection 1, examine the locks using *sp_lock*

```
1> sp_lock
2> go
The class column will display the cursor name for locks associated
with a cursor
for the current user and the cursor id for other users.
 spid   locktype                      table_id    page        dbname
         class
 ------ ----------------------------- ----------- -----------
---------------
         ------------------------------

      5 Sh_intent                       384004399           0 master
         Non Cursor Lock
      5 Sh_intent                     1472008275           0 pubtune
         curs_auth

(2 rows affected, return status = 0)
1>
```

   d. On connection 2, examine the locks

```
1> sp_lock
2> go
The class column will display the cursor name for locks associated
with a cursor
for the current user and the cursor id for other users.
 spid   locktype                      table_id    page        dbname
         class
 ------ ----------------------------- ----------- -----------
---------------
         ------------------------------
```

```
        1 Sh_intent                      384004399          0 master
          Non Cursor Lock
        5 Sh_intent                     1472008275          0 pubtune
          Cursor Id 65537

(2 rows affected, return status = 0)
1>
```

e. How are they different?

On connection 1, the cursor name is displayed in the *class* column. On connection 2, the cursor id is displayed.

f. On connection 2, attempt to delete the table inside a transaction.

```
1> begin tran
2> go
1> delete from authors_id
2> go
```

g. What happens?

The process hangs because it cannot get an exclusive lock on the table

h. On connection 1, close the cursor

```
1> close curs_auth
2> go
1>
```

i. What happens on connection 2, and why?

Now the delete goes through, as the cursor lock was released
(5000 rows affected)

1>

j. On connection 2, roll back the transaction to undo the delete.

```
1> rollback tran
2> go
```

2. *Identify cursor database locking (scenario #2)*

a. On connection 1, reopen curs_auth

```
1> open curs_auth
2> go
1>
```

b. Run sp_lock

```
1> sp_lock
2> go
```

```
The class column will display the cursor name for locks associated
with a cursor
 for the current user and the cursor id for other users.
 spid   locktype                     table_id    page        dbname
         class
 ------ ---------------------------- ----------- -----------
---------------
        ------------------------------
      5 Sh_intent                      384004399           0 master
         Non Cursor Lock
      5 Sh_intent                     1472008275           0 pubtune
         curs_auth

(2 rows affected, return status = 0)
1>
```

c. What locks are set up?

There is a shared intent cursor lock on *authors_id*

d. Fetch a row and write down its au_id:

```
1> fetch curs_auth
2> go
 au_id       au_lname                                 au_fname
 ----------- ----------------------------------------
--------------------
 A1000091556 Garske                                   Ray

(1 row affected)
1>
```

e. Close the cursor and reopen it

```
1> close curs_auth
2> go
1> open curs_auth
2> go
1>
```

f. On connection 2, attempt to delete a row inside a transaction.

```
1> begin tran
2> go
1> delete from authors_id
2> where au_id = "A1000091556"
3> go
(1 row affected)
1>
```

g. Were you successful? Why is this different from the last time? (Hint: examine the locks that are held by both processes.)

This time connection 2 is deleting just one row, and this works. Connection 2 is inside a transaction and is holding an exclusive intent lock on the *authors_id* table and an exclusive page lock on the page where the deletion occurred.

h. On connection 2, roll back the transaction.

```
1> rollback tran
2> go
```

i. On connection 1, close the cursor.

```
1> close curs_auth
2> go
1>
```

3. *Identify cursor database locking (scenario #3)*

a. Declare an update cursor

```
1> declare curs_upd cursor for
2> select au_id, au_lname, au_fname
3> from authors_id
4> where au_id like 'A1%'
5> for update
6> go
1>
```

b. Run *sp_lock*

```
1> sp_lock
2> go
The class column will display the cursor name for locks associated
with a cursor
for the current user and the cursor id for other users.
 spid   locktype                    table_id    page        dbname
         class
 ------ --------------------------- ----------- -----------
---------------
        ------------------------------
      5 Sh_intent                     384004399           0 master
         Non Cursor Lock

(1 row affected, return status = 0)
1>
```

c. What cursor locks have been taken?

No cursor locks have been taken so far

d. Open the cursor

```
1> open curs_upd
2> go
```

e. Run *sp_lock*

```
1> sp_lock
2> go
The class column will display the cursor name for locks associated
with a cursor
for the current user and the cursor id for other users.
 spid   locktype                     table_id    page        dbname
         class
 ------ ---------------------------- ----------- -----------
---------------
        ------------------------------
      5 Sh_intent                      384004399           0 master
         Non Cursor Lock
      5 Sh_intent                     1472008275           0 pubtune
         curs_upd

(2 rows affected, return status = 0)
1>
```

f. What locks have been taken now?

There is a shared intent lock on the *authors_id* table.

g. Fetch a row.

```
1> fetch curs_upd
2> go
 au_id       au_lname                                 au_fname
 ----------- ----------------------------------------
--------------------
 A1000091556 Garske                                   Ray
```

h. Run *sp_lock*

```
(1 row affected)
1> sp_lock
2> go
The class column will display the cursor name for locks associated
with a cursor
for the current user and the cursor id for other users.
 spid   locktype                     table_id    page        dbname
         class
 ------ ---------------------------- ----------- -----------
---------------
        ------------------------------
      5 Sh_intent                      384004399           0 master
         Non Cursor Lock
      5 Sh_intent                     1472008275           0 pubtune
         curs_upd
      5 Update_page                   1472008275        5080 pubtune
```

```
          curs_upd

(3 rows affected, return status = 0)
1>
```

i. What locks have been taken?

In addition to the shared intent lock on the table, there is an update page lock on the page the cursor is on.

j. Fetch 100 rows

```
1> fetch curs_upd
2> go 100
```

k. Run *sp_lock*

```
1> sp_lock
2> go
The class column will display the cursor name for locks associated
with a cursor
for the current user and the cursor id for other users.
 spid   locktype                    table_id    page        dbname
          class
 ------ --------------------------- ----------- -----------
---------------
          ------------------------------

      5 Sh_intent                     384004399           0 master
          Non Cursor Lock
      5 Sh_intent                    1472008275           0 pubtune
          curs_upd
      5 Update_page                  1472008275        5084 pubtune
          curs_upd

(3 rows affected, return status = 0)
1>
```

l. What locks have been taken?

In addition to the shared intent lock on the table, there is an update page lock on a different page--the page the cursor is on at the 101st fetch.

f. Close the cursor and look at the locks again.

```
1> close curs_upd
2> go
1> sp_lock
2> go
The class column will display the cursor name for locks associated
with a cursor
for the current user and the cursor id for other users.
 spid   locktype                    table_id    page        dbname
          class
 ------ --------------------------- ----------- -----------
```

```
---------------
        ------------------------------
      5 Sh_intent                     384004399           0 master
         Non Cursor Lock

(1 row affected, return status = 0)
1>
```

Lab 15 – Using CPU Resources

(Student Guide, page 15-24)

15

Exercise Overview

Goals

- Determine the overall CPU use of the system and the ratio of CPU and I/O time for a set of transactions

General Tasks

- Determine the overall CPU statistics of the SQL Server for the *OrderEntryBatch* and *SalesBatch* procedures
- Determine CPU usage at operating system level (unix)

Lab Setup

For the following exercises must:

1. be executed in groups by server (gather round one student)
2. be executed serially (one person/group after another)
3. be executed as a demo by instructor

Only one person or group may run the processes on the server in order to achieve accurate CPU utilization figures.

Exercise Instructions...

Detailed Instructions

In this lab, you will determine the overall CPU use of the system and the ratio of CPU and I/O time for a set of transactions.

Note:

For the following exercises must:

1. **be executed in groups by *server* (gather round one student)**
2. **be executed *serially* (one person/group after another)**
3. **be executed as a *demo by instructor***

Only one person or group may run the processes on the server in order to achieve accurate CPU utilization figures.

1. Determine the overall CPU statistics of the SQL Server for the *OrderEntryBatch* and SalesBatch procedures as follows:

   a. Execute *sp_monitor* twice to establish a baseline

   b. Record the last value you get for *cpu_busy* as the baseline figure.

CPU Statistics: OrderEntryBatch & SalesBatch

	cpu_busy - %
Baseline (sp_monitor)	Ø

   c. Execute *OrderEntryBatch* and *SalesBatch* consecutively

   d. Then execute *sp_monitor* again to check CPU statistics

   e. Subtract the baseline figures from the *cpu_busy* statistics

   f. Record your results *cpu_busy* in table below

CPU Statistics: OrderEntryBatch & SalesBatch

	cpu_busy - %
Baseline (sp_monitor)	
OrderEntryBatch & SalesBatch	

2. Determine the CPU statistics for the CPUBatch procedure

   a. Display CPUBatch using *sp_helptext*

   b. Execute each of the procedures contained in the CPUBatch procedure individually to see which uses the CPU most intensively. Use the method described in the previous exercise, that is, execute *sp_monitor* twice, then your procedure, then *sp_monitor* again, subtracting your baseline CPU figures to get your readings. Record your results in the table below:

Procedure	cpu_busy - %
CPUBatch	26%
CheckForTitle	0%
UpdateQty	0%
SalesByPub	13%
SalesByStore	0%
SalesGrowthByStore	0%

   c. Which procedure uses the CPU most intensively?

3. Determine CPU usage at operating system level (unix)

   a. Open a window to the host where the SQL Server is running

   b. Execute the **showserver** command

   c. Record the server's CPU utilization

   d. Using your first Server connection, re-execute the CPUBatch stored procedure

   d. During execution of CPUBatch execute the **showserver** command

   e. Record the server's CPU utilization

   e. What do you observe?

4. Determine the current setting for 'max online engines'.

   a. Run *sp_configure*

Solutions

Note:

For the following exercises must:

1. **be executed in groups by *server* (gather round one student)**
2. **be executed *serially* (one person/group after another)**
3. **be executed as a *demo by instructor***

Only one person or group may run the processes on the server in order to achieve accurate CPU utilization figures.

1. Determine the overall CPU statistics of the SQL Server for the OrderEntryBatch and SalesBatch procedures as follows:

a. Execute *sp_monitor* twice to establish a baseline

```
1> sp_monitor
2> go
 last_run                   current_run                 seconds
 -------------------------- --------------------------  -----------
        Sep  5 1994  9:50AM        Sep  5 1994  9:50AM            4

 cpu_busy                  io_busy                   idle
 ------------------------- ------------------------- -------------
 897(0)-0%                 0(0)-0%                   331523(4)-100%

 packets_received          packets_sent              packet_errors
 ------------------------- ------------------------- -------------
 1490(1)                   1096(2)                   7(0)

 total_read          total_write         total_errors
connections
 ------------------- ------------------- ------------------- -----
 1834(0)             296407(52)          0(0)                13(0)

(return status = 0)
1>
```

b. Record the last value you get for cpu_busy as the baseline figure.

CPU Statistics: OrderEntryBatch & SalesBatch

	cpu_busy - %
Baseline (sp_monitor)	0%

c. Execute *OrderEntryBatch* and *SalesBatch* consecutively

```
1> OrderEntryBatch
2> go
 TitleID Title                  Publisher       Price    Available
 ------- ---------------------- --------------- -------- ---------
 T81002  RIP Version 2 Protocol McGraw-Hill        44.95         0

(1 row affected)
 TitleID Title                  Publisher       Price    Available
 ------- ---------------------- --------------- -------- ---------
 T63365  Assigned numbers       Bantam Books       58.95         0
 T62154  Assigned numbers       Howard W. Sams     44.95         1
 T6544   Assigned numbers       Wadsworth Publi    51.95         1
 T69002  Assigned numbers       O'Reilly & Asso     3.95         1

(4 rows affected)
 TitleID Title                   Publisher       Price    Available
 ------- ----------------------- --------------- -------- ---------
 T63002  Explaining the role of  John Wiley & So    74.95         0
 T64412  Explaining the role of  Prentice-Hall       4.95         1

(2 rows affected, return status = 0)
1> SalesBatch
2> go
 S878     Books R Us:  Store 491                 0               0
 S978     Books R Us:  Store 492                 0               0
 S179     Books R Us:  Store 493                 0               0
 S279     Books R Us:  Store 494                 0               0
 S379     Books R Us:  Store 495                 0               0
 S479     Books R Us:  Store 496                 0               0
 S579     Books R Us:  Store 497                 0               0
 S879     Books R Us:  Store 498                 0               0
 S979     Books R Us:  Store 499                 0               0
 S18      Books R Us:  Store 500                 0               0

(500 rows affected, return status = 0)
1>
```

d. Then execute *sp_monitor* again to check CPU statistics

```
1> sp_monitor
2> go
 last_run                   current_run                seconds
 -------------------------- -------------------------- -----------
        Sep  5 1994  9:50AM        Sep  5 1994  9:53AM         166

 cpu_busy                  io_busy                   idle
 ------------------------- ------------------------- ------------
 922(25)-15%              0(0)-0%                   331664(140)-84%

 packets_received          packets_sent              packet_errors
 ------------------------- ------------------------- ------------
 1496(6)                   1916(820)                  7(0)

 total_read          total_write          total_errors
connections
 ------------------- ------------------- ------------------- -----
 2775(941)           296981(574)          0(0)                 13(0)

(return status = 0)
1>
```

e. Subtract the baseline figures from the cpu_busy statistics

```
25-0=25
```

f. Record your results cpu_busy in table below

CPU Statistics: OrderEntryBatch & SalesBatch

	cpu_busy - %
Baseline (sp_monitor)	0-1%
OrderEntryBatch & SalesBatch	25%

2. *Determine the CPU statistics for the CPUBatch procedure*

a. Display CPUBatch using *sp_helptext*

```
1> sp_helptext CPUBatch
2> go
 # Lines of Text
 ---------------
               1

(1 row affected)

         text

-------------------------------------------------------------------
--------------

Create procedure CPUBatch
as
exec CheckForTitle
'%Explaining%','computer','10/10/1979',0.00,1000.00
exec UpdateQty 'S657','1994-0357-000002','T17188', 57
exec SalesByPub
exec SalesByStore
exec SalesGrowthByStore
```

```
(1 row affected, return status = 0)
1>
```

b. Execute each of the procedures contained in the CPUBatch procedure individually to see which uses the CPU most intensively. Use the method described in the previous exercise, that is, execute *sp_monitor* twice, then your procedure, then *sp_monitor* again, subtracting your baseline CPU figures to get your readings. Record your results in the table below:

Procedure	cpu_busy - %
CPUBatch	26%
CheckForTitle	0%
UpdateQty	0%
SalesByPub	13%
SalesByStore	0%
SalesGrowthByStore	0%

c. Which procedure uses the CPU most intensively?

SalesByPub

3. *Determine CPU usage at operating system level (unix)*

a. Open a window to the host where the SQL Server is running

b. Execute the **showserver** command

```
godzilla% showserver
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
sybase    1458  0.0 20.5  712 6196 ?  S    Sep  1 72:11
/curdevl/server1001/bin/dataserver -d/devices/PTEME2_master.dat
-sPTEME2 -e/curdevl/server1001/install/PTEME2_errorlog
```

c. Record the server's CPU utilization

0.0%

f. Using your first Server connection, re-execute the CPUBatch stored procedure

```
1> CPUBatch
2> go
```

g. During execution of CPUBatch execute the **showserver** command

```
godzilla% showserver
USER        PID %CPU  %MEM   SZ  RSS TT STAT START  TIME COMMAND
sybase     1458  21.9 20.5  712 6232  ?  S     Sep  1 72:20
/curdevl/server1001/bin/dataserver -d/devices/PTEME2_master.dat

-sPTEME2 -e/curdevl/server1001/install/PTEME2_errorlog
```

h. Record the server's CPU utilization

21.9%

d. What do you observe?

You should notice higher CPU utilization during execution.

4. *Determine the current setting for 'max online engines'*

a. Run *sp_configure*

```
1> sp_configure "max online engines"
2> go
 name                    minimum     maximum  config_value  run_value
 --------------------  ----------- -------- ------------  ---------
 max online engines              1       32             1          1

(return status = 0)
1>
```

Lab C – Introduction to Problem Analysis

(Student Guide, page C-13)

Exercise Overview

Goals

- become familiar with your copy of the pubtune database

General Tasks

- Execute a stored procedure against its tables
- Measure average response time for that procedure
- Display I/O statistics

Exercise Instructions...

Detailed Instructions

In this lab, you will become familiar with your copy of the pubtune database, execute a stored procedure against its tables, measure average response time for that procedure, and display i/o statistics.

1. Access the database assigned to your user number, and display its objects:

```
use pubtuneN    /* where 'N' is your user number */
go
sp_help
go
```

2. Find out how many rows are in each of the following ten tables, and fill in the table below. Use `select count(*) from table_name` to get your answers.

Table Row Count

Table	Rows	Table	Rows
titles		salesdetail	
titleauthor		stores	
authors		shipments	
blurbs		publishers	
sales		roysched	

3. Limit the number of rows returned (this is important!), then display columns and rows from each of these tables to become familiar with their data.

```
set rowcount 10
go
select * from titles
go
etc.
```

4. Reset rowcount using `set rowcount 0`.

5. Display the text of the stored procedure "CheckForTitle".

   ```
   sp_helptext 'CheckForTitle'
   ```

   What does this stored procedure do?

6. Measure the average response time for the CheckForTitle procedure by inserting select getdate() before and after the procedure and using the datediff function, as shown below. Perform at least 3 separate test runs including at least 10 instances of the stored procedure per test run to get valid measurements. (Use vi after you run your batch to capture it and run it again.)

   ```
   1> declare @start datetime
   2> select @start = getdate()
   3> exec CheckForTitle
   4> exec CheckForTitle
   5> select "Total Response Time" = datediff(ms, @start, getdate())
   6> go
   ```

   Response times:

   Average response time:

   ---

7. Turn statistics io on. Execute your procedure again, by itself. What gets displayed on your screen? Turn statistics io off again.

Optional Exercises

1. Start the SQL Monitor Client

   a. Get the monitor server name from the instructor

   b. Start the **sqlmon** executable with the correct parameters

   ```
   gamera% sqlmon -U user_name -P passw -M monitor_server &
   ```

2. Observe page IO using SQL Monitor cache window

   Double click on the **Cache** window option

Solutions

1. *Access the database assigned to your user number, and display its objects:*

```
use pubtuneN    /* where 'N' is your user number */
go
sp_help
go
```

2. *Find out how many rows are in each of the following ten tables, and fill in the table below. Use* `select count(*) from table_name` *to get your answers.*

Table Row Count

Table	Rows	Table	Rows
titles	5000	salesdetail	20
titleauthor	6250	stores	500
authors	5000	shipments	10
blurbs	6	publishers	30
sales	10	roysched	4000

3. *Limit the number of rows returned (this is important!), then display columns and rows from each of these tables to become familiar with their data.*

```
set rowcount 10
go
select * from titles
go
etc.
```

4. *Reset rowcount (off)*

```
1> set rowcount 0
2> go
```

5. *Display the text of the stored procedure "CheckForTitle".*

```
sp_helptext 'CheckForTitle'
```

What does this stored procedure do?

It looks for and displays titles (and associated information) of a given type, publication date, and price range.

6. *Measure the average response time for the CheckForTitle procedure by inserting select getdate() before and after the procedure and using the datediff function, as shown below. Perform at least 3 separate test runs including at least 10 instances of the stored procedure per test run to get valid measurements. (Use vi after you run your batch to capture it and run it again.)*

```
1> declare @start datetime
2> select @start = getdate()
3> exec CheckForTitle
4> exec CheckForTitle
5> select "Total Response Time" = datediff(ms, @start, getdate())
6> go
(1 row affected)
 TitleID Title                        Publisher      Price    Avail
 ------- ---------------------------- -------------- -------- --
 T27317  Aggregation Support in the NSF Addison-Wesley    38.95  0
 T2903   Comments on memory allocation  QED Information   10.95  1
 T290    Commercialization of the Inter Specialized Sys   39.95  1
 T22242  Comparison of Proposals for Ne Que Corporation   23.95  1
 T24049  Domain names - concepts and fa SAMS, A Divisio   26.95  1
 T26377  Feast or famine? A response to Addison-Wesley    34.95  1
 T20017  IMP-Host interface flow diagra Trilithon Press   41.95  0
 T22102  Interim NETRJS specifications  Specialized Sys   48.95  1
 T26206  Internet numbers               QED Information   44.95  0
 T21485  Mapping between X.400(1988) /  ACM Press / Add    7.95  0
 T23149  Network Debugging Protocol     MIS Press         69.95  0
 T23341  Reliable Data Protocol         QED Information   70.95  0
 T2137   Simple Network Management Prot Bell Laboratori   19.95  0
 T2335   The MD5 Message-Digest Algorit Osborne-McGraw    47.95  0
 T202    The UNIX System V Environment  ACM Press / Add   28.95  1
 T22111  UNIX/Xenix Text Processing Ref MIS Press         67.95  1
 T23051  Using TSO at CCN               John Wiley & So    8.95  1

(17 rows affected, return status = 0)
 TitleID Title                        Publisher      Price    Avail
 ------- ---------------------------- -------------- -------- --
 T27317  Aggregation Support in the NSF Addison-Wesley    38.95  0
 T2903   Comments on memory allocation  QED Information   10.95  1
 T290    Commercialization of the Inter Specialized Sys   39.95  1
 T22242  Comparison of Proposals for Ne Que Corporation   23.95  1
 T24049  Domain names - concepts and fa SAMS, A Divisio   26.95  1
 T26377  Feast or famine? A response to Addison-Wesley    34.95  1
 T20017  IMP-Host interface flow diagra Trilithon Press    41.95  0
 T22102  Interim NETRJS specifications  Specialized Sys    48.95  1
 T26206  Internet numbers               QED Information    44.95  0
 T21485  Mapping between X.400(1988) /  ACM Press / Add     7.95  0
 T23149  Network Debugging Protocol     MIS Press          69.95  0
 T23341  Reliable Data Protocol         QED Information    70.95  0
 T2137   Simple Network Management Prot Bell Laboratori    19.95  0
 T2335   The MD5 Message-Digest Algorit Osborne-McGraw     47.95  0
 T202    The UNIX System V Environment  ACM Press / Add    28.95  1
 T22111  UNIX/Xenix Text Processing Ref MIS Press          67.95  1
 T23051  Using TSO at CCN               John Wiley & So     8.95  1

(17 rows affected, return status = 0)

  Total Response Time
 --------------------
                  243

(1 row affected)
1>
```

Response times: (243ms + Additional Executions)

Average response time: (These will vary.)

7. *Turn statistics io on. Execute your procedure again, by itself. What gets displayed on your screen? Turn statistics io off again.*

```
1> declare @start datetime
2> select @start = getdate()
3> exec CheckForTitle
4> exec CheckForTitle
5> select "Total Response Time" = datediff(ms, @start, getdate())
6> go
(1 row affected)
 TitleID Title                          Publisher       Price    Avail
 ------- ------------------------------ --------------- -------- --
 T27317  Aggregation Support in the NSF Addison-Wesley     38.95  0
 T2903   Comments on memory allocation  QED Information    10.95  1
 T290    Commercialization of the Inter Specialized Sys    39.95  1
 T22242  Comparison of Proposals for Ne Que Corporation    23.95  1
 T24049  Domain names - concepts and fa SAMS, A Divisio    26.95  1
 T26377  Feast or famine? A response to Addison-Wesley     34.95  1
 T20017  IMP-Host interface flow diagra Trilithon Press    41.95  0
 T22102  Interim NETRJS specifications  Specialized Sys    48.95  1
 T26206  Internet numbers               QED Information    44.95  0
 T21485  Mapping between X.400(1988) /  ACM Press / Add     7.95  0
 T23149  Network Debugging Protocol     MIS Press          69.95  0
 T23341  Reliable Data Protocol        QED Information    70.95   0
 T2137   Simple Network Management Prot Bell Laboratori    19.95   0
 T2335   The MD5 Message-Digest Algorit Osborne-McGraw     47.95   0
 T202    The UNIX System V Environment  ACM Press / Add    28.95   1
 T22111  UNIX/Xenix Text Processing Ref MIS Press          67.95   1
 T23051  Using TSO at CCN              John Wiley & So     8.95   1
Table: titles    scan count 1, logical reads: 624,physical reads: 0
Table:publishers scan count 17,logical reads: 34,physical reads: 0
Table:Worktable  scan count 0, logical reads: 23,physical reads: 0
Total writes for this command: 0
Total writes for this command: 0


(return status = 0)
 TitleID Title                         Publisher       Price    Avail
 ------- ----------------------------- --------------- -------- --
 T27317  Aggregation Support in the NSF Addison-Wesley     38.95   0
 T2903   Comments on memory allocation  QED Information    10.95   1
 T290    Commercialization of the Inter Specialized Sys    39.95   1
 T22242  Comparison of Proposals for Ne Que Corporation    23.95   1
 T24049  Domain names - concepts and fa SAMS, A Divisio    26.95   1
 T26377  Feast or famine? A response to Addison-Wesley     34.95   1
 T20017  IMP-Host interface flow diagra Trilithon Press    41.95   0
 T22102  Interim NETRJS specifications  Specialized Sys    48.95   1
 T26206  Internet numbers              QED Information    44.95   0
 T21485  Mapping between X.400(1988) /  ACM Press / Add     7.95   0
 T23149  Network Debugging Protocol     MIS Press          69.95   0
 T23341  Reliable Data Protocol        QED Information    70.95   0
 T2137   Simple Network Management Prot Bell Laboratori    19.95   0
 T2335   The MD5 Message-Digest Algorit Osborne-McGraw     47.95   0
 T202    The UNIX System V Environment  ACM Press / Add    28.95   1
 T22111  UNIX/Xenix Text Processing Ref MIS Press          67.95   1
 T23051  Using TSO at CCN              John Wiley & So     8.95   1
Table: titles    scan count 1, logical reads: 624,physical reads: 0
Table:publishers scan count 17,logical reads: 34,physical reads: 0
Table:Worktable  scan count 0, logical reads: 23,physical reads: 0
Total writes for this command: 0
Total writes for this command: 0

(return status = 0)
  Total Response Time
  -------------------
                  300
Total writes for this command: 0

(1 row affected)
1>
```