Locking in DB2 for MVS/ESA Environment

August 1996

International Technical Support Organization
San Jose Center
Contents

Figures ........................................................................ vii

Tables ........................................................................ ix

Preface ........................................................................ xi
How This Redbook Is Organized ........................................... xi
The Team That Wrote This Redbook ................................. xii
Comments Welcome ..................................................... xiii

Chapter 1. DB2 Lock Management ................................. 1
1.1 Introduction ............................................................ 1
1.2 How Locking Works ............................................... 1
1.3 Three Basic Reasons for Locking .............................. 2
1.3.1 Case 1: Losing Updated Data .............................. 2
1.3.2 Case 2: Reading Uncommitted Data ...................... 4
1.3.3 Case 3: Repeatable Read Within a Unit of Work .......... 5
1.4 Unit of Work and Unit of Recovery ......................... 6
1.4.1 Online Transaction Processing ............................ 7
1.4.2 Batch and Distributed Environments ..................... 7

Chapter 2. DB2 Serialization Mechanisms ...................... 9
2.1 Transaction Locking ................................................ 9
2.1.1 Lock Hierarchy ................................................ 11
2.1.2 Lock Sizes .................................................... 12
2.1.3 Duration of Locks ............................................ 12
2.1.4 Mode of Locks ............................................... 17
2.1.5 Lock Escalation .............................................. 19
2.1.6 Lock Suspension ............................................. 21
2.1.7 Deadlock ...................................................... 21
2.1.8 Locking Control Options ................................... 21
2.1.9 Lock Avoidance ............................................. 22
2.1.10 Lock Avoidance Control ................................. 26
2.1.11 IRLM – DB2 Resource Lock Manager ................. 28
2.1.12 Two-Phase Commit Considerations .................... 35
2.2 Restricted States .................................................... 36
2.3 DB2 Subsystem Object Locking ............................... 39
2.3.1 Locks on the DB2 Catalog and Directory ............... 40
2.3.2 Locks on Skeleton Tables ................................ 41
2.3.3 Locks on Database Descriptors ......................... 41
2.4 Claims and Drains ............................................... 42
2.4.1 Claims ....................................................... 42
2.4.2 Drains ....................................................... 43
2.4.3 Interaction between Claims and Drains .................. 43
2.5 Partition Independence .......................................... 48
2.6 Latches ............................................................ 50
2.7 Type 1 and Type 2 Index Considerations .................... 51
2.7.2 Application Scenarios ...................................... 56
2.7.3 End-of-File and Mass-Delete Type 2 Index Lock ....... 58

Chapter 3. How to Prevent Locking Problems .................. 63
3.1 Database Design .................................................. 63
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 3: DELETE</td>
<td>146</td>
</tr>
<tr>
<td>Case A</td>
<td>147</td>
</tr>
<tr>
<td>Case B</td>
<td>149</td>
</tr>
<tr>
<td>Case C</td>
<td>151</td>
</tr>
<tr>
<td>Case D</td>
<td>153</td>
</tr>
<tr>
<td>Case E</td>
<td>155</td>
</tr>
<tr>
<td>Summary</td>
<td>157</td>
</tr>
</tbody>
</table>

### Chapter 6. Data Sharing Considerations

- Logical Locks | 160
  - Table Space Logical Locks | 160
- Physical Locks | 160
  - Page-Set Physical Locks | 161
  - Page Physical Locks | 161
  - Lock Negotiation | 161
- Contention in a Data Sharing Environment | 163
- Claims and Drains | 164
- Lock Avoidance | 164
- Application Design Considerations | 165

### Appendix A. Special Notices | 167

### Appendix B. Related Publications | 169
- International Technical Support Organization Publications | 169
- Other Publications | 170

### How To Get ITSO Redbooks | 171
- How IBM Employees Can Get ITSO Redbooks | 171
- How Customers Can Get ITSO Redbooks | 172
- IBM Redbook Order Form | 173

### List of Abbreviations | 175

### Index | 177
## Figures

1. Scenario to Illustrate Reasons for Locking ........................... 3
2. Example of a Unit of Work ........................................... 6
3. TheGranularity of Locking ........................................... 10
4. Locking Hierarchy .................................................. 11
5. Acquire and Release of Table Space and Table Locks .................... 13
6. Duration of Page or Row Locks in a Read-Only Environment .............. 15
7. Duration of Page or Row Locks in a Read and Write Environment ...... 16
8. CLSN Test Process .................................................. 23
9. PUNC Test Process .................................................. 24
11. Locking Data Model ................................................ 29
12. Lock Processing Flow ............................................... 32
13. IRLM Locking Implementation ....................................... 34
14. Restricted State: Illustration ....................................... 39
15. Concurrency of Transactions and Utilities ................................ 45
16. Example of Partition Independence ................................... 49
17. Locking in an Index-Based Scan Access ................................ 54
18. Locking in an Index-Only Scan Access ................................ 55
19. Locking in an Insert Process ........................................ 56
20. Insertions in Type 1 and Type 2 Indexes ................................ 57
21. RR Readers and Concurrent Insert Process ............................. 59
22. End-of-Index Lock .................................................. 60
23. Units of Recovery Showing Three DB2 Locking Strategies .............. 73
24. Lock Avoidance ..................................................... 76
25. Surprising Results of Short S-locks .................................. 78
26. Australian Way to Update a Hot Page ................................ 80
27. UK Solution ........................................................ 81
28. Lock Duration for the Australian Way and Swiss Solution .............. 83
29. Distributed Client/Server Environment ................................ 85
30. Quality Card for a Table ............................................. 89
31. Quality Card for ORDER Table ....................................... 91
32. Quality Card for NEXTORDERNO Table .............................. 93
33. Quality Card for ORDER Table (Customer 1001) ...................... 94
34. Quality Card for ORDER Table at the Beginning of the Month .......... 96
35. Australian Way to Update a Hot Page ................................ 98
36. Swiss Solution to the Hot-Page Problem ................................ 99
37. UK Solution to the Hot-Page Problem ................................ 100
38. Quality Card for ORDER Table (Row Locks) .......................... 102
39. Quality Card for ORDER Table with Changed Clustering Sequence .... 104
40. Quality Card for NEXTORDERNO Table (With Swiss or UK Solution) ... 105
41. Quality Card for Type 1 Index ....................................... 106
42. The Chain Effect Example ........................................... 110
43. Lock Monitoring Sources ............................................. 112
44. DISPLAY DATABASE LOCKS command ............................. 114
45. EXPLAIN Statement Showing Columns with Relevant Locking Information ........................................... 117
46. DB2 Locks Requested and TSLOCKMODE Values ...................... 118
47. Extract from DB2 PM Accounting Report ................................ 120
48. Result of the DISPLAY THREAD Command ........................... 122
49. Result of the DISPLAY DATABASE USE Command ................. 124
50. Extract from DB2 PM Statistics Report ................................ 125
51. Result of the DISPLAY DATABASE CLAIMERS Command ........ 129
52. DB2 PM Lock Suspension Report .......................... 131
53. DB2 PM Lockout Trace ................................... 135
54. Scenario 1: Program Flow .............................. 139
55. Number of Commits, Elapsed Time, and CPU Consumption . 141
56. Number of Commits, CPU Consumption, and Maximum Pages Locked 142
57. Proposed Commit Strategy ............................... 143
58. LOCKSIZE, Elapsed Time and CPU Consumption ........... 146
59. Scenario 3: Case A – Accounting Trace ................... 148
60. Scenario 3: Case B – Accounting Trace ................... 150
61. Scenario 3: Case C – Accounting Trace ................... 152
62. Scenario 3: Case D – Accounting Trace ................... 154
63. Scenario 3: Case E – Accounting Trace ................... 156
64. DB2 Data Sharing Overview ............................... 159
65. Negotiation of Physical Locks (P-Locks) .................... 162
# Tables

1. Basic Lock Compatibility Matrix ........................................... 2
2. DB2 Serialization Mechanism ............................................... 9
3. Compatibility of Page and Row Lock Modes ............................ 19
4. Compatibility of Table and Table Space Lock Modes .................. 19
5. Possible Values of LOCKMAX .............................................. 20
6. Default Value of LOCKMAX .................................................. 20
7. Impact of CURRENTDATA Option .......................................... 28
8. Timeout Period and Execution Environment .............................. 30
9. Deadlock SQL Return Codes ................................................ 33
10. Restricted States ............................................................ 37
11. Restricted State Check During SQL Program Process ................ 38
12. DB2 Subsystem Locking Activity ......................................... 42
13. Differences Between the Latch and the Lock ............................ 50
14. Coolers for ORDER and NEXTORDERNO Tables ..................... 101
15. Result Table: Commit Frequency Based on Number of Updates ...... 140
16. Result Table: Commit Frequency Based on Number of Seconds ...... 144
17. Result Table: Comparison of Table, Page, and Row Locking ........ 145
18. EXPLAIN Information: Case A ............................................. 147
19. EXPLAIN Information: Case B ............................................. 149
20. Summary of Measurements ................................................. 158
This redbook describes locking in a DB2 for MVS/ESA environment.

It is written for DB2 professionals in general and application developers in particular.

Locking and concurrency issues can have a significant impact on the performance of a DB2 application. This means the application developer needs a good understanding of how the various serialization mechanisms work in DB2. This book has been written to help application developers better understand these mechanisms and how they influence application design decisions.

Several practical examples are presented to demonstrate the impact of locking on the performance of DB2 applications.

This book is used as a supporting document for Education and Training course CF97, “DB2 for MVS/ESA Version 4: Locking for Application Developers.”

Some knowledge of DB2 locking is assumed.

How This Redbook Is Organized

This redbook contains 182 pages. It is organized as follows:

• Chapter 1, “DB2 Lock Management”
  This chapter discusses the basic need for a DB2 locking mechanism and defines unit of work and unit of recovery.

• Chapter 2, “DB2 Serialization Mechanisms”
  This chapter explains the DB2 serialization mechanisms and how they work and discusses the considerations for Type 1 and Type 2 indexes.

• Chapter 3, “How to Prevent Locking Problems”
  This chapter discusses ways of preventing locking problems. Database design, bind and SQL parameter considerations, application design and programming considerations, and quality card analysis to determine the probability of locking are discussed.

• Chapter 4, “How to Identify and Analyze Locking Problems”
  This chapter provides a set of rules to identify and analyze application, concurrency, global, and locking problems.

• Chapter 5, “Locking Scenarios”
  This chapter analyzes locking scenarios and suggests solutions for better performance.

• Chapter 6, “Data Sharing Considerations”
  This chapter discusses locking considerations in a DB2 data sharing environment.
Chapters 1 through 5 do not discuss locking mechanisms that only exist in a DB2 data sharing environment.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization—San Jose Center.

Ravi Kumar is a Senior Information Technology Consultant at the International Technical Support Organization, San Jose Center. He writes extensively and teaches IBM classes worldwide on all areas of DB2 for MVS/ESA. Before joining the ITSO two years ago, Ravi worked in IBM Australia as a DB2 specialist.

Carlos Guardia is a Systems Engineer at IBM Spain where he specializes in database management systems and parallel technologies. He has 10 years of experience in the field of information technology. Carlos holds a degree in Electronics Engineering and a Masters in Business Administration from the Politechnics University of Barcelona, Spain. His areas of expertise include DB2 for MVS/ESA, decision support systems, and data warehousing applications. Carlos also has intimate knowledge of banking, finance, and securities sectors. He has written extensively on DB2 for MVS/ESA performance and parallel optimization techniques.

Hans Ulrik Tetens is an IBM Certified Solutions Architect at IBM Denmark. He has 15 years of experience in the field of information technology. Hans Ulrik holds a Master of Science degree from the Danish Technical University. His areas of expertise include application design and development, DB2 database design, and client/server design and construction.

We thank Tapio Lahdenmaki from IBM Finland and Bernhard Baumgartner from IBM Switzerland for their help and support throughout. Tapio and Bernhard are the developers of Education course CF97, “DB2 for MVS/ESA Version 4 Locking for Application Developers.”

Thanks to the following people for their invaluable contributions to this project:

- John Campbell IBM United Kingdom
- Viviane-Anavi Chaput IBM France
- Karelle Cornwell IBM Development, Santa Teresa
- David Drane IBM United Kingdom
- Hans Duerr IBM Spain
- Jan Henderyckx Education Consultant, Belgium
- Namik Hrle DB2 Consultant, Germany
- Bryce Krohn IBM Education System Center, San Jose
- Christina Lee IBM Development, Santa Teresa
- Bob Lyle IBM Development, Santa Teresa
- Roger Miller IBM Development, Santa Teresa
- Chris Mills IBM Development, Santa Teresa
- Rudolf Metz IBM Germany
- Reg Munusamy ISSC, Australia
- Akira Shibamiya IBM Development, Santa Teresa
- Horacio Terrizzano IBM Development, Santa Teresa
Thanks to Shirley Weinland Hentzell for editing the document, and Stephanie Manning for editorial assistance.

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!
Chapter 1. DB2 Lock Management

In this chapter we discuss the basic need for a DB2 locking mechanism and define unit of work and unit of recovery.

1.1 Introduction

Locking and lock management are a must in any multithread environment that uses a shared, updatable database, so that all users access the same version at any one time. With the proliferation of online databases and terminal networks, locking is not a function that application programmers should have to invoke explicitly. Locking must therefore be a fundamental and implicit system service.

Locking is a database management system (DBMS) function required to allow interprocess concurrency and avoid data integrity problems; that is, avoid exposing uncommitted updates, avoid missing updates, and provide for repeatable read. Two or more independent application processes must be prevented from simultaneously accessing and updating specific data occurrences, whether the occurrences be called rows as in DB2, segments as in information management system (IMS), or logical records as for the virtual storage access method (VSAM).

It is the combination of concurrency, updating, and the need for consistency provided by the DBMS and the application that drive granularity and locking. The job is a partnership. Everything that the DBMS can “guarantee,” the application can undo with a combination of caching data across commits, failure to check return codes, or inadequate and incorrect restart logic.

Without proper locking, application processes that update would have to be single-threaded, thereby reducing throughput and increasing response time. Running batch processes concurrently with online processes would be nearly impossible.

But the lock manager must do more than set and release locks. Management of processes suspended because of requests for locked resources is essential. Dispatching these suspended processes when the resource becomes available is a basic task. Suspended processes that represent a deadlock situation must be identified and the deadlock relieved, while the integrity of the database is maintained.

With DB2 serving more interactive users through various attachments such as time sharing option (TSO), IMS, customer information control system (CICS), and call attach facility (CAF), and with emerging distributed database technology, timeout situations can occur. As with deadlocks, processes that have timed out must be backed out while database integrity is maintained.

1.2 How Locking Works

DB2 takes locks on table spaces, tables (if table space is segmented), and pages or rows. For detailed explanations of table space, table, page and row locks and how they work together see 2.1, “Transaction Locking” on page 9.
Table 1 on page 2 is a compatibility table showing the locks that the DB2 locking mechanism can grant at the same time. You must know these rules to basically understand the three reasons for locking discussed in 1.3, “Three Basic Reasons for Locking” on page 2.

### Table 1. Basic Lock Compatibility Matrix

<table>
<thead>
<tr>
<th>Held Lock</th>
<th>Share</th>
<th>Update</th>
<th>Exclusive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Share</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Update</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Exclusive</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Notes:**

The Share, Update, and Exclusive lock modes in the table apply to row or page locks. These are applicable only if there is an intent lock on the table space, and also an intent lock on the table if the table space is segmented.

**Share**—The lock owner and any concurrent process can read, but not change, the locked DB2 object. Other concurrent processes may acquire Share or Update locks on the DB2 object. Also called S-lock.

**Update**—The lock owner can read the DB2 object and intends to change it. Concurrent processes may acquire Share locks and read the DB2 object but no other process can acquire an Update lock. Update locks must be promoted to Exclusive locks before DB2 actually changes the DB2 object. Promotion to Exclusive lock may cause a suspension if other processes are holding Share locks. Also called U-lock.

**Exclusive**—Only the lock owner can read or change the locked data, with the following exceptions. Concurrent applications using uncommitted read isolation can read the exclusively locked data. Also, if lock avoidance techniques indicate the exclusively locked data is committed, the data can be returned to concurrent cursor-stability applications that do not require currency of data. Also called X-lock.

### 1.3 Three Basic Reasons for Locking

In this section, we illustrate the three basic reasons for locking with the following cases:

- Case 1 – Losing updated data
- Case 2 – Reading uncommitted data
- Case 3 – Repeatable read within a unit of work

#### 1.3.1 Case 1: Losing Updated Data

Figure 1 on page 3 shows the scenario used to illustrate Cases 1 and 2.
1.3.1.1 Case 1 – Without Locking
1. User A reads column value, 100, into a host variable.
2. User B reads the same column value into a host variable.
3. User A adds 10 to the host variable value and saves the new value, 110, in the DB2 table column.
4. User B adds 20 to the host variable value and saves the new value, 120, in the DB2 table column.
This case shows that, without locking, the updated value in the column depends on who commits first. If User A commits first, then the updated column value is 120 and the update of User A is lost. If User B commits first, then the updated column value is 110 and the update of User B is lost.

1.3.1.2 Case 1 – With Locking
In the following scenario, assume an updatable cursor:

1. User A reads column value 100 into a host variable with the intention of updating the value:
   • DB2 grants an Update (U) lock to User A.
2. User B wants to read the same column value into a host variable with the intention of updating the value:
   • According to the compatibility matrix in Table 1 on page 2, DB2 does not grant User B a U-lock on the DB2 object containing column value 100.
   • Since DB2 does not grant a U-lock to User B, User B must wait to read the column value until User A releases the lock.
3. User A adds 10 to the host variable value and wants to save the new value, 110, in the DB2 table column:
   • DB2 changes the U-lock to an Exclusive (X) lock on the DB2 object containing the column value.
4. User A commits:
   • DB2 releases the X-lock on the DB2 object containing the column value.
• DB2 grants the U-lock to User B on the DB2 object containing the column value, unless User B has timed out.

5. The host variable specified by User B now contains the updated value, 110.

6. User B adds 20 to the host variable value and saves the new value, 130, in the table column:
   • DB2 changes the U-lock to an X-lock on the DB2 object containing the column value.

7. User B commits:
   • DB2 releases the X-lock on the DB2 object containing the column value.

If updatable cursors are not used, then DB2 grants an S-lock to User A instead of a U-lock in Step 1 and consequently grants an S-lock to User B in Step 2. When User A and User B try to update the column value, they get into a deadlock situation. When a deadlock exists, DB2 decides whether User A or User B should be rolled back. If User A is rolled back, User A releases the locks, and User B is allowed to complete the process. Conversely, if User B is rolled back, User B releases the locks, and User A is allowed to complete the process.

---

**Note**

The application programmer is responsible for avoiding or at least minimizing deadlock situations. In Case 1, to facilitate the use of U-locks, you should use a `FOR UPDATE OF` clause in the `DECLARE CURSOR` SQL (structured query language) statement. When User A requests a U-lock, User B (who is also requesting a U-lock) must wait until User A has committed the work. According to the compatibility table (Table 1 on page 2), you cannot have two U-locks at the same time on the same DB2 object.

---

### 1.3.2 Case 2: Reading Uncommitted Data

Case 2 uses the same scenario as used in Case 1, Figure 1 on page 3.

#### 1.3.2.1 Case 2 – Without Locking

1. User A updates value 100 to 0 into the DB2 table column.

2. User B reads the updated value 0 and makes program decisions based on that value.

3. User A cancels the process and changes value 0 back to value 100 for the DB2 table column.

This case shows that, without locking, User B made a wrong program decision, and because of that the business data in the database is potentially corrupted:

#### 1.3.2.2 Case 2 – With Locking

1. User A attempts to update value 100 to 0 in the DB2 table column:
   • DB2 grants an X-lock to User A on the DB2 object containing the column value to be updated.

2. User B attempts to read the updated column value 0 and make program decisions based on the value 0:
   • DB2 does not allow User B to read the updated column value 0.
   • User B tries to get an S-lock on the DB2 object that currently has an X-lock. User B has to wait until User A commits or rolls back the work.
3. User A cancels the process and changes value 0 back to value 100 for the DB2 table column:
   - DB2 changes the value 0 to 100 for the table column and releases the X-lock for User A.
   - DB2 grants an S-lock to User B on the DB2 object containing the column value and reads the value 100.

User B, with locking, made the right program decision, and because of that the business data in the database is not corrupted.

Note
In Case 2, with locking, the application programmer has no influence on the DB2 locking mechanism. An update SQL statement always has to have an X-lock. The application programmer must know that S-locks prevent X-locks on a given DB2 object. X-locks persist until commit or rollback.

1.3.3 Case 3: Repeatable Read Within a Unit of Work
User A wants to read the same data twice. It is essential that another user has not updated the data between the two read processes.

Let us look at the following example:

```
SELECT * FROM EMP
  WHERE SALARY >
     (SELECT AVG(SALARY) FROM EMP)
```

This SQL statement reads the EMP (employee) table twice:
1. Find the average value of salary from the EMP table
2. Find all rows in the EMP table that have a salary value greater than the average value.

1.3.3.1 Case 3 – Without Locking
Without any locking between the two read processes, another user can update the EMP table between the two read processes. This update can lead to a wrong result for user A.

1.3.3.2 Case 3 – With Locking
With DB2 locking, there are the following options:

1. Use the package or plan isolation level repeatable read (RR), or use the WITH RR clause in the SQL SELECT statement.
2. Lock the table in share or exclusive mode, although exclusive mode is an overkill in this particular case:
   - SQL statement: LOCK TABLE EMP IN SHARE MODE
   - or LOCK TABLE EMP IN EXCLUSIVE MODE
3. Mark the table space Read-only
   - DB2 command: STA DB(Database name) SPACENAM(Table space name) ACCESS(RO)

Depending on the need for concurrency, only Options 1 and 2 are recommended.
1.4 Unit of Work and Unit of Recovery

A unit of work (UW) is a sequence of actions that must be completed before any of the individual actions in the sequence can finish. For example, the actions of decrementing an inventory file and incrementing a reorder file by the same quantity constitute a unit of work. Both steps must be completed before either step is complete. If one action occurs and not the other, the database loses its integrity, or consistency.

The UW is strictly within the domain of application designers, who must recognize the makeup of each logical application process that changes the content of the database.

Figure 2 illustrates a DB2 process that consists of one unit of work (two UPDATE statements).

It does not make sense to remove parts from one area of control in a work-in-process inventory system unless the parts are accounted for as they move to a new area of control. In classic double-entry bookkeeping terms, we cannot credit one account without debiting another.

```
UPDATE DEPT-INVENTORY
SET ONHAND = ONHAND - 144
WHERE DEPTNUM = 'MACHINING'
AND PARTNUM = 244925

UPDATE DEPT-INVENTORY
SET ONHAND = ONHAND + 144
WHERE DEPTNUM = 'ASSEMBLY'
AND PARTNUM = 244925
```

Figure 2. Example of a Unit of Work

In this situation, because update operations are involved, the unit of work is also the unit of recovery (UR). That is, a UR is the work done by DB2 for an application that changes DB2 data from one point of consistency to another. A point of consistency (also called a sync point or commit point) is a time when all recoverable data that an application program accesses is consistent with other data. A UR begins with the first change to the data after the beginning of the job (or after the last point of consistency) and ends at the next point of consistency.

During a UR, DB2 must enforce strict locking rules to avoid database contamination. URs can end in one of two ways: first, the application process signals successful completion; second, the application or the controlling environment indicates an abnormal situation that drives the process back to the beginning of the UR. In other words, an abnormality is removed by backing out any inflight database changes to the last point of consistency. URs must be 0% or 100% complete; no middle ground is tolerated.

In IMS, the get unique from the message queue includes the commit.

In CICS, the commit can be implicit (return) or explicit with a SYNC call.
Distributed, batch, and TSO applications must be designed with commit logic in mind. A commit or sync point generally releases all locks on the DB2 objects. For batch operation, the UW may consist of one or more transactions if the program is transaction driven or one or more database rows if the program is data driven. For TSO and CAF batch operation, there is an implicit commit at program termination. Intermediate commit points must be explicitly requested.

The term **logical unit of work** (LUW) is used in transaction managers such as IMS and CICS to represent a UW. In general UW, LUW, UR, and commit scope are synonymous.

### 1.4.1 Online Transaction Processing

Online transaction processing typically fits easily into the concept of a logical unit of work. The transaction managers IMS and CICS coordinate with DB2 for proper synchronization of logging and journaling to ensure a unified commit, and signaling to the lock manager that locks held by a transaction can be released. Each update transaction represents an LUW and a UR. The transaction boundary represents an implicit UR.

### 1.4.2 Batch and Distributed Environments

If we think only of online transactions, the term UR seems redundant. When we run DB2 in distributed, batch, or TSO environments, however, we lose the boundaries of the transaction manager. Away from the control of the transaction manager, if one UW involves multiple DBMSs, then it may consist of multiple URs. In a distributed environment, a **distributed unit of work** (DUW) with multisite updates consists of as many URs as the number of relational DBMSs that participate in the coordinated two-phase commit process.

In distributed, batch, or TSO environments, URs are one or more UWs that an application process performs against a database between explicit commit points. For TSO, batch, and distributed procedures, the logic for when a UW has been completed and a UR can be signaled is the application’s responsibility. The DB2 COMMIT statement is used with TSO, TSO batch and distributed tasks or the IMS checkpoint (CHKP) call within IMS-controlled batch operation. Either indicates that the process has completed a so-called **durable** function that should be reflected in the database even across system failures.
We have explained locking in simple terms using examples. However, the term locking should not be used in a generic sense because locking is just one of the serialization mechanisms employed by DB2. DB2 uses different mechanisms for serialization to achieve its goal of maximizing concurrency without losing the integrity of the database and with a minimum cost in CPU, I/O, and storage resources.

Different serialization mechanisms used in DB2 are:

- Transaction locking
- Claims and drains
- Restrictive states
- Latching.

Table 2 shows which serialization mechanisms DB2 uses when SQL, DB2 utilities, and DB2 commands access data in DB2 databases.

<table>
<thead>
<tr>
<th>Table 2. DB2 Serialization Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SQL</strong></td>
</tr>
<tr>
<td>SQL</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DB2 UTILITIES</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DB2 COMMANDS</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Notes:

(a) See DB2 V4 Utility Guide and Reference

(b) CLAIM for COPY and RUNSTATS with SHRLEVEL(REFERENCE)

The following scenario illustrates the use of Table 2:

- A user accesses a table from a TSO DB2 program. At the same time another user starts a utility against the same table.
- DB2 uses the CLAIM and DRAIN process to ensure concurrency and data integrity between the processes.

### 2.1 Transaction Locking

The granularity of locking as shown in Figure 3 on page 10 within a data base management system represents a definite trade-off between concurrency and CPU overhead. Whenever we ask for finer granularity of locking, we may increase the use of available CPU resources because locking in general increases CPU path length. No I/O operations are done, but each lock request requires two-way communication between DB2 and the internal resource lock manager (IRLM). However, it is also possible there may or may not be an increase in the number of potential lock requests. For example, for read-only SQL with highly effective lock avoidance you may not see any increase in the number of DB2 lock requests to the IRLM.
A DB2 thread makes lock requests through IRLM services. Transaction locks are owned by the work unit or thread and managed by the IRLM.

DB2 objects that are candidates for transaction locking are:

- Table space
- Partition
- Table
- Page
- Row.

We have to consider locks on any of the following objects:

- User data in target tables
  - A target table is a table that is accessed specifically in an SQL statement. We have the most control over the locks on target tables.
- User data in related tables
  - Operations subject to referential constraints can require locks on related tables. For example, if we delete a row from a parent table, DB2 might delete rows from the dependent table as well. In that case, DB2 locks data in the dependent table as well as in the parent table.
- User data in indexes
  - For transaction locking with Type 1 indexes, only leaf pages and subpages are locked.
- DB2 subsystem objects
  - Portions of the DB2 catalog
  - The skeleton cursor table (SKCT) for an application plan
  - The skeleton package table (SKPT) for a package
  - The database descriptor (DBD) for a DB2 database.

For more explanation on locking DB2 subsystem objects, see 2.3, “DB2 Subsystem Object Locking” on page 39.
2.1.1 Lock Hierarchy

To access data in a DB2 table, locks are acquired starting at the top of the locking hierarchy and working down until DB2 has decided that the lock is strong enough to serialize access. Figure 4 illustrates the locking hierarchy top-down approach.

Example: In a segmented table space, if DB2 decides to lock a data row, then according to the locking hierarchy, DB2 locks the table space and the table before locking the row.

Note

The way DB2 locks a partitioned table space is always by taking a lock on each partition. However, if you implement the authorized program analysis report (APAR) PN85387 on your DB2 V4.1 system, DB2 locks the partitioned table space by locking only the last partition.
2.1.2 Lock Sizes

Looking at specifics on the lock sizes for DB2 objects, we have the LOCKSIZE option on the CREATE TABLESPACE and ALTER TABLESPACE statements. The choices are ANY, ROW, PAGE, TABLE, or TABLESPACE, with ANY as the default. ANY permits DB2 to make the final choice and DB2 favors page locking as a good compromise between high concurrency and high CPU consumption. Most often, DB2 implicitly locks at the page level, but certain SQL DML statements may be such that DB2 has the option to use table or table-space level or even row locking as an alternative. PAGE is a good design default to allow for interprocess concurrency with read and write intent against a table. Row locking can be considered to improve concurrency still further (reduce suspensions, timeouts) but it may increase locking cost for sequential update processing and introduce more opportunities for deadlocks.

The TABLE or TABLESPACE option should rarely be used in an online environment that emphasizes shared data. The TABLE or TABLESPACE option is rarely used where there is interprocess read and write interest in the table or table space. These options are useful where the data is read-only; a single user or an update process requires exclusive control over the table. TABLE or TABLESPACE options could be useful for private data within an information or decision-support system using products like Query Management Facility (QMF) or the Application System (AS). Here, many of the tables are the exclusive property of a single user. Taking one lock on the entire table or table space saves the CPU overhead of locking individual pages as they are accessed. The more pages accessed by the process, the greater the savings.

The explicit SQL statement LOCK TABLE is another option available. LOCK TABLE is not more limiting to concurrency than LOCKSIZE TABLE or TABLESPACE. What is stated for LOCKSIZE TABLESPACE is also true for LOCK TABLE. The SHARE option reduces the table, and possibly the whole table space, to read-only status. No other concurrent process can issue INSERT, UPDATE, or DELETE SQL statements. EXCLUSIVE also reduces the table, and possibly the whole table space, to the exclusive ownership of a single process which now can issue any authorized SQL DML statement with respect to that table.

2.1.3 Duration of Locks

The duration of a lock is defined as the length of time a lock is held and varies according to the type of lock. Page or row locks are acquired when individual pages or rows are accessed and are released at different times depending on whether the data is accessed for read-only or for update and on the isolation level specified. Table and table space level locks are typically acquired implicitly, based on options specified at BIND or REBIND time.

Because DB2 does implicit table and table space locking, it is critical that the application designer understand the options that influence when table and table space locks are acquired, when they are released, and the mode of the lock.
2.1.3.1 Table Space and Table Locks

Every plan executed in DB2 must acquire a table-space lock and, if the table space is segmented, a table lock also. The exception to this rule is that locks are not acquired on the table space or table if uncommitted read isolation is specified. However, DB2 does acquire a mass delete lock.

DB2 provides options to control the point at which the table space and table locks are imposed. Figure 5 illustrates the effect of the BIND and REBIND options ACQUIRE and RELEASE on when table space and table locks are acquired and released.

The possible values for the ACQUIRE option are ALLOCATE and USE. If you use ACQUIRE(ALLOCATE) all of the needed locks on all of the table spaces and tables used by the plan are acquired at the first SQL call. If you use ACQUIRE(USE), only the needed locks on the table spaces and tables are acquired as and when the table spaces and tables are accessed.

DB2 packages and dynamic SQL always use ACQUIRE(USE).

The time at which the table space and table locks are released is influenced by the RELEASE option. The possible values for this option are DEALLOCATE and COMMIT. If you use RELEASE(DEALLOCATE), the locks are released only when
the thread terminates. If you use RELEASE(COMMIT), the locks are released at commit time. RELEASE(DEALLOCATE) favors the online transaction processing environment (CICS and IMS) where it is more efficient to hold the locks until the thread terminates. RELEASE COMMIT favors the TSO long-running interactive process and results in the locks being held until a COMMIT statement is executed explicitly or implicitly (at successful completion).

The most common option pair is ACQUIRE(USE) and RELEASE(DEALLOCATE). This is probably the best design default. For example, consider two transactions using the same thread. Assume page or row level locking. All page and row locks are always released at commit time regardless of the RELEASE parameter. Table space locks and table locks are released, based on the RELEASE option. If a mixture of RELEASE(COMMIT) and RELEASE(DEALLOCATE) is used across plans and packages, then RELEASE(DEALLOCATE) takes precedence.

RELEASE(DEALLOCATE) is particularly good for thread reuse in an online transaction processing (OLTP) environment and for batch programs taking frequent intermediate commit points. The disadvantage of RELEASE(DEALLOCATE) is that the table space locks and table locks taken as a result of lock escalation are retained across the commit point. Lock escalation is not a good thing. If IMS wait for input (WFI) or IMS/VS fast path (IFP) regions are used with RELEASE(DEALLOCATE), the table spaces accessed are never eligible for a deferred data-set close. ACQUIRE(USE) is best where only a small subset of SQL requests are processed in a typical transaction execution.

2.1.3.2 Page or Row Locks

The duration of page or row locks depends on whether they are acquired and released in a read-only or read and write environment.

Read-Only: Figure 6 on page 15 illustrates the effect of the BIND and REBIND ISOLATION option on when page or row locks are acquired and released in a read-only environment.
The duration and number of page or row locks depends on the nature of the process and the value for the ISOLATION option specified at BIND or REBIND. The most common ISOLATION option is cursor stability or CS. That is the most common, not the default, which is repeatable read or RR.

With cursor stability as the ISOLATION option, DB2 returns committed data and guarantees stability of data under updatable cursors. If a query is answered through a work file, for example, a concurrent thread may update the data in the database that generated the data in the work file.

The repeatable read isolation retains page locks even though the process logic moves through several pages in the same table during a unit of work. If the pages are accessed again, repeatable read guarantees repeatability: the data cannot be changed by any other concurrent process. This is true for read stability isolation also.

A nonstandard query to list all employees who make more than the average salary is an example of a process that would benefit from repeatable read. Two passes of the table are required: the first calculates the average salary while the second tests the average against each employee’s current salary. If employees could be added or deleted, or if salaries could be updated by other concurrent processes, a decision is needed. Either lock-out all the concurrent...
updating processes until the query is complete (repeatable read on the query would do that), or use cursor stability on the query, permitting concurrent access to the table with the knowledge that the query output may contain zero to many erroneous rows.

If the option WITH HOLD is used in a SQL DECLARE CURSOR statement, not all locks are released at commit time:

- All page or row locks except locks held on the cursor positioned page are released.
- Any page or row lock at the current cursor position is retained (S-lock) or downgraded (X or U -> S).
- Table space, table, and DBD locks are not released even if RELEASE(COMMIT) is specified.

**Read and Write:** Figure 7 illustrates the effect of the BIND and REBIND ISOLATION option on when page or row locks are acquired and released in a read and write environment.

A process that uses a cursor adds another dimension to locking. Pages read acquire a U-lock, initially indicating update intent. But before a column can be
updated or a row can be deleted, the U-lock on the page is changed to an X-lock and is released at commit.

2.1.4 Mode of Locks

The mode or the state of a lock determines the type of access to the locked object permitted to the lock owner and to any concurrent application processes. Page or row locks are acquired while a process is inflight, whereas the table space locks are acquired at the first SQL call (if ALLOCATE is specified for ACQUIRE option at bind or rebind) or the first actual use of the table (if USE is specified for ACQUIRE option at bind or rebind).

Two or more processes can concurrently access data by using a page or row share lock (S-lock). However, two or more processes cannot concurrently access data using a page or row update-intent lock (U-lock). That is, only one process can acquire a U-lock on a specific page or row at any instant. Even though a page or row may be U-locked however, other processes can access that page or row (but for read-only) by acquiring an S-lock. If the process holding the U-lock wants to do an update or delete, the U-lock must be promoted to X-lock before this can be done. However, the promotion to X-lock does not occur if any other concurrent process holds an S-lock on that page or row. Eventually, when the S-locks are released, the U-lock is promoted to an X-lock. While a page or row is U-locked, if any other concurrent process wants to do an update or delete, it must first acquire a U-lock, but has to wait and eventually either the U-lock is acquired or a timeout occurs.

2.1.4.1 Table and Table Space Lock Mode

We have the following table and table space lock modes.

- **Intent share (IS)**
  The lock owner can read data in the table or table space but not change it. The lock owner acquires an S-lock on a page or row. Concurrent processes may both read and change data.

- **Intent exclusive (IX)**
  The lock owner and concurrent processes can read and change data in the table or table space. The lock owner acquires U-lock or S-lock on a page or row. When the data is to be changed, the U- or S-lock is promoted to an X-lock.

- **Share (S)**
  The lock owner and concurrent processes can read, but not change, data in the table or table space. There is no need for S-lock on a page or row.

- **Update (U)**
  The lock owner can read, but not change, the locked data. However, the owner can promote the U-lock to an X-lock and then change the data. Processes concurrent with the U-lock can acquire S-locks and read the data, but no concurrent process can acquire a U-lock. The lock owner does not need page or row locks.
  
  U-locks reduce the chance of deadlocks when the lock owner is reading data to determine whether to change it.

- **Share with intent exclusive (SIX)**
The lock owner can read and change data in the table or table space. Concurrent processes can read the data in the table or table space, but not change it. Only when the lock owner changes data, is the X-lock acquired on a page or row.

- Exclusive (X)
  The lock owner may read and change data in the table space or table. Only the concurrent processes using uncommitted read isolation may access the table space or table.

We can classify these six modes into two subsets:
- The intent modes of IS and IX
- The gross modes of S, U, and X.

The lock mode SIX is actually a hybrid state. From a read perspective, it is a gross lock; from a write perspective, it is an intent lock.

Any of the gross locks severely limits concurrency. S restricts the entire application population to read-only status. U also restricts concurrent usage to read-only to the point where the U-lock owner exercises the option to promote the lock to an X-mode. Assuming the lock owner can make that promotion without a timeout or deadlock, the object is now exclusively locked for a single thread. Such locking is a little less severe than using only the X-mode from the outset, but is still very restrictive relative to concurrent use.

### 2.1.4.2 Page or Row Lock Mode

We have the following page or row lock modes.

- Share (S)
  The lock owner and any concurrent process can read, but not change, the locked page or row. Concurrent processes may acquire S- or U-lock on a page or row, or might read the data without acquiring a page or row lock. The table and table space must have either IS- or IX-locks.
- Update (U)
  The lock owner can read, but not change, the locked page or row. However, the owner can promote the U-lock to an X-lock and then change the page or row. Promotion to X-lock may cause a suspension if concurrent processes hold S-locks.

  U-locks reduce the chance of deadlocks when the lock owner is reading data to determine whether to change it. Table and table space must have IX-locks.
- Exclusive (X)
  Only the lock owner can read or change the locked page or row. A concurrent process can access the data only if the process runs with uncommitted read isolation. The table space and table must have either IX- or SIX-locks.

### 2.1.4.3 Compatibility Rules for Lock Modes

The major effect of the lock mode is to determine whether one lock mode is compatible with another. Locks of some modes do not shut out all other users. Assume that a process A holds a lock on a table space which process B also wants to access. DB2 requests, on behalf of Process B, a lock of some particular
mode. If the lock mode of Process A permits a request from Process B, then the two lock modes are said to be compatible.

If the two locks are not compatible, Process B cannot proceed. It must wait until Process A releases the lock.

Compatibility of Page and Row Lock Modes: Compatibility for page and row locks is easy to define. Table 3 shows whether page locks of any two modes, or row locks of any two modes are compatible (Yes) or not (No). No question of compatibility of page lock with a row lock can arise, because a table space cannot use both page and row locks:

<table>
<thead>
<tr>
<th>Held Lock</th>
<th>Requested Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S</td>
</tr>
<tr>
<td>S</td>
<td>Yes</td>
</tr>
<tr>
<td>U</td>
<td>Yes</td>
</tr>
<tr>
<td>X</td>
<td>No</td>
</tr>
</tbody>
</table>

Compatibility of Table and Table Space Lock Modes: Compatibility for table and table space locks is slightly more complex. Table 4 shows whether or not table and table space locks of any two modes are compatible:

<table>
<thead>
<tr>
<th>Held Lock</th>
<th>Requested Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IS</td>
</tr>
<tr>
<td>IS</td>
<td>Yes</td>
</tr>
<tr>
<td>IX</td>
<td>Yes</td>
</tr>
<tr>
<td>S</td>
<td>Yes</td>
</tr>
<tr>
<td>U</td>
<td>Yes</td>
</tr>
<tr>
<td>SIX</td>
<td>Yes</td>
</tr>
<tr>
<td>X</td>
<td>No</td>
</tr>
</tbody>
</table>

2.1.5 Lock Escalation

Lock escalation is the releasing of all the page or row locks that a process holds on a single table or table space in order to acquire a table or table space S- or X-lock mode instead.

Lock escalation balances concurrency with performance by using page or row locks while a process accesses relatively few pages or rows, then changing to table space or table locks when the process accesses many.

LOCKMAX is a new option on the CREATE TABLESPACE and ALTER TABLESPACE statements. It defines the maximum number of page or row locks an application process can hold simultaneously in the table space. If a program requests more than that number, locks are escalated. The page or row locks are released, and the intent lock on the table space or segmented table is promoted to S or X. Table 5 on page 20 shows the possible values of LOCKMAX.
If LOCKMAX is omitted in the CREATE TABLESPACE or ALTER TABLESPACE statements, the default value of LOCKMAX depends on the specified value of LOCKSIZE. Table 6 summarizes the default value of LOCKMAX.

### Table 5. Possible Values of LOCKMAX

<table>
<thead>
<tr>
<th>LOCKMAX Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>Use the value specified in LOCKS PER TABLE(SPACE) (DSNZPARM NUMLKT(S)) on the DSNTIPJ installation panel</td>
</tr>
<tr>
<td>0</td>
<td>Number of locks allowed is unlimited and escalation does not occur</td>
</tr>
<tr>
<td>1 to 2147483647</td>
<td>Number of locks allowed before escalating</td>
</tr>
</tbody>
</table>

**Note:** Although the maximum value of LOCKMAX is 2147483647, if you must assign a LOCKMAX value that represents more than half the pages or rows in the table space, you should review your application or database design to see if it can be reduced.

### Table 6. Default Value of LOCKMAX

<table>
<thead>
<tr>
<th>LOCKSIZE</th>
<th>Default LOCKMAX for CREATE TABLESPACE Statement</th>
<th>Default LOCKMAX for ALTER TABLESPACE Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLESPACE or TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PAGE or ROW</td>
<td>0</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ANY</td>
<td>SYSTEM</td>
<td>SYSTEM</td>
</tr>
</tbody>
</table>

DB2 V4 extends lock escalation in the following ways:

- The count of locks applies to row locks in the same way as to page locks.
- Lock escalation applies not only to table spaces defined with LOCKSIZE ANY, but also to those defined with LOCKSIZE PAGE or ROW.
- You can choose a different lock escalation cutoff point for a specific table space by specifying LOCKMAX for that table space. You can also disable the lock escalation of a table space by setting LOCKMAX to 0.

Lock escalation occurs when the number of locks held by a single process exceeds the LOCKMAX value on a table or table space. Lock escalation involves obtaining a table or table-space lock, then releasing all of the page or row locks.

Lock escalation is a safety valve that DB2 provides just in case your application overuses system resources. LOCKMAX 0 means that you are disabling this extremely useful feature. No loss of concurrency is incurred with lock escalation when most of the rows or pages in a table space are already locked, as long as RELEASE(COMMIT) is used. Lock escalation in such a situation reduces storage use but much more important, reduces the CPU time needed to traverse a lock hash synonym chain. Lock escalation also reduces the internal resource lock manager (IRLM) latch suspension time, which also reduces CPU time.

A new column, LOCKMAX, is added to the SYSIBM.SYSTABLESPACE catalog table. It contains the maximum number of locks per user on the table or table space before escalating to the next locking level. Any escalation process is suspended during the execution of SQL statements for ALTER, CREATE, DROP, GRANT, and REVOKE.
2.1.6 Lock Suspension

A process is suspended when it requests a lock on an object that is already locked by another concurrent process and the two locks are not compatible. The suspended process temporarily stops running.

Incoming lock requests are queued. Requests for lock promotion and requests for a lock by a process that already holds a lock on the same object take precedence over requests for locks by new processes. The suspended process resumes running when:

- All concurrent processes that hold conflicting locks release them.
- The requesting process times out or deadlocks and the process resumes to deal with an error condition.

**Example: Using an Application for Inventory Control:** Two users attempt to reduce the quantity on hand of the same item at the same time. The two lock requests are queued. The second request in the queue is suspended and waits until the first request releases its lock.

2.1.7 Deadlock

A deadlock occurs when two or more processes hold locks on resources that the others need and without which they cannot proceed.

DB2 scans for deadlocked processes at regular intervals. The field DEADLOCK TIME on DB2 installation panel DSNTIPJ sets the length of the interval. The default value is 5 seconds. DB2 can roll back the current unit of work for one of the processes or request a process to terminate. That frees the locks and allows the remaining processes to continue.

It is possible for two processes to be running on separate DB2 subsystems, each trying to access a resource at the other location. In that case, neither subsystem can detect that the two processes are in deadlock; the situation is resolved only when one process times out.

2.1.8 Locking Control Options

You can influence DB2 locking after tables have been created, by using the different options when binding or rebinding an application plan or package or with individual SQL statements.

The options at bind or rebind of the application plan or package are these:

- **ACQUIRE**, with possible values **USE** and **ALLOCATE**
- **RELEASE**, with possible values **COMMIT** and **DEALLOCATE**
- **ISOLATION**, with possible values **CS**, **RR**, **UR**, and **RS**
- **CURRENTDATA**, with possible values **YES** and **NO**.

The options with SQL statement are these:

- **SELECT ... WITH CS, RR, UR, or RS**
- **INSERT ... WITH CS, RR, or RS**
- **UPDATE ... WITH CS, RR, or RS**
- **DELETE ... WITH CS, RR, or RS**
- **DECLARE CURSOR ... FOR UPDATE OF**
- **DECLARE CURSOR ... WITH HOLD**
- **DECLARE CURSOR ... FOR UPDATE OF**
- **LOCK TABLE**.
2.1.9 Lock Avoidance

Lock avoidance is a mechanism used under certain circumstances by DB2 to increase concurrency, at the same time avoiding calls to IRLM to request a page or row lock.

In a lock-avoidance situation, DB2 reads a page without first taking an IRLM lock; hence, the page might contain uncommitted data. To prevent retrieval of uncommitted data, DB2 uses a combination of mechanisms:

- Page latching controlled by DB2 is a very efficient way to ensure physical consistency of the page, just as locks ensure that data is committed.
- Commit Log Sequence Number (CLSN) to check for the committed state at a page level
- Possibly UNCommitted (PUNC) a deeper checking at a row level, done only if the CLSN check fails.

In general, index-only access with a Type 2 index implies the data page is locked. However, there can be lock avoidance for index-only access and also index access with data reference because of the CLSN and PUNC checking on the index page.

We call those rows in a DB2 page that satisfy the SQL predicate conditions and are returned to the program issuing the SQL call qualifying rows. Nonqualifying rows are also handled by DB2 but they are discarded and not returned to the program issuing the SQL call, as they do not satisfy the SQL predicate conditions.

Users have no direct control over the use of lock avoidance. To take advantage of lock-avoidance mechanisms, applications must use:

- Read-only or Ambiguous cursor
- ISOLATION(CS)

For CURRENTDATA(NO), lock avoidance is attempted for all rows (qualifying and nonqualifying). For CURRENTDATA(YES), lock avoidance can occur for nonqualifying rows only. COPY and RUNSTATS utilities with the SHRLEVEL(CHANGE) option do not acquire transaction locks.

2.1.9.1 Commit Log Sequence Number

The CLSN technique takes advantage of the page log relative byte address in the header of every page in the page set. The page log relative byte address value represents the last change RBA to the page.

Before the point in time represented by the CLSN value for the page set, there are no uncommitted updates against it; Then, if the page log relative byte address value of the page is less than the CLSN for the page set, all of the rows in the page are in a committed state and no lock is required to ensure the logical consistency of the data in the page.

At page-set open time, before any update activities can be made against the page set, the CLSN is set to the log RBA of the end of the log. As a result, the page log relative byte address of every page in the page set is less than the page set’s CLSN. Figure 8 on page 23 shows the CLSN test process.
The CLSN technique works as follows:

1. Determine the CLSN for the page set or partition.
2. Latch the page in S mode to ensure that accesses to the page are made in sequence and the page is physically consistent.
3. Compare the data page log relative byte address of the page with the CLSN for the page set or partition. If the page log relative byte address is less than the CLSN, all of the rows in the page are in a committed state. If the page log relative byte address of the data page is greater than CLSN, the PUNC technique is used.

CLSN can avoid a lot of locks when the transactions running against the page set are not very long.

### 2.1.9.2 Possibly Uncommitted

When the CLSN technique is not enough to determine the commit state of the page, DB2 uses the possibly uncommitted (PUNC) technique. The CLSN technique is page oriented and checks whether all rows in a page are committed. If CLSN fails, the DB2 goes deeper asking for the state of each row in the page. The row oriented PUNC technique checks whether an individual row in a page is committed.
The PUNC technique may mark the row as possibly uncommitted.

Figure 9 shows the PUNC test process.

The PUNC test is always done using a page latch.

The row is definitely committed when the row is not marked possibly uncommitted. However, when the row is marked possibly uncommitted, it indicates that the row may be uncommitted. When an update process updates the row, it marks the row as possibly uncommitted.

The possibly uncommitted mark is removed at either the data-page level or row level.

At the data-page level, the mark indicating that a row is possibly uncommitted is removed, and this is done for all such rows in the page, when the following are true:

- The update process is running with page or row locking.
- The page log relative byte address is less than the CLSN for the page set.
- More than 25% of the rows are marked possibly uncommitted.
The number of rows that are marked possibly uncommitted is recorded in the page header.

At the row level, the possibly uncommitted mark is removed only for the row being read.

Only the readers that do not use uncommitted read isolation read committed data. As the reader reads the row, if the row is marked possibly uncommitted, that mark is removed.

2.1.9.3 Lock-Avoidance Flow Process

Figure 10 shows the lock-avoidance flow process that results in avoiding the lock on the page if the specified conditions are met. This process applies to data pages only. For index pages, unqualified rows are rejected before the lock-avoidance techniques are applied.

The lock-avoidance flow process steps are these:

1. DB2 requests a GETPAGE and takes an S-latch on the referenced page.

2. DB2 implements the first lock-avoidance checking mechanism, CLSN. The cursor characteristics, such as whether the cursor is ambiguous or not, are not important here:
If the CLSN test fails, DB2 goes to the next level of lock avoidance checking, the PUNC test.

· If CLSN test is OK, then the process continues with Step 4.

3. If CLSN test fails, DB2 checks the integrity of the data at a row level using the PUNC test:
   · If the PUNC test fails, an IRLM S-lock is requested, as there is no guarantee that the row is in a committed state. When the S-lock is granted, DB2 reads the page. Lock avoidance is not possible in this situation.
   · If the PUNC test is OK, then the process continues with Step 4.

4. DB2 now checks if the row qualifies (that is, is a row that will be returned to the calling program):
   · If the row does not qualify Stage 1 predicates, no lock is taken and the next row on the page is retrieved to continue the process with Step 1.
   · If the row qualifies Stage 1 predicates, then to determine if the row qualifies Stage 2 predicates, the row will be locked (even if eventually it does not qualify), subject to the isolation level and data currency specifications. At that stage, the CURRENTDATA test is done for the CS and RS scanners only. If the row does not qualify Stage 2 predicates, next row in the page is retrieved to continue the process with Step 1. If the row qualifies, then the process continues with Step 5.

5. DB2 checks the CURRENTDATA option of the plan or package containing the SQL call:
   · If CURRENTDATA is YES, lock is requested.
   · If CURRENTDATA is NO, lock avoidance is effective and the row is passed to the calling program without requesting an IRLM lock.

6. If this is the last row in the page, the page is unlatched.

Plans and packages have a better chance for lock avoidance if the bind options used are ISOLATION(CS) and CURRENTDATA(NO).

For nonqualifying rows, lock avoidance can be applied for all cursors. That is, if SELECT ... FROM ... WHERE ... FOR UPDATE OF ... is specified and not many rows meet the conditions in the WHERE clause, then lots of lock avoidance can take place.

DB2 uses also lock avoidance mechanisms when accessing the parent referential integrity structures. For update primary key SQL statements or delete SQL statements with a restrict referential integrity constraint, DB2 uses lock-avoidance techniques to search for dependent rows.

2.1.10 Lock Avoidance Control

For unambiguous read-only cursors with CURRENTDATA(NO) specified, DB2 uses the data page lock-avoidance technique to read all rows in the table without taking any IRLM locks.

The page latch, CLSN, and PUNC techniques are used. However, in the read-only cursors with CURRENTDATA(NO) semantic, the stability of the qualifying rows is not protected by the IRLM lock. As soon as the row qualifies under the protection of a data page latch, the row is passed to the application, and the latch is
released. Therefore the content of the qualified row might have changed immediately after it was passed to the application. To continue processing further rows in a page, DB2 must latch the page again.

The user has no direct control over DB2's use of lock avoidance techniques. However, the user can develop programs that meet the needed criteria to qualify for lock avoidance, if doing so fits in with the application requirements.

Lock avoidance can be monitored with DB2 trace facilities using DB2 Performance Monitor (DB2 PM) reports.

- **Performance trace class 6** reports general lock-avoidance usage in instrumentation facility component identifier (IFCID) record 218. It reports when lock avoidance by page set occurs in a unit of work. One record is externalized at each commit or abort point.

- **Performance trace class 7** reports every time DB2 successfully uses a lock-avoidance mechanism in IFCID 223. It does not, however, indicate the technique employed.

  Starting this facility generates a high volume of trace records.

### 2.1.10.1 BIND Options

The following options can be specified at a plan or package level, both in bind and rebind processes:

- **CURRENTDATA**: With CURRENTDATA(NO) DB2 considers lock avoidance techniques to access the data. This is the recommended option and must be explicitly specified. Lock avoidance is not considered for qualifying rows if the application is bound with CURRENTDATA(YES).

  Default value: YES

- **ISOLATION**: Cursor Stability (CS) increases the concurrency and also the possibility of lock avoidance, and is therefore recommended.

  Default value: RR

  ISOLATION can also be specified in the SQL statement using the WITH clause.

  An example is the following SQL:

  ```sql
  SELECT name, address
  FROM user.table
  WHERE dept = 110
  WITH CS;
  ```

  The ISOLATION value specified using the WITH clause in the SQL statement overrides the value specified for the ISOLATION option for the plan or package.

### 2.1.10.2 Lock Avoidance, Parallelism, and Block Fetch Enablement

With the new Type 2 index in DB2 V4, lock avoidance is enabled for ambiguous cursors with CURRENTDATA(NO). In DB2 V2.3, when ISOLATION(CS) is specified, the application can be certain that the row a cursor has just fetched is locked. However, when CURRENTDATA(NO) is in effect, this certainty may not exist. The effect of the CURRENTDATA option on locking behavior for
ISOLATION(CS) applications is to change the default of CURRENTDATA from NO to YES in DB2 V4.

The change of the CURRENTDATA default has also affected I/O and CP parallelism and block fetching in distributed applications. Table 7 summarizes the impact of the CURRENTDATA option.

<table>
<thead>
<tr>
<th>Is currency of data required?</th>
<th>Ambiguous Cursor</th>
<th>Read-only Cursor</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES (default in DB2 V4)</td>
<td>• Lock avoidance is considered for ISOLATION(CS) applications.</td>
<td>• Lock avoidance is considered for ISOLATION(CS) applications.</td>
</tr>
<tr>
<td></td>
<td>• I/O and CP parallelism are not allowed.</td>
<td>• I/O and CP parallelism are allowed.</td>
</tr>
<tr>
<td></td>
<td>• Block fetching does not apply for distributed applications.</td>
<td>• Block fetching applies for distributed applications.</td>
</tr>
<tr>
<td>NO</td>
<td>• Lock avoidance is considered for ISOLATION(CS) applications.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• I/O and CP parallelism are allowed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Block fetching applies for distributed applications.</td>
<td></td>
</tr>
</tbody>
</table>

Note: In both cases of CURRENTDATA option, lock avoidance is considered on unqualified rows for ISOLATION(CS) applications.

If you want your ISOLATION(CS) applications, with default CURRENTDATA in DB2 V3, to have the same locking behavior in DB2 V4, you must rebind your applications, with CURRENTDATA(NO) added to your BIND REPLACE statement, when you migrate to DB2 V4.

2.1.11 IRLM – DB2 Resource Lock Manager

DB2 application programs request data from DB2 by issuing a call to the DB2 language interface, which directs this request to DB2. DB2 retrieves the required data. But before the retrieval can happen, DB2 must check, by calling the IRLM, that this data is consistent and it is not being used by another concurrent application program. However, DB2 does not always have to use IRLM services. Qualifying rows for read-only SQL using cursor stability isolation can be accessed if a check of the CLSN indicates that the data can be guaranteed to be committed and consistent (discussed in 2.1.9, “Lock Avoidance” on page 22). DB2 can access data, latch the page, check that it is not down level without first acquiring a read-only lock.

The IRLM handling of lock elements can be illustrated with an entity-relationship diagram. Essentially we have a many-to-many relationship between resources and application threads. Resources are locked by application threads that hold locks on resources. A resource may be requested by many application threads but an application thread may be waiting for only one locked resource.

Figure 11 on page 29 shows the locking data model.
The IRLM should have a very high MVS dispatching priority for detecting and breaking deadlocks, for error management, and for lock resumption.

### 2.1.11.1 IRLM and DSNZPARM Options

The IRLM options can be specified in the DSNZPARM DSN6SPRM macro, in the IRLM startup procedure, or during DB2 installation on the panels DSNTIPI and DSNTIPJ.

The significant system parameters related to application locking and performance of the DB2 subsystem are these:

- **Deadlock Time:** This is the entry DEADLOCK TIME on the DB2 installation panel DSNTIPJ and corresponds to the IRLM PROC parameter DEADLOK. A deadlock is a situation where two or more requesters are waiting for resources held by the other.

  Deadlock Time specifies the time in seconds of the local deadlock detection cycle and must be less than the value specified for the IRLM resource wait timeout option.

  *Default value:* 5 seconds

- **IRLM Resource Wait Timeout:** This is the entry RESOURCE TIMEOUT on the DB2 installation panel DSNTIPJ and corresponds to the DSNZPARM...
IRLMRWT. It specifies the number of seconds IRLM waits before detecting a timeout. A timeout is a lock request that has waited for a resource longer than the number of seconds specified for this option. IRLM uses the deadlock time to initiate timeout detection, so the real timeout period can be more than IRLMRWT and is given by the following formula:

\[
\text{Timeout wait time} = \text{UPPER}(\frac{\text{IRLMRWT}}{\text{DEADLOK}}) \times \text{DEADLOK}
\]

When the IRLM Resource Wait Timeout system parameter is a multiple of the Deadlock Time system parameter, the initialization time is zero and the maximum time to wait for a resource is IRLMRWT. You have seen the importance of this parameter to backout programs that are waiting for a request that may be locked for a long period of time. This way you avoid the cascade problem that is produced if this locked program can also lock resources needed for other programs in an endless chain. Other important processes should be allowed to wait more than the transactions to get the resource they need. So IRLM considers multiples of IRLMRWT for these processes as shown in Table 8.

**Default value:** 60 seconds

<table>
<thead>
<tr>
<th>Program Execution Environment</th>
<th>Multiples of IRLMRWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSO Online</td>
<td>1</td>
</tr>
<tr>
<td>TSO Batch</td>
<td>1</td>
</tr>
<tr>
<td>Call Attach Facility</td>
<td>1</td>
</tr>
<tr>
<td>IMS MPP</td>
<td>1</td>
</tr>
<tr>
<td>IMS Fast Path</td>
<td>6</td>
</tr>
<tr>
<td>IMS BMP</td>
<td>4 (a)</td>
</tr>
<tr>
<td>IMS DL/I Batch</td>
<td>6 (b)</td>
</tr>
<tr>
<td>CICS</td>
<td>1</td>
</tr>
<tr>
<td>BIND processing</td>
<td>3</td>
</tr>
<tr>
<td>DB2 utilities</td>
<td>6 (c)</td>
</tr>
</tbody>
</table>

**Notes:**

(a) Can be modified (BMPTOUT DSNZPARM parameter)

(b) Can be modified (DLITOUT DSNZPARM parameter)

(c) Can be modified (UTIMOUT DSNZPARM parameter)

**Utility Timeout:**

This is the entry UTILITY TIMEOUT on the DB2 installation panel DSNTIPI and corresponds to the ZPARM UTIMOUT. Utility Timeout specifies the number of IRLM Resource Wait Timeout values (IRLMRWT) that a utility or DB2 command waits for a lock or for all claims on a resource of a particular claim class to be released.

**Default value:** 6 seconds

**U lock for RR or RS isolation:** This is the entry U LOCK FOR RR/RS on the DB2 installation panel DSNTIPI and corresponds to the ZPARM RRULOCK. It specifies whether DB2 will use a U-lock when the isolation of the program is
repeatable read (RR) or read stability (RS). This is applicable for the SQL statements:

- SELECT with FOR UPDATE OF
- UPDATE and DELETE, without a cursor

If the value specified is YES, the lock used in these cases is update (U). If the value specified is NO, the lock used in these cases is shared (S). If your applications make frequent updates with RR or RS isolation, the U-lock reduces potential deadlocks. In read-only processes or when updates are less frequent, S-locks generally provide more concurrency and better performance (reduced CPU time). If you want concurrency with a mix of read and write, use the value NO.

*Default value: NO*

- **Cross-address space program call:**

  This is the entry CROSS MEMORY on the DB2 installation panel DSNTIPJ and corresponds to the IRLM PROC parameter PC. It specifies whether the IRLM is to use the cross-address-space program call and affects where the IRLM lock control block structure is stored. PC = NO puts the IRLM lock control block structure in the extended common storage area (ECSA). This requires less processor time but can reduce the range of addresses available to private address spaces. PC = YES puts the lock control block structure in the IRLM private address space; the lock manager has the lock elements in private virtual storage above the 16 megabyte line as opposed to their being in ECSA with other sharable system elements. The larger the ECSA, the smaller the private area for all MVS address spaces.

  Additionally, PC = NO results in a shorter path length for lock manager functions. This is because MVS cross-memory services are not required for DB2 and IRLM communication for each lock request. To minimize CPU usage, choose PC = NO.

  Although PC=NO is recommended to reduce CPU consumption, you must make sure that ECSA which is shared is sized correctly to support the IRLM requirement in addition to other users. If ECSA is not sized correctly, it spills to CSA and when that is exhausted MVS ends abnormally (abends).

  *Default value: NO*

- **Maximum locks per table space:** This is the entry LOCKS PER TABLE(SPACE) on the DB2 installation panel DSNTIPJ and corresponds to the DSNZPARM NUMLKTS. This value becomes the default value for the LOCKMAX clause of the CREATE TABLESPACE for user table spaces.

  *Default value: 1,000*

- **Maximum locks per user:**

  This is the entry LOCKS PER USER on the DB2 installation panel DSNTIPJ and corresponds to the DSNZPARM NUMLKUS. It specifies the maximum number of page or row locks that a single application can hold concurrently on all table spaces.

  The maximum includes locks on data pages, rows (and index pages and subpages if you are using Type-1 indexes) that the program acquires when it accesses table spaces. The limit applies to all table spaces defined with the LOCKSIZE PAGE, LOCKSIZE ROW, or LOCKSIZE ANY options. A value of 0 means that there is no limit to the number of page and row locks a program can acquire.
Default value: 10,000

- **Deadlock cycle:** This is the entry DEADLOCK CYCLE on the DB2 installation panel DSNTIPJ and corresponds to IRLM PROC DEADLOK. This option is used only for DB2 data sharing and is the number of local deadlock cycles that must expire before the IRLM starts next global detection cycle.

  Default value: 1

### 2.1.11.2 Lock Processing Flow

Figure 12 shows the logic flow in DB2 lock processing.

![Diagram of Lock Processing Flow](image)

An SQL request (SELECT, FETCH, INSERT, DELETE, or UPDATE) from an application thread results in lock request from DB2 to the IRLM, unless either uncommitted read isolation is used or lock avoidance is possible.

IRLM checks whether the requested resource is available for the type of lock requested. If the request can be granted, a new lock element is added to the lock manager’s queue of elements and chained off the queue for the thread holding the lock.

If the request cannot be granted, it is queued on a suspend queue to wait until:
• The resource is freed by the holder reaching the end of a UR, or the holder moves to a different page through a cursor select and no update, or there is an explicit or implicit rollback as a result of abend by holder.

• IRLM determines that the suspended thread has timed out.

• IRLM determines that the suspended thread is a participant in a deadlock situation.

IRLM monitors the length of any suspension. When the specified installation values for deadlock time or the timeout period are reached, deadlock and timeout processing occurs. A deadlocked program receives a -911 or -913 SQLCODE while a program that times out receives a -911, -913 or -923 SQLCODE.

Table 9 shows the SQL return code in case of deadlocks in different environments.

<table>
<thead>
<tr>
<th>Result of Deadlock</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TSO and CAF</td>
</tr>
<tr>
<td>BACKOUT</td>
<td>Yes</td>
</tr>
<tr>
<td>PSEUDO ABEND</td>
<td>-</td>
</tr>
<tr>
<td>SQL-Code</td>
<td>-911</td>
</tr>
</tbody>
</table>

Notes:
(a) Only the current SQL statement is backed out.
(b) Only in a DB2 abend situation.

After granting a requested lock, DB2 checks that:

• The total number of page or row locks currently held for the thread does not exceed the specified installation maximum (NUMLKUS).

• The total number of page or row locks currently held for the thread on the target table space does not exceed the specified installation maximum (LOCKMAX).

Exceeding NUMLKUS results in the return of a unique SQLCODE -904 to the calling program, which can be interpreted as indicating the need for more frequent COMMIT points in your program.

Exceeding LOCKMAX can trigger lock escalation in case the table space being accessed is defined with a locksize of ANY, PAGE, or ROW.

Therefore, the application program can receive two possible results from an SQL request. The program either gets the requested lock and continues processing with or without a lock suspension wait or receives an SQL return code indicating an abnormal situation was detected such as exceeding NUMLKUS or a timeout or deadlock.

Once DB2 has finished processing the data, and depending on the isolation options, a call to the IRLM is performed to release the lock on the object.
2.1.11.3 IRLM Locking Implementation

Figure 13 shows how the IRLM locking is implemented.

DB2 requests a lock from the IRLM. A hashing algorithm based on the data set identifier and page number is used to locate a hash table entry pointing to a chain of hashing synonyms. If a matching entry is found in the hash anchor table, and the lock request is compatible, the application can continue processing without requiring a lock-wait suspension. If a matching entry is found in the hash anchor table, and the lock request is not compatible, the application requesting the lock is placed in a suspended, lock-wait state. If a matching entry is not found in the hash anchor table, a new lock element is added to the chain off the hash table anchor point, control is returned to DB2, and the application can continue processing without requiring a lock-wait suspension.

The amount of storage needed for each lock held in the IRLM table is 250 bytes.

Suspended threads are the only ones examined periodically to determine if they are so queued for resources such that a deadlock situation exists or that the thread should be timed-out.

DB2 is also responsible for coordinating logging and commit functions. This requires a threadwide release of all locks held by a thread that is terminating or
doing commit processing. While releasing the locks, IRLM also tests if there is another thread on the suspend queue waiting for the object being unlocked. If so, that thread can be granted the lock and end its lock suspension wait.

The lock management requires pure CPU time. Except for possible paging, there is no input or output operation or any inherent delay in the lock management.

Also, the CPU cost for doing lock management work is a function of:

- The frequency of locking requests issued by DB2
- The overall number of existing lock elements chained off a hash table entry at any time
- The frequency with which threads terminate or commit
- The frequency with which the IRLM deadlock cycle is initiated.

An installation should want to push as many SQL calls through its system as possible. Therefore, the first factor should not be considered as a tuning option, while the others are. The overall number of lock elements reflects the installation’s discipline with respect to UR size:

- Transaction threads should have short URs and not hold an excessive number of locks.
- Read-only SQL can use lock avoidance techniques to avoid locking. Other lock avoidance or reduction techniques are use of CURRENTDDATA(NO), Type 2 indexes, ACQUIRE(USE), RELEASE(DEALLOCATE), isolation, locksize, and frequent commits
- An unplanned query using uncommitted read isolation will not hold locks.

The major contributor to long chains of lock elements is usually the concurrent batch job with a poor COMMIT strategy and overly long URs. The solution, as discussed in 3.3, “Application Design and Programming Considerations” on page 72 is good program design and programming discipline.

Obviously with good discipline, the search time for chained lock elements, as well as the amounts of real and virtual storage, will be minimized with improved concurrency.

### 2.1.12 Two-Phase Commit Considerations

Programs running in IMS and CICS environments use what is known as *two-phase commit*. In these environments, the commit process is controlled by the coordinator (IMS or CICS) and DB2 is the participant in the process: Thus, the coordinator requests that the participant *Prepare to commit* and waits for the answer. If the answer is positive, the coordinator requests the commit and the participant commits the changes.

This process involves writes to the DB2 log; the writes may be dual.

Locks are released when all changes made to the database are secured; that is, the writes to the log containing the changes are completed. Therefore, the time needed for DB2 to write the log data sets is critical to the concurrency of the system; faster writes release locks sooner, decreasing the risk of suspensions. In the case of dual logging, writes to the second log data set follow the completion of the writes to the first log data set. Therefore, the total time
needed to complete the log writes is doubled, increasing the time needed for commit and therefore for the release of the locks held.

In two-phase commit environments, the commit time is increased by the fact that two writes to the log must be completed (corresponding to Phase 1 and Phase 2) and the increase is even more critical for dual logging (four writes). The two subsystems (coordinator and participant) have a message communications delay that must also be taken into account. However, the writes to the log are serial when the log control intervals are rewritten, and the writes to the log are parallel when log control intervals are written for the first time. All these facts must be considered when planning the system as logging time can be a concurrency limiting factor.

2.2 Restricted States

Restricted states can be set on DB2 objects:

- Table space
- Partitioned table space
- Index space
- Partitioned index space

by either a DB2 utility or a DB2 command.

IRLM does not know about restricted states.

Resetting a restricted state on a DB2 object can only be done by executing

- A new DB2 utility against the DB2 object
- START DATABASE (database) SPACENAM (table space or index space) ACCESS(FORCE)
- Using the Repair utility against the DB2 object.

Only CHKP, COPY, PSRCP and RECP restricted states can be reset.

Table 10 on page 37 shows all the possible restricted states, how they are set, and how they can be reset on the DB2 objects. Table 11 on page 38 shows what is checked.
Table 10. Restricted States

<table>
<thead>
<tr>
<th>Object State</th>
<th>State set by</th>
<th>State reset by</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHKP table space, table space partition</td>
<td>LOAD ENFORCE NO</td>
<td>CHECK DATA</td>
</tr>
<tr>
<td></td>
<td>LOAD REPLACE parent table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALTER TABLE ADD FOREIGN KEY</td>
<td></td>
</tr>
<tr>
<td>COPY table space, table space partition</td>
<td>LOAD, REORG LOG NO</td>
<td>COPY FULL YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPL table space, table space partition, index, index partition, logical index partition</td>
<td>SQL</td>
<td>START DATABASE, RECOVER, LOAD REPLACE, SQL DROP</td>
</tr>
<tr>
<td>OPENF table space, table space partition, index, index partition</td>
<td>Data set open with error</td>
<td>Open data set without error</td>
</tr>
<tr>
<td>PSRCP index space</td>
<td>RECOVER TS error</td>
<td>RECOVER</td>
</tr>
<tr>
<td>RECP table space, table space partition, index, index partition, logical index partition</td>
<td>RECOVER TS index error</td>
<td>RECOVER</td>
</tr>
<tr>
<td>REST table space, table space partition, index, index partition</td>
<td>START DATABASE</td>
<td>End of start</td>
</tr>
<tr>
<td>STOPE table space, table space partition, index, index partition</td>
<td>DB2 internal error</td>
<td>correct error</td>
</tr>
<tr>
<td>STOPP table space, table space partition, index, index partition</td>
<td>STOP DATABASE</td>
<td>Release Claims</td>
</tr>
</tbody>
</table>

During execution of a program, DB2 checks if any restricted states exist on the referenced DB2 objects.

DB2 has grouped the restricted states into two different levels of checking.

- **Level 1**
  - STOP
  - START RO
  - START UT
  - CHKP

- **Level 2**
  - UTUT
  - UTRO
  - UTRW
  - COPY
  - PSRCP
  - RECP
  - CHKP
  - STOP AT (COMMIT)

Level 1 restricted states are always checked while Level 2 restricted states are checked only when an application program makes a claim.

Table 11 on page 38 shows when the restricted state checks are done during SQL program processing.
### Table 11. Restricted State Check During SQL Program Process

<table>
<thead>
<tr>
<th></th>
<th>ACQUIRE(ALLOCATE) RELEASE(DEALLOCATE)</th>
<th>ACQUIRE(USE) RELEASE(COMMIT)</th>
<th>ACQUIRE(USE) RELEASE(DEALLOCATE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1 Restricted</td>
<td>First reference and once per execution</td>
<td>First reference and for each unit of recovery</td>
<td>First reference and once per execution</td>
</tr>
<tr>
<td>states</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2 Restricted</td>
<td>First reference and each unit of recovery</td>
<td>First reference and for each unit of recovery</td>
<td>First reference and for each unit of recovery</td>
</tr>
<tr>
<td>states</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ACQUIRE(ALLOCATE) is possible only if data base request modules (DBRM) are directly bound to a plan. Packages are always bound with ACQUIRE(USE). For a plan bound with ACQUIRE(ALLOCATE), all table-space locks that are needed in the program are acquired after the thread is created. Until the object is actually used, no claims are required. In this case a command or utility can set or hold a restricted state on an object not yet referenced in the application program.

If the plan or package is bound with RELEASE (DEALLOCATE), table-space locks are not released at commit point. The claim on the object is lost and a command or a utility can set a restricted state on this object.

Figure 14 on page 39 illustrates this.
Notes:

1. Activate application program.
2. Create DB2 thread and read DB2 plan locking information. IRLM locks all DB2 page sets referenced in the DB2 plan. No claims are taken on any of the page sets.
3. Application program accesses the first table in page set 1. DB2 takes a claim on page set 1. Additional locks can be acquired on page set 1.
4. Application program receives a OK SQLCODE.
5. Outside the program control, a restricted mode UTUT is set on page set 2. During execution of the utility a drain lock is set on page set 2.
6. The program references page set 2. DB2 checks level 1 and level 2 restricted states on page set 2. The page set 2 is marked UTUT and any program reference to page set 2 is not allowed, and the application program receives SQLCODE -911 or -913.
7. If SQLCODE is OK in Step 6, the second reference to page set 2 only checks whether any Level 1 restricted states exist. Level 2 restricted states are not checked.
8. At commit point, all claims on page sets are released. Locks remain until end of program.

Figure 14. Restricted State: Illustration

2.3 DB2 Subsystem Object Locking

There is some locking activity that is not related to user data but takes place in shared DB2 subsystem objects. This locking activity can be caused by user application programs. It can also be caused by DB2 system plans processing. You are never aware of most of this locking activity, although in some cases it can produce suspension problems in your system.

The DB2 system plans that generate locking activity are these:
• **BCT**, the plan used to perform service tasks and handle requests from DB2 resource managers. A lock issued by this plan is usually of short duration.

• **ACT**, the authorization plan used in the process of validating a user’s authority to access a specified plan. A lock issued by this plan is of short duration, but ACT is used frequently, with every create thread.

• **DSNBIND**, the plan used for all binds. It is major contributor to the length of time it takes to complete a bind.

• **DSNUTIL**, the plan used by DB2 utility control program DSNUTILB. The duration of locks held by this plan varies according to the specific DB2 utility function being invoked.

The DB2 subsystem objects involved in locking activity are:
- DB2 catalog and directory
- Skeleton cursor table(SKCT) and skeleton package table(SKPT)
- Database descriptors(DBDs).

### 2.3.1 Locks on the DB2 Catalog and Directory

The DB2 Catalog, which consists of tables of data about everything that is defined to the DB2 system, and the DB2 directory, whose tables contain information that DB2 uses to control normal operation, are subject to update activity that must be serialized.

Although the catalog and directory are designed to minimize contention among application processes, update activity may lead to locking problems in concurrency situations.

Five main activities can cause suspension problems in the DB2 catalog and directory:

• DB2 tasks that update DB2 catalog and directory tables. An example is writing to SYSLGRNG every time a table space or partition is opened and updated.

• The BIND, REBIND, and FREE process, which reads some DB2 catalog tables, such as SYSIBM.SYSTABLES, and inserts and/or updates others, such as SYSIBM.SYSPACKAGE.

• Data definition processes, such as CREATE, ALTER and DROP, which also insert, update, or delete catalog and directory entries (for example, DBD01).

• Data control processes, such as GRANT and REVOKE, which affect the concurrency of the catalog authorization tables (for example, SYSIBM.SYSUSERAUTH).

• Utility processing, which may also lock some directory tables (for example, SYSUTILX).

To avoid most of these concurrency problems, process all the data definition, data control and BIND activities in a dedicated window outside production hours. You can also greatly relieve limitations on concurrency by converting all catalog and directory indexes to Type 2, thus avoiding all index locking.
2.3.2 Locks on Skeleton Tables

The SKCT is located in the SCT02 table space in the directory and describes the structure of SQL statements in application plans. The SKPT is located in the SPT01 table space in the directory and applies to packages.

When you bind a plan, DB2 creates an SKCT in SCT02. When you bind a package, DB2 creates an SKPT in SPT01.

The following operations require exclusive control of the related SKCT and SKPT as they are updating them:

- Using BIND, REBIND, and FREE for the plan or package
- Dropping a resource or authority on which the plan or package depends
- In some cases, altering a resource or authority on which the plan or package depends.

When you run a plan or package, DB2 takes a shared lock on it, so that changes cannot be made to the object while it is being executed in an application program. Thus, the execution of a plan or package contends with operations listed above.

2.3.3 Locks on Database Descriptors

The database descriptors (DBDs) describe the DB2 databases. Each DBD fully describes a database and all of its objects, such as table spaces, tables, indexes, and referential relationships, and contains other access information. DBDs are located in the table space DBD01 in the directory.

Two main processes have to read and lock a DBD in shared mode:

- Dynamic SQL statements
- Active utilities.

Static SQL does not take any locks on the DBD if the DBD is cached in the environment descriptor management (EDM) pool. Therefore DB2 does not have to go to the DB2 directory table space to retrieve the DBD.

Each time a database or a dependent object definition is modified, the DBD object in the directory must be updated. This update process requires an exclusive lock on the DBD which is incompatible with dynamic SQL and utility execution. If the DBD is in use for a plan or package, dynamic SQL and utility execution are suspended.

Table 12 on page 42 summarizes the DB2 subsystem locking activity.
### Table 12. DB2 Subsystem Locking Activity

<table>
<thead>
<tr>
<th>Object of Locking</th>
<th>Static SQL</th>
<th>Dynamic SQL</th>
<th>BIND</th>
<th>Create Table</th>
<th>Alter Table</th>
<th>Drop Table Space</th>
<th>Grant</th>
<th>Revoke</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalog Table spaces</td>
<td>IS (a)</td>
<td>IS (b)</td>
<td>IX</td>
<td>IX</td>
<td>IX</td>
<td>IX</td>
<td>IX</td>
<td>IX</td>
</tr>
<tr>
<td>SKCT or SKPT</td>
<td>S</td>
<td>S</td>
<td>X</td>
<td>-</td>
<td>X (c)</td>
<td>X (d)</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>DBD</td>
<td>- (e)</td>
<td>S</td>
<td>S</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Notes:**

(a) IS locks on the catalog table spaces are held only for a short time to check EXECUTE authority if the plan or package is not public or the authorization list is not cached in the EDM pool.

(b) Except when checking EXECUTE authority (see Note a), IS locks on the catalog table spaces are held until the COMMIT point.

(c) SKCT or SKPT is marked invalid if a referential constraint (such as a new primary key or foreign key) is added or changed, or the AUDIT attribute is changed in the table.

(d) SKCT or SKPT is marked invalid as a result of a drop table space operation.

(e) If the DBD is not in the EDM pool, S-locks are acquired on the DBD table space, which effectively locks the DBD.

### 2.4 Claims and Drains

In an online DB2 environment the compatibility between transactions or batch programs and utilities or DB2 commands has always been an issue. Some utilities and commands require full control over an object, which is difficult to achieve as there is almost always a program that locks just one page in the table space. In many cases, users have to stop applications to be able to run utilities. DB2 overcomes this problem by using claims and drains, which enable DB2 utilities and commands to take over access to some objects independently of any transaction locks held on the objects.

A classic example is wanting to run QUIESCE to establish a point of consistency without stopping the WFI application. Another example of the need for the claim and drain process is IMS IFP or WFI region with plan bound with RELEASE(DEALLOCATE) enables utility drainer to break in.

#### 2.4.1 Claims

A claim registers to DB2 an SQL agent’s active use of an object. A claim count is maintained at a page set level (table space, index space, or partition). The counter is incremented when the application program first accesses the page set (data or an index) within a unit of work. Claim processing is not dependent on the ACQUIRE option of the BIND command or any other BIND, SQL, or installation option.

Claims are acquired only for partitions accessed within the commit scope. Claims are always released at a commit point. When claims are released, the claim counter is decremented, and the application must request a new claim if it...
needs to access the page set again. There is an exception: If the cursor is defined WITH HOLD, the claim is not released at commit point, provided there is a fetch on the cursor since the previous commit. This is true for read-only cursors. In the case of updatable cursors, the claim is downgraded (the counter is not decremented) and when required the claim is upgraded (the counter is incremented). To be very precise, claims held by updatable cursors are not handled any differently than claims held by read-only cursors. At commit, DB2 downgrades the duration of all claims held at "hold" duration to "commit" duration and then updates the claims with cursor-hold back to "hold" duration. The net effect is that all claims that do not need to be held now have commit duration and all claims that need to be held have "hold" duration. There is no concept of downgrading a write claim to a read claim.

Claimers declare their active presence on a page set by making a claim on it (which in essence means incrementing a count of claimers maintained for the page set). Typically, making a claim is as simple as incrementing a count. Once a claim action has been performed, SQL transactions can still request locks to serialize with other SQL transactions accessing the same data.

DB2 supports three claim classes. The claim classes indicate the manner in which the SQL uses the data. The claim classes are:

- **CS**: Read-only, with CS, UR, or RS isolation
- **RR**: Read-only, with RR isolation
- **WRITE**: Read and write accesses.

Different claim counters are kept at a page set level for each of the three classes.

### 2.4.2 Drains

A *drain* is the action that DB2 utilities and commands perform to wait for claimers; that is, to wait for the SQL transactions actively using the object to quit actively using them, that is, release their claim.

The drain acquires a special lock, which is known as a *drain lock*. It is a real IRLM lock, located in the page set, and blocks new claimers. It then waits for all current claims on the object to be released.

The effect of the drain is to make active applications quiescent by allowing them to reach a commit point while preventing new applications from accessing the object by making a new claim.

The real purpose of the drain lock is to implement the claimer’s wait for the drainer to leave. When the object is drained, an exclusive (X) drain lock is held on the object. The claimer then requests a shared (S) drain lock and waits for the drain lock to be released.

### 2.4.3 Interaction between Claims and Drains

Claimers have to request and wait for the drain lock to be released only when a drain is in control of the page set. Otherwise, the increment of the claim count by the claimer occurs without acquiring the lock. When claimer acquires the lock, it holds it only until the count is incremented. The claim is granted and the claim count is incremented as long as no incompatible exception states are present.
When no more claims are present in the object (the claim count is set to zero), the DB2 utility or command takes control of the object.

There are three drains that correspond to the three claim classes:

- **CS**: Drains CS claims.
- **RR**: Drains RR claims.
- **WRITE**: Drains WRITE claims.

A drainer does not always need to have complete control of the object. It could drain:

- Only the WRITE claims
- Only the RR claims
- All claim classes.

The drain lasts until the DB2 command occurs or the utility finishes processing.

So far we have seen that utilities and commands do not follow the normal IRLM locking process to serialize with running transactions, and thus they do not claim objects. There is one exception to this statement: `COPY SHRLEVEL(CHANGE)` and `RUNSTATS SHRLEVEL(CHANGE)` request a read claim; they do not drain any class.

### 2.4.3.1 Compatibility Rules

Claim and drain processing can occur on the following DB2 objects:

- Simple table spaces
- Segmented table spaces
- Partitions of table spaces
- Nonpartitioned index spaces
- Partitions of index spaces
- Logical partitions of nonpartitioned Type 2 indexes.

The concurrent execution of two different utilities is not typically controlled by the drain and claim process, except for a set of compatibility rules that dictate that a new utility can start only after it is checked against all other utilities running on the same object and found to be compatible.

Basically, some utilities such as the LOAD utility require exclusive read and write access to the object. Others, such as `COPY SHRLEVEL(REFERENCE)`, allow concurrent read-only access.

For information about the compatibility of utilities, please refer to each utility description in the *DB2 Utility Guide and Reference*.

### 2.4.3.2 Example:

The example in Figure 15 on page 45 illustrates a situation where transactions and utilities coexist.

In our case, during a certain period of time four SQL programs run concurrently and access the same table space partition with three different utilities. All claim, drain, and interutility compatibility mechanisms play a role in the example.
**Time T1:** Application 1 executes the following SQL statement by taking a write claim on Partition 1 of the target table space, and locks some of its rows or pages.

```sql
UPDATE USER.STAFF
SET ZIPCODE = 1100
WHERE ZIPCODE = 1000
WITH CS;
```

Writers (updaters) make an implicit read claim, which can be either CS read claim or RR read claim. Application 1 makes an implicit CS read claim because the UPDATE statement has the WITH CS clause. The write claim count on Partition 1 of the target table space is 1 and the CS read claim count on Partition 1 of the target table space is 1.

**Time T2:** Application 2 executes the following SQL statement by taking a CS read claim on Partition 1 of the target table space and uses lock avoidance techniques to avoid IRLM locking.
SELECT NAME, ADDRESS, PHONE
FROM USER.STAFF
WHERE DEPT = 30
FOR FETCH ONLY
WITH CS;

The write claim count on Partition 1 of the target table space is 1 and the CS read claim count on Partition 1 of the target table space is 2.

_Time T3:_ Application 3 executes the following SQL statement by making a write claim on Partition 1 of the target table space and locking some of its rows or pages.

DELETE
FROM USER.STAFF
WHERE DEPT = 70
WITH CS;

Writers (updaters) make an implicit read claim, which can be either CS read claim or RR read claim. Application 3 makes an implicit CS read claim because the DELETE statement has the WITH CS clause. The write claim count on Partition 1 of the target table space is 2 and the CS read claim count on Partition 1 of the target table space is 3.

_Time T4:_ Utility 1 (COPY SHRLEVEL(REFERENCE)) on Partition 1 of the target table space starts and drains all write classes, although it allows concurrent SQL read claimers. Since the write claim count on Partition 1 of the target table space is 2 because of Applications 1 and 3, the utility goes into a suspended state.

_Time T5:_ Application 3 is suspended because Application 1 is holding an X-lock on a page that it needs to access.

_Time T6:_ Application 1 commits, freeing all locks that it held. The write claim count on Partition 1 of the target table space is 1, and the CS read claim count on Partition 1 of the target table space is 2. Application 3, which was suspended, resumes its execution.

_Time T7:_ Application 3 commits, freeing all locks that it held. The write claim count on Partition 1 of the target table space is 0, and the CS read claim count on Partition 1 of the target table space is 1. Utility 1 can now write-drain Partition 1 and start. Utility 1 takes IX drain lock. The write claim count on Partition 1 of the target table space is 0, and the CS read claim count on Partition 1 of the target table space is 1.

_Time T8:_ Application 4 requests S drain lock on Partition 1 of the table space and gets suspended because the S drain lock is not compatible with the IX drain lock held by Utility 1. The SQL to be executed is:
UPDATE USER.STAFF
SET DEPT = 20
WHERE NAME = 'RYAN' WITH CS;

**Time T9:** Utility 2 (RUNSTATS TABLESPACE SHRLEVEL(REFERENCE)) requests IX drain lock on Partition 1 of the target table space. Although this request is compatible with the IX drain lock already held on Partition 1 of the target table space, IRLM queues this request behind the S drain lock request of Application 4.

**Time Ta:** Utility 3 (MODIFY RECOVERY TABLESPACE) is started on Partition 1. This utility is not compatible with Utility 1 (COPY). As a result, it fails.

**Time Tb:** Application 2 commits, freeing all locks that it held. The write claim count on Partition 1 of the target table space is 0, and the CS read claim count on Partition 1 of the target table space is 0.

**Time Tc:** Utility 1 completes. Application 4, which was suspended, gets the S drain lock on Partition 1 of the target table space and resumes. Writers (updaters) make an implicit read claim, which can be either CS read claim or RR read claim. Application 4 makes an implicit CS read claim because the UPDATE statement has the WITH CS clause. The write claim count on Partition 1 of the target table space is 1 and the CS read claim count on Partition 1 of the target table space is 1.

**TimeTd:** Application 4 commits, freeing all locks that it held. The write claim count on Partition 1 of the target table space is 0, and the CS read claim count on Partition 1 of the target table space is 0. Utility 2 which was queued behind Application 4 gets the IX drain lock on Partition 1 of the target table space and starts executing.
### Notes

**Times T5-T6**
During this period of time, Application 3 is waiting for a page that is held by Application 1. Assuming an installation IRLM Resource Wait Timeout value of 25 and Deadlock Time value of 10, the maximum time Application 3 can be suspended without being backed out is:

\[
\text{Upper}(25 / 10) \times 10 = 30 \text{ seconds}
\]

**Times T4-T7**
Assuming an installation IRLM Resource Wait Timeout value of 25, Deadlock Time value of 10, and Utility Timeout value of 6, the maximum time Utility 1 waits without being timed out is:

\[
30 \text{ seconds} \times 6 = 180 \text{ seconds}
\]

**Times T8-Tc**
During this time period, Application 4 is waiting for an S drain lock which is incompatible with the IX drain lock held by Utility 1. Assuming an installation IRLM Resource Wait Timeout value of 25 and Deadlock Time value of 10, the maximum time Application 4 can be suspended without being backed out is:

\[
\text{Upper}(25 / 10) \times 10 = 30 \text{ seconds}
\]

**Times T9-Td**
During this time period, Utility 2 is waiting behind application 4 for the IX drain lock on Partition 1 of the target table space. Assuming an installation IRLM Resource Wait Timeout value of 25, Deadlock Time value of 10, and Utility Timeout value of 6, the maximum time Utility 2 waits without being timed out is:

\[
30 \text{ seconds} \times 6 = 180 \text{ seconds}
\]

---

### 2.5 Partition Independence

Partition independence provides parallel processing support on the different partitions of a partitioned table space and its related indexes. Partition independence allows parallel execution of all utility jobs, SQL applications, and DB2 commands as long as they address different partitions of a given table space.

Partition independence provides the following benefits:

- It is possible to independently access different partitions of a table space or index:
  - Different utilities can run on different partitions concurrently.
  - SQL applications run on partitions not in use by utilities.
  - It is possible to isolate unavailability of data to a single partition.
- It is possible to start and stop a single partition.
- It is possible to independently access logical partitions of a nonpartitioned index.
- Different utilities can access the nonpartitioned index concurrently.
- SQL applications can access the logical partitions not in use by utilities.
- It is possible to start and stop a logical partition.

A logical partition is the set of key and row identification (RID) pairs in a nonpartitioned index that are associated with a particular partition. The RID contains the partition number to which the key entry belongs. Logical partitions are determined on this number.

Logical partitions are supported only for nonpartitioned indexes created as Type 2 indexes.

Figure 16 shows an example of partition independence.

Figure 16 shows that the LOAD utility BUILD phases and SQL applications can operate concurrently on the different partitions, including the logical partitions of a nonpartitioned index. A logical partition constitutes an independent entity that can be drained without impacting other logical partitions.
2.6 Latches

Latches and locks are both used to control access to shared DB2 data.

While locks ensure that the data is logically consistent—that is, committed and not subject to rollback or abort—latches guarantee that the page is physically consistent and is not in a partially changed state.

The structure of a page may be inconsistent while changes (inserts, updates) are being made. During the time when the page may not be consistent, a page latch is used to prevent access to the page.

Page latching instead of locking is used for concurrency improvement, not necessarily for CPU time reduction. Depending on the specific situation, there may be either a slight increase or a reduction in CPU time as a result of using page latch instead of lock.

Page latches are held for a shorter duration only while a program is accessing a page, cannot last after a commit point (locks taken in a CURSOR WITH HOLD do last) and are requested in such a way that no timeouts or deadlocks involving latches can occur.

To prevent latch-lock deadlocks, a latch is never held by a process while waiting for an IRLM lock. However, a process can hold more than one latch at a time. When an update to the data page requires information to be updated in the space map page, DB2 latches the data page before latching the space map page to avoid the possibility of a deadlatch.

There are only two types of page latches: S (shared) and X (exclusive).

The S-latch is used to avoid acquiring IRLM locks. Readers use the S-latch to ensure the physical consistency of the page. If the commit state of the data page or row cannot be determined, an IRLM lock is requested.

The X-latch is used to serialize access with S-latches and other X-latches for the update process. Before a process can update the data on the page, it must acquire both an X-latch and an X-lock.

Table 13 summarizes the differences between the latch and the lock.

<table>
<thead>
<tr>
<th>Latch</th>
<th>Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensures physical consistency and is inexpensive.</td>
<td>Ensures logical consistency but is relatively expensive.</td>
</tr>
<tr>
<td>Deadlatch is not possible.</td>
<td>Deadlock is detected.</td>
</tr>
<tr>
<td>Latch can be suspended and timeout detection is required.</td>
<td>Lock can be suspended and timeout detection is required.</td>
</tr>
<tr>
<td>The object of a latch is a page.</td>
<td>The object of a lock is a table space, table, page, or row.</td>
</tr>
<tr>
<td>Duration is very short.</td>
<td>Duration is short to long.</td>
</tr>
<tr>
<td>Latch types are S and X.</td>
<td>Lock types are IS, IX, SIX, S, U, and X.</td>
</tr>
</tbody>
</table>
2.7 Type 1 and Type 2 Index Considerations

To provide for more concurrency, better performance, and high data availability and also to introduce new functions, DB2 V4 implements the Type 2 index. Type 2 indexes need no locks on their pages. A lock on the data page or row locks the index key. DB2 latches the Type 2 index page to preserve the physical consistency of the index page.

DB2 also supports the previous existing index, which is now called Type 1, to distinguish it from the new Type 2 index.

The following list is a summary of the main advantages users can expect from the conversion of Type 1 indexes to Type 2 indexes:

- **Better concurrency**
  - *Elimination of index page locks*
    
    Most of the contention with Type 1 indexes is caused by locking index pages; a lock on an index page could potentially make many key entries unavailable to other users. Eliminating index locks improves data availability and concurrency.

    With Type 2 indexes, locks are not acquired on index pages during retrieval, insert, and delete processing or structure modification. The lock suspensions, timeouts, and possibly deadlocks are reduced, thereby increasing concurrency.

    DB2 uses lock avoidance on data pages in programs with ISOLATION(CS) and CURRENTDATA(NO).

  - *Elimination of index subpages*

    To reduce the impact of index locking, the Type 1 index locking unit is called subpage. Index subpages are specified at create index time, and are the number of locking units for each physical index page. Number of subpages can be 1, 2, 4, 8, and 16. If number of subpages is 1, it means there are no subpages, the entire index leaf page is locked, and provides least concurrency. If number of subpages is 16, it means that only 1/16 index keys in physical index leaf page is locked, and provides maximum concurrency.

    Since the Type 2 index promotes even higher concurrency by not locking index pages, subpages have been eliminated in Type 2 indexes.

  - *Row locking*

    Row-locking granularity allows for improved concurrency of update operations while preserving logical data integrity. Because of the small size of lock granularity, lock suspensions and timeouts can be reduced. However, deadlocks may increase. For example, it is impossible to have deadlocks among rows on the same page if page locking is used, but it is possible with row locking.

    DB2 applications do not have to be modified to take advantage of row locking. All you have to do is modify the lock size of the selected table spaces.

  - *Uncommitted reads*

    Uncommitted read, also known as “dirty read,” is the new isolation level in DB2 V4. ISOLATION(UR) is valid for read-only operations, such as
SELECT, SELECT INTO, and FETCH using a read-only cursor. Data retrieved using uncommitted read isolation may not be committed and should be considered dirty, as the program may have been reading through existing locks.

**Concurrent structure modification**

A structure modification is an index operation, such as a page split or page deletion that updates the nonleaf portion of the index tree and changes the structure of the tree.

For a Type 1 index, the entire index tree is locked and unavailable until the structure modification is completed. The modified pages are locked until the commit point. Index-tree traversal is prevented.

For a Type 2 index, locking is also required, but the granularity is improved, so that other applications can traverse the index tree while the structure is being modified. Modified pages are locked for a shorter duration.

**Deletion of index entries**

Unless the table or table space is locked in mode X, DELETE does not immediately delete an entry; it only marks it for deletion. The space in the index cannot be used to insert another entry until the data is committed. Hence, if the deletion is rolled back, the deleted entry can be reinstated without splitting the page. After the deletion is committed, a marked entry can be physically deleted when space is needed for an insert or when the percentage of entries marked on the page reaches an internally defined threshold.

**DASD allocation**

A Type 2 index is created as a two-level index with a root page that points to an empty leaf page.

You may experience less or more DASD space allocation for Type 2 indexes, depending on the following factors:

**Less DASD**

The absence of subpages and the use of key suffix truncation in nonleaf pages mean that less DASD space allocation is required. Other contributing factors are these:

- When inserts are made to the end of indexes, no space is wasted during page splits.
- In nonunique indexes there is no need to add columns for uniqueness and no need to repeat the key value every 255 RIDs.
- In both unique and nonunique indexes there is no need to have high PCTFREE or dummy columns to avoid index contention because of the absence of subpages and key suffix truncation in nonleaf pages.

**More DASD**

The new page structure of Type 2 indexes requires more DASD space allocation:

- Two extra bytes per key for the key map are needed to preserve the key order.
- One extra byte per RID is used for marking the delete and lock avoidance support.

An average increase of 10% to 20% in the number of Type 2 index leaf pages can be expected because of two extra bytes per key and one extra byte per RID. The increase in percentage depends on key length: the smaller the key length, the larger the percentage increase.

The net increase in the number leaf pages is hopefully reduced because of the three factors that contribute to less DASD space allocation.

The calculation of Type 2 index DASD space requirements is discussed in Chapter 2 of the *IBM DATABASE 2 for MVS/ESA Version 4 for MVS/ESA Version 4 Installation Guide*. The IBM tool, Estimator for DB2, also provides the space calculation.

- **Support for CP parallelism**

  Complex queries that require a lot of CP processing qualify for query CP parallelism. DB2 starts multiple coordinated tasks to reduce the elapsed time required to complete the query.

- **Logical partition independence**

  Programs and utilities can benefit from the availability advantages of full partition independence, with support for logical partitions in nonpartitioned indexes defined on partitioned table spaces.

Although the general recommendation is to gradually migrate all existing application indexes to Type 2, coexistence of Type 1 and Type 2 indexes may be required during the transition phase. This is the reason why, though Type 2 index is assumed in general, this section is dedicated to discussing a few aspects about Type 1 indexes.

In case you are considering retaining Type 1 indexes, you must take into consideration that the following DB2 V4 enhancements are not supported by Type 1 indexes:

- Row locking
- Uncommitted reads ("dirty reads")
- Logical partition independence
- Query CP parallelism.

If you are planning to migrate your existing indexes to Type 2, DB2 gives you the facility of doing it with a single SQL statement:

`ALTER INDEX index_name CONVERT TO TYPE 2`

The index is left in recover pending state and the index change does not take place until the index is recovered or reorganized.

The following examples show the different behavior of Type 1 and Type 2 indexes in some typical access paths. Assume there are no subpages for the Type 1 indexes.
2.7.1.1 Example 1: Index-Based Scan
This is a typical example of an index used in an access path. DB2 uses the index structure to locate the data pages containing rows that satisfy the part of the predicate involving indexed columns. Assume also that no lock-avoidance techniques can be used, so locking is required.

Figure 17 shows the impact of Type 1 and Type 2 indexes on locking in an index-based scan.

![Figure 17. Locking in an Index-Based Scan Access](image)

With a Type 1 index, index subpages, and data pages are S-locked, while data pages are S-latched too. With a Type 2 index, only data pages are locked while index pages are latched. With Type 2 indexes, concurrency can be improved further by using row locking; in this case, only data rows that are accessed are locked and the remaining rows in the page can be accessed by other concurrent processes.

2.7.1.2 Example 2: Index-Only Scan
This is similar to the index-based scan, but here DB2 can satisfy the SQL request without accessing data pages. This is the typical case of a \textit{SELECT C1, C2} or \textit{SELECT COUNT(*)} where the predicate conditions refer only to index columns or do not exist. Assume also that no lock-avoidance techniques can be used.

Figure 18 on page 55 shows the impact of Type 1 and Type 2 indexes on locking in an index-only scan.
With a Type 1 index, only index subpages are locked, while data pages are not accessed and are thus not locked. With Type 2 indexes, index pages are latched but data consistency also requires data pages to be locked (though not accessed). With row locking, data rows are locked.

This is the only case when the use of Type 2 indexes can reduce concurrency. With a Type 1 index, in the best case only 1/16th of the index entries are locked and programs not using this index are not affected; with a Type 2 index, in the worst case, the entire data page is locked.

2.7.1.3 Example 3: Changing the Database
Updaters (that is, SQL INSERT, UPDATE, or DELETE) require higher locking granularity than readers.

Figure 19 on page 56 shows the impact of Type 1 and Type 2 indexes on locking when inserting new data into the database.
With a Type 1 index, an X-lock is acquired both on the data page and on the index subpage updated; with a Type 2 index, an X-lock is acquired on the data page or row, but updates to index pages are serialized with latches. This serialization greatly reduces the insert contention that is critical in the case of ascending keys, where all inserts concentrate in the last page, by eliminating the index-locking overhead.

2.7.2 Application Scenarios

Type 2 index generally improves the system concurrency and eases and even eliminates existing Type 1 index concurrency problems.

**Hot spot insertions:** This problem occurs while doing many key insertions in ascending order, if the index key is a timestamp, for instance.

When a new insertion comes, if there is no more space in the last page, a Type 1 index allocates a new page and splits the last page in two, moving half of the entries to the new page and inserting the new key in the new page allocated.

Figure 20 on page 57 shows how the insertions into hot spots are done with Type 1 and Type 2 indexes.
With a Type 1 index, an X-lock is acquired on the index subpage and another X-lock on the data page. In case of page splitting, there is a need to go into the structure modification process, which means that the index tree is not available during the change. There is no concurrency at all in this process, which leads many times to timeouts and deadlocks that are very difficult to avoid from an application point of view.

This process also leads to having all leaf pages (the last pages of the index tree) half full, wasting a lot of space.

Type 2 indexes do not move half of the entries to the new page, thus leading to full index leaf pages. Moreover, Type 2 indexes do not lock index pages during insertions and, as concurrent access is permitted during structure modifications, the so-called hot spot problem is minimized.

**The phantom problem:** This well known theoretical problem refers to the case where a physically deleted index key in a unique index is skipped by a repeatable-read reader-scanning program, and then the key reappears later when the delete program aborts and rolls back all the changes.

Type 1 and Type 2 indexes solve this problem using different approaches.
The Type 1 index always physically erases deleted index entries and X-locks the affected index subpages, preventing other programs from accessing them.

Type 2 indexes solve the phantom problem by pseudo deletion of index entries: These entries can be physically removed only when the deleting program commits, thus eliminating the phantom problem.

The Type 1 index approach requires the locking of index keys located in the subpage but not involved in the process, while the Type 2 index does not lock any index key.

*Delete from Nonunique indexes:* Type 1 indexes do not keep RIDs in order, so a deletion of one entry means a sequential scan of many leaf pages, with much lock and unlock activity (the Type 1 index locks index pages) and CPU-consuming getpage and release page activity.

Type 2 indexes keep RIDs in an ordered ascending sequence that allows a direct addressing to the leaf page containing the entry, which is much more efficient.

### 2.7.3 End-of-File and Mass-Delete Type 2 Index Lock

Two types of lock are special to Type 2 indexes: the End-of-File (EOF) lock and the mass-delete lock. These are new locks created for the Type 2 index. A Type 1 index handles the EOF problem by using index page locks. Since the Type 2 index acquires page latches only, there is need for an EOF index lock. The mass-delete lock is used to serialize against transactions using uncommitted read isolation, that do not conflict with the table space lock.

#### 2.7.3.1 Locking for RR Transactions: End-Of-File Lock

The EOF lock is related to the provision of repeatable read (RR) semantics when accessing a table using a Type 2 index.

Consider the following situation: Program A is bound with ISOLATION(RR), and is executing the following SQL statement:

```sql
SELECT COLA, COLB
FROM USERID.TABLE
WHERE COLC < 100;
```

For simplicity, assume row locking is used for the table space containing the USERID.TABLE. Consider that the USERID.TABLE contains the values 10, 70, 90, 110, and 130 for key COLC.

Concurrently, Program B is assumed to be executing the following SQL statement:

```sql
INSERT INTO USERID.TABLE
(COLA, COLB, COLC)
VALUES(‘COLA’, ‘COLB’, 95);
```

Figure 21 on page 59 illustrates the steps executed.
If Program A is S-locking table rows with key COLC up to value 90, the Program B insert statement would execute and commit perfectly; But this would break the RR semantics for program A, as a subsequent read of the same table would provide an additional value for key 95.

To avoid these problems, RR readers take an additional S-lock in the first row that does not qualify as the RR predicate; that is, in this example, key 110.

Inserting programs always check for the existence of an S-lock with an RR attribute in the next greater key than the one to be inserted: If an S-lock exists, the insert process is suspended until the RR reader commits and releases the lock as insertion would violate the RR scanner’s repeatability; If not, the insert is allowed.

The problem is the same when the RR program reads up to the last key value of the table. An example of this case is:

```sql
SELECT COLA, COLB  
FROM USERID.TABLE  
WHERE COLC > 100;
```
Now there is no greater key to be locked to make sure the RR semantics are respected.

The EOF lock is used when an RR reader is executed and the highest key in the index satisfies the query. The RR scanner needs to lock the first key that does not qualify (and no more keys exist), so the reader locks the end of index.

This means that until the RR reader commits, no high keys can be inserted in the index, as the EOF index lock prevents insertion of any key value greater than the highest existing key value.

Figure 22 shows this.

---

**2.7.3.2 Mass-Delete Lock**

Mass delete refers to a SQL DELETE statement without a *WHERE* clause, so all index entries and all rows are deleted; for example,

\[
\text{DELETE FROM USERID.TABLE}
\]
The mass-delete operation has been the subject of various performance improvements since DB2 V2R1.

Prior to DB2 V2, performing mass delete required that every data row be accessed and deleted.

In V2R1, segmented table space was introduced and mass delete in such table spaces was optimized. To delete everything in a table at that point, only the space map pages needed to be accessed and the page segments were simply marked "deallocated."

The mass-delete lock was introduced to serialize a mass-delete operation with insert. The mass-deleter would get an exclusive mass-delete lock. Then, Insert (if a user wanted to allocate a deallocated segment of pages) would ask for a shared mass-delete lock to make sure the mass deleter that released the segment had committed, eliminating any possibility that the mass deleter could roll back and reallocate the segment.

In DB2 V4, with Type 2 indexes and support for uncommitted read isolation, new conflicting possibilities open: mass delete in index can free pages for reuse. Upon reuse, pages that were leaf pages can become nonleaf pages, and vice versa. To overcome this, DB2 serializes uncommitted readers on Type 2 indexes with mass delete using a mass-delete lock.
Chapter 3. How to Prevent Locking Problems

Almost all locking problems occur because of application considerations such as:

- Index, table, and table space design
- Program logic
- SQL statements.

In most cases, DSNZPARM parameters such as timeout value, deadlock detection interval, and lock escalation thresholds do not cause locking problems. The objectives of this chapter are to make the application developer aware of the most common DB2 locking problems and to enable the application developer to actively prevent locking problems.

The application developer has to consider locking problems when participating in database design, developing programs, and constructing SQL statements.

The application programmer must always try to minimize lock duration, CPU use, and I/O activity, without losing the logical integrity of data in the database. In most cases, the application programmer also has to allow for maximizing the concurrent access to data by users or programs.

How much effort the application programmer has to put in on these different issues of course depends on the business needs of the environment.

3.1 Database Design

This section discusses physical design and implementation for table spaces, tables, and indexes, but focusing mainly on DB2 locking and concurrency considerations.

Physical database design is the outcome from the logical database design. Throughout the transformation (for instance, denormalization of data) from logical to physical database design, decisions have to be taken that influence DB2 locking and overall performance.

A highly denormalized design results in reducing the number of tables in a system. But duplication of entity fields in different tables accompanies the reduction. With a denormalized design, we must update the same entity field in different tables. That is, we have to do more DB2 work (update or insert) in the same logical unit of work. More DB2 work produces more DB2 locking in these situations.

On the other hand, read-only operations are often simpler (not so many table joins), resulting in less DB2 work (including DB2 locking) when we use a denormalized design.

During the database design, we must always identify what are called Hot pages or even Hot rows, pages or rows that are locked for more than 10% of the time. Eliminating these may have a considerable effect on the table design in the database. We always have to ask where there is a possibility of hot pages, as they can result in significant lock waits.
Hot pages can be detected very early when the table is being designed, with the quality card proposed in 3.4, "Quality Card Analysis to Determine the Probability of Locking" on page 87.

Common ways to prevent hot-page problems are:
- Clustering index design
- Row locking.

3.1.1 Table Space Considerations
When creating a table space, consider the following factors because they affect DB2 locking:
- Type of table space
- Page size
- Lock size
- Free space
- Maximum number of page or row locks
- Data compression

3.1.1.1 Page Size
BUFFERPOOL identifies the buffer pool to be used for the table space and determines the page size of the table space. For 4KB page buffer pools (BP0 to BP49), the page size is 4KB. For 32KB page buffer pools (BP32K to BP32K9), the page size is 32KB. If the BUFFERPOOL clause is not specified, the default buffer pool of the database is used. Using page locking in table spaces defined with 32KB pages increases concurrency problems. Each page lock locks more rows in a 32KB page than in a 4KB page. We recommend that you use 4KB pages. Use 32KB pages only when you can justify defining rows longer than 4KB.

3.1.1.2 Lock Size
LOCKSIZE specifies the size of locks used within the table space. LOCKSIZE ANY is the default for CREATE TABLESPACE, allowing DB2 to choose the locksize, usually LOCKSIZE PAGE and LOCKMAX SYSTEM. Before you choose LOCKSIZE TABLESPACE or LOCKSIZE TABLE, you consider whether concurrency is likely to be affected.

LOCKSIZE ROW increases concurrency. However, weigh this advantage against the CPU overhead for locking.

3.1.1.3 Maximum Number of Page or Row Locks
LOCKMAX specifies the maximum number of page or row locks an application process can hold simultaneously in the table space. If a program requests more than that number, locks are escalated. The page or row locks are released and the intent lock on the table space or segmented table is promoted to S or X mode. You can specify one of the following:
- LOCKMAX n (n can be from 1 to 2147483647). Specifies the maximum number of page or row locks that a single application process can hold on the table space before those locks are escalated.
- LOCKMAX SYSTEM. Specifies that n is effectively equal to the system default value (entry LOCKS PER TABLE(SPACE) on the DB2 installation panel...
DSNTIPJ corresponding to the DSNZPARM NUMLKTS) set by the DB2 database administrator.

- LOCKMAX 0. Disables lock escalation entirely. That is, zero indicates that the number of locks allowed is unlimited and escalation does not occur.

We recommend that you use the LOCKMAX SYSTEM in the beginning. Later on, if you experience lock escalations in your environment, consider changing to LOCKMAX n (n has to be greater than the default value).

3.1.1.4 Data Compression
A table space contains more rows in a page when the data is compressed compared to the number of rows in a page when the data is not compressed. Consequently, DB2 uses fewer locks on a table space when the data is compressed. However, there is less concurrency. Compressing data can result in a higher processing cost, depending on the actual SQL work load.

If you have concurrency problems with table spaces when the data is compressed, then consider using row locks.

3.1.1.5 Free Space
FREEPAGE and PCTFREE are used to control free space. Assuming that there is very little SQL INSERT activity, more free space means fewer rows per page, hence fewer rows locked by a page lock. There is trade-off, however. It takes extra space, and can degrade scanning performance for clustered data. Another (and maybe a better) solution is to use row locking instead of increasing the free space.

3.1.1.6 Type of Table Space
You can choose from three types of table spaces:

- Simple table space, which can contain more than one table. The simple table consists of pages, and each page can contain rows from many tables.

- Segmented table space, which can also contain more than one table. The segmented table space consists of groups of pages called segments. Each segment is dedicated to holding rows of a single table.

- Partitioned table space, which can contain only a single table. The partitioned table space is divided into partitions based on the key range of a nominated partitioning index. Each such partition can be processed separately by utilities, possibly allowing concurrent access by other utilities and SQL statements.

**Simple table space:** For simple table spaces containing more than one table, we have no provision for locking a single table. If LOCK TABLE is specified for a table contained in a simple table space, then all tables in the table space are locked: A page lock in a simple table space can mean that data from other tables is locked as well.: Use of simple table space with multiple tables should be avoided to prevent problems in DB2 locking and concurrency.

**Segmented table space:** For segmented table space containing more than one table, we can lock one table, page or row. The lock does not interfere with access to segments of other tables. This is generally true, although some DB2 utilities operate on a table space or partition basis only. These utilities include LOAD REPLACE, RECOVER, COPY and REORG.:
Partitioned table space: Partitioned table space is generally of interest only for large tables with many rows. In terms of DB2 locking and concurrency the most important consideration is the ability to allow a DB2 utility to execute on part of the data at a time, while allowing concurrent access by other DB2 utilities or applications on other partitions. In this way, several DB2 utility jobs could, for example, load all partitions of a table space concurrently.

3.1.2 Table Considerations

For tables, you have no direct access to the locking mechanism. There are no keywords in the CREATE TABLE statement that directly influence locking. However, referential integrity rules defined on tables can implicitly have a great impact on locking and concurrency.

A delete operation on a parent table must acquire locks on the dependent tables, or at least on their indexes. This locking can make those tables less readily available for concurrent use.

Locks on these table spaces and index spaces are acquired only when they are used. Table spaces and index spaces that are required to be accessed only for enforcing referential constraints are not affected by the ACQUIRE(ALLOCATE) option of the BIND PLAN command. How long the locks are acquired for referential integrity checking operations is also a performance consideration. With the RELEASE(DEALLOCATE) option, they are released, like all other table spaces and index spaces, only when the plan terminates.

3.1.3 Index Considerations

Consider using Type 2 indexes to minimize locking and increase concurrency.

INSERT, UPDATE, or DELETE operations require a lock on every affected page or subpage of a Type 1 index, but not on a Type 2 index page. With a Type 2 index, only the affected data pages or rows are locked. Because there are usually fewer rows to a data page than there are index entries to an index page or subpage, locking only the data when you lock pages is likely to cause less contention than locking the index. Locking only data rows is likely to cause even less contention.

Changes can split an index leaf page, which locks out concurrent access to a Type 1 index, and if the page has more than one subpage, there can be additional splitting for subpages. Type 2 indexes have no subpages.

Probably the most common locking problem with Type 1 indexes is the contention on the last leaf page because of an ascending key. This problem is practically eliminated with Type 2 indexes: with isolation CS, the leaf pages are only latched, and not locked. The duration of a latch is likely to be less than one millisecond. Thus, it would take more than 100 inserting transactions per second to make the last leaf page hot, that is, the leaf page is locked for more than 10% of the time.

A Type 2 index is always needed for:
- Row locking on a table space
- Uncommitted read isolation for an access path
- Processing query by multiple parallel tasks (query CP parallelism)
- Concurrent access to logical partitions in a nonpartitioned index.
If you insert data with a constantly increasing key, use a Type 2 index. The Type 1 index splits the last index page in half and adds the new key at the end of the list of entries. It continues to add new keys after that, wasting one-half of the old split page. The Type 2 index merely adds the new highest key to the top of a new page, without splitting the page.

When inserting at the end of a Type 2 index (inserting data with a consistently increasing key or inserting keys higher than any existing key), DB2 V4 fills up the last index page and adds the new key to the top of a new, empty index page without splitting the page. This special insert processing is implemented for both unique and nonunique indexes. It also applies for an insert at the end of a partitioned index. For ascending indexes containing null values, an insert of the highest key and RID value in the index still uses the special insert processing. However, the null byte is stored as X'FF' at the front of the key, so all keys containing null values sort at the end of the index.

If a key without any null values is inserted, it is before the start of the nulls, so the special insert processing is not used. However, if a key that contains null values is inserted and it is also the highest key entry in the index, the special insert processing is used.

During insert, a lock on the next key is acquired to check for repeatable-read users, and an end-of-file index lock is established if there is a repeatable-read claim on this index.

### 3.2 Bind Option Considerations

This section discusses the advantages and disadvantages of using different bind options as input to the package or plan bind process. The following package and plan bind options influence locking and concurrency of data:

- **ACQUIRE**
- **RELEASE**
- **ISOLATION**
- **CURRENTDATA**

#### 3.2.1 ACQUIRE and RELEASE

The combination of ACQUIRE and RELEASE options determines the duration of table or table space locks. If you specify USE for ACQUIRE, then you can specify either COMMIT or DEALLOCATE for RELEASE. If you specify ALLOCATE for ACQUIRE, then you can specify only DEALLOCATE for RELEASE. If not specified, the defaults used are ACQUIRE USE and RELEASE COMMIT. For package bind, ACQUIRE USE is always used implicitly and RELEASE can be either COMMIT or DEALLOCATE.

#### 3.2.1.1 ACQUIRE(ALLOCATE) and RELEASE(DEALLOCATE)

Avoids deadlocks by locking out other users as soon as the program starts to run. The advantages are these:

- All tables or table spaces used in DBRM bound directly to the plan are locked when the plan is allocated. All tables or table spaces are unlocked only when the plan terminates.
• The locks used are the most restrictive needed to execute all SQL statements in the plan, regardless of whether the statements are actually executed.

• Restricted states for a page set are not checked until the page set is accessed. Locking when the plan is allocated ensures that the job is compatible with other concurrent SQL jobs. Waiting until the first access to check restricted states provides greater availability; however, it is possible that an SQL transaction could:
  − Hold a lock on a page set or partition that is stopped
  − Acquire a lock on a page set or partition that is started for DB2 utility access only, ACCESS(UT)
  − Acquire an exclusive lock (IX, X) on a page set or partition that is started for read-only access, ACCESS(RO), thus prohibiting access by readers.

The disadvantages are that this combination reduces concurrency and can lock up resources in high demand for longer than necessary.

Use this combination if processing efficiency is more important than concurrency. It is a good choice for batch jobs that release table and table space locks only to acquire them again almost immediately. It might even improve concurrency, by allowing batch jobs to finish sooner. Generally, however, do not use this combination if your application contains many SQL statements that are often not executed.

3.2.1.2 ACQUIRE(USE) and RELEASE(DEALLOCATE)
This combination results in the most efficient use of processing time in most cases. The advantages are these:

• A table or table space used by the plan or package is locked only when it is accessed.

• All tables or table spaces are unlocked only when the plan terminates.

• The least restrictive lock needed to execute each SQL statement is used (except that, if a more restrictive lock remains from a previous statement, that lock is used without change).

The disadvantage is that this combination can increase the frequency of deadlocks. Because all locks are acquired in a sequence that is predictable only in an actual run, more delays for concurrent access might occur. The possibility of deadlocks can be reduced by using Type 2 indexes.

3.2.1.3 ACQUIRE(USE) and RELEASE(COMMIT)
This is the default combination and provides the greatest concurrency, but it requires more processing time if the application commits frequently. The advantages are these:

• A table or table space is locked only when needed. That is important if the process contains many SQL statements that are rarely used, or statements that are intended to access data only in certain circumstances.

• Locks held by cursors that are defined WITH HOLD are retained beyond commit points. Except for these, table space locks are released at the next commit point.
• The least restrictive lock needed to execute each SQL statement is used; except that, if a more restrictive lock remains from a previous statement, that lock is used without change.

The disadvantage is that this combination can increase the frequency of deadlocks. Because all locks are acquired in a sequence that is predictable only in an actual run, more delays for concurrent access might occur. The possibility of deadlocks can be reduced by using Type 2 indexes.

3.2.2 ISOLATION

You specify the isolation value at package or plan bind time. You can override this value for a particular SQL statement by specifying the isolation for that SQL statement using the WITH clause.

3.2.2.1 ISOLATION (RR)

Allows the application to read the same row or page more than once without allowing any change by another concurrent process. All rows or pages are locked, even if they do not satisfy the predicate. The considerations to keep in mind are these:

• Use repeatable read (RR) only if it is needed to accomplish the purpose of the application process. If your application can tolerate having new rows inserted into an existing answer set, use read stability (RS) isolation for greater concurrency. And for even more concurrency, use cursor stability (CS) or uncommitted read, if your application can tolerate those options. Applications using RR can leave rows or pages locked for longer periods, especially in a distributed environment, and they can claim more logical partitions than similar applications using CS. They are not compatible with utility operations that drain all claim classes on a logical partition.

• If you use RR, plan for frequent commit points.

• An installation option determines the mode of lock chosen for a cursor defined with the clause FOR UPDATE OF and bound with RR isolation.

3.2.2.2 ISOLATION (RS)

Allows the application to read the same rows more than once without allowing qualifying rows to be updated or deleted by another concurrent process. It can offer greater concurrency than RR because although other applications cannot change rows that are returned to the original application, they can insert new rows, or update rows that did not satisfy the original application’s search condition. This option offers greater concurrency than RR because only those rows that qualify hold locks until commit.

While the RR isolation level provides the highest level of consistency of the result set within a unit of work (repeated executions of the same query within a read-only transaction return the same set of rows), for some applications it is too restrictive. Namely, an RR query with range predicates locks not only rows that qualify, but also the rows that do not qualify the query, even the so-called phantom rows, which do not exist at the time the query is executed but could be inserted into the query range and change the results in a subsequent execution of the query. Consider an example. At Time t1, Table TAB has 100 rows; the key range is COL1 = 1-100, and LOCKSIZE=ROW. The statement executed is SELECT * FROM TAB WHERE COL1 > 97. A matching index access is selected and the sequence of events and associated row locks are as follows:
In the period from Time t6 to t9, this transaction holds four locks, whereas the end-of-index lock is the most restrictive one. It prevents an insert transaction that would come at Time t7 or t8 to insert a row with the key greater than 100.

While in some cases this is the desired behavior (the same query, if repeated before commit, is expected to return the same set of rows, not more than the first time), in other cases preventing the inserts at the end of the range can severely impede concurrency. That is a case where the RS isolation level might be the answer.

The RS isolation is similar to CS isolation with CURRENTDATA(YES) because it acquires locks only on the rows (or pages, if page level locking is used) that satisfy query predicates. It is, however, different from CS and more like RR in that it does not release the lock as the cursor moves off the row. For the above example, the sequence of events in the case of RS is as follows:

<table>
<thead>
<tr>
<th>Time</th>
<th>SQL event</th>
<th>Lock requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>OPEN CURSOR</td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td>FETCH</td>
<td>lock row COL1 = 98</td>
</tr>
<tr>
<td>t3</td>
<td>FETCH</td>
<td>lock row COL1 = 99</td>
</tr>
<tr>
<td>t4</td>
<td>FETCH</td>
<td>lock row COL1 = 100</td>
</tr>
<tr>
<td>t5</td>
<td>FETCH (+100)</td>
<td>end-of-index lock</td>
</tr>
<tr>
<td>t6</td>
<td>CLOSE CURSOR</td>
<td></td>
</tr>
<tr>
<td>t7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t9</td>
<td>COMMIT</td>
<td>all locks released</td>
</tr>
</tbody>
</table>

You can see that the insert inhibiting end-of-index lock is not made, which enables insert applications to add ever-increasing key rows. At the same time, the result set is preserved until commit from the updates and deletes of the qualifying rows.

Because locks are required only on the rows that qualify, DB2 uses lock avoidance to avoid acquiring locks on nonqualifying rows wherever possible. There is a situation, though, where a lock is taken on a row that does not qualify. This happens if the row cannot be rejected by evaluating Stage 1 predicates but is rejected by applying Stage 2 predicates. In this case, the row is not part of the result set, but a lock on it is still held.

Use RS only if it is needed to accomplish the purpose of the application process. For more concurrency, use CS or uncommitted read. Applications using RS isolation can leave rows or pages locked for long periods, especially in a distributed environment. If you use RS isolation, plan for frequent commit points.
3.2.2.3 ISOLATION (CS)
Allows maximum concurrency with data integrity. However, after the process releases a row or page, another process can change the data. If the first process returns to read the same row or page, the data is not necessarily the same. Consider these consequences of that possibility:

- For table spaces created with LOCKSIZE ROW, PAGE, or ANY, a change can occur even while executing a single SQL statement, if the statement reads the same row more than once. In the following example:

  ```sql
  SELECT * FROM T1
  WHERE COL1 = (SELECT MAX(COL1) FROM T1);
  ```

  data read by the inner SELECT can be changed by another transaction before it is read by the outer SELECT.

- In particular, if your process reads a row and returns later to update it, you can erase an intervening update made by some other process. If you use ISOLATION (CS) with UPDATE, your process might need to lock out concurrent updates. One method is to bind with the option CURRENTDATA(YES), which is the default. Another method is to declare a cursor with the clause FOR UPDATE OF.

3.2.2.4 ISOLATION (UR)
Allows the application to read while acquiring almost no locks (see 3.3.2.1, “Uncommitted Read Considerations” on page 75), at the risk of reading uncommitted data. Uncommitted read isolation applies only to read-only operations.

3.2.2.5 Some Considerations
Use of the following clauses in the SQL statements can influence the type of ISOLATION level that DB2 chooses to use:

- WITH RR, RS, CS, or UR in the SELECT statement within the DECLARE CURSOR or a singleton SELECT statement
- WITH RR, RS, or CS in an INSERT from subselect statement
- WITH RR, RS, or CS in a searched UPDATE statement
- WITH RR, RS, or CS in a searched DELETE statement.

For each SQL statement, DB2 decides, based on the isolation level in the plan, the package, or the SQL statement, which isolation level to use. If you use WITH RR, RS, CS, or UR in a SQL statement, DB2 generally uses this isolation level for the particular SQL statement.

If the rules to use the isolation level specified in the WITH clause are not met, however, DB2 changes the isolation level to a valid one. DB2 does not provide any warning when it changes the isolation level as a result of a violation of rules. Furthermore the access path can be affected.

For example, assume the following situation:

- Table EMP with EMPNO as primary key
- Type 1 Index on the EMPNO primary key
- SQL statement (package bound with ISOLATION(RR)).

  ```sql
  SELECT EMPNO FROM EMP
  ```

  DB2 reads EMPNO using the Type 1 index (index scan).
• SQL statement (package bound with ISOLATION(RR)).

```
SELECT EMPNO FROM EMP WITH UR
```

- DB2 reads EMPNO without using the Type 1 index; instead, it does a table space scan.

DB2 does not use uncommitted read isolation with a Type 1 index, nor does DB2 issue any warning about this inconsistency. Look at explain information in the table PLAN_TABLE to see what DB2 has done.

### Note

If DB2 is not using an effective access path, check whether there are any inconsistencies in the SQL options specified.

#### 3.2.3 CURRENTDATA

Determines whether to require data currency for read-only and ambiguous cursors when CS isolation is in effect. It also determines whether block fetching can be used for distributed, ambiguous cursors.

The option CURRENTDATA applies only to statements bound with ISOLATION(CS) and only to cursors that are read-only or ambiguous. A cursor is ambiguous if DB2 cannot definitely determine whether or not it is read-only. An ambiguous cursor is found only in applications containing dynamic SQL statements. We recommend eliminating the ambiguity by declaring the cursor using one of the clauses FOR FETCH ONLY (read-only cursor) or FOR UPDATE OF (non-read-only cursor).

##### 3.2.3.1 CURRENTDATA(YES)

The semantics of ISOLATION(CS) are to return committed data and, for updatable cursors, to provide stability of the data under the cursor. CURRENTDATA(YES) allows the semantic value to be strengthened, such that the stability of data under read-only and ambiguous cursors will also be guaranteed for cursors doing local access to a base table. It does not affect the semantic value of cursors accessing a work file.

##### 3.2.3.2 CURRENTDATA(NO)

CURRENTDATA(NO) means that standard ISOLATION(CS) semantics apply. That is, the committed data is returned and, for updatable cursors, the stability of data under the cursor is guaranteed.

#### 3.3 Application Design and Programming Considerations

In the past, application programming was concentrated on host-based online and batch applications. Executable programs, transactions, presentation logic, data and presentation were managed by only one platform. Now, application programming has to deal with the client-server approach where business, presentation and data are managed on different platforms and possibly at different locations.

New programming techniques and approaches have been developed, for example, object oriented methods. However, even with new programming techniques, DB2’s basic locking strategy is the same and should be understood by all application programmers.
3.3.1 Unit of Work and Unit of Recovery Considerations

Application programmers must think of designing programs or modules in terms of the scope of the unit of recovery.

Basically there are three alternative strategies for online applications. Figure 23 illustrates the three different strategies for unit of recovery.

![Figure 23. Units of Recovery Showing Three DB2 Locking Strategies](image)

3.3.1.1 Strategy 1 – Conversation Integrity (Not Recommended)
A commit point is created at the end of the conversation. The lock duration may include user think time. Therefore, this strategy should not be used in applications where other users need concurrent access to the tables.

Even if there is no need for concurrent processes, this strategy is generally not recommended when you need to update data. DB2 may cancel a program that is inactive (waiting for user input) for more than a few minutes in a distributed environment. This is another reason for always creating a commit point when sending a response to the user.

Read-only application might get away with this strategy in some cases. S-locks are normally released before commit point, but an S-locked page or row cannot
be updated by another process and an S-lock may have a very long duration. It may include user think time.

We do not recommend this strategy even for read-only applications, although it may be the most convenient one from a programming point of view.

Conversation integrity is the default in TSO, an option in CICS, and not possible in IMS.

### 3.3.1.2 Strategy 2 – Interaction Integrity

A commit point is created (and all locks are released) when the application sends a response to the user.

As no locks are held between user actions: while User A is thinking, User B may update any row. The application program may need logic to detect such situations and to react in a proper way. For example, the application program may have to check row-level timestamps (time of last change), or read the data again in each transaction to ensure integrity, or specify in the WHERE clause of the SQL statement `AND COL = :old_value`. It may also be necessary to save the input per user transaction in a private work area (CICS temporary storage or IMS scratch pad area) and perform the database updates in the last step.

In online and distributed transactions, interaction integrity strategy should be the default.

### 3.3.1.3 Strategy 3 – Intermediate Commit Points

A long-running transaction may take several seconds or minutes. With interaction integrity, such a transaction keeps some pages locked for a long time. If the changes are not strictly related (depending on the business rules), intermediate commit points may be added in the application to reduce waiting time for locks. No commit interval should exceed 5 seconds.

In CICS transactions, an intermediate commit point is created with a SYNCPPOINT; in IMS transactions, it is done with a program-to-program switch; and in TSO transactions; a COMMIT is used.

### 3.3.2 Read-Only Considerations

Exclusive locks are acquired for SQL statements INSERT, UPDATE, and DELETE and are released at commit or roll back.

Normally lock avoidance works and allows readers to get in and out with no page or row locks in case of Read-only SQL statements SELECT and FETCH. In situations when locks have to be acquired, the read-only locks are shared, not exclusive. SELECT and FETCH SQL statements acquire S- or U- locks on table space, table, page or row.

Because an S-lock forces an updater to wait, Read-only SQL statements cannot be ignored in locking considerations.

Let us make the following assumptions:

- Isolation level is CS.
- Lock avoidance is deactivated; that is, CURRENTDATA is YES.
- DB2 does not create a temporary table.
A SELECT or a FETCH now takes an S-lock or a U-lock on a page or a row. The lock is released when one of the following conditions is met:

- DB2 locks another data page or another row.
- End-of-file is reached.
- Cursor is closed.
- The program has completed a singleton SELECT.
- COMMIT is executed.

### 3.3.2.1 Uncommitted Read Considerations

Uncommitted read takes latches but almost always no locks. However, there are restrictions on concurrent access and in two situations, locks are acquired:

- An uncommitted read operation makes a claim on the table space. Therefore an application using uncommitted read isolation cannot run concurrently with a utility that drains all claim classes.

- An application using uncommitted read isolation cannot risk reading data from a target table or table space subject to a mass delete operation from another application. Mass delete is a DELETE statement without a WHERE clause and that operation acquires the mass-delete lock in X-mode on the table or table space. The application using uncommitted read isolation tries to acquire a mass-delete lock in S-mode on the target table or table space. This is not possible because of the incompatibility with the mass-delete lock in X-mode, so the two applications cannot run concurrently.

- An application using uncommitted read isolation takes an IX lock on any table space used in the work file database. That lock prevents dropping the table space while the application is running.

- Dynamic SQL readers using uncommitted read isolation take a DBD lock, which prevents operations such as DROP or ALTER, which accesses the same DBD.

An applications using uncommitted read isolation runs fast because of little contention, but it reads uncommitted data. Do not use uncommitted read isolation unless you are sure that your application and the end users can accept the logical inconsistencies that may occur.

Uncommitted read isolation applies only to read-only operation: SELECT, SELECT INTO, or FETCH using a read-only cursor.

You cannot use uncommitted read isolation with

- Any statement that requires access through a Type 1 index
- INSERT, UPDATE, or DELETE operations
- Any cursor defined with a FOR UPDATE OF clause.

If you bind a program with ISOLATION(UR), and some statements in the program are not compatible with uncommitted-read rules, then DB2 uses CS isolation, excepting for statements using the clause WITH RR or WITH RS.
3.3.2.2 Temporary Result Tables
Depending on the access path, DB2 may build a result table when a read-only cursor is opened.

In that case, acquiring and releasing of locks takes place during OPEN CURSOR while the temporary table is being built. When the OPEN CURSOR processing is completed, no locks remain if CS isolation is used. Rows are retrieved from the temporary table and no locking is done at FETCH time.

Lock duration is shorter than in the normal case when the result table is built row by row at FETCH time.

3.3.2.3 Lock avoidance
Lock avoidance is discussed in 2.1.9, “Lock Avoidance” on page 22. Lock avoidance is possible only when the candidate row or page is what we call cold. A candidate row or page is cold if DB2 is able to determine quickly (just by looking at timestamps and possibly uncommitted bits, without accessing the locked resource table) that the page or the row is committed. This is the normal case. If all other conditions required for lock avoidance are satisfied, DB2 simply returns the row to the application program, without imposing a lock. The physical consistency of the page is ensured by a page latch.

If the candidate row or page is not cold, lock avoidance fails. DB2 imposes a lock (in addition to the latch) and releases it according to the rules discussed in 3.3.2, "Read-Only Considerations" on page 74.

From an application point of view, lock avoidance and to some extent also the DB2 use of temporary result tables have to be analyzed. Figure 24 illustrates why you have to think about lock avoidance when you design your application program.

![Figure 24. Lock Avoidance. We assume the criteria for using lock avoidance are met.](image)

Notes:

1. User A reads a page with a read-only SQL statement. DB2 only latches and unlatches the page.
2 User B updates the page and imposes an X-lock on the page. This is possible because the page is not locked.
3 User B commits.
4 User A updates the page based on data read from Step 1. DB2 imposes an X-lock on the page.
5 User A commits.

To avoid the loss of data integrity illustrated in Figure 24 on page 76, disable lock avoidance by doing any of the following:
• Bind plan or package with ISOLATION(RR).
• Bind plan or package with CURRENTDATA(YES).
• Add FOR UPDATE OF clause to read-only cursors.
• Add WITH RR clause to read-only SQL statements;
  RR isolation always disables lock avoidance
• Add WITH RS clause to read-only SQL statements. RS is actually CS with CURRENTDATA(YES). It employs lock avoidance techniques where it can and locks only qualifying rows.

3.3.3 SQL Design

The SQL design should minimize lock durations. However, no lock should be released so early that the integrity of the data is at risk. How to use the different bind options to meet these criteria is discussed in 3.2, “Bind Option Considerations” on page 67. This section discusses program design.

Locks released too soon may result in surprising results. Locks kept too long may cause excessive waiting and even bottlenecks for hot pages. Hot pages are discussed in 3.3.4, “Hot pages” on page 82.

3.3.3.1 Surprising Results

Figure 25 on page 78 illustrates a common problem with S-locks.

We have an ORDER table containing order numbers and some order details. The next available primary key value for the ORDER table is maintained in a separate NEXTORDERNO table with only one row.
Figure 25. Surprising Results of Short S-locks

Notes:

1. User A reads value 777 from table NEXTORDERNO using a singleton SELECT. With ISOLATION(CS) and lock avoidance, DB2 latches the page only during the time data is retrieved. If lock avoidance fails, DB2 acquires a lock as well, but this lock is released as soon as the SELECT is completed, if the isolation level is CS.

2. User B reads the same value (777) from table NEXTORDERNO using a singleton SELECT. DB2 again acquires only a latch or both a latch and a short S-lock, if the isolation level is CS.

3. User A changes value 777 to 778 using UPDATE. This process X-locks the page until commit point.

4. User B tries to update NEXTORDERNO but is suspended because of the X-lock held by user A.

5. User A inserts a new order 777 into the ORDER table and commits.

6. After updating the NEXTORDERNO to 779, user B tries to insert a new order 777 into the ORDER table. If there is no unique index on the order number column in the ORDER table, DB2 inserts an order 777 and the database is corrupted. If there is a unique index on the order number column in the TORDER table, the insert fails and the program...
must check for the SQL code and include the logic to handle the situation.

Note

With isolation level CS, the S-lock, if acquired, is released immediately after DB2 processes the singleton SELECT. With isolation level RR or RS, DB2 keeps the S-lock until commit point. Use of RR or RS in this scenario thus results in a deadlock:

• At Step 3, User A, requesting an X-lock, waits for User B to release the S-lock from Step 2.
• At Step 4, User B, requesting an X-lock, waits for User A to release the S-lock acquired in Step 1.

This scenario with any of the isolation levels CS, RR, or RS is unacceptable. Therefore, we have to consider different solutions.

3.3.3.2 The Normal Solution: U-Lock Instead of S-Lock

Instead of singleton SELECT, use SELECT defined with a cursor. Replace the SELECT ORDERNO FROM NEXTORDERNO (Steps 1 and 2 in Figure 25 on page 78) with a FETCH referring to a cursor defined with SELECT ORDERNO FROM NEXTORDERNO FOR UPDATE OF ... Close the cursor after the UPDATE.

This causes the following:

• User A acquires a U-lock at Step 1. This is promoted to an X-lock at Step 3 and released at commit time.
• User B requesting a U-lock at Step 2 waits until User A commits.

This solution prevents a deadlock with CS, RR, or RS isolation levels.

This solution requires the following steps in the application program:

• Declare cursor and should include the for update of clause
• Open cursor
• Fetch cursor
• Update using the where current of cursor clause
• Close cursor
• Insert
• Commit.

3.3.3.3 The Australian Way: X-Lock Instead of S-Lock

The Australian way shown in Figure 26 on page 80 has the same effect as the normal solution described in 3.3.3.2, "The Normal Solution: U-Lock Instead of S-Lock" but uses fewer SQL statements and therefore has a lower CPU cost.
Figure 26. Australian Way to Update a Hot Page

Notes:

1. User A updates value 777 to 778 using a noncursor UPDATE, which X-locks the page.
2. User B tries to update NEXTORDERNO, but is suspended because of the X-lock held by User A. When User A commits, User B continues from Step 5.
3. User A reads value 777 (778 - 1) from table NEXTORDERNO using a singleton SELECT.
4. User A inserts a new order 777 into the ORDER table and commits. After the commit by User A, Step 2 is processed: NEXTORDERNO is incremented to 779.
5. User B reads value 778 (779 - 1) from table NEXTORDERNO with a singleton SELECT.
6. User B inserts a new order 778 and commits.

This solution requires the following steps in the application program:

- Singleton update
- Singleton select
- Insert
- Commit.

The Australian way executes four SQL calls to accomplish what the normal solution does with six SQL calls at run time.
Note

In this scenario, the NEXTORDERNO table has no index and has only one row. However, if the table has indexes and contains many rows, then this solution requires complete access from the top of an index tree for both Update and Select, whereas in the previous solution, only one probe would be required because of the use of only one cursor. Therefore, for IMS, TSO, Batch, and CAF, there would be no significant difference in CPU use between these two. For CICS, or with performance trace on for each SQL call, four calls is better than six.

3.3.3.4 UK Solution
The UK solution is shown in Figure 27.

```
USER A  PAG  HV  PAG  HV  USER B
       !    !    
SELECT ORDERNO S  777
                  S  777  SELECT ORDERNO
UPDATE          S  777
    X  778  <
NEXTORDERNO
WHERE ORDERNO   SUS
   = :HVORDNO
    UPDATE
    SUS
    IF SQLCODE = 100
    THEN RESTART
    NEXTORDERNO
    WHERE ORDERNO
    = :HVORDNO
INSERT ORDER    777
COMMIT          v  v  v
                X
                IF SQLCODE = 100
                THEN RESTART
                778 INSERT ORDER
                v  COMMIT
```

- PAG = Only page of NEXTORDERNO table
- HV = Content of host variable :HVORDNO
- SUS = Suspension

Figure 27. UK Solution

Note

This solution works well when no other program runs concurrently with ours. The UPDATE is enhanced with a WHERE clause, ensuring that nobody changes the NEXTORDERNO table between the SELECT and the UPDATE. If there is a change in between, an SQLCODE of 100 is returned and the program restarts at the SELECT.
3.3.4 Hot pages

A data page is hot if it is locked for more than 10% of the time. However, the criterion can vary, depending on your environment and requirements. In any case, a hot page causes significant lock waiting if page locks are used. According to queuing theory:

\[
\text{Average wait time} = \frac{P(PL)}{1 - P(PL)} \times \text{Average lock duration}
\]

where \(P(PL)\) is the probability of page being locked.

If \(P(PL)\) of a page is 10%, the page is locked for 10% of the time. The average queuing time in front of that page is then, according to basic queuing theory,

\[
\text{Average wait time} = \frac{0.1}{1 - 0.1} \times \text{Average lock duration}
\]

\[= 0.11 \times \text{Average lock duration}\]

As the average lock duration is normally less than a second, 10% can be considered a reasonable alarm limit; lower values are safe, higher ones may imply excessive lock wait times.

Row locking may significantly reduce the waiting caused by a hot page. It is, however, not an adequate solution if it is a single row that is locked more than 10% of the time.

The only page of the NEXTORDERNO in our example is likely to become a hot page if there are many concurrent users. The Australian way is a little faster than the normal solution, but the difference is not dramatic (less than one millisecond of elapsed time in most environments). The main factor influencing the duration of the NEXTORDERNO lock is the number of synchronous reads. Assuming five indexes pointing to the ORDER table, the duration of the NEXTORDERNO could be roughly 0.1 second (four synchronous reads to the indexes with random access and synchronous log writes to the disk control unit). According to our definition of hot page, the only time the page of the NEXTORDERNO becomes hot is when the insert rate exceeds one transaction per second: 1 transaction/second * 0.1 second/transaction = 0.1 = 10%.

To achieve higher transaction rates without significant lock waiting, the hot page must be cooled. Obviously, row locking does not help, as our only row is exactly as hot as our only page.

If the hot page is detected early enough, the primary key of the order could be changed to a timestamp (NEXTORDERNO table can be eliminated) or to a set of consecutive numbers—for instance one set of keys per user (several rows in NEXTORDERNO table and row locking helps).

If the hot page is detected by measurements in a stress test or in production, it is probably too late to change the primary key of the ORDER table. The only solution, then, is to reduce the duration of the lock by decreasing the number of synchronous reads affecting the NEXTORDERNO lock. This could be done by increasing the size of database buffer pools so much that the leaf pages of the ORDER indexes stay in the buffer pools. Increasing buffer pool size normally requires more central or expanded storage, however. If the cost of additional storage is too high, the Swiss solution described in 3.3.4.1, “Swiss Solution” on
page 83 may be a good compromise between performance and programming convenience.

### 3.3.4.1 Swiss Solution

The Swiss solution reduces lock duration without increasing storage requirements. The idea is to perform all synchronous reads on a randomly accessed index before taking any X-locks. The duration of the X-locks then encompasses only CPU time, CPU queuing time, and log write time. In many environments, this duration is in the order of 20 milliseconds. With a lock duration equal to 20 milliseconds, the X-locked pages do not become hot until the transaction rate exceeds 5 transactions per second. Figure 28 illustrates lock duration time for the Australian way from Figure 26 on page 80. Figure 28 also shows the lock duration time in the Swiss solution.

![Figure 28. Lock Duration for the Australian Way and Swiss Solution](image)

From Figure 28, we can see that X-lock duration time for the Swiss solution on table NEXTORDER is less than the X-lock duration time for the Australian way.

The X-lock duration times on NEXTORDER table are the same in both the solutions, if the ORDER table has no indexes. Each additional randomly accessed index on the ORDER table results in an increase of X-lock duration time of approximately 20 milliseconds when using the Australian way.
Compared to the Swiss solution, an additional index on ORDER table increases the X-lock duration on NEXTORDER table.

The disadvantage in using the Swiss solution is that the application program has to contain one dummy SQL statement, for each ORDER index, to make a matching index scan with no data page reference. If you want to use the Swiss solution in an existing application, you have to code and execute these dummy SQL statements against each ORDER index before updating the NEXTORDER table.

3.3.5 Application Access Strategy

To prevent excessive lock waits, the application programmer should structure the program logic flow:

• Estimate the longest commit interval before implementing the program. If the estimate exceeds the alarm limit (typically 5 seconds), the program should be redesigned to reduce the longest lock duration using one or more of the following:

  − Update data at the end of the logical unit of work (as late as possible). To minimize the duration of X-locks try to update tables as near the commit time as possible.
  − Close cursors as soon as possible. If you define a cursor with the WITH HOLD option, some locks are held past a commit point. Use the CLOSE CURSOR statement as soon as possible in your program, to release any locks and free the resources they hold.
  − Commit work as soon as possible. To avoid unnecessary lock contentsions, issue a COMMIT statement as soon as possible after reaching a point of consistency, even in read-only applications that are bound with RR isolation.
  − Roll back work as soon an error is detected. To prevent unsuccessful SQL statements (such as PREPARE) from holding locks, issue a ROLLBACK statement after a failure.

• Access data in consistent sequence to reduce deadlocks. When different applications access the same data, they should do so in the same sequence. For example, two applications accessing the data should access rows 1,2,3,5 in that order. Of course, the first application to access the data makes the second application wait, but the two applications do not deadlock. For the same reason, the different applications should access the same tables in the same order.

3.3.6 Distributed Application Considerations

A distributed application can be defined as an application running on a PC client (or on some other requestor platform that has the executable code) and accessing DB2 tables on DB2 for MVS/ESA server through the network. Figure 29 on page 85 illustrates such an environment.
In such an environment the network plays an important role in how well distributed applications perform. Delay in the network can be the biggest problem.

### 3.3.6.1 Reducing Network Traffic

DB2 uses two different methods to reduce the number of messages sent across the network when fetching data using a cursor:

- **DRDA** (application-directed) access can use limited block fetch. It optimizes data transfer by guaranteeing the transfer of a minimum amount of data in response to each request from the requesting system.

- **DB2 private protocol** (system-directed) access can use continuous block fetch, which sends a single request from the requester to the server. The server fills a buffer with data it retrieves and transmits it back to the requester. Processing at the requester is asynchronous with the server; the server continues to send blocks of data to the requester without further prompting.

To ensure either type of block fetch, DB2 must determine that the cursor is not used for update or delete. Indicate that in your program by adding **FOR FETCH ONLY** or **FOR READ ONLY** in the DECLARE CURSOR statement for the query. If you do not use **FOR FETCH ONLY** or **FOR READ ONLY**, DB2 still uses block fetch for the query if:

- The result table of the cursor is read-only.
- The result table of the cursor is not read-only, but the cursor is ambiguous, and the **BIND** option **CURRENTDATA** is **NO**.

DB2 does not use continuous block fetch if:

- The cursor is referred to in the statement **DELETE WHERE CURRENT OF** elsewhere in the program.
- The cursor statement appears that it can be updated at the requesting system. DB2 does not check whether the cursor references a view at the server that cannot be updated.
3.3.6.2 Prefetch

Distributed database connection services (DDCS) V2.3 prefetch improves the response times for queries that return large result sets. This function combines the blocking technique used in previous versions of DDCS, with the DDCS’s ability to fetch rows of data for an open cursor before the application actually requests the data.

Prefetch is performed automatically and transparently every time you issue an SQL OPEN or FETCH statement. Basically, when you issue an SQL OPEN statement, DDCS gets a block of data. If the query has more data on the server side, DDCS asynchronously issues another DRDA FETCH to get the next block of data. The first block is processed by subsequent SQL FETCH. The only difference is that the next time DDCS has to send out another DRDA FETCH request for the next block, chances are the next block is already in the workstation. At this time DDCS issues another asynchronous DRDA FETCH to get a subsequent block. This process improves performance for queries that have a large answer set.

DDCS V2.3 prefetch is different from DB2 for MVS/ESA continuous block fetching in that DDCS does not need a separate LU6.2 session and does not tie up extra virtual telecommunications access method (VTAM) buffers along the way. Continuous block fetching keeps pumping the data from the application server (AS) to the application requestor (AR) that is buffered and paused by VTAM. DDCS requests only one extra block at a time. In practice an extra block at a time is sufficient for most machines and uses. Any more than that could tie up the machine cycle and buffers because the application is too slow to fetch all of the data.

3.3.6.3 Stored Procedures

Stored procedures help to reduce the network traffic in a distributed environment.

A stored procedure is a user-written program that executes on the DB2 for MVS/ESA server. If a stored procedure is used, then the DB2 calls and logic shown as existing in the PC program in Figure 29 on page 85 can be taken out and instead executed as part of a stored procedure on the DB2 for MVS/ESA server.

There are only two network messages in calling a stored procedure. The first message is to invoke the stored procedure. The second message is to signal the commit or rollback of the transaction. Locks acquired during the stored procedure processing are held until the commit or rollback.

When using stored procedure in a client/server environment, tune the database as if the data is accessed locally.

3.3.6.4 DSNZPARM Parameters

Figure 29 on page 85 also illustrates another task to be taken care of in a distributed environment. Assume that a user switches off the PC power just before the PC program executes the SQL COMMIT. In this situation, DB2 holds locks acquired by the PC program and potentially can hold these locks for a long time.

To avoid this situation, DB2 has a mechanism to automatically back out and disconnect distributed threads that are not doing any work. This is controlled by
3.4 Quality Card Analysis to Determine the Probability of Locking

3.4.1 Background

We have stated (3.3.4, “Hot pages” on page 82), that the threshold is 10% for \( P(PL) \); that is, a page is considered hot if it is locked more than 10% of the time.

The general formula for \( P(PL) \) is

\[
P(PL) = 100 \times \left( \frac{A \times B \times C}{D} \right) \%,
\]

where

- \( A \) = the number of table pages locked by transaction
- \( B \) = transaction rate
- \( C \) = lock duration
- \( D \) = the number of pages accessed.

Every time DB2 reads an index entry or a data row, that read is considered as a touch. If the touches are not distributed equally, \( B \) = the number of transactions touching a particular page, and \( D = 1 \).

Assume the following:

- A table has 10 pages.
- Each transaction is updating and locking one table page; Therefore, \( A = 1 \).
- The transaction rate (\( B \)) is 5 transactions per second.
- Lock duration (\( C \)) is 1 second per transaction. The load is assumed to be spread uniformly on all ten pages; Therefore, \( D = 10 \).
- All ten pages are hot.

Then, \( P(PL) \) is calculated as 50%. That is, the probability is that the pages are locked 50% of the time.

Which \( P(PL) \) are we talking about? The example just discussed is limited to X-locks. Actually, there are three different \( P(PL) \)s, depending on the lock level requested:

- \( P(PL;X) \) is the probability that a page is X-locked when there is a request for an S-lock on that page.
- \( P(PL;U,X) \) is the probability that a page is U- or X-locked when there is a request for a U-lock on that page.
- \( P(PL;S,U,X) \) is the probability that a page is S-, U- or X-locked when there is a request for an X-lock on that page.

\( P(PL;U,X) \) is the easiest to estimate because all read-only programs can be ignored. Fortunately, it is normally fairly close to \( P(PL;S,U,X) \), the \( P(PL) \) seen by
a program requesting an X-lock. The lock avoidance technique and ISOLATION(UR) reduce the difference between P(PL;U,X) and P(PL;S,U,X).

3.4.2 The INSERT Case

An important special case is the insert case. Inserts never wait for an X-lock on the table pages, but try another page if the best possible page is already locked. An insert lock is known as a conditional lock.

Inserts are thus important for locking purposes only if they suspend other programs doing updates or deletes (or some selects). They cannot be suspended themselves.

3.4.3 Simplified P(PL) Estimate

The P(PL) of the hottest page should be estimated by the database designer for each table very early, as soon as a new table is needed.

This may seem an impossible requirement for two reasons:

- Lock durations depend on the structure of the program. The programs are normally not yet designed when a table requirement comes up.
- Transaction rates per program are difficult to predict.

The first issue is resolved by a pessimistic approximation: assume that all lock durations equal the execution time of the program which is local response time with transactions (default 1 second) and commit interval with batch (default 5 seconds). Actually, of course, the average lock duration is about one-half of the program execution time. If estimates for the updating programs are not yet available, default values can be used for local response time and commit interval.

The second issue is lessened by ignoring read-only programs, an optimistic approximation. For the remaining programs, a really rough estimate is acceptable: the worst period of the year, and a rate the system should be able to handle. It is important to realize that we are not trying to predict the average peak-hour rate here.

The database designer should keep the two approximations in mind when the simple formula leads to an alarm. Critical cases may require a better estimate for lock duration.

3.4.4 Quality Card and Predicting P(PL)

Most applications have some hot pages that are detected too late in a stress test or in production. Often the best way to eliminate locking bottlenecks would be to make fundamental changes to the application, such as changing the primary key of a table. It is very difficult to make such changes when many programs are already implemented.

Most of the hot pages can be identified very early in the development cycle, if the database designer familiar with the application can answer some simple questions at the time a new table is designed.

With tables, you have to answer a single question: Will this table have a hot page? The simple answer of YES or NO can be adequate if the application is very small and the database designer is very experienced.
However, in most cases, it is a good idea to document the assumptions and the calculations behind the YES or NO answer. Quality card is a quality control tool, a checklist for the database designer. With Type 1 indexes, there are more things than just hot pages to consider, for example, the number of RIDs for the most common value in a nonunique index. Quality cards are required only for tables and Type 1 indexes.

Figure 30 shows a blank quality card for a table.

<table>
<thead>
<tr>
<th>INSERTS</th>
<th>Rows/minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATES</td>
<td>Rows/minute</td>
</tr>
<tr>
<td>DELETES</td>
<td>Rows/minute</td>
</tr>
</tbody>
</table>

LOCK SIZE PAGE ROW

Hottest Page/Row:

<table>
<thead>
<tr>
<th>How many pages/rows locked by transaction</th>
<th>Peak load or commit share</th>
<th>How many pages/rows locked by transaction</th>
<th>Peak load or commit share</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAGES/XACT ROWS/XACT</th>
<th>XACT/MIN</th>
<th>S/XACT</th>
<th>PAGES ROWS LOCKED (U,X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XACT INSERT 2 X</td>
<td>X</td>
<td>X</td>
<td>=</td>
</tr>
<tr>
<td>XACT UPDATE 2 X</td>
<td>X</td>
<td>X</td>
<td>=</td>
</tr>
<tr>
<td>XACT DELETE 2 X</td>
<td>X</td>
<td>X</td>
<td>=</td>
</tr>
<tr>
<td>BTCH INSERT 2 X</td>
<td>X</td>
<td>X</td>
<td>=</td>
</tr>
<tr>
<td>BTCH UPDATE 2 X</td>
<td>X</td>
<td>X</td>
<td>=</td>
</tr>
<tr>
<td>BTCH DELETE 2 X</td>
<td>X</td>
<td>X</td>
<td>=</td>
</tr>
</tbody>
</table>

ALARM LIMIT = 10%

*Figure 30. Quality Card for a Table*
The formula in the quality card gives an estimate for $P(PL)$, the probability for a page being locked. Actually, to simplify the estimating process, we estimate $P(PL;U,X)$, the probability that a page is $U$- or $X$-locked. This is an acceptable approximation with CURRENTDATA(NO) and ISOLATION(CS).

**XACT** represents a transaction.

**BTCH** represents a batch job.

**LRT** represents local response time.

**PAGES/XACT** is the number of pages locked by the average transaction. For row locking, **ROWS/XACT** is the number of rows locked by the average transaction.

**XACT/MIN** is the peak transaction rate—the maximum rate the system is expected to tolerate, the worst hour of the year. If in doubt, round up. If you have no idea, set XACT/MIN = X and solve the equation with $P(PL)=10\%$. For batch, this is the number of commit intervals per minute.

**S/XACT** is the time for lock duration interval. If you have no idea, use pessimistic default values like 1 second for transaction and 5 seconds for batch.

**PAGES** is the number of pages sharing the locking load. This is equal to the number of pages if the access is random. Otherwise, PAGES=1 and XACT/MIN should reflect the number of transactions locking the hottest page. For row locking ROWS is the number of rows sharing the locking load. This is equal to the number of rows if the access is random. Otherwise, ROWS=1 and XACT/MIN should reflect the number of transactions locking the hottest row.

The multiplier $2$ is $100 / 60 = 1.7$ rounded up; 100 because the result is expressed as a percentage, and 60 because XACT/MIN is multiplied by S/XACT.

The batch runs present in the worst possible situation should be included in the estimate.

### 3.4.5 Using Quality Card

Consider the following scenario:

- ORDER table has 10,000 pages consisting of 1,000,000 rows.
- ORDER table has a primary key on ORDERNO, clustering index on CUSTNO, and index on ORDERDATE. There are seven other indexes. All 10 indexes are Type 2.
- NEXTORDERNO table has just one row.
- ORDER and NEXTORDERNO tables have page locks.
- The maximum is 100 inserts per minute to the ORDER table. One transaction inserts one order, and local response time = 1 second.
- The maximum is 50 updates per minute to the ORDER table. Order status is updated, and local response time = 1 second.
- A batch job deletes old orders (WHERE ORDERDATE < :Host Variable). The commit interval is 5 seconds. On average, 25 rows are deleted per commit interval, and 1% of the orders are old. As there is no batch window, the batch program must run during the online period.
• The ORDER table is always in the ascending ORDERNO sequence (primary key of ORDER).

• The NEXTORDERNO table row always has the next ORDERNO, that is, one higher than the current ORDERNO.

3.4.5.1 Quality Card for ORDER Table

Figure 31 shows the quality card for the ORDER table. The quality card shown in Figure 31 assumes at most one inserting batch job, one update job, and one delete job. In real life, more lines may be required.

<table>
<thead>
<tr>
<th>INSERTS</th>
<th>100</th>
<th>Rows/minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATES</td>
<td>50</td>
<td>Rows/minute</td>
</tr>
<tr>
<td>DELETES</td>
<td>300</td>
<td>Rows/minute</td>
</tr>
</tbody>
</table>

LOCK SIZE X PAGE ROW

HOTTEST PAGE: ANY

<table>
<thead>
<tr>
<th></th>
<th>Peak load</th>
<th>LRT or commit</th>
<th>How many pages locked by transaction</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>P(PL;U,X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGES/XACT</td>
<td>XACT/MIN</td>
<td>S/XACT</td>
<td>PAGES LOCKED (U,X)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XACT INSERT 2 X</td>
<td>1 X 100 X 1</td>
<td>/ 10000</td>
<td>= 0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XACT UPDATE 2 X</td>
<td>1 X 50 X 1</td>
<td>/ 10000</td>
<td>= 0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XACT DELETE 2 X</td>
<td>X X</td>
<td>/</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTCH INSERT 2 X</td>
<td>X X</td>
<td>/</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTCH UPDATE 2 X</td>
<td>X X</td>
<td>/</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTCH DELETE 2 X</td>
<td>25 X 12 X 5</td>
<td>/ 10000</td>
<td>= 0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ALARM LIMIT = 10%  0.33

Figure 31. Quality Card for ORDER Table
The 300 deleted rows per minute come from the assumptions about our batch program. It takes 5 seconds to delete 25 rows, and, by conversion, it takes 60 seconds to delete 300 rows.

As the table is clustered on CUSTNO, all activities will be randomly spread across the whole table. Therefore, D = 10,000 for all programs.

For the batch program, we assume a commit interval of 5 seconds, C = 5. In 1 minute, there are 12 commit intervals, so that B = 12. As the 25 deleted rows per commit interval are distributed randomly on the table, (Rows are accessed in ORDERDATE sequence; the table is clustered in CUSTNO sequence) A = 25.

ALARM LIMIT = 10% because a page is considered hot if it is locked more than 10% of the time. For the ORDER table, P(PL;U,X) is 0.03%, which is below the ALARM LIMIT so there is no hot page problem.

3.4.5.2 Quality Card for NEXTORDERNO Table
Figure 32 on page 93 shows the quality card for the NEXTORDERNO table.
**INS E R T S** 0 Rows/minute

**U P D A T E S** 100 Rows/minute (Updates/minute)

**D E L E T E S** 0 Rows/minute

**L O C K S I Z E X** PAGE   **R O W**

**HOTTEST PAGE:** THE ONLY ONE

<table>
<thead>
<tr>
<th>How many pages locked by transaction</th>
<th>Peak load</th>
<th>LRT or commit</th>
<th>How many pages locked</th>
<th>How many share interval peak load</th>
<th>P(PL;U,X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAGES/XACT</th>
<th>XACT MIN</th>
<th>S/XACT</th>
<th>PAGES LOCKED(U,X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XACT INSERT 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td>XACT UPDATE 2 X</td>
<td>1</td>
<td>X</td>
<td>100</td>
</tr>
<tr>
<td>XACT DELETE 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td>BTCH INSERT 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td>BTCH UPDATE 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td>BTCH DELETE 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
</tr>
</tbody>
</table>

**ALARM LIMIT = 10% 200**

*Figure 32. Quality Card for NEXTORDERNO Table*

This quality card is easy to fill. The only thing to be aware of is the fact that the NEXTORDERNO table is updated once for every insert transaction on ORDER. Therefore, \( B = 100 \).

ALARM LIMIT = 10% because a page is considered hot if it is locked more than 10% of the time. For the NEXTORDERNO table \( P(PL;U,X) \) is 200%, which is above the ALARM LIMIT, and hot page problem exists.
### 3.4.5.3 Refinement of Quality Card Estimates

More analysis reveals that 50% of all status updates relate to Customer 1001 (evenly distributed on typically 100 orders of that customer). Based on this knowledge, the quality card for the ORDER table is updated.

<table>
<thead>
<tr>
<th>INSERTS</th>
<th>100</th>
<th>Rows/minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATES</td>
<td>50</td>
<td>Rows/minute</td>
</tr>
<tr>
<td>DELETES</td>
<td>300</td>
<td>Rows/minute</td>
</tr>
</tbody>
</table>

**LOCK SIZE**

**HOTTEST PAGE: THE TWO WITH CUSTOMER 1001**

<table>
<thead>
<tr>
<th>How many pages locked by transaction</th>
<th>Peak load</th>
<th>LRT interval</th>
<th>How many share peak load</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>P(PL;U,X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XACT INSERT 2 X</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>/</td>
<td>10000</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>XACT UPDATE 2 X</td>
<td>1</td>
<td>25</td>
<td>1</td>
<td>1</td>
<td>/</td>
<td>2</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>XACT DELETE 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTCH INSERT 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTCH UPDATE 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTCH DELETE 2 X</td>
<td>25</td>
<td>12</td>
<td>5</td>
<td>1</td>
<td>/</td>
<td>10000</td>
<td>0.3</td>
<td></td>
</tr>
</tbody>
</table>

**ALARM LIMIT = 10% 25**

*Figure 33. Quality Card for ORDER Table (Customer 1001)*

Figure 33 shows the updated quality card for the ORDER table.
Assuming that 50% of the updates relate to Customer 1001, the hot spot for the update transaction moves to one or two particular pages with data for Customer 1001.

Because the table is clustered on CUSTNO, all assumed 100 rows for Customer 1001 are consecutive. As one page contains 100 rows (1,000,000 rows/10,000 pages), the hot spot is 1 page (unrealistic, all 100 rows would have to be in the same page), or 2 pages (the 100 rows are partially in one page and partially in the next page). So, \( D = 2 \). But as only 50% of the transactions touch that page, \( B = 25 \). All values on the quality card relate to what happens on the hot page(s).

With these changes, \( P(PL;U,X) \) for the ORDER table is 25%, which is above the ALARM LIMIT and a hot page problem exists.

We also learn at this stage that all orders are deleted at the end of each month (moved to a history table, for instance). Based on this knowledge, the quality card for the ORDER table is further updated.

Figure 34 on page 96 shows the updated quality card for the ORDER table.
I N S E R T S 100 Rows/minute

U P D A T E S 50 Rows/minute

D E L E T E S 300 Rows/minute

L O C K S I Z E X PAGE ROW

HOTTEST PAGE: THE ONLY ONE

How many Peak LRT How many
pages load or share
locked by commit interval peak load
transaction

A B C D P(PL;U,X)

HOTTEST PAGE

PAGES/XACT PAGES LOCKED

XACT INSERT 2 X 1 X 100 X 1 / 1 = 200
XACT UPDATE 2 X 1 X 50 X 1 / 1 = 100
XACT DELETE 2 X X X / =
BTCH INSERT 2 X X X / =
BTCH UPDATE 2 X X X / =
BTCH DELETE 2 X 1 X 12 X 5 / 1 = 120

ALARM LIMIT = 10% 420

Figure 34. Quality Card for ORDER Table at the Beginning of the Month

At the beginning of a new month, the ORDER table is empty, meaning that the table has only one page. Therefore, until there are some inserted orders, D = 1 for all programs.

Even if there are still 50% of the updates related to customer 1001, B = 50 as all update transactions access the only available page. The same applies to the batch program.
The assumption that the batch program runs on that empty table is theoretical, but even without this batch program, the only page on ORDER is very hot (300%).

\[ A = 1 \text{ as the number of pages locked cannot be higher than the total number of pages (D).} \]

### 3.4.6 Cooling Hot Pages and Hot Rows

You can take one of the following steps to cool hot pages:

- Decrease **A** (number of table pages locked by the program)
  - Establish intermediate commit points.
  - Change clustering order.
- Decrease **B** (transaction rate).
- Decrease **C** (lock duration)
  - Establish intermediate commit points (transactions only).
  - Implement Swiss solution.
  - Implement UK solution.
- Increase **D** (number of pages possibly accessed)
  - Change from page locking to row locking.
  - Change clustering order.
  - Add dummy columns (or rows) to reduce the number of rows in a page.

You cannot decrease **B** without severely damaging the user’s business, as you would limit the transaction rate to a value below what the users need to do their business.

Setting intermediate commit points in a batch program does decrease **C**, but **B** would be increased accordingly. Remember that for batch programs, **B \times C = 60**. So, intermediate commit points to decrease **C** are not an alternative to cool down hot pages for batch programs. You can take one of the following steps to cool hot rows:

- Decrease **A** (number of table rows locked by the program) by establishing intermediate commit points.
- Decrease **B** (transaction rate). However, this is not possible without decreasing user satisfaction!
- Decrease **C** (lock duration)
  - Establish intermediate commit points (transactions only).
  - Implement Swiss solution.
  - Implement UK solution.
- Increase **D** (number of rows possibly accessed), which is not possible.

### 3.4.7 Decrease C (Lock Duration)

The Australian way discussed in 3.3.3.3, “The Australian Way: X-Lock Instead of S-Lock” on page 79 is presented in Figure 35 on page 98. This approach is presented as a starting point to discuss how the Swiss and UK solutions help to cool the hot pages.
## Australian Way to Update a Hot Page

The basis for discussing Swiss and UK solutions.

<table>
<thead>
<tr>
<th>UPDATE NEXTORDERNO</th>
<th>SELECT NEXTORDERNO</th>
<th>INSERT ORDER + 10 INDEXES COMMIT</th>
</tr>
</thead>
</table>

- (2) = 1 T + 0 SR = 0.3 ms (including CPUQ)
- (3) = 1 T + 0 SR = 0.3 ms (including CPUQ)
- (4) = 12 T + 10 SR = 203.6 ms (including CPUQ)
- (5) = 10 ms (assuming DASD Fast Write)

**Note:**

T = Touch, SR = Synchronous Read, and CPUQ = CPU Queuing

In (4), T includes one touch to locate the page to insert the row.

(1) is omitted because it corresponds to doing a SELECT on ORDER table (see Figure 36 on page 99) which is not required in this solution.

---

All figures are according to quick upper-bound estimate. Please refer to *DB2—Quick Upper Bound Estimate: An Application Design Methodology* (SG24-2549). Assume a 9021-711 processor, 3390 Model 1 disks and a CPU load of 33% (resulting in a CPU Queuing factor of 50%).

In this situation, there are 10 indexes on the ORDER table, and one index (not the clustering index), is on ORDERNO, which must be inserted in ascending sequence without holes.

The X-lock on NEXTORDERNO is held for 214.2 ms. Based on the rule that no page should be locked more than 10% of the time, this means that a maximum of \( \frac{60000}{(10 * 214.2)} = 28 \) transactions per minute can be afforded.

For low to medium transaction rates, there is no extra step to take.

Figure 36 on page 99 shows the Swiss solution to the hot-page problems.
Most of the 214.2 ms are I/Os (10 * 20 ms). The Swiss solution consists of moving all I/O activities before the X-lock on NEXTORDERNO is taken. This can be done by issuing SELECTs, based on the host variables used in the INSERT, to read all needed pages in the buffer pools.

The result is a small increase in local response time (+3 ms), but the X-lock on NEXTORDERNO is held only for 13.6 ms, meaning that a transaction rate of (60000 / (10 * 13.6)) = 441 transactions per minute can now be tolerated.

Besides the higher transaction rates, the Swiss solution could lead to less need for buffer pools, as not so many objects have to be nearly resident to avoid long locking times.

The Swiss solution has two small disadvantages. It requires more programming effort, and, in case an index is added to the ORDER table after program implementation, the program has to be enhanced with a SELECT for that new index.

Figure 37 on page 100 shows the UK solution to the hot-page problems.
UPDATE
SELECT INSERT NEXTORDerno
NEXTORDerno ORDER + WHERE ORDERNO
INTO :HVORDNO 10 INDEXES = :HVORDNO COMMIT

(3) (4) (2) (5)

Lock on \(0.3\) ms \(10.3\) ms
NEXTORDerno (5) \(\geq\) X (6)

Lock on \(213.9\) ms
ORDER X (7)

\(214.2\) ms
< (8) >

- (2) = 1 T + 0 SR = 0.3 ms (including CPUQ)
- (3) = 1 T + 0 SR = 0.3 ms (including CPUQ)
- (4) = 12 T + 10 SR = 203.6 ms (including CPUQ)
- (5) = 10 ms (assuming DASD Fast Write)

Note:

T = Touch, SR = Synchronous Read, and CPUQ = CPU Queuing

In (4), T includes one touch to locate the page to insert the row.

(1) is omitted because it corresponds to doing a SELECT on ORDER table (see Figure 36) which is not required in this solution.

From a mathematical point of view, the UK solution is better than the Swiss solution. The X-lock on NEXTORDerno is held for 10.6 ms, meaning a transaction rate of \((60000 / (10 \times 10.6)) = 566\) transactions per minute.

The program must include some logic after the INSERT on ORDER. If the SQLCODE is -803 (duplicate index keys in a unique index, assumed here on ORDERNO), someone else has already inserted a row with the same ORDERNO, and the program must start again at the SELECT on NEXTORDerno.

In the case of high transaction rates however, the probability that a program must restart is high. So 566 transactions per minute will be lessened by the number of restarts becoming necessary.
3.4.8 Coolers for the Hot Pages in ORDER and NEXTORDERNO Tables

Table 14 shows the coolers that can be used for the ORDER and NEXTORDERNO tables.

<table>
<thead>
<tr>
<th>Hot pages</th>
<th>Coolers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 NEXTORDERNO Table THE ONLY PAGE</td>
<td>Intermediate commit (plus list of holes for auditors) or Swiss solution or UK solution</td>
</tr>
<tr>
<td>2 ORDER Table ORDERS FOR CUSTNO 1001</td>
<td>Row locking or Change clustering</td>
</tr>
<tr>
<td>3 ORDER Table ONLY PAGE (if orders are deleted)</td>
<td>Do not delete physically or Swiss solution or Mills solution</td>
</tr>
</tbody>
</table>

Figure 38 on page 102 shows the quality card table for ORDER table assuming row locks instead of page locks.
INSERTS 100 Rows/minute

UPDATES 50 Rows/minute

DELETES 300 Rows/minute

LOCK SIZE PAGE X ROW

HOTTEST ROW: ANY WITH CUSTOMER 1001

<table>
<thead>
<tr>
<th>How many rows locked by transaction</th>
<th>Peak load LRT or commit interval</th>
<th>How many share peak load</th>
<th>P(RL;U,X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

ROWS/XACT | XACT/MIN | S/XACT | ROWS
---|---|---|---
XACT INSERT 2 X | 1 X 100 X 0.02 / 1000000 = 0
XACT UPDATE 2 X | 1 X 25 X 1 / 100 = 0.5
XACT DELETE 2 X | X X / =
BTCH INSERT 2 X | X X / =
BTCH UPDATE 2 X | X X / =
BTCH DELETE 2 X | 25 X 12 X 5 / 1000000 = 0

ALARM LIMIT = 10% 0.5

Figure 38. Quality Card for ORDER Table (Row Locks)

Changing the ORDER locksize from page to row causes the following changes:

- D now reflects the number of rows that could be touched, and, therefore, D = 1000000 for the insert transaction and the batch program, and D = 100 for the update transaction, still assuming that 50% of the updates relate to customer 1001. (Remember customer 1001 has an average of 100 rows.) With this assumption, B = 25 again.
• C = 0.02 second reflects the changes that we made to the insert transaction to cool down the NEXTORDERNO table. The Swiss solution is assumed here. With the UK solution, C would be about 0.3 second.

• ALARM LIMIT = 10% because a row is considered hot if it is locked more than 10% of the time. For the ORDER table \( P(PL;U,X) \) is 0.5% and is below the ALARM LIMIT and so there is no hot row problem.

Figure 39 on page 104 shows the quality card table for ORDER table assuming clustering sequence is changed. Page locks are used.

By changing the sequence of the table rows, we are back to a random access by the update program. Thus, \( D = 100 \), the number of possible pages. The ALARM LIMIT = 10%. For the ORDER table \( P(PL;U,X) \) is 0.8%, which is below the ALARM LIMIT, so that no hot-page problem exists.
<table>
<thead>
<tr>
<th>Operation</th>
<th>Rate</th>
<th>Rows/minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inserts</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Updates</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Deletes</td>
<td>300</td>
<td></td>
</tr>
</tbody>
</table>

**Lock Size**

**Hottest Page:** The Only One

<table>
<thead>
<tr>
<th>How many pages locked by transaction</th>
<th>Peak load</th>
<th>LRT or commit share interval</th>
<th>How many pages peak load</th>
<th>( P(PL;U,X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

**Pages/XACT**

<table>
<thead>
<tr>
<th>XACT INSERT 2 X</th>
<th>1</th>
<th>X</th>
<th>100</th>
<th>X</th>
<th>0.02</th>
<th>/</th>
<th>10000</th>
<th>=</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>XACT UPDATE 2 X</td>
<td>1</td>
<td>X</td>
<td>50</td>
<td>X</td>
<td>1</td>
<td>/</td>
<td>100</td>
<td>=</td>
<td>0.5</td>
</tr>
<tr>
<td>XACT DELETE 2 X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTCH INSERT 2 X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTCH UPDATE 2 X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTCH DELETE 2 X</td>
<td>25</td>
<td>X</td>
<td>12</td>
<td>X</td>
<td>5</td>
<td>/</td>
<td>10000</td>
<td>=</td>
<td>0.3</td>
</tr>
</tbody>
</table>

**Alarm Limit:** 10% 0.8

---

*Figure 39. Quality Card for ORDER Table with Changed Clustering Sequence*

*Figure 40 on page 105 shows the quality card table for NEXTORDERNO table using either the Swiss or the UK solution.*
How many Peak LRT How many
pages load or pages
locked by commit share
transaction interval peak load

A   B   C   D   P(PL;U,X)

PAGES/XACT       XACT/MIN   S/XACT   PAGES LOCKED

XACT INSERT 2 X   X   X   /   =
XACT UPDATE 2 X   1   X   100   X   0.02   /   1   =   4
XACT DELETE 2 X   X   X   /   =
BTCH INSERT 2 X   X   X   /   =
BTCH UPDATE 2 X   X   X   /   =
BTCH DELETE 2 X   X   X   /   =

ALARM LIMIT = 10% 4

Figure 40. Quality Card for NEXTORDERNO Table (With Swiss or UK Solution)

For the Swiss solution, a typical lock duration of 20 ms is a realistic value. For the
UK solution, lock duration depends on how many times the transaction has to retry,
so B could increase significantly.
### 3.4.9 Quality Card for Type 1 Index

Figure 41 shows a blank quality card table for a Type 1 index.

#### Quality Card for Table _________________________________

**Index _________________________________**

**COLUMNS**

<table>
<thead>
<tr>
<th>INSERTS</th>
<th>RANDOM</th>
<th>LAST PAGE</th>
<th>NONE</th>
<th>&gt;</th>
</tr>
</thead>
</table>

**LONGEST RID CHAIN**

**KEY:**

ALARM LIMIT=100

**HOTTEST PAGE:**

(Insert + Update + Delete)

<table>
<thead>
<tr>
<th></th>
<th>PAGES/XACT</th>
<th>XACT/MIN</th>
<th>S/XACT</th>
<th>PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>XACT INSERT 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
<td>=</td>
</tr>
<tr>
<td>XACT UPDATE 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
<td>=</td>
</tr>
<tr>
<td>XACT DELETE 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
<td>=</td>
</tr>
<tr>
<td>BTCH INSERT 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
<td>=</td>
</tr>
<tr>
<td>BTCH UPDATE 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
<td>=</td>
</tr>
<tr>
<td>BTCH DELETE 2 X</td>
<td>X</td>
<td>X</td>
<td>/</td>
<td>=</td>
</tr>
</tbody>
</table>

ALARM LIMIT = 10 %

---

### 3.4.10 Who Should Do What?

The database designer should fill in the quality card and then let the application developers check the assumptions. Every one involved should understand that this is an order-of-magnitude estimate or actually a binary estimate: Is P(PL) more than 10% or not?
3.5 Summary of Recommendations

1. Choose the weakest isolation level and data currency level the application can tolerate. Often this means ISOLATION(CS) and CURRENTDATA(NO): the application does not read uncommitted data but data may be updated after a read-only SQL call.

2. Lock duration should never include user think time; create a commit point when writing a response to the user.

3. No page or row should be locked for more than 5 seconds, not even with the worst input during peak load.

   This applies to all programs in interactive and batch, update and read-only, centralized and distributed environments. The application developer should identify the exceptions at the time the program is designed.

4. No page or row should be locked for more than 10% of the time.

   The database designer should identify the exceptions at the time the table is designed.

5. Rows should be accessed in consistent sequence.

6. If a row is retrieved before update, use a cursor with the FOR UPDATE OF clause.
Chapter 4. How to Identify and Analyze Locking Problems

This chapter provides a set of rules to identify and analyze locking problems.

4.1 Introduction

Some of these problems are visible only when they become critical as production applications abend with timeouts and deadlock codes. In many cases, deadlocks are the evidence of a large hidden problem with concurrency, leading to increasing suspension times in DB2 programs. This results in longer waits for response times for the online applications and a need for a longer batch window.

Locking problems can have different causes: In some cases, the problem resides in the program or database design, which was not planned to allow concurrent access. Sometimes the problem is due to a system-wide abnormal situation or to more active programs than planned.

This chapter discusses how to handle such problems from a bottom-up perspective. At the bottom is the application problem, which most of the time is the cause of the locking problem; in the middle is the concurrency problem caused too many programs running in the DB2 system, and at the top are system-wide (global) problems. These should be explored in case the investigation at the bottom and middle layers does not yield positive results.

Once the cause is analyzed, a deeper analysis of the locking problem can be done using the trace facilities in DB2.

There are three major ways to detect suspension problems in DB2 applications:

- User warning
- Console messages
- Periodic monitoring.

These three are listed in order of complexity, quantity of information provided, and increasing system overhead.

It is also important to point out that there are no fixed thresholds for locking problems in an installation. The existence of a problem depends on subjective factors, such as how critical are the application programs the company processes, and also on objective factors, such as the number of deadlocks or timeouts per million program executions. The DB2 system administrator must evaluate the impact of lock suspensions in the installation.

4.1.1 User Warning

This is the main source of information to the DB2 data base or system administrator about possible suspension problems in the system.

DB2 suspension times are part of the total elapsed time of the program and are the outcome of diverse factors, such as:

- Excessive CPU load, which results in CPU waits for the programs
- Input/output (I/O) times
Lock suspension times.

All these factors vary from one execution of the program to another. Factors like CPU load are not constant over time, and sometimes they compensate one another: For example, I/O suspensions moderate program CPU requirements at a given moment so that CPU requirements for this program are eased in this period of time.

While they compensate one another, it is also important to notice the chain effect they produce in the system: For instance, a program that holds an X-lock on one page and is waiting for CPU, holds the lock longer, creating new locking suspension problems for other programs active in the system.

For example, consider three DB2 programs (A, B, and C) all CPU-bound (that is, there are no waits for I/O) and running concurrently on one DB2 system and updating the same DB2 row.

The chain effect is illustrated in Figure 42

![Figure 42. The Chain Effect Example](image)

Application A holds an X-lock on the table for a much longer time than usual because of the CPU constraints in the system. This X-lock in turn means longer suspension time for Application B. Consequently, Application C has to wait for the lock for longer than the maximum time allowed by the timeout installation parameter and it abends.
This is the typical way in which locking suspension problems are detected. No system messages are issued to warn of impending timeouts or deadlocks.

The end-user sees only an increasing response time in the programs, but is in most cases unable to point out the reason. In this situation, the DB2 database or system administrator must use the tools DB2 provides to determine if this is a locking suspension problem, that is if the program is waiting for a resource that is held by another application program or utility.

4.1.2 Console Messages

There are no console messages in the case of normal resumption after a suspension. Console messages log abend, time-out, and deadlock situations. Frequent console messages indicate a problem that is sometimes hidden and should lead to a careful analysis of concurrency in the installation.

Timeout messages can indicate possible lock suspension problems. However, some deadlocks may be unavoidable and a certain number of deadlocks should be accepted in the installation. The exact number is subjective, and depends on the type of application: for instance, a deadlock in a CICS transaction may mean no more than a longer response time, as it is automatically launched again, which is not the case for a TSO or batch program. As a first guess, one deadlock for every 25,000 programs executed is an acceptable number.

Timeout messages should be quickly routed to the DB2 database or system administrator for further analysis of the causes that led to this concurrency problem.

4.1.3 Periodic Monitoring

This is the third source of warnings of suspensions and is the only source of warnings that go directly to the DB2 database or system administrator without an intermediary.

Periodic monitoring, which can be online or batch, provides suspension warnings through the following:

- **DB2 commands** give the DB2 database or system administrator complete information about the status of the DB2 programs and objects.
- The **DB2 Explain** facility provides the access path and lock information for the program.
- **DB2 PM Exception reports** are very useful as watch dogs to monitor the system. These reports identify the programs that surpass a defined threshold. We recommend monitoring programs whose LOCK/LATCH suspension time is more than 5 seconds.
- **DB2 trace records** provide the maximum information. The three trace records mainly used are these:
  - **DB2 Statistics** trace records are written, for example, at the statistics interval that can be controlled by a ZPARM (default value is 30 minutes). You can judge the frequency of concurrency problems by noting the number of suspensions for lock waits, the number of time-outs, and the number of deadlocks in the system. Statistics do not identify the affected programs because they give only global counters for the whole system.
  - **DB2 Accounting** trace records are written once for every DB2 program termination and give a complete picture of the execution of this program. Again, these trace records show whether or not a particular program is
involved in concurrency problems, but do not provide details about contending resources or about other programs contending for these resources. Because DB2 generates accounting records for every executed program, some kind of summarization is recommended if periodic monitoring is done.

- **DB2 Performance** trace records give the complete picture of the concurrency problem, but at the cost of high system overhead.

- **User Reports** often cause extra work, but help a great deal in finding the reasons for suspension. An example of such reports can be a program that has as input an exception report showing the suspended programs and the accounting trace records (which can be stored in a DB2 table for ease of use) and generates a list of the programs responsible for causing lock suspensions. These are the programs that executed during the suspension intervals, that lasted for at least 5 seconds, and that did not suffer contention themselves.

Figure 43 shows the relationship in terms of the effort to analyze and the system overhead to gather the locking information:

![Figure 43. Lock Monitoring Sources](image-url)
4.2 Identify Application Problems

If you experience lock-suspensions, you initially suspect the application and start analyzing the application.

If it is an application problem, the concurrency can be improved either by changing the database design, that is, the SQL data definition language or by changing the program itself, that is the SQL data manipulation language embedded in the program, or even changing the logic structures in the program.

Typical database design options leading to concurrency problems are:

• Use of Type 1 indexes
• Table spaces containing more than one table
• Small tables with page locking
• Disorganized tables.

Indicators of program-design problems that hurt concurrency include these:

• Hot pages or hot rows (locked more than 10% of the time)
• Updates at the beginning of the program resulting in locks being held for a long time (lock too soon)
• Low commit frequency in batch programs (unlock too late)
• Use of restrictive ISOLATION options (RR or RS) when not needed
• Bad ACQUIRE and RELEASE strategy
• Use of CURRENTDATA(YES) when data currency is not needed
• Long IMS or CICS transactions without intermediate commit points
• Closing a cursor too late
• LOCK TABLE statement in program
• FOR UPDATE OF clause missing from the DECLARE CURSOR statement.

These problems when identified are really difficult to solve in many cases, as they very often involve disruptive database structure changes, program logic changes, or both.

There are four main sources of information to identify application concurrency problems, listed in the increasing order of complexity and cost:

• DISPLAY DATABASE LOCKS command
• EXPLAIN statement
• DB2 PM Accounting report
• DB2 PM Online Monitor windows Locked Resources and Threads Holding Resource.
4.2.1 DISPLAY DATABASE LOCKS command

Using this DB2 command you can visualize the status and the locks currently held at a database, table space, or index space level.

The syntax of the command is:

```
DISPLAY DATABASE (db) SPACENAM(sp) LOCKS
```

where `db` is the database name and `sp` is the table space or index space name. The option SPACENAM is optional. If not used or if SPACENAM(*) is specified, DB2 displays lock information for all table spaces and index spaces in the database.

For example, Figure 44 shows the display of DISPLAY DATABASE LOCKS command.

![Figure 44. DISPLAY DATABASE LOCKS command]

Message DSNT360I in Figure 44 displays the overall status of the database, which can hold one of the following values:

- `RO`, the database is started for read-only activity.
- `RW`, the database is started for read and write activity.
- `STOP`, the database is stopped.
- `STOPP`, the database is in stop pending status or stop is in progress.
- `UT`, the database is started for utility processing only.

Message DSNT397I in Figure 44 displays all the lock-related information at a table space or index space level:
• **NAME** is the name of table space or index.

• **TYPE** is the type of DB2 object being displayed and can hold one of two values:
  - **TS**, for table space
  - **IX**, for index.

• **PART** refers to the partition number in a partitioned table space or index and is blank for a nonpartitioned table space or index.

• **STATUS** shows the current status of the table space or index and can hold one of the following values:
  - **RW**, the object is started for read and write activity.
  - **RO**, the object is started for read-only activity.
  - **STOP**, the object is stopped.
  - **STOPP**, the object is in stop pending status or stop is in progress.
  - **STOPE**, the object is implicitly stopped for a problem with the log relative byte address in a page, indicating inconsistency.
  - **REST**, the object is being restarted.
  - **CHKP**, the object is in check pending status.
  - **COPY**, the object is in copy pending status. An image copy is required for this object.
  - **RECP**, the object is in recovery pending status.
  - **RECP***, the logical partition is in recovery pending status, so the entire index is inaccessible to SQL. The logical partition needs to be recovered.
  - **PSRCP**, the index is in a page set recovery pending status.
  - **UT**, the object is started for utility processing only.
  - **UTRO**, a utility is in process on the object that allows read-only access.
  - **UTRW**, a utility is in process on the object that allows read and write access.
  - **UTUT**, a utility is in process on the object that allows access only to utilities.
  - **LPL**, the object has pages or ranges of pages that are unavailable because of logical or physical damage.
  - **LSTOP**, the logical partition is stopped.

• **CONNID** is the connection identifier for the thread and can be one of the following:
  - **BATCH**, if the thread is from a batch region.
  - **TSO**, if the thread is from a TSO terminal.
  - **UTILITY**, if the thread is from a utility.
  - An **IMS identifier**, if the thread is from an IMS terminal.
  - A **CICS identifier**, if the thread is from a CICS terminal.
  - A **CONSOLE** if the thread is from the console.

• **CORRID** is the correlation identifier of the thread associated with the space name and can be one of the following:
  - A **job-name** if the thread is from a batch region.
  - A **TSO logon identifier** if the thread is from a TSO terminal.
− A PST#.PSBNAME, if the thread is from an IMS terminal.
− An entry ID.thread number.tran id if the thread is from a CICS terminal.
− CONSOLE, if the thread is from the console.

• LOCKINFO is the most important in terms of locking information. The information provided in this column can be classified in four types:

− **Lock qualifier**
  - H, if the lock is held by the process.
  - W, if the process is waiting for the lock.

− **Lock identifier** shows the type of lock:
  - IS, a lock with read intentions.
  - IX, a lock with write intentions.
  - S, a lock with read-only capability.
  - U, a lock with update capability.
  - SIX, a lock with a protocol that does not lock a page while reading, but locks the page with update intention while updating.
  - X, an exclusive lock.

− **Lock object** shows the object being locked:
  - S, a table space lock.
  - T, a table lock.
  - C, a cursor-stability-read drain lock.
  - R, a repeatable-read drain lock.
  - W, a write drain lock.
  - P, a partition lock.

− **Lock duration** shows when the lock is freed if it is held (lock qualifier H), and the position in the wait queue if the process is waiting for the lock (lock qualifier W):
  - A, lock is freed at deallocation time (End of application).
  - C, lock is freed at commit time.
  - H, lock is freed when all cursors are closed (Held across commit).
  - M, lock is freed by the system (Manual).
  - P, lock is freed when the plan is complete.

### 4.2.2 EXPLAIN Statement

The SQL EXPLAIN statement records in the plan table information as to which locks DB2 chooses as part of the access strategy.

EXPLAIN can be invoked as an isolated SQL statement for a single SQL statement or can be specified as a BIND or REBIND optional parameter (EXPLAIN YES/NO) so that each time the plan is bound or rebound, an automatic EXPLAIN will be done.

The syntax of the EXPLAIN statement is:
EXPLAIN PLAN SET QUERYNO = nn FOR
explainable SQL statement;

where QUERYNO is an optional identifier for the explained statement.

EXPLAIN places its result in a user table called PLAN_TABLE. This table can be
queried to analyze the result of the EXPLAIN statement. The table PLAN_TABLE
has many columns, but the columns with relevant locking information are listed in
Figure 45.

![Figure 45. EXPLAIN Statement Showing Columns with Relevant Locking Information](image)

Information related to locking contained in column TSLOCKMODE in the
PLAN_TABLE indicates the lock mode to be acquired on the table
CREATOR.TNAME or on the table space holding the table.

If the isolation can be determined at BIND time, TSLOCKMODE can have one of
the following values:

- IS, intent share lock
- IX, intent exclusive lock
- S, share lock
- U, update lock
- X, exclusive lock
- SIX, share with intent exclusive lock
- N, no lock.

If the isolation cannot be determined at BIND time and must be determined at RUN
time, TSLOCKMODE can have one of the following values:

- NS, S-lock (for uncommitted read isolation, no lock)
- NIS, IS-lock (for uncommitted read isolation, no lock)
- NSS, IS-lock for CS and RS isolation, S-lock for RR isolation, and no lock for uncommitted read isolation
- SS, IS-lock for uncommitted read, CS, and RS isolation, and S-lock for RR isolation.

Based on these TSLOCKMODE values, the user can identify the mode of lock and the scope of the lock, which also depends on the table space structure (segmented or nonsegmented) as you can see in Figure 46.

![Figure 46. DB2 Locks Requested and TSLOCKMODE Values](image-url)

In the SQL example in Figure 45 on page 117, assuming the table space is segmented, you can see that DB2 acquires an IS-lock on the table or table space, and page S-locks in case lock avoidance is not possible.

### 4.2.3 DB2 PM Accounting Report

DB2 accounting trace records contain relevant locking information at a plan or package level. This information can be formatted using the DB2 Performance Monitor (DB2 PM).

Accounting trace gives information about a particular program, as an accounting record is written at plan deallocation, or thread reuse.
In order to generate these records, DB2 must activate the accounting trace facility. This facility is started with the following command:

```
START TRACE (ACCTG) CLASS (1, 3, 8) DEST (SMF)
```

Class 8 provides accounting information on individual packages.

The `DEST` parameter specifies where the trace output is to be recorded. The recommendation is to write it in the System Management Facility (SMF) data set as the volume of records (one per plan or package executed) is low. Other destinations are Generalized Trace Facility (GTF) if you need to keep your trace records isolated, and Online Performance monitor destination (OPX) for online monitoring with the `MONITOR` trace started also.

Once trace information is recorded, DB2 PM can be used to generate a DB2 PM Accounting report. The DB2 PM command to generate the report is:

```
ACCOUNTING REPORT LAYOUT (layout: name)
INCLUDE (PLANNAME: plan)
```

You can also use the DB2 PM Online Monitor Collect Report Data Facility. DB2 PM automatically starts the desired DB2 traces and captures the trace data in a user-specified data set that can be used as input for DB2 PM Batch processing.

Figure 47 on page 120 contains an extract from the DB2 PM Accounting report for the process. Notice that the shaded fields are relevant when analyzing locking problems in the report.
DB2 PM summarizes all occurrences of the plan or program, but it is also possible to extract a single occurrence of the program by adding more conditions in the DB2 PM command used. Field #OCCURRENCES gives information on how many program executions are summarized.

To analyze locking problems, the most important field to look at is CLASS 3 SUSP. LOCK/LATCH, that is, the time the program waits for locks, or IRLM and DB2 latches and comparing this time with the total execution time for the program in DB2 CLASS 2 ELAPSED TIME. PAGE LATCH is the page latch contention time.

It is also important to look at counters for TIMEOUTS and DEADLOCKS to know if the program could not access the required resources and how many times?

Other interesting fields are the number of LOCK REQUEST and UNLOCK REQUEST, LOCK SUSPENS and LATCH REQUEST. You may be able to estimate the CPU overhead for locking and also guess whether the program uses lock-avoidance techniques: in fact, this is the only place to make a quick guess. Possible escalations are also very interesting indicators.

You can also see the number of COMMITS in the program.

Figure 47. Extract from DB2 PM Accounting Report
MAX LOCKS HELD is useful in determining whether the program is well designed for concurrency.

4.3 Identify Concurrency Problems

Having an unexpected number of programs running concurrently produces locking queues that lead to suspensions, increasing the elapsed time needed for the individual programs to end normally, or abend because of timeout or deadlock.

Information-technology installations are sized to support a certain level of concurrency. If this threshold is exceeded, a well-designed application may experience concurrency problems.

If concurrency problems are rare, installations usually tend to keep the same peaks and assume longer response times in their programs. If concurrency problems are more common, there is little that database or system administrators, or even application developers can do. The only possible solution is to increase the system capacity by adding more CPU power, or increasing the I/O subsystem capacity by adding more DASD or control units.

This concurrency problem sometimes is the first one investigated by the DB2 database or system administrator before the application analysis is done.

Three tools that the DB2 database or system administrator can use in this situation are:

- DISPLAY THREAD command
- DISPLAY DATABASE USE command
- DB2 PM Statistics report.

4.3.1 DISPLAY THREAD Command

The DB2 command DISPLAY THREAD displays current status information about DB2 threads. The threads can be active, inactive, or indoubt. The command also displays distributed threads.

By using this command, the DB2 database or system administrator can, at a glance, see how many users or programs are connected to the DB2 system, and how many of them are doing active work.

The command can be issued from an MVS console, from a DSN session under time sharing option (TSO), a DB2 Interactive (DB2I) panel, an IMS or CICS terminal, or a program using the Instrumentation Facility Interface (IFI). The syntax of the command is:

```-DISPLAY THREAD(*)```

This command lists all threads. Some can be inactive, meaning that no real work is being done, although they may still be holding locks, which is the reason why it can be important to be aware of them.

You can see only the active users by using the option TYPE. The command is then:

```-DISPLAY THREADS(*) TYPE(ACTIVE)```

The result of the command execution can be seen in Figure 48 on page 122.
Status (ST) displays the information related to the state of the thread. It can hold the following values:

- **N**, the thread is in either IDENTIFY or SIGNON status.
- **T**, an allied nondistributed thread is established.
- **D**, the thread is in the process of termination as a result of the termination of the associated allied thread.
- **QD**, the thread is queued for termination as a result of the termination of the associated allied task.
- **QT**, the CREATE THREAD request is queued.
- **RA**, the distributed thread is performing a remote access location. A request is presented to a DB2 DDF service task that establishes a system conversation. The thread is suspended until the request is processed.
- **RQ**, a distributed thread is performing a remote access on request from another location. The thread is suspended because the maximum number of active database access threads has been reached.
- **RN**, a distributed thread is performing a remote access on behalf of a request from another location.
- **SP**, a thread is executing within a stored procedure.
• SW, a thread is waiting for a stored procedure to be scheduled.
• TN, an allied distributed thread to access data at another DB2 location is suspended because DB2 must first connect to the partner location.
• TR, an allied thread distributed to access data at another location is established.

**Active flag** displays an asterisk if the thread is active.

**Requests (REQ)** displays the number of DB2 requests for the thread. It is a wrap-around counter, which can give a rough idea of the amount of work performed by DB2 on behalf of the thread.

### 4.3.2 DISPLAY DATABASE USE Command

This DB2 command displays information about the status of DB2 databases and its associated objects (table spaces, tables in segmented table spaces, indexes, and partitions) and additional information about who is using the database. It shows connection IDs, correlation IDs, and authorization IDs for all programs accessing the database.

It gives a more detailed display at a database level of the **DISPLAY THREAD** command.

The syntax of the command is:

```
DISPLAY DATABASE(db) SPACENAM(sp) USE
```

db is the database name and sp is the space or index space name. The option SPACENAM is optional. If not used or SPACENAM(*) is specified, DB2 displays information for all the objects in the database.

Figure 49 on page 124 shows a sample result of the DISPLAY DATABASE USE command.
Figure 49. Result of the DISPLAY DATABASE USE Command

Figure 49 shows that two concurrent users (USERID1 and USERID9) are accessing the table space TSTCUST in the database EXAMPLE.

4.3.3 DB2 PM Statistics Report

DB2 accumulates system statistics from the time it is started until it is stopped. The DB2 performance monitor writes trace records, for example, at the statistics interval that can be controlled by a ZPARM (default value is 30 minutes). These records contain accumulated fields, high-water-mark fields, and current information.

DB2 STATISTICS trace should be started to generate these records. The overhead of the statistics trace is very low, so starting this trace is highly recommended.

The syntax of the DB2 command is:

```
-START TRACE(STAT) CLASS(1,3) DEST(SMF)
```

We also recommend generating DB2 PM Statistics Report - Short, which gives all needed locking information.

The DB2 PM command to be used is:

```
STATISTICS
```

Figure 50 on page 125 contains an extract from the DB2 PM Statistics report.
In order to analyze locking activity, you must look at the fields in the block labeled **LOCKING ACTIVITY** (fourth block down in Figure 50). The fields **DEADLOCKS**, **TIMEOUTS**, **SUSPENSIONS-LOCK**, **LOCK REQUESTS** and **UNLOCK REQUESTS** contain the accumulated locking information for the DB2 subsystem.

**LOCK ESCALAT(SH)** and **LOCK ESCALAT(EX)** contain information related to lock escalation processes.

**DRAIN REQUESTS** and **CLAIM REQUESTS** contain information about claim and drain processing.
4.4 Identify Global Problems

Global problems are those problems not directly related to database design or program logic issues. Although these problems occur during periods of normal activity with a number of concurrent users and DB2 tasks, they cannot be classified as concurrency problems.

Global problems are usually related to the MVS system configuration, hardware and software, or to the DSNZPARM parameters. For example, an inadequate DB2 buffer pool can lead to a high increase in CPU usage and long elapsed times for DB2 applications. On the other hand, large DB2 buffer pools not backed up by adequate real storage can result in MVS paging activity that reduces the performance of DB2 applications.

We provide some guidelines on how to get the necessary information, both from a DB2 point of view and from the MVS system point of view, to analyze the situation.

We consider three basic tools:
- DB2 PM Statistics report
- DB2 PM I/O Activity report
- RMF reports.

4.4.1 DB2 PM Statistics Report

The DB2 PM Statistics report is an excellent tool to identify global problems affecting normal DB2 performance. DB2 PM statistics report provides information in several blocks:
- Highlights
- CPU times
- Open/close activity
- SQL DML
- SQL DCL
- SQL DDL
- Locking activity
- RID list
- Stored procedures
- Query parallelism
- Plan/package processing
- Subsystem services
- Log activity
- EDM pool
- Buffer pool activity
- DDF activity.

The important performance data is analyzed in further detail in Chapter 5 of *DB2 for MVS/ESA Administration Guide Volume 2 (SC26-3265-00).*
4.4.2 DB2 PM I/O Activity Report

The tracking of the I/O volumes and service times is key in monitoring system performance. DB2 performance data reflects reads and writes throughout the system, allowing us to monitor database buffer pools, the EDM pool, active and archive logs, and bootstrap data-set activity to detect abnormal situations leading to performance degradation and identify existing or potential bottlenecks related to I/O activity.

For further detail, please refer to DB2 PM Report Reference Volume 1 (SH12-6163-00).

4.4.3 RMF Reports

The MVS Resource Measurement Facility (RMF) provides a set of reports based on MVS trace records to report all kinds of events related to the MVS system and can be helpful in identifying problems not related to the DB2 system. Please refer to the MVS bookshelf library for more detailed information.

4.5 Analyze Locking Problems

At this point, you have identified a locking problem in your installation. To solve this problem, you may want to gather the following data:

- Details of locking activity in the DB2 system
- Average time by DB2 resource and workload category attributable to lock suspension delays
- Number of timeouts and deadlocks occurring in the system
- Users involved in a time-out or deadlock and the resources held
- Deadlock and timeout situations in the system and the participants.

The main tools for investigation are more DB2 commands, and DB2 trace records that can be used to generate useful reports using a monitoring and reporting tool such as DB2 PM.

The DB2 PM Locking report set provides various levels of detail about concurrency control within DB2 in the form of reports, summaries, or traces of individual events showing information about:

- Transaction locks, which are locks on table spaces, tables, pages or rows and are used primarily to control access by SQL statements
- Drains and claims that control access by DB2 utilities
- Lock-avoidance techniques and related locking data, such as page-latch waits.

Some of the DB2 traces started to generate locking records are quite expensive and can affect the overall performance of the system. They could also require a large amount of space; For this reason, we recommend you specify GTF as the trace destination.
4.5.1 Suspensions

There are various reasons for a program to be suspended, that is, to be waiting for an external process to deliver something the program needs to continue and has requested:

- In I/O suspensions, the program waits for the synchronous read to finish and make the page read available in buffers.
- In lock suspensions, the program waits for a resource locked by another program.
- In other suspensions, the program may wait for example, a service task switch.

There are a number of cases when an IRLM request cannot be satisfied and the agent requesting that service is suspended. The most common reason for suspension is that the requested service (for example, lock request) addresses an object that is already locked and has an incompatible lock state. The suspension of an agent can be resolved in one of the following ways:

- By removing the cause for the suspension, for example, releasing the locked resource.
- By a timeout, when the agent waits for the resource for longer than a preset interval, in which case control is passed back to DB2.
- By detecting a deadlock as the cause for the suspension and having DB2 roll back, one of the deadlocked application processes.

If programs are involved in lock contention problems, the main symptom is suspensions, that is, elapsed time waiting for a resource that is exclusively held by another program. This situation increases the response time of the programs. Wait time for locks is not externalized unless there is a timeout or deadlock.

There may be a chain effect if the program suspended for a lock is already holding locks requested by other programs, making them wait as well.

Assuming that you have identified the programs whose lock suspension times are an important part of the total execution time, you have the following tools to analyze the problem in more detail:

- DISPLAY DATABASE LOCKS command
- DISPLAY DATABASE CLAIMERS command
- DB2 PM Lock Suspension report
- DB2 PM Lock Suspension trace
- DB2 PM Lock Detail trace.

The DB2 PM Lock Detail trace gives information about every lock acquired and released. It should be used only when none of the other sources provides the specific information needed because, the DB2 PM Lock Detail trace requires DB2 Performance trace class 7 records, which can cause high CPU overhead.

4.5.1.1 DISPLAY DATABASE LOCKS Command

This command (discussed in 4.2.1, “DISPLAY DATABASE LOCKS command” on page 114) displays a snapshot of the locks requested (held and waiting) over a database and its associated objects.
4.5.1.2 DISPLAY DATABASE CLAIMERS Command

This command displays, in addition to the information related to the status of the database, information about all the claims on the database objects (table spaces, indexes and partitions) and the logical partitions that have logical claims, and the claims associated with them.

The syntax of the command is:

```
DISPLAY DATABASE(db) SPACENAM(sp) CLAIMERS
```

where `db` is the database name and `sp` is the space or index space name. Specifying SPACENAM is optional. If not specified, or SPACENAM(*) is specified, DB2 displays information about claimers on all the objects in the database.

For example, Figure 51 shows the display resulting from the DISPLAY DATABASE CLAIMERS command.

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Part</th>
<th>Status</th>
<th>ConnId</th>
<th>Corrid</th>
<th>ClaimInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSTCUST</td>
<td>TS</td>
<td>RW</td>
<td>RW</td>
<td>DB2CALL USERID1</td>
<td>(RR,C)</td>
<td></td>
</tr>
<tr>
<td>TSTCUST</td>
<td>TS</td>
<td>RW</td>
<td>RW</td>
<td>DB2CALL USERID0</td>
<td>(WR,C)</td>
<td></td>
</tr>
</tbody>
</table>

---

**Claim class descriptor**

**Claim duration**

---

**Figure 51. Result of the DISPLAY DATABASE CLAIMERS Command**

**CLAIMINFO** is the type and duration of currently held claims. The claim is composed of a claim class descriptor (CC) and a claim duration (CD) descriptor in the format: (CC,CD).

The CC descriptor has one of the following values:

- CS claim class
- RR claim class
- Write (WR) claim class.

The CD descriptor describes the duration of the claim as follows:

- A, the claim is held until deallocation.
• C, the claim is held until the next commit point.
• H, the claim is held across commit points.

4.5.1.3 DB2 PM Lock Suspension Report
The Lock Suspension report is a summary of lock suspensions in the reporting interval for a unique combination of selected DB2 PM identifiers. In addition to the number of occurrences and elapsed times, the report shows the cause of the suspension and the reason for resumption of processing.

This report summarizes the suspensions in the system during the trace interval, and allows the DB2 database or system administrator to identify the suspended plans, the resources causing the suspensions, the average elapsed time for each suspensions and its type, and the reason for the resumption of processing if the resource was freed before a timeout or a deadlock occurred.

This performance trace can produce some CPU overhead in the system so it is recommended to start it for the particular plan or user we need to analyze further.

The DB2 command needed to generate the required trace records is:

```
-START TRACE (PERFM) IFCID (44, 45, 105, 107, 213, 214, 215, 216, 226, 227)
DEST (GTF) PLAN (plan) AUTHID (user)
```

Once you have the trace data, the DB2 PM report command is:

```
LOCKING
```

Figure 52 on page 131 illustrates a DB2 PM Lock Suspension report:
<table>
<thead>
<tr>
<th>PRIMAUTH</th>
<th>PLANNAME</th>
<th>TYPE</th>
<th>NAME</th>
<th>SUSPENDS</th>
<th>LATCH</th>
<th>IRMLQ</th>
<th>OTHER</th>
<th>NMBR</th>
<th>AET</th>
<th>NMBR</th>
<th>AET</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMF001</td>
<td>DSNTEP3</td>
<td>INDEXOF</td>
<td>DB =DSNDB06</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.938008</td>
<td>0</td>
<td>N/C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CB =GSMRDG02</td>
<td>PART= 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSNUTIL</td>
<td>UTILSER</td>
<td>N/A</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.375616</td>
<td>0</td>
<td>N/C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>TOTAL</em></td>
<td>ADMF001</td>
<td>N/P</td>
<td>N/P</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.656812</td>
<td>0</td>
<td>N/C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADMF004</td>
<td>DSNTEP3</td>
<td>N/P</td>
<td>N/P</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.568704</td>
<td>0</td>
<td>N/C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SUM OF DSNTEP3**

<table>
<thead>
<tr>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
<th><strong>TOTAL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>6 0.656812</td>
<td>2 0.375616</td>
<td>2 0.568704</td>
<td>6 0.938008</td>
<td>0.273924</td>
<td>0.543525</td>
<td>0.122589</td>
<td>0.738858</td>
<td>0.111919</td>
<td>0.111919</td>
<td>0.738858</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 52. DB2 PM Lock Suspension Report*

The categorized reasons for suspensions (SUSPEND REASONS in Figure 52) are:

- **LOCAL**, the local resource is locked.
- **LATCH**, the resource access is serialized by latches.
- **IRLMQ**, IRLM queued request.
- **GLOB.**, global contention
- **S.NFY**, intersystem message sending.
- **OTHER**, suspensions are other than those listed above.

The resumption reasons (RESUME REASONS in Figure 52) can be:

- **NORMAL NMBR**, the number of suspensions that ended in a normal resumption of the process, both for locks and latches.
- **NORMAL AET**, the average elapsed time of a suspension that ended normally.
- **TIMEOUT NMBR**, the number of waits to access the locked resources that resulted in a timeout.
- **TIMEOUT AET**, the average elapsed time until the resumption due to timeout.
- **CANCEL NMBR**, the number of page latch suspensions that ended with the latch request being canceled.
- **CANCEL AET**, the average elapsed time of latch suspension that ended with the latch request being canceled.
4.5.1.4 DB2 PM Lock Suspension Trace
When the average numbers produced by the DB2 PM lock suspension report are not enough to explain the suspension problem, you need to have information for every suspension.

The lock suspension trace lists each lock suspension occurrence individually.

This trace identifies applications that have been suspended after requesting a lock on a resource that is not available. The trace shows an entry for each of the following events:

- Suspension of an IRLM request
- Suspension of an IRLM request where the resource type is a drain lock
- Suspension of a drain request where the drainer has to wait for the claim count on the particular resource to become zero
- Suspension of a page latch request.

The DB2 command needed to generate the required trace records is:

\[
\text{\textasciitilde START TRACE (PERFM) IPCID (44, 45, 105, 107, 213, 214, 215, 216, 226, 227) DEST (GTF) PLAN (plan) AUTHID (user)}
\]

Once you have the trace data, the DB2 PM command is:

\[
\text{\textasciitilde LOCKING TRACE LEVEL (SUSPENSION)}
\]

For further details, please refer to DB2 PM Report Reference Volume 1 (SH12-6163-00).

4.5.2 Timeouts and Deadlocks
It is important to monitor for timeouts and deadlock. Frequent timeouts and deadlocks indicate concurrency problems. These can either be due to locks being held for long durations or to applications not accessing the DB2 resources in the proper sequence.

We strongly recommend that the statistics class 3 trace always be active. It does not cause significant overhead and provides very important information about timeouts and deadlocks.

4.5.2.1 Timeouts
An application program is timed out when it is terminated because it has been suspended for longer than the installation-specified timeout interval.

After waiting for an interval specified by the timeout parameter, DB2 rolls back the unit of work. Timeouts can be considered the unsatisfactory resolution of a suspension state.
When the statistics class 3 trace is started, DB2 records one IFCID 196 record with the information of the timeout situation.

**Indications of a Timeout:** The indications for a timeout depend partly on the environment in which the DB2 program is executed:

- **TSO, Batch, CAF and CICS environments**  
  The programs return one of these SQLCODES:
  - -911 (SQLSTATE '40001')
  - -913 (SQLSTATE '57033')

- **IMS environments:** The programs return only SQLCODE -911 (SQLSTATE '40001').

**DB2 Console:** DB2 issues the `DSNT376I` system message each time a timeout situation is detected. The format of the message is:

```
DSNT376I PLAN plan-id1 WITH CORRELATION-ID id1 CONNECTION-ID id2 LUW-ID IS TIMED OUT. ONE HOLDER OF THE RESOURCE IS PLAN plan-id2 WITH CORRELATION-ID id4 CONNECTION-ID id5 LUW-ID id6 ON MEMBER id7
```

This message is usually followed by a `DSNT500I` (resource unavailable message) where the conflicting object can be identified. Also, additional information can be gathered by looking at the REASON field.

### 4.5.2.2 Deadlocks

A deadlock occurs when two or more application programs each hold locks on resources that the others need and without which they cannot proceed. Most common deadlock situations involve two programs, but cases involving more than two programs are not unusual.

After waiting for an interval specified by the deadlock time parameter, DB2 decides to roll back the unit of work of the process that did the least work so that all conflicting locks are released and the other program or programs can continue processing.

When the statistics class 3 trace is started, DB2 records one IFCID 172 record with the information on the deadlock and its resolution.

**Indications of a Deadlock:** The indications of a deadlock depend on the environment in which the DB2 program is executed:

- **TSO, Batch and CAF Environments:** When a deadlock occurs in these environments, DB2 decides whether or not to roll back one of the application processes. As a result, the application process receives one of two error codes in the SQLCODE field of the SQLCA:
  - -911: a ROLLBACK statement is issued on behalf of the application process. All updates during the current unit of work have been undone (SQLSTATE '40000').
  - -913: a ROLLBACK statement is not issued, but the application process itself is requested either to issue a ROLLBACK statement or to terminate (SQLSTATE '40502').

- **IMS:** If you are using IMS, and a deadlock or timeout occurs, the IMS attachment facility informs IMS, and IMS passes control to the Abort/Continue exit of the IMS attachment facility.
In any IMS environment except DL/I batch, one of the following actions occurs:

- DB2 performs a roll back on behalf of the application process. All DB2 updates during the current unit of recovery are undone. DB2 places SQLCODE -911 in the SQLCA.

- IMS issues a pseudo-abend and backs out all changes. For an MPP or IFP region, IMS reschedules the transaction. The application does not receive an SQLCODE.

If a deadlock occurs when a DL/I batch program is running, the application process abends with ABEND 04E, and either reason code 00D44033 or 00D44050 is placed in register 15.

**CICS:** If you are using CICS and a deadlock occurs, the CICS attachment facility decides whether or not to roll back one of the application processes, based on the value of the ROLBE or ROLBI parameter. If the application process is chosen for rollback, it receives one of two SQLCODEs in the SQLCA:

- -911: a SYNCPOINT command with the ROLLBACK option is issued on behalf of the application process. All updates (CICS commands and DL/I calls, as well as SQL statements) during the current unit of work have been undone (SQLSTATE '40001').

- -913: a SYNCPOINT command with the ROLLBACK option is not issued. DB2 rolls back only the incomplete SQL statement that encountered the deadlock or timed out. CICS does not roll back any resources. The application process itself should either issue a SYNCPOINT command with the ROLLBACK option or terminate (SQLSTATE '57033').

**DB2 Console:** DB2 issues the DSNT375I system message each time a deadlock is detected. The format of the message is:

```
DSNT375I PLAN plan-id1 WITH CORRELATION-ID id1 CONNECTION-ID id2 LUW-ID id3 IS DEADLOCKED WITH PLAN plan-id2 WITH CORRELATION-ID id4 CONNECTION-ID id5 LUW-ID id6 MEMBER id7
```

This message is usually followed by a DSNT500I (resource unavailable message) where the conflicting object can be identified. Also, additional information can be gathered by looking at the REASON field.

**4.5.2.3 DB2 PM Lockout Trace**

DB2 PM processes IFCID records 172 (deadlocks), 196 (timeouts) generated from statistics trace class 3, and IFCID record 105 (translating the database and object identifiers) generated from statistics trace class 1.

The DB2 PM Lockout trace shows when a timeout occurred and provides details of the resource involved in the timeout and information about the threads that held the resource or waited to use the resource.

The DB2 PM Lockout trace also contains an entry for every occurrence of a deadlock during a specified time period. The information includes when a deadlock occurred, details of the resources involved in the deadlock and about the threads that held the resource or waited to use the resource. If the resource is held by more than one agent and not all of them are actively involved in the deadlock, the holder cannot be determined and is not printed.

The DB2 command needed to generate the required trace records is:
Once you have the trace data, the DB2 PM command is:

**LOCKING TRACE LEVEL (LOCKOUT)**

Figure 53 shows a DB2 PM Lockout trace.

---

**Figure 53. DB2 PM Lockout Trace**

---
Look first at the EVENT field. This field contains either TIMEOUT or DEADLOCK indicating whether the associated information in the block is about a timeout or a deadlock.

If the EVENT field contains TIMEOUT, the relevant fields in this block are:

**EVENT TIMESTAMP**

The time at which the timeout occurred.

**RELATED TIMESTAMP**

The time at which the suspended request was timed out.

**LOCK RESOURCE TYPE**

The type of locked resource.

**LOCK RESOURCE NAME**

The name of the resource involved in the timeout.

**EVENT SPECIFIC DATA**

The data specific to the timeout for each resource involved.

If the EVENT field contains DEADLOCK, the relevant fields in this block are:

**EVENT TIMESTAMP**

The time at which the deadlock occurred.

**RELATED TIMESTAMP**

The time of the suspended request selected as the victim of this deadlock.

**LOCK RESOURCE TYPE**

The type of locked resource.

**LOCK RESOURCE NAME**

The name of the resource.

**EVENT SPECIFIC DATA**

The data specific to the deadlock. For each resource involved in a deadlock, there is a block of data on programs waiting and optionally, on their holders.

For more details, please refer to *DB2 PM Report Reference Volume 1 (SH12-6163-00)*.

### 4.6 Summary of Recommendations

Notification of a suspension may or may not mean a problem that needs resolution, but it does indicate a need for analysis. Many times the DB2 database or system administrator must rely on periodic monitoring to know if the system is suffering from contentions.

These three steps can generally identify the existence of locking problems, and how to analyze them:

- A locking problem exists if one of the following is true:
  - The lock suspension times for all the DB2 programs in the system is more than 1% of the total elapsed execution time. These numbers can be found in the DB2 PM Accounting report summary.
  - The number of transactions with more than 5 seconds of suspension times is more than 0.01% of the total.
  - The number of TSO or batch programs with more than 60 seconds of lock suspension times is more than 0.01% of the total.
• Find the most common victims. This can be easily done with the DB2 PM Exception report, setting a threshold locking suspension time of 5 seconds. Gather a list of the programs that are victims of suspensions.

• Find the culprits. This can be a far more difficult task that requires more dedication and resources. Some culprits are, at the same time victims of another program, typically a long-running with many updates per commit. Such cases are hard to analyze. A costly but useful approach is to employ User customized reports listing programs active during intervals when the analyzed plan is suspended. These customized reports can check resource dependencies in the DB2 catalog.

In cases involving mainly short duration transactions, the analysis may be much easier: DB2 PM Exception reports contain all transactions with class 3 elapsed time more than 5 seconds. The transaction that ends just before a program becomes a lock victim is the probable culprit. This can be confirmed when it commits and releases locks and all victims are able to complete their processing.
Chapter 5. Locking Scenarios

5.1 Scenario 1: Commit Frequency Test

This scenario illustrates how commit frequency affects the performance of DB2 applications.

In 3.5, “Summary of Recommendations” on page 107, we made some recommendations related to the need to commit frequently to avoid locking so much data as to compromise system concurrency. Of course, committing frequently to release locks has a cost. The commit frequency must be a compromise between availability and concurrency of data, and also performance of DB2 applications.

The example in Scenario 1 shows updating with a cursor using page locking. The table has 90972 rows contained in 2178 pages. The access to data is through a matching index, and 8664 rows are updated.

Figure 54 shows the program logic.

The program logic implements conditional intermediate commits so that commit frequency is input as an external parameter. The cursor CSR is defined WITH HOLD so that the program does not have to reposition after each commit point.
The scenarios test a variety of commit frequencies, from one commit per update to only one commit per program (or 8664 updates).

The results of the test are shown in Table 15.

<table>
<thead>
<tr>
<th>Commit Frequency</th>
<th>Elapsed Class 2</th>
<th>CPU Class 2</th>
<th>Commits</th>
<th>Updates per Commit</th>
<th>Getpages</th>
<th>Maximum Locks Held</th>
<th>STS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every 8664 updates</td>
<td>3.13</td>
<td>1.46</td>
<td>1</td>
<td>8664</td>
<td>242</td>
<td>242</td>
<td>0.55</td>
</tr>
<tr>
<td>Every 1000 updates</td>
<td>5.19</td>
<td>1.86</td>
<td>9</td>
<td>963</td>
<td>247</td>
<td>28</td>
<td>2.49</td>
</tr>
<tr>
<td>Every 500 updates</td>
<td>8.82</td>
<td>1.91</td>
<td>18</td>
<td>482</td>
<td>254</td>
<td>15</td>
<td>5.73</td>
</tr>
<tr>
<td>Every 100 updates</td>
<td>17.79</td>
<td>1.98</td>
<td>86</td>
<td>101</td>
<td>321</td>
<td>4</td>
<td>13.28</td>
</tr>
<tr>
<td>Every 50 updates</td>
<td>31.58</td>
<td>2.00</td>
<td>170</td>
<td>51</td>
<td>406</td>
<td>3</td>
<td>27.81</td>
</tr>
<tr>
<td>Every 10 updates</td>
<td>90.21</td>
<td>2.37</td>
<td>867</td>
<td>10</td>
<td>998</td>
<td>2</td>
<td>80.50</td>
</tr>
<tr>
<td>Every 1 update</td>
<td>686.20</td>
<td>6.92</td>
<td>8664</td>
<td>1</td>
<td>8678</td>
<td>1</td>
<td>669.8</td>
</tr>
</tbody>
</table>

Notes:
- Elapsed Class 2: Elapsed time executing DB2 calls.
- CPU Class 2: CPU time executing DB2 calls.
- Commits: Number of commits (Close to number of rows updated, 8664, divided by Commit Frequency).
- Updates per Commit: Number of rows updated, 8664, divided by Commits.
- Getpages: Number of getpage requests.
- Maximum Locks Held: High-water-mark for the number of locks held by the program.
- STS: Service task switch suspensions (in this example, corresponds to synchronous log writes).

5.1.1 Elapsed and CPU Time Analysis

As can be seen in Figure 55 on page 141, the elapsed time of the program is the factor most affected by commit frequency. The increase in elapsed time is mainly because of service task switch suspensions, or times spent waiting for the committed data to be written to the DB2 active logs.

The order-of-magnitude increase in elapsed time is also due to the slow 3380 devices that contain the log data sets in the test scenario. If the logs are located on faster 3390 devices with DASD fast write (DFW) capabilities, the elapsed time increase is much less significant.

As can be seen in Figure 55 on page 141, the CPU cost increases considerably as the commit frequency rises. Based on the data in Table 15, the best approach appears to be committing every 1000 updates, as the increase in elapsed and CPU times is not high. This strategy may not be good for other programs, however. Here, the 1000 updated rows reside in fewer than 28 pages (See column Maximum Locks Held in Table 15). If updating 1000 rows would cause 1000 pages to be locked, concurrency would definitely be affected.
Figure 55 on page 141 shows the elapsed time and CPU consumption for increasing numbers of commits.

### 5.1.2 Analysis of CPU time and Maximum Pages Locked

The objective of this analysis is basically to show that concurrency has its cost. As can be seen in Figure 56 on page 142, as the maximum pages locked decreases (thus increasing the possible concurrency for the system), the CPU consumption of the program increases (due to the CPU cost of the additional commit points).

Figure 56 on page 142 shows the CPU consumption and maximum pages locked for increasing numbers of commits.
5.1.3 When to Commit?

The question, then, is when to commit. We have seen that frequent commits is expensive because of increased CPU consumption and elapsed time, and infrequent commits hurts concurrency. Given a theoretical commit frequency interval of 50 updates, the total interval time for those 50 updates (that is, the total time the pages are actually locked), depends on the program access path, the physical disposition of the updates (in case of page or row locking), and the CPU cost of the accesses.

Adding more logic to the program, so that it commits every 0.5 to 5 seconds, appears to be a useful approach. The exact value can be determined based on the needs of the environment, the characteristics of the program, and the longest running SQL statement in the program.

If any SQL call takes more than 5 seconds, any possible commit interval based control is of no use, as previous locks remain until the SQL finishes. We recommend modifying the database design (for instance, adding one more index) or the logic of the program. If this is not possible, then the only solution is to insert a commit point before the long SQL call so that all previous locks are not held until the long SQL call finishes.
The general structure of the proposed commit strategy is shown in Figure 57 on page 143.

To test this recommended approach, the initial program is modified so that the commit interval is now defined in seconds and not by the actual number of updates. This way, the program can self-control and avoid problems associated with different absolute commit time intervals that have different access costs.

The results of the test are shown in Table 16 on page 144.

As can be seen in Table 16 on page 144, the optimal commit strategy for this program seems to be to commit every 0.5 to 1 second.
Table 16. Result Table: Commit Frequency Based on Number of Seconds

<table>
<thead>
<tr>
<th>Commit Frequency</th>
<th>Elapsed Class 2</th>
<th>CPU Class 2</th>
<th>Commits</th>
<th>Updates per Commit</th>
<th>Getpages</th>
<th>Maximum Locks Held</th>
<th>STS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every 1 second</td>
<td>2.22</td>
<td>1.50</td>
<td>3</td>
<td>2888</td>
<td>226</td>
<td>76</td>
<td>0.65</td>
</tr>
<tr>
<td>Every 0.5 second</td>
<td>2.03</td>
<td>1.52</td>
<td>6</td>
<td>1444</td>
<td>230</td>
<td>39</td>
<td>1.2</td>
</tr>
<tr>
<td>Every 0.1 second</td>
<td>46.28</td>
<td>2.14</td>
<td>358</td>
<td>25</td>
<td>576</td>
<td>24</td>
<td>42.21</td>
</tr>
<tr>
<td>Every 0.01 second</td>
<td>687.54</td>
<td>6.57</td>
<td>8664</td>
<td>1</td>
<td>8676</td>
<td>1</td>
<td>669.82</td>
</tr>
</tbody>
</table>

Notes:
- **Elapsed Class 2**: Elapsed time executing DB2 calls.
- **CPU Class 2**: CPU time executing DB2 calls.
- **Commits**: Number of commits depending on Commit Frequency.
- **Updates per Commit**: Number of rows updated (8664) divided by Commits.
- **Getpages**: Number of getpage requests.
- **Maximum Locks Held**: High water mark for the number of locks held by the program.
- **STS**: Service Task Switch suspensions, basically in this example this time corresponds to synchronous log writes.

5.2 Scenario 2: CPU Cost of Row Locking

Row locking is a feature introduced with Type 2 indexes in DB2 V4.

Row locking can help to increase the concurrency in hot tables, as programs lock only the rows that are actually updated. But, just as in Scenario 1, a decrease in the amount of data locked is associated with an increase in CPU consumption. With row locking, DB2 requires an individual call to IRLM for each row that is updated, compared with page locking where one lock is sufficient for all rows in the page.

From the IRLM point of view, a row lock has the same characteristics as a page lock and uses the same resources in memory and CPU usage. That is why the cost of locking a row is the same as the cost of locking the whole page. Programs that update many rows in the same page suffer most when changing from page locking to row locking.

The test in Scenario 2 reproduces a single dynamic SQL update statement that updates 23826 rows using a 50 MIPS processor machine under three conditions:

- Table locking (LOCKSIZE TABLE)
- Page locking (LOCKSIZE PAGE)
- Row locking (LOCKSIZE ROW)

The results of the test are contained in Table 17 on page 145.
### Table 17. Result Table: Comparison of Table, Page, and Row Locking

<table>
<thead>
<tr>
<th>LOCKSIZE</th>
<th>Elapsed Class 2</th>
<th>CPU Class 2</th>
<th>Lock Requests</th>
<th>Max. Locks Held</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>13.37</td>
<td>1.98</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>PAGE</td>
<td>12.89</td>
<td>2.08</td>
<td>660</td>
<td>607</td>
</tr>
<tr>
<td>ROW</td>
<td>21.68</td>
<td>3.27</td>
<td>23983</td>
<td>23826</td>
</tr>
</tbody>
</table>

**Notes:**

- **Elapsed Class 2**: Elapsed time executing DB2 calls.
- **CPU Class 2**: CPU time executing DB2 calls.
- **Max. Locks Held**: High-water mark for the number of locks held by the program.

The CPU cost of locking from Table 17 can be derived as the difference in CPU time divided by the difference in lock requests. In the test case, changing from page locking to row locking, the CPU cost of locking is:

\[
\frac{(3.27 - 2.08)}{(23983 - 660)} = 51
\]

That is, the CPU uses an average of 51 microseconds for the lock, change, and unlock operation, or about 17 microseconds per lock operation in a 50 MIPS processor.

This CPU cost per lock can be significant if the program needs to update many rows. Therefore, we recommend the use of row locking only when the concurrency requirements are very high.

Figure 58 on page 146 is a graphical representation of elapsed time and CPU consumption values for table, page, and row locks from Table 17.
Row locking is then recommended instead of old methods of pseudo-row locking, such as table column padding (that is, adding dummy columns to increase the total row length of the table so that only one row fits on one page), or high PCTFREE values (that is, defining PCTFREE table space parameter to 100 to avoid having more than one row per page) when high concurrency is needed.

5.3 Scenario 3: DELETE

In this example, we examine the alternatives for deleting 1% of the table ORDER keys (437000 keys in total). The table is clustered on the key sequence.

The ORDER table has two additional indexes. All indexes are Type 2. The ORDER table is in a segmented table space. The table has 437000 rows located in 5532 pages.

The table space resides in buffer pool BP0, and the indexes reside in buffer pool BP2.

The ORDER table has three Type 2 indexes. One of them is a unique clustering index with a cluster ratio of 100. Two indexes are nonunique, nonclustering, and the
cluster ratio is 5. All the indexes have three levels. The number of leaf pages in the clustering index is 1325 and in each of the other two indexes 768.

In this scenario we are going to evaluate five possible programming alternatives to execute the DELETE statement. We will evaluate them based mainly on elapsed time, CPU consumption, number of maximum locks held, and maximum lock duration.

The five alternatives to be considered are:

1. Case A: Singleton DELETE statement
2. Case B: DELETE using a CURSOR WITH HOLD, one COMMIT per key deleted
3. Case C: DELETE using a CURSOR WITH HOLD, one COMMIT per second
4. Case D: DELETE using a CURSOR, reposition after COMMIT (no HOLD), one COMMIT per second
5. Case E: DELETE using a CURSOR WITH HOLD, one COMMIT per second (same as case C) but using row locking.

All programs are bound with ISOLATION(CS) and CURRRENDATA(NO), and table space is defined with LOCKSIZE ANY except for Case E, so pages are the lock units.

5.4 Case A

This is the simplest case from a programming point of view. The DB2 program just issues a singleton SQL DELETE statement that deletes 1% of the rows from the ORDER table (4370 rows).

The SQL statement used in the program is:

```
EXEC SQL DELETE FROM ORDER
WHERE ORDERNO BETWEEN 300001 AND 304370;
```

The access path for the program is shown in Table 18.

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Matching Columns</th>
<th>Index Only</th>
<th>Sorts</th>
<th>TS Lock Mode</th>
<th>Prefetch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A</td>
<td>Index</td>
<td>1</td>
<td>Yes</td>
<td>IX</td>
<td>Sequential Prefetch</td>
</tr>
</tbody>
</table>

The DB2 PM Accounting trace for the highlights, timings, locking, and buffer pool information for the execution of this program is shown in Figure 59 on page 148.
<table>
<thead>
<tr>
<th>Times/Events</th>
<th>Appl (Class 1)</th>
<th>Db2 (Class 2)</th>
<th>Ifi (Class 5)</th>
<th>Class 3 Susp.</th>
<th>Elapsed Time</th>
<th>Events</th>
<th>Highlights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time</td>
<td>44.205122</td>
<td>44.136785</td>
<td>N/P</td>
<td>Lock/Latch</td>
<td>0.016716</td>
<td>17</td>
<td>Thread Type: Allied</td>
</tr>
<tr>
<td>Cpu Time</td>
<td>18.725589</td>
<td>18.712875</td>
<td>N/P</td>
<td>Synchron., I/O</td>
<td>0.928788</td>
<td>72</td>
<td>Term.Condition: Normal</td>
</tr>
<tr>
<td>Tcb</td>
<td>18.725589</td>
<td>18.712875</td>
<td>N/P</td>
<td>Other Read I/O</td>
<td>0.000000</td>
<td>0</td>
<td>Invoke Reason: Dealoc</td>
</tr>
<tr>
<td>Tcb-SubProc</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A</td>
<td>Other Write I/O</td>
<td>19.771009</td>
<td>123</td>
<td>Commits: 1</td>
</tr>
<tr>
<td>Cpu-Parall.</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A</td>
<td>Ser.,Task Switch</td>
<td>0.597646</td>
<td>6</td>
<td>Rollback: 0</td>
</tr>
<tr>
<td>Not Account.</td>
<td>N/A</td>
<td>4.109751</td>
<td>N/P</td>
<td>Arc.,Log(Ques)</td>
<td>0.000000</td>
<td>0</td>
<td>Increm.BINDS: 0</td>
</tr>
<tr>
<td>Db2 Ent/Exit</td>
<td>N/A</td>
<td>5</td>
<td>N/A</td>
<td>Arc.,Log Read</td>
<td>0.000000</td>
<td>0</td>
<td>Update/Commit: 1.00</td>
</tr>
<tr>
<td>Bin/Bx-SubProc</td>
<td>N/A</td>
<td>0</td>
<td>N/A</td>
<td>Drain Lock</td>
<td>0.000000</td>
<td>0</td>
<td>Programs: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Claim Release</td>
<td>0.000000</td>
<td>0</td>
<td>Parallellism: No</td>
</tr>
<tr>
<td>Dcap.,Descri.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/P</td>
<td>Page Latch</td>
<td>0.000000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Log Extract.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/P</td>
<td>Stored Proc.</td>
<td>0.000000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Notify MSGs</td>
<td>0.000000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Global Cont.</td>
<td>0.000000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total Class</td>
<td>21.314159</td>
<td>218</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sql Dml</th>
<th>Total</th>
<th>Sql Dcl</th>
<th>Total</th>
<th>Sql Dl</th>
<th>Create</th>
<th>Drop</th>
<th>Alter</th>
<th>Locking</th>
<th>Total</th>
<th>Data Sharing</th>
<th>Total</th>
<th>Data Capture</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>0</td>
<td>Lock</td>
<td>Table</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Timouts</td>
<td>0</td>
<td>Lock Request</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert</td>
<td>0</td>
<td>Grant</td>
<td>Index</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Deadlocks</td>
<td>0</td>
<td>Unlock Request</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td>0</td>
<td>Request</td>
<td>TableSpace</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Escal.(SRM)</td>
<td>0</td>
<td>Change Request</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deltek</td>
<td>1</td>
<td>Set</td>
<td>HVar.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Max.Lck Held</td>
<td>56</td>
<td>Unlock-Xes</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Describe</td>
<td>0</td>
<td>Set</td>
<td>Degree</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>Lock Request</td>
<td>67</td>
<td>Change-Xes</td>
<td>N/P</td>
<td></td>
</tr>
<tr>
<td>Desc.Tbl</td>
<td>0</td>
<td>Set</td>
<td>Rules</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>Unlock Request</td>
<td>10</td>
<td>Susp. - IRM</td>
<td>N/P</td>
<td></td>
</tr>
<tr>
<td>Prepare</td>
<td>0</td>
<td>Connect</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>Query Request</td>
<td>0</td>
<td>Susp. - XEs</td>
<td>N/P</td>
<td></td>
</tr>
<tr>
<td>Open</td>
<td>0</td>
<td>Connect</td>
<td>2</td>
<td>0</td>
<td>Package</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
<td>Change Request</td>
<td>0</td>
<td>Susp. - False</td>
<td>N/P</td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>0</td>
<td>Set</td>
<td>Connect</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Other Request</td>
<td>0</td>
<td>Incomp.Lock</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td>0</td>
<td>Release</td>
<td>Total</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Lock Susp.</td>
<td>0</td>
<td>Notify Sent</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dml-All</td>
<td>1</td>
<td>Dcl-All</td>
<td>Comment</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Other Susp.</td>
<td>0</td>
<td>Total Susp.</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rid List</th>
<th>Total</th>
<th>Query Parallell.</th>
<th>Total</th>
<th>Stored Proc.</th>
<th>Total</th>
<th>Drain/Claim</th>
<th>Total</th>
<th>Data Capture</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used</td>
<td>0</td>
<td>Maximum Degree</td>
<td>0</td>
<td>Call Stmts</td>
<td>0</td>
<td>Drain Request</td>
<td>0</td>
<td>Ifi Calls</td>
<td>N/P</td>
</tr>
<tr>
<td>Fail-No Storage</td>
<td>0</td>
<td>Groups Executed</td>
<td>0</td>
<td>Proc. Abends</td>
<td>0</td>
<td>Drain Failed</td>
<td>0</td>
<td>Reg.Captured</td>
<td>N/P</td>
</tr>
<tr>
<td>Fail-Limit Exc.</td>
<td>0</td>
<td>Planned Degree</td>
<td>0</td>
<td>Call Timet</td>
<td>0</td>
<td>Claim Request</td>
<td>6</td>
<td>Log Rec.Read</td>
<td>N/P</td>
</tr>
<tr>
<td>Reduced-No Buff</td>
<td>0</td>
<td>Call Reject</td>
<td>0</td>
<td>Claim Failed</td>
<td>0</td>
<td>RonG Return</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seq - Cursor</td>
<td>0</td>
<td>Seq - No Buff</td>
<td>0</td>
<td>Seq - No Buff</td>
<td>0</td>
<td>Seq - Xls.</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seq - No Ea</td>
<td>0</td>
<td>Seq - Xls.</td>
<td>0</td>
<td>Seq - Xls.</td>
<td>0</td>
<td>Seq - Xls.</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seq - Encl. Ser.</td>
<td>0</td>
<td>Seq - Xls.</td>
<td>0</td>
<td>Seq - Xls.</td>
<td>0</td>
<td>Seq - Xls.</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parall.Disabled</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bp0</th>
<th>Total</th>
<th>Bp2</th>
<th>Total</th>
<th>Total</th>
<th>Total</th>
<th>Total</th>
<th>Total</th>
<th>Total</th>
<th>Total</th>
<th>Total</th>
<th>Total</th>
<th>Total</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expansions</td>
<td>N/A</td>
<td>Expansions</td>
<td>N/A</td>
<td>Expansions</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GetPages</td>
<td>166</td>
<td>GetPages</td>
<td>39346</td>
<td>GetPages</td>
<td>39512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffer Updates</td>
<td>4480</td>
<td>Buffer Updates</td>
<td>13110</td>
<td>Buffer Updates</td>
<td>17590</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronous Write</td>
<td>0</td>
<td>Synchronous Write</td>
<td>0</td>
<td>Synchronous Write</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronous Read</td>
<td>57</td>
<td>Synchronous Read</td>
<td>15</td>
<td>Synchronous Read</td>
<td>72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential Prefetch</td>
<td>0</td>
<td>Sequential Prefetch</td>
<td>3</td>
<td>Sequential Prefetch</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List Prefetch</td>
<td>0</td>
<td>List Prefetch</td>
<td>0</td>
<td>List Prefetch</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Prefetch</td>
<td>0</td>
<td>Dynamic Prefetch</td>
<td>0</td>
<td>Dynamic Prefetch</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pages Read Async.</td>
<td>0</td>
<td>Pages Read Async.</td>
<td>33</td>
<td>Pages Read Async.</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hpool Writes</td>
<td>0</td>
<td>Hpool Writes</td>
<td>0</td>
<td>Hpool Writes</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hpool Writes Failed</td>
<td>0</td>
<td>Hpool Writes Failed</td>
<td>0</td>
<td>Hpool Writes Failed</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pages Read-Hpool</td>
<td>0</td>
<td>Pages Read-Hpool</td>
<td>0</td>
<td>Pages Read-Hpool</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hpool Reads</td>
<td>0</td>
<td>Hpool Reads</td>
<td>0</td>
<td>Hpool Reads</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hpool Reads Failed</td>
<td>0</td>
<td>Hpool Reads Failed</td>
<td>0</td>
<td>Hpool Reads Failed</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 59. Scenario 3: Case A – Accounting Trace
There is only the singleton SQL DELETE statement that locks a maximum of 56 pages, that is 1% of the total number of pages. These pages are locked until the COMMIT point; that is, these 56 data pages are locked for the whole duration of the program inside DB2, 44.14 seconds.

Observe that the number of unlock requests does not match the number of lock requests. This is because DB2 requests a so-called blank unlock that releases all locks at one time at a page-set level.

5.5 Case B

Assume that the DB2 system administrator does not like the idea of having some data pages locked for such a long time in Case A and is also concerned about the buffer pool requirements for the changed and not-committed pages. For these reasons, a change in the program is requested.

In the revised solution the program performs the DELETE as in Case A, but with a cursor-based delete and doing one COMMIT per row deleted. This cursor is defined WITH HOLD so that cursor position is not lost after the COMMIT statement is executed and there is no need to program reposition logic.

The SQL statement embedded in the program is:

```sql
EXEC SQL DECLARE CSR CURSOR WITH HOLD FOR
SELECT * FROM ORDER
WHERE ORDERNO BETWEEN 300001 AND 304370
FOR UPDATE OF ORDERNO;
EXEC SQL FETCH CSR INTO DCLORD;
EXEC SQL DELETE FROM ORDER
WHERE CURRENT OF CSR;
```

The access path for this program is shown in Table 19.

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Matching Columns</th>
<th>Index Only</th>
<th>Sorts</th>
<th>TS Lock Mode</th>
<th>Prefetch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case B</td>
<td>Index</td>
<td>1</td>
<td>No</td>
<td>IX</td>
<td>List Prefetch</td>
</tr>
</tbody>
</table>

The DB2 PM Accounting trace for the highlights, timings, locking, and buffer pool information for the execution of this program is shown in Figure 60 on page 150.
<table>
<thead>
<tr>
<th>TIMES/EVENTS</th>
<th>APPL (CLASS 1)</th>
<th>DB2 (CLASS 2)</th>
<th>IFI (CLASS 5)</th>
<th>CLASS 3 SUSP.</th>
<th>ELAPSED TIME</th>
<th>EVENTS</th>
<th>HIGHLIGHTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELAPSED TIME</td>
<td>1:23.036673</td>
<td>1:22.336432</td>
<td>N/P</td>
<td>LOCK/LATCH</td>
<td>0.002259</td>
<td>19</td>
<td>THREAD TYPE : ALLIED</td>
</tr>
<tr>
<td>CPU TIME</td>
<td>29.010772</td>
<td>28.221776</td>
<td>N/P</td>
<td>SYNCHRON. I/O</td>
<td>15.133765</td>
<td>1583</td>
<td>TERM.CONDITION: NORMAL</td>
</tr>
<tr>
<td>TCB</td>
<td>29.010772</td>
<td>28.221776</td>
<td>N/P</td>
<td>OTHER READ I/O</td>
<td>0.047580</td>
<td>1</td>
<td>INVOKE REASON : DEALLOC</td>
</tr>
<tr>
<td>TCB-SPROCP</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A</td>
<td>OTHER WRITE I/O</td>
<td>2.384304</td>
<td>64</td>
<td>COMMITS : 4371</td>
</tr>
<tr>
<td>CPU-PARALL.</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A</td>
<td>SER.TASK SWITCH</td>
<td>36.38463</td>
<td>4368</td>
<td>ROLLBACK : 0</td>
</tr>
<tr>
<td>NOT ACCOUNT.</td>
<td>N/A</td>
<td>N/C</td>
<td>N/P</td>
<td>ARC.LOG(QUIES)</td>
<td>0.000000</td>
<td>0</td>
<td>TERM.MEM.BINDS : 0</td>
</tr>
<tr>
<td>DB2 ENT/EXIT</td>
<td>N/A</td>
<td>26227</td>
<td>N/A</td>
<td>ARC.LOG READ</td>
<td>0.000000</td>
<td>0</td>
<td>UPDATE/COMM. : 1.00</td>
</tr>
<tr>
<td>RNDK-YK-SPROCP</td>
<td>N/A</td>
<td>0</td>
<td>N/A</td>
<td>DRAIN LOCK</td>
<td>0.000000</td>
<td>0</td>
<td>PROGRAMS : 1</td>
</tr>
<tr>
<td>DCA.P DESC.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/P</td>
<td>PAGE LATCH</td>
<td>0.000000</td>
<td>0</td>
<td>PARALLELISM : NO</td>
</tr>
<tr>
<td>LOG EXTRACT.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/P</td>
<td>STORED PROC.</td>
<td>0.000000</td>
<td>0</td>
<td>NO COMM. SENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMES/EVENTS</td>
<td>SQL DML TOTAL</td>
<td>SQL DCL TOTAL</td>
<td>SQL DDL CREATE</td>
<td>DROP</td>
<td>ALTER</td>
<td>LOCKING</td>
<td>TOTAL</td>
</tr>
<tr>
<td>SELECT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INSERT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UPDATE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DELETE</td>
<td>4370</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PREPARE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OPEN</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FETCH</td>
<td>4371</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLOSE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DML-ALL</td>
<td>8742</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RFD LIST TOTAL</th>
<th>QUERY PARALLEL</th>
<th>STORED PROC. TOTAL</th>
<th>DRAIN/CLAIM TOTAL</th>
<th>DATA CAPTURE TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>USED</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FAIL-NO STORAGE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FAIL-LIMIT EXC.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP0 TOTAL</td>
<td></td>
<td>BP2 TOTAL</td>
<td>TOTAL</td>
<td></td>
</tr>
<tr>
<td>EXPANSIONS</td>
<td>N/A</td>
<td>EXPANSIONS</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>GEITPages</td>
<td>4483</td>
<td>GEITPages</td>
<td>39355</td>
<td>43838</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>13110</td>
<td>BUFFER UPDATES</td>
<td>13110</td>
<td>17590</td>
</tr>
<tr>
<td>SYNCHRONOUS WHITE</td>
<td>0</td>
<td>SYNCHRONOUS WHITE</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>2</td>
<td>SYNCHRONOUS READ</td>
<td>1561</td>
<td>1563</td>
</tr>
<tr>
<td>SEQUENTIAL PREFETCH</td>
<td>0</td>
<td>SEQUENTIAL PREFETCH</td>
<td>114</td>
<td>114</td>
</tr>
<tr>
<td>LIST PREFETCH</td>
<td>0</td>
<td>LIST PREFETCH</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>DYNAMIC PREFETCH</td>
<td>0</td>
<td>DYNAMIC PREFETCH</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PAGES READ ASYNCHR.</td>
<td>56</td>
<td>PAGES READ ASYNCHR.</td>
<td>93</td>
<td>149</td>
</tr>
<tr>
<td>HPOOL WRITES</td>
<td>0</td>
<td>HPOOL WRITES</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HPOOL READS - FAILED</td>
<td>0</td>
<td>HPOOL READS - FAILED</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PAGES READ-HPOOL</td>
<td>0</td>
<td>PAGES READ-HPOOL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HPOOL READS</td>
<td>0</td>
<td>HPOOL READS</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HPOOL READS - FAILED</td>
<td>0</td>
<td>HPOOL READS - FAILED</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 60. Scenario 3: Case B – Accounting Trace
Compared with Case A, there is almost a doubling of the elapsed time for the program, mainly because of SER.TASK SWTCH Class 3 suspensions. These are service task switch (write to log) suspensions. There are 4368 of these suspensions and the suspension time is 36.64 seconds. The 4368 suspensions contrasted to 4,371 commits means that a suspension occurs every time a commit takes place.

CPU times have also increased, due to the additional COMMIT calls and the RID list sort processing that is done once.

The number of lock and unlock requests increases (4386 and 4384 respectively) because a commit is performed after each row is deleted. Also 4373 change requests result from the change from U-lock to X-lock.

As the maximum number of locks held is only one, concurrency of the table is increased.

Cursor WITH HOLD keeps an S-lock in the processing page across commits. Qualifying rows for the ORDER table reside in 5,532 * 0.01 = 55 pages, meaning that the same page is locked across commits 4371 / 55 = 79.47 times. This makes the maximum time a page is locked = 79.47 * (83.03 / 4371) = 1.51 seconds, which is far below the recommended maximum of 5 seconds.

The number of buffer updates for the data pages (buffer pool BP0) is the same as the number of getpages. As soon as a data row is deleted, a commit forces a new getpage for the same page. This also contributes to the CPU increase.

The main reason for the total CPU time increase (almost 10 seconds) is the additional SQL calls (4370 fetches, deletes, and commits).

Why is the optimizer choosing a list prefetch?. The answer is in the FOR UPDATE OF ORDERNO clause and the fact that the program uses the key index as the value to update. In other cases, we would expect an index scan with dynamic prefetch. When a column other than ORDERNO is used in the FOR UPDATE OF, the optimizer chooses an index scan with dynamic prefetch.

5.6 Case C

Assume that the users complain about the new version of the program (Case B) that takes almost twice as long to complete as in Case A.

The main cause of the long elapsed time is the write to logs that is done for each commit, so we decide to implement the recommended solution of adding some logic to control the commit interval time. We set this time to 1 second.

The SQL statements embedded in the program are exactly the same as in Case B, and so is the access path.

The DB2 PM Accounting trace for the highlights, timings, locking, and buffer pool information for the execution of this program is shown in Figure 61 on page 152.
### DB2 Performance Monitor (V4)

**Accounting Trace - Long**

**Requested From:** Not Specified  
**Actual From:** 03/03/96 14:45:06.44

**Member:** N/P  
**To:** Not Specified  
**Subsystem:** DB2F

<table>
<thead>
<tr>
<th>TIMES/EVENTS</th>
<th>APPL (CLASS 1)</th>
<th>DEB (CLASS 2)</th>
<th>IFI (CLASS 5)</th>
<th>CLASS 3 SUSP.</th>
<th>ELAPSED TIME</th>
<th>EVENTS</th>
<th>HIGHLIGHTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELAPSED TIME</td>
<td>1:12.73199</td>
<td>1:11.97361</td>
<td>N/P LOCK/LATCH</td>
<td>0.00588</td>
<td>24 THREAD TYPE : ALLIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU TIME</td>
<td>22.665167</td>
<td>21.895209</td>
<td>N/P SYNCHRON. 1/O</td>
<td>25.576321</td>
<td>1566 TERM.CONDITION: NORMAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>22.665167</td>
<td>21.895209</td>
<td>N/P OTHER READ 1/O</td>
<td>0.040798</td>
<td>1 INVOKE REASON : DEALLOC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP-IFIT</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A OTHER WRITE 1/O</td>
<td>19.241713</td>
<td>98 COMMITS : 66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU-PARALL.</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A SER.TASK SWITCH</td>
<td>2.776857</td>
<td>81 ROLLBACK : 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT ACCOUNT.</td>
<td>N/A</td>
<td>2.380577</td>
<td>N/P ARC.LOCK(QUES)</td>
<td>0.000000</td>
<td>0 INKREM.BINDS : 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEB ENT/EXIT</td>
<td>N/A</td>
<td>17617</td>
<td>N/P ARC.LOCK READ</td>
<td>0.000000</td>
<td>0 UPDATE/COMM : 66.21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BN/BK-IFIT</td>
<td>0</td>
<td>N/A</td>
<td>N/A DRAIN LOCK</td>
<td>0.000000</td>
<td>0 PROGRAMS : 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CLAIM RELEASE</td>
<td>0.000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PARALLELISM : NO</td>
<td></td>
</tr>
<tr>
<td>DCSPT.DESC.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/P PAGE LATCH</td>
<td>0.000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOG EXTRACT.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A STORED PROC.</td>
<td>0.000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NOTIFY MSGS</td>
<td>0.000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GLOBAL CONT.</td>
<td>0.000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TOTAL CLASS 3</td>
<td>47.641576</td>
</tr>
</tbody>
</table>

**SQL DML**

<table>
<thead>
<tr>
<th>SQL DML</th>
<th>TOTAL</th>
<th>SQL DCL</th>
<th>TOTAL</th>
<th>SQL DDL</th>
<th>CREATE</th>
<th>DROP</th>
<th>ALTER</th>
<th>TOTAL</th>
<th>LOCKING</th>
<th>TOTAL</th>
<th>DATA SHARING</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>0</td>
<td>LOCK TABLE</td>
<td>0</td>
<td>TABLE 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LOCK REQUEST 0</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSERT</td>
<td>0</td>
<td>GRANT</td>
<td>0</td>
<td>INDEX 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DEADLOCKS 0</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td>0</td>
<td>RESTORE</td>
<td>0</td>
<td>TABLESPACE 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>EXCL.(SHAR) 0</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>0</td>
<td>SIT SQILD</td>
<td>4370</td>
<td>DATABASE 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>EXCL.(EXCL) 0</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESCRL</td>
<td>0</td>
<td>SIT H.VAR.</td>
<td>0</td>
<td>STGROUP 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MAX.LCK HELD 3</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESC.TEL</td>
<td>0</td>
<td>SIT RULES</td>
<td>0</td>
<td>VIEW 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A LOCK REQUEST 134</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PREPARE</td>
<td>0</td>
<td>CONNECT 1</td>
<td>4371</td>
<td>ALIAS 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A UNLOCK REQUEST 79</td>
<td>SUSP - ILRM N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPEN</td>
<td>1</td>
<td>CONNECT 2</td>
<td>4371</td>
<td>PACKAGE N/A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A CHANGE REQUEST 121</td>
<td>SUSP - FALSE N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FETCH</td>
<td>4371</td>
<td>SIT CONNEC</td>
<td>0</td>
<td>OTHER REQUEST</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A INCOMP.LOCK 0</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOSE</td>
<td>0</td>
<td>RELEASE</td>
<td>0</td>
<td>TOTAL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LOCK SUSP. 0</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DML-ALL</td>
<td>8742</td>
<td>DCL-ALL</td>
<td>0</td>
<td>COMMENT ON 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LATCH SUSP. 0</td>
<td>N/P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**RID LIST**

<table>
<thead>
<tr>
<th>RID LIST</th>
<th>TOTAL</th>
<th>QUERY PARALLEL</th>
<th>TOTAL</th>
<th>STORED PROC.</th>
<th>TOTAL</th>
<th>DRAIN/CLAIM</th>
<th>TOTAL</th>
<th>DATA CAPTURE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>USED</td>
<td>1</td>
<td>MAXIMUM DEGREE</td>
<td>0</td>
<td>CALL STMTS 0</td>
<td>0</td>
<td>DRAIN CLAIM 0</td>
<td>IFI CALLS 0</td>
<td>N/P</td>
<td></td>
</tr>
<tr>
<td>FAIL-NO STORAGE</td>
<td>0</td>
<td>GROUPS EXECUTED</td>
<td>0</td>
<td>PROC. ABAND</td>
<td>0</td>
<td>DRAIN FAILED</td>
<td>REC.CAPTURED</td>
<td>N/P</td>
<td></td>
</tr>
<tr>
<td>FAIL-LIMIT EXEC.</td>
<td>0</td>
<td>GROUPS EXECUTED</td>
<td>0</td>
<td>PROC. ABAND</td>
<td>0</td>
<td>DRAIN FAILED</td>
<td>REC.CAPTURED</td>
<td>N/P</td>
<td></td>
</tr>
</tbody>
</table>

**BP0**

<table>
<thead>
<tr>
<th>BP0</th>
<th>TOTAL</th>
<th>EXPANSIONS</th>
<th>TOTAL</th>
<th>GETPAGES</th>
<th>TOTAL</th>
<th>BUFFER UPDATES</th>
<th>TOTAL</th>
<th>SYNONYMS WRITE</th>
<th>TOTAL</th>
<th>SYNCHRONOUS READ</th>
<th>TOTAL</th>
<th>SEQUENTIAL PREFETCH</th>
<th>TOTAL</th>
<th>LIST PREFETCH</th>
<th>TOTAL</th>
<th>DYNAMIC PREFETCH</th>
<th>TOTAL</th>
<th>PAGES READ ASYNCHR</th>
<th>TOTAL</th>
<th>HPOOL WRITES</th>
<th>TOTAL</th>
<th>HPOOL WRITES-FAILED</th>
<th>TOTAL</th>
<th>HPOOL READS</th>
<th>TOTAL</th>
<th>HPOOL READS FAILED</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPANSIONS</td>
<td>N/A</td>
<td>EXPANSIONS</td>
<td>N/A</td>
<td>EXPANSIONS</td>
<td>N/A</td>
<td>GETPAGES</td>
<td>231</td>
<td>GETPAGES</td>
<td>39555</td>
<td>BUFFER UPDATES</td>
<td>4480</td>
<td>BUFFER UPDATES</td>
<td>13110</td>
<td>SYNONYMS WRITE</td>
<td>0</td>
<td>SYNONYMS WRITE</td>
<td>0</td>
<td>SYNCHRONOUS READ</td>
<td>2</td>
<td>SYNCHRONOUS READ</td>
<td>1561</td>
<td>SEQUENTIAL PREFETCH</td>
<td>0</td>
<td>SEQUENTIAL PREFETCH</td>
<td>114</td>
<td>LIST PREFETCH</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 61. Scenario 3: Case C – Accounting Trace
The elapsed time is reduced by over 10% (72.73 seconds) and the CPU consumption is reduced by about 25% (21.90 seconds). This is basically due to the reduction in service task switch time (now 2.78 seconds) as there are only 66 commits. The fewer number of commits decreases the total CPU processing cost.

Numbers of lock, unlock, and change requests have dropped quite sharply.

The maximum number of locks held is three compared with one lock held in Case B. But the cursor WITH HOLD, means that the same page is locked across commits 66 / 55 = 1.2 times. This makes the maximum time a page is locked = 1.2 * (72.73 / 55) = 1.59 seconds, which is less than the maximum recommended locking time (5 seconds.), making Case C an acceptable solution.

The drop in the number of getpage requests in buffer pool BP0 helps to reduce the CPU consumption. The rest of the counters are pretty much the same as in Case B.

5.7 Case D

The maximum time any process can lock a page in the ORDER table (1.59 seconds) in Case C is not acceptable if the ORDER table is heavily accessed.

Assume, therefore, that a new program design is proposed and tested. The solution consists in avoiding the use of cursor WITH HOLD and the related S-locks held across commits. Instead, Case D uses a normal cursor and a repositioning logic after each commit to reopen the closed cursor.

Commit interval continues to be 1 second.

SQL statements in the program do not change, so that the access path is exactly the same as in Case B.

The DB2 PM Accounting trace for the highlights, timings, locking, and buffer pool information for the execution of this program is shown in Figure 62 on page 154.
### DB2 Performance Monitor (V4)

**Accounting Trace - Long**

**REQUESTED FROM:** Not Specified  
**TO:** Not Specified  
**ACTUAL FROM:** 05/03/96 17:27:28.37  
**SUBSYSTEM:** DB2F  
**GROUP:** N/P Accounting Trace - Long  
**LOCATION:** ST11DB2F

**Member:** N/P  
**To:** Not Specified  
**Subsystem:** DB2F  
**Actual From:** 05/03/96 17:27:28.37

---

<table>
<thead>
<tr>
<th>TIMES/EVENTS</th>
<th>APPL (CLASS 1)</th>
<th>DB2 (CLASS 2)</th>
<th>IFI (CLASS 5)</th>
<th>CLASS 3 SOSP.</th>
<th>ELAPSED TIME</th>
<th>EVENTS</th>
<th>HIGHLIGHTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time</td>
<td>1:48.896187</td>
<td>1:46.577989</td>
<td>N/P Lock/Latch</td>
<td>0.027297</td>
<td>19</td>
<td>THREAD TYPE : ALLIED</td>
<td></td>
</tr>
<tr>
<td>CPU Time</td>
<td>25.039094</td>
<td>24.172304</td>
<td>N/P Synchron. 1/O</td>
<td>30.900192</td>
<td>1548</td>
<td>TERM, CONDITION: NORMAL</td>
<td></td>
</tr>
<tr>
<td>TCB</td>
<td>25.039094</td>
<td>24.172304</td>
<td>N/P Other Read 1/O</td>
<td>0.048432</td>
<td>1</td>
<td>INVOKE REASON : DEALLOC</td>
<td></td>
</tr>
<tr>
<td>TCB-TPROCESS</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A Other Write 1/O</td>
<td>2.123792</td>
<td>38</td>
<td>COMMITS : 106</td>
<td></td>
</tr>
<tr>
<td>CPU-PAARAL.</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A Server, Task Switch</td>
<td>5.606537</td>
<td>115</td>
<td>ROLLBACK : 0</td>
<td></td>
</tr>
<tr>
<td>NOT ACCOUNT.</td>
<td>N/A</td>
<td>43.699454</td>
<td>N/P Arc.Log(Ques)</td>
<td>0.000000</td>
<td>0</td>
<td>INCREM, Binds : 0</td>
<td></td>
</tr>
<tr>
<td>DB2 ENT/EXIT</td>
<td>N/A</td>
<td>17907</td>
<td>N/A Arc.Log Read</td>
<td>0.000000</td>
<td>0</td>
<td>UPDATE/COMMIT : 41.23</td>
<td></td>
</tr>
<tr>
<td>BN/BK-STPROC</td>
<td>N/A</td>
<td>0</td>
<td>N/A Drain Lock</td>
<td>0.000000</td>
<td>0</td>
<td>PROGRAMS : 1</td>
<td></td>
</tr>
<tr>
<td>TCB</td>
<td>0</td>
<td>0</td>
<td>N/A Drain Lock</td>
<td>0.000000</td>
<td>0</td>
<td>PROGRAMS : 1</td>
<td></td>
</tr>
<tr>
<td>DCAPT, DESCR.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/P Page Latch</td>
<td>0.000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOG EXTRACT.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/P Stored Proc.</td>
<td>0.000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>END-EXEC</td>
<td>N/A</td>
<td>0</td>
<td>N/A Store Latch</td>
<td>0.000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2 ENT/EXIT</td>
<td>N/A</td>
<td>17907</td>
<td>N/A ROLLBACK : 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SQL DML**

<table>
<thead>
<tr>
<th>SQL DML</th>
<th>TOTAL</th>
<th>SQL DCL</th>
<th>TOTAL</th>
<th>SQL DDL</th>
<th>CREATE</th>
<th>DROP</th>
<th>ALTER</th>
<th>TOTAL</th>
<th>LOCKING</th>
<th>TOTAL</th>
<th>DATA SHARING</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>0</td>
<td>0</td>
<td>TABLE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LOCK REQUEST</td>
<td>N/P</td>
</tr>
<tr>
<td>INSERT</td>
<td>0</td>
<td>0</td>
<td>INDEX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>UNLOCK REQUEST</td>
<td>N/P</td>
</tr>
<tr>
<td>UPDATE</td>
<td>0</td>
<td>0</td>
<td>TABLESPACE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ESCAL, (SHARE)</td>
<td>N/P</td>
</tr>
<tr>
<td>DELETE</td>
<td>4370</td>
<td>0</td>
<td>DATABASE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ESCAL, (SHR)</td>
<td>N/P</td>
</tr>
<tr>
<td>DESCR.</td>
<td>0</td>
<td>0</td>
<td>SYNONYM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ESCAL, (SHARE)</td>
<td>N/P</td>
</tr>
<tr>
<td>DESC, TEL</td>
<td>0</td>
<td>0</td>
<td>VIEW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LOCK REQUEST : 489</td>
<td>N/P</td>
</tr>
<tr>
<td>PREPARE</td>
<td>0</td>
<td>0</td>
<td>ALIAS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LOCK - XES</td>
<td>N/P</td>
</tr>
<tr>
<td>OPEN</td>
<td>0</td>
<td>0</td>
<td>PACKAGE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>UNLOCK - XES</td>
<td>N/P</td>
</tr>
<tr>
<td>FETCH</td>
<td>4371</td>
<td>0</td>
<td>TABLE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>UNLOCK - XES</td>
<td>N/P</td>
</tr>
<tr>
<td>CLOSE</td>
<td>0</td>
<td>0</td>
<td>TOTAL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>NOTIFY SENT</td>
<td>N/P</td>
</tr>
<tr>
<td>DML-ALL</td>
<td>8847</td>
<td>0</td>
<td>DCL-ALL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LATCH SUSP.</td>
<td>0</td>
</tr>
</tbody>
</table>

**Rid List**

<table>
<thead>
<tr>
<th>RId List</th>
<th>TOTAL</th>
<th>QUERY PARALLEL</th>
<th>TOTAL</th>
<th>STORED PROC.</th>
<th>TOTAL</th>
<th>DATA SHARE</th>
<th>TOTAL</th>
<th>TOTAL</th>
<th>TOTAL</th>
<th>TOTAL</th>
<th>TOTAL</th>
<th>TOTAL</th>
<th>TOTAL</th>
<th>TOTAL</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>USED</td>
<td>106</td>
<td>MAXIMUM DEGREE</td>
<td>0</td>
<td>CALL STMTS</td>
<td>0</td>
<td>DRAIN CLAIM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FAL-FAL STORAGE</td>
<td>0</td>
<td>GROUPS EXECUTED</td>
<td>0</td>
<td>PROC. ABNDS</td>
<td>0</td>
<td>DRAIN FAILED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FAL-LIMIT EXEC.</td>
<td>0</td>
<td>GROUPS EXECUTED</td>
<td>0</td>
<td>PROC. ABNDS</td>
<td>0</td>
<td>DRAIN FAILED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RECALLS</td>
<td>0</td>
<td>MAXIMUM DEGREE</td>
<td>0</td>
<td>CALL STMTS</td>
<td>0</td>
<td>DRAIN CLAIM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SEQ - CURSOR</td>
<td>0</td>
<td>seq - cursor</td>
<td>0</td>
<td>seq - no buffer</td>
<td>0</td>
<td>seq - no buffer</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SEQ - CURSOR</td>
<td>0</td>
<td>seq - cursor</td>
<td>0</td>
<td>seq - no buffer</td>
<td>0</td>
<td>seq - no buffer</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PARALL. DISABLED</td>
<td>YES</td>
<td>PARALLEL DISABLED</td>
<td>0</td>
<td>seq - encl. ser.</td>
<td>0</td>
<td>seq - encl. ser.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**BD0**

<table>
<thead>
<tr>
<th>BD0</th>
<th>TOTAL</th>
<th>BD2</th>
<th>TOTAL</th>
<th>TOTAL</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPANSIONS</td>
<td>N/A</td>
<td>EXPANSIONS</td>
<td>N/A</td>
<td>EXPANSIONS</td>
<td>N/A</td>
</tr>
<tr>
<td>GETDATES</td>
<td>274</td>
<td>GETDATES</td>
<td>40463</td>
<td>GETDATES</td>
<td>40737</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>4480</td>
<td>BUFFER UPDATES</td>
<td>13110</td>
<td>BUFFER UPDATES</td>
<td>17590</td>
</tr>
<tr>
<td>SYNCHRONOUS WRITE</td>
<td>0</td>
<td>SYNCHRONOUS WRITE</td>
<td>0</td>
<td>SYNCHRONOUS WRITE</td>
<td>0</td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>2</td>
<td>SYNCHRONOUS READ</td>
<td>1544</td>
<td>SYNCHRONOUS READ</td>
<td>1546</td>
</tr>
<tr>
<td>SEQUENTIAL PREFETCH</td>
<td>0</td>
<td>SEQUENTIAL PREFETCH</td>
<td>0</td>
<td>SEQUENTIAL PREFETCH</td>
<td>0</td>
</tr>
<tr>
<td>LIST PREFETCH</td>
<td>199</td>
<td>LIST PREFETCH</td>
<td>0</td>
<td>LIST PREFETCH</td>
<td>0</td>
</tr>
<tr>
<td>DYNAMIC PREFETCH</td>
<td>0</td>
<td>DYNAMIC PREFETCH</td>
<td>77</td>
<td>DYNAMIC PREFETCH</td>
<td>77</td>
</tr>
<tr>
<td>PAGES READ ASYNCHR.</td>
<td>56</td>
<td>PAGES READ ASYNCHR.</td>
<td>31</td>
<td>PAGES READ ASYNCHR.</td>
<td>87</td>
</tr>
<tr>
<td>HPPOOL WRITES</td>
<td>0</td>
<td>HPPOOL WRITES</td>
<td>0</td>
<td>HPPOOL WRITES</td>
<td>0</td>
</tr>
<tr>
<td>HPPOOL WRITES-FAILED</td>
<td>0</td>
<td>HPPOOL WRITES-FAILED</td>
<td>0</td>
<td>HPPOOL WRITES-FAILED</td>
<td>0</td>
</tr>
<tr>
<td>HPPOOL READS</td>
<td>0</td>
<td>HPPOOL READS</td>
<td>0</td>
<td>HPPOOL READS</td>
<td>0</td>
</tr>
<tr>
<td>HPPOOL READS-FAILED</td>
<td>0</td>
<td>HPPOOL READS-FAILED</td>
<td>0</td>
<td>HPPOOL READS-FAILED</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Figure 62. Scenario 3: Case D – Accounting Trace**

---

**DB2 for MVS Locking**
The elapsed time is about 50% more than in Case C (108.90 seconds), and there is also an increase of about 10% in the CPU consumption (24.17 seconds). The larger number of commits (106) does not justify this increase.

The increase in CPU consumption is due to the increase in the number of open cursor calls (106). As this cursor is doing list prefetch, every time DB2 opens the cursor, it must additionally reposition to the first nondeleted key and sort the RIDS of the qualifying rows.

Service task switch time is also more than in Case C, as it is directly related to the number of COMMIT points in the program.

The maximum number of locks held is two compared with three locks held in Case C. The maximum time interval a page is locked is now 108.90 / 106 = 1.03 seconds, less than the 1.59 seconds in Case C. This may be an acceptable solution, even though it entails a 50% increase in elapsed time and about 10% more CPU consumption, than in Case C.

An interesting fact to be noticed here is that DB2 is using now dynamic prefetch to access the index pages (buffer pool BP2) instead of sequential prefetch.

5.8 Case E

The last case that is tested is the possibility of locking at a row level. This increases the concurrency. However, the price is increased CPU consumption and perhaps a higher risk of deadlocks because of the finer granularity of the locks.

The program is the one used in Case C, with the LOCKSIZE value of ORDER table changed to ROW.

There is no change in the access path because of LOCKSIZE ROW.

The DB2 PM Accounting trace for the highlights, timings, locking, and buffer pool information for the execution of this program is shown in Figure 63 on page 156.
### DB2 PERFORMANCE MONITOR (V4)

**LOCATION:** ST11DB2F
**GROUP:** N/P
**ACCOUNTING TRACK - LONG**
**MEMBER:** N/P
**SUBSYSTEM:** DB2F
**REQUESTED FROM:** NOT SPECIFIED
**TO:** NOT SPECIFIED
**ACTUAL FROM:** 03/05/96 22:25:01.53
**DB2 VERSION:** V4

**TIMES/EVENTS**

<table>
<thead>
<tr>
<th>APPL (CLASS 1)</th>
<th>DB2 (CLASS 2)</th>
<th>IFI (CLASS 5)</th>
<th>CLASS 3 SUSP.</th>
<th>ELAPSED TIME</th>
<th>EVENTS</th>
<th>HIGHLIGHTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELAPSED TIME</td>
<td>1:21.783862</td>
<td>1:19.692032</td>
<td>N/P LOCK/LATCH</td>
<td>0.034070</td>
<td>26</td>
<td>THREAD TYPE : ALLIED</td>
</tr>
<tr>
<td>CPU TIME</td>
<td>24.56312</td>
<td>23.706904</td>
<td>N/P SYNCHRON. 1/O</td>
<td>24.295778</td>
<td>1548</td>
<td>TERM. CONDITION: NORMAL</td>
</tr>
<tr>
<td>TCB</td>
<td>24.56312</td>
<td>23.706904</td>
<td>N/P OTHER READ 1/O</td>
<td>0.036613</td>
<td>1</td>
<td>INVOKE REASON : DEALLOC</td>
</tr>
<tr>
<td>TCB-TPROCC</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A OTHER WRITE 1/O</td>
<td>2.891589</td>
<td>63</td>
<td>COMMITS : 81</td>
</tr>
<tr>
<td>CPU-PARALL.</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A SER.TASK SWITCH</td>
<td>5.452173</td>
<td>90</td>
<td>ROLLBACK : 0</td>
</tr>
<tr>
<td>NOT ACCOUNT.</td>
<td>N/A</td>
<td>23.272904</td>
<td>N/P ARC.LOG (QUIES)</td>
<td>0.000000</td>
<td>0</td>
<td>INCREM.BINDS : 0</td>
</tr>
<tr>
<td>DB2 EN/T/EXIT</td>
<td>N/A</td>
<td>17647</td>
<td>N/P ARC.LOG READ</td>
<td>0.000000</td>
<td>0 UPDATE/COMMIT : 53.95</td>
<td></td>
</tr>
<tr>
<td>BN/BX-TPROCC</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A DRAIN LOCK</td>
<td>0.000000</td>
<td>0 PROGRAMS : 1</td>
<td></td>
</tr>
<tr>
<td>NOT ACCOUNT.</td>
<td>N/A</td>
<td>23.272904</td>
<td>N/P ARC.LOG READ</td>
<td>0.000000</td>
<td>0 UPDATE/COMMIT : 53.95</td>
<td></td>
</tr>
<tr>
<td>TCB</td>
<td>24.56312</td>
<td>23.706904</td>
<td>N/P OTHER READ 1/O</td>
<td>0.036613</td>
<td>1</td>
<td>INVOKE REASON : DEALLOC</td>
</tr>
<tr>
<td>TCB-TPROCC</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A OTHER WRITE 1/O</td>
<td>2.891589</td>
<td>63</td>
<td>COMMITS : 81</td>
</tr>
<tr>
<td>CPU-PARALL.</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A SER.TASK SWITCH</td>
<td>5.452173</td>
<td>90</td>
<td>ROLLBACK : 0</td>
</tr>
<tr>
<td>NOT ACCOUNT.</td>
<td>N/A</td>
<td>23.272904</td>
<td>N/P ARC.LOG (QUIES)</td>
<td>0.000000</td>
<td>0</td>
<td>INCREM.BINDS : 0</td>
</tr>
<tr>
<td>DB2 EN/T/EXIT</td>
<td>N/A</td>
<td>17647</td>
<td>N/P ARC.LOG READ</td>
<td>0.000000</td>
<td>0 UPDATE/COMMIT : 53.95</td>
<td></td>
</tr>
<tr>
<td>BN/BX-TPROCC</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A DRAIN LOCK</td>
<td>0.000000</td>
<td>0 PROGRAMS : 1</td>
<td></td>
</tr>
<tr>
<td>NOT ACCOUNT.</td>
<td>N/A</td>
<td>23.272904</td>
<td>N/P ARC.LOG READ</td>
<td>0.000000</td>
<td>0 UPDATE/COMMIT : 53.95</td>
<td></td>
</tr>
<tr>
<td>TCB</td>
<td>24.56312</td>
<td>23.706904</td>
<td>N/P OTHER READ 1/O</td>
<td>0.036613</td>
<td>1</td>
<td>INVOKE REASON : DEALLOC</td>
</tr>
<tr>
<td>TCB-TPROCC</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A OTHER WRITE 1/O</td>
<td>2.891589</td>
<td>63</td>
<td>COMMITS : 81</td>
</tr>
<tr>
<td>CPU-PARALL.</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/A SER.TASK SWITCH</td>
<td>5.452173</td>
<td>90</td>
<td>ROLLBACK : 0</td>
</tr>
<tr>
<td>NOT ACCOUNT.</td>
<td>N/A</td>
<td>23.272904</td>
<td>N/P ARC.LOG (QUIES)</td>
<td>0.000000</td>
<td>0</td>
<td>INCREM.BINDS : 0</td>
</tr>
<tr>
<td>DB2 EN/T/EXIT</td>
<td>N/A</td>
<td>17647</td>
<td>N/P ARC.LOG READ</td>
<td>0.000000</td>
<td>0 UPDATE/COMMIT : 53.95</td>
<td></td>
</tr>
<tr>
<td>BN/BX-TPROCC</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A DRAIN LOCK</td>
<td>0.000000</td>
<td>0 PROGRAMS : 1</td>
<td></td>
</tr>
</tbody>
</table>

**SQL DML**

<table>
<thead>
<tr>
<th>SELECT</th>
<th>INSERT</th>
<th>UPDATE</th>
<th>DELETE</th>
<th>DESCRIBE</th>
<th>DESC.IN</th>
<th>PREPARE</th>
<th>OPEN</th>
<th>FETCH</th>
<th>CLOSE</th>
<th>DML-ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4371</td>
<td>0</td>
<td>8742</td>
</tr>
</tbody>
</table>

**RDF LIST**

<table>
<thead>
<tr>
<th>USED</th>
<th>FAIL-NO STORAGE</th>
<th>FAIL-LIMIT EXC.</th>
<th>MAXIMUM DEGREE</th>
<th>SQL STMTS</th>
<th>DRAIN CLAIM</th>
<th>IPI CALLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**BP0**

<table>
<thead>
<tr>
<th>EXPANSIONS</th>
<th>TOTAL EXPANSIONS</th>
<th>TOTAL BP0</th>
<th>TOTAL TOTAL</th>
<th>TOTAL TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>4880</td>
<td>4880</td>
<td>249</td>
</tr>
</tbody>
</table>

**Figure 63. Scenario 3: Case E – Accounting Trace**
There is about a 13% increase in elapsed time over Case C (81.78 seconds compared with 72.73 seconds) that is related to an increase of the suspension times due to service task switch (15 additional COMMITs).

CPU times also increase a little (23.71 seconds, or about 10%) mainly because of the additional lock and change requests produced by row locking (4383 + 4373 = 8756 as against 134 + 121 = 255 in Case C). The 8501 additional IRLM calls cause an additional approximate CPU usage of 23.71 - 21.90 = 1.81, which means a cost for each lock and change operation of 21 microseconds.

The maximum number of row locks held is 118, which is equivalent to approximately 1.5 pages. The cursor WITH HOLD keeps the positioned row locked across commits, so the maximum time a row is locked is 1.48 seconds, which is less than the 1.59 seconds in Case C.

The values related to the buffer pool are similar to the ones in Case C, as we could expect, as the only difference is the level of locking.

### 5.9 Summary

We summarize the five different cases introduced, pointing out the pros and cons of the proposed solutions. We evaluate the solutions on the basis of these five significant factors:

- **Elapsed Time**, or the time the program needs to complete. This is a main performance indicator, and one to which users are highly sensitive.

- **CPU time**, or the CPU processor time consumption requirements. This is also a main performance indicator and is intimately linked to elapsed time.

- **Lock and Change requests**. This factor shows whether or not the program suffers many contentions in a concurrent environment.

- **Maximum number of locks held**. If this value is high, the program may cause suspensions to other concurrent programs that need to access the pages.

- **Maximum time a lock is held**. If this value is high, we can expect an increase in the chain factor affecting concurrent programs and an increase in the number of timeouts in the system.

The results of these five tests are summarized in Table 20 on page 158.
Table 20. Summary of Measurements

<table>
<thead>
<tr>
<th>Case</th>
<th>Elapsed Time (Seconds)</th>
<th>CPU Time (Seconds)</th>
<th>Lock and Change Requests</th>
<th>Maximum Number of Locks Held</th>
<th>Maximum Time a Lock is Held</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A</td>
<td>44.21</td>
<td>18.71</td>
<td>77</td>
<td>56</td>
<td>44.14</td>
</tr>
<tr>
<td>Case B</td>
<td>83.04</td>
<td>82.34</td>
<td>13143</td>
<td>1</td>
<td>1.51</td>
</tr>
<tr>
<td>Case C</td>
<td>72.73</td>
<td>21.90</td>
<td>334</td>
<td>3</td>
<td>1.59</td>
</tr>
<tr>
<td>Case D</td>
<td>108.90</td>
<td>24.17</td>
<td>765</td>
<td>2</td>
<td>1.03</td>
</tr>
<tr>
<td>Case E</td>
<td>81.78</td>
<td>23.71</td>
<td>8846 (a)</td>
<td>118 (a)</td>
<td>1.48</td>
</tr>
</tbody>
</table>

Notes:

All magnitudes are for locks in pages unless otherwise specified.
(a) These lock magnitudes are in rows rather than pages

Case A is the simplest from a programming point of view, with fewer SQL calls, and has the lowest elapsed and CPU times. However, it can be considered only if it is to be executed in a dedicated window. Also, a large number of locks are held for a long time (up to 44.14 seconds), resulting in concurrency problems.

Case B offers much better concurrency, but the excessive write log requirements result in a high elapsed time, making it unsatisfactory as an alternative solution.

Case C is a possible choice, as the elapsed and CPU times are reasonable, the time the locks are held is small, and extra repositioning logic, which always complicates the program logic, is not required.

Case D is the best choice for concurrency, but at the cost of an increase in the CPU consumption. This alternative is better than Case C for installations where concurrency is more critical than CPU consumption by the programs.

Case E can be a good choice, but row locking can considerably increase the chances of deadlocks in the environment.
Chapter 6. Data Sharing Considerations

DB2 implements data sharing using the shared data architecture, where each system is a symmetrical multiprocessor (SMP) with its own local memory and the disk is shared by all the systems using an enterprise systems connection (ESCON) link.

Therefore, each DB2 system has a pool of local memory and shares all or some databases on shared DASD. Each DB2 system can also have some local disks to contain databases that are not shared.

Figure 64 presents an overview of DB2 data sharing.

For data consistency in a data sharing environment, locks must be known and respected among all members. DB2 data sharing uses global locks to ensure that members are aware of each others’ locks. A global lock is one that guarantees serialization across the data sharing group. In contrast, a local lock guarantees serialization for one member only.
In data sharing, all locks on database resources are global locks. Some global locks are propagated from the local IRLM through the cross-system extended services (XES) to the coupling facility (CF), and some global locks do not need to be propagated.

DB2 data sharing uses two types of global locks: logical locks and physical locks.

### 6.1 Logical Locks

DB2 uses transaction locks for data consistency. With DB2 data sharing, these locks are also called logical locks, or L-locks and correspond to the lock concept in a non-data-sharing environment.

Logical locks are used for intertransaction concurrency controls within a data sharing group. Logical locks are owned by a transaction; the lock duration can be commit, plan deallocation, or manual, and can have deadlocks and timeouts.

The logical locks are of four types:

- Table space locks
- Table locks
- Page locks
- Row locks.

#### 6.1.1 Table Space Logical Locks

DB2 uses table space locks to verify access compatibility between different processes within a DB2 subsystem. For example, if one transaction has a table space intent share (IS) lock, a batch program requiring an exclusive (X) lock on the same table space is not allowed. In a data sharing environment, these locks need to be known by all members; they must be registered in the lock structure of the CF.

For a partitioned table space, accessing one partition causes logical locks to be acquired on the last partition of the target table space.

DB2 data sharing introduces a concept called *explicit hierarchical locking* to optimize performance by reducing the number of logical locks to be propagated. Hierarchical locking relates parent locks (table space logical locks or partition logical locks) to child locks (table, page, or row logical locks). Therefore IRLM can decide which locks to propagate to the lock structure in the CF.

### 6.2 Physical Locks

DB2 data sharing introduces physical locks, or P-locks. Some P-locks are used to ensure inter-DB2 data coherency, others are used to ensure physical data consistency. Physical locks are owned by a DB2 member and their duration is determined by the interest of the member.

Physical locks do not have deadlocks or timeouts.

There are two types of P-locks: page-set physical locks are used to track inter-DB2 read-write interest at the page set or partition level, while page physical locks are used to provide write serialization between members, ensuring the physical consistency of a page just as latches do in a non-data-sharing environment.
6.2.1 Page-Set Physical Locks

Page-set physical locks are used to track inter-DB2 read-write interest or different members for a page set or partition. Each member acquires one (and only one) page-set physical lock for each open table space or index, or for any partition of a partitioned table space or partitioned index. Because the physical locks are owned by the member, they are independent of the transactions.

Page-set physical locks are always propagated to the lock structure of the CF at member level.

The inter-DB2 read-write interest of a page-set physical lock for a given page set by a given member is determined by the following activities on the page set:

- Physical open, read-only interest
- Physical close, no interest
- First update or pseudo-open, read-write interest
- Read-only switching or pseudo-close, read-only interest.

6.2.2 Page Physical Locks

Page physical locks are used to ensure physical consistency of a page across the members of a group. They are used in these three cases:

- The page set is shared.
- There is a need for subpage concurrency (in Type 1 indexes).
- No page logical locks protect the page.

The page physical lock protects the page while its structure is being changed, ensuring that a sharing member does not read it.

There are two types of page physical locks: X-locks and S-locks.

X-locks are requested in cases such as the following:

- Update of a shared data page, while row locking is in effect
- Update of a shared space map page
- Update of a shared Type 2 index page
- Update of certain DB2 directory pages.

S-type page physical locks are requested in only three cases:

- Repeatable read index scan with LOCKSIZE ROW for data pages and Type 2 index leaf pages.
- Referential integrity (RI) checking.
- Space map pages (occasional use).

6.2.3 Lock Negotiation

Data sharing introduces the concept of negotiable physical locks. Such locks can change state (mode) based upon changes in the inter-DB2 read-write interest on a particular resource. For example, if a physical lock is held by a member in exclusive mode (X-lock), and another member wants the physical lock in share mode (S-lock), the member holding the physical X-lock may choose to downgrade it to share mode.
thus permitting the other member access to the data. Physical lock negotiation is done by DB2 exits, driven by the IRLM.

Figure 65 shows an example of physical lock (P-lock) negotiation.

In Figure 65, the following symbols are used:
- **P(x,y)**: Request for a physical lock in X-mode by Member y
- **TS T**: Table space T
- **GM**: Group manager
- **GBP**: Group buffer pool (GBP) dependent
- **DB2y**: Subsystem ID for Member y
- **IRLMy**: IRLM ID for Member y
- **SLM**: System lock manager
- **CF**: Coupling facility

Figure 65 shows the following:
- Initially, both DB21 and DB22 hold P-lock in S-mode as they are only reading data. Member DB22 now wants to update the data.
- IRLM2 cannot grant the physical lock in SIX-mode for TS T, as there is already a physical lock in S-mode held by DB21 (P(S,1)) in the CF. The request from IRLM2 is rejected by the CF, notifying IRLM2 that P(S,1) exists. The CF is not involved for the remainder of this negotiation.
- IRLM2 then uses the cross coupling facility (XCF) to communicate with IRLM1 and request P(SIX,2).
- IRLM1 drives the contention exit in DB21 and finally grants the P(SIX,2) by making table space T GBP-dependent.
- IRLM2 becomes group manager (GM) for TS T.
- IRLM2 informs DB22 about the inter-DB2 read-write interest.
- DB22 makes table space T GBP-dependent and forthwith registers all updated pages in the CF and writes copies of the updated pages to the CF.
6.3 Contention in a Data Sharing Environment

The lock structure in the coupling facility has two components:

1. A lock table, that is used to detect contentions.
2. A modified resource list that saves information on modified locks, which can be either logical or physical locks.

Contention occurs when different members need to use the same resource at the same time. There are three types of contention in a data sharing environment:

- **Real contention**: occurs when lock requests are incompatible, for instance one program holds a logical S-lock on a page while another program requests an X-lock on the same page.
- **False contentions**: occurs when the hashing algorithm provides the same hash value for two different resources.
- **XES contentions** are contentions that arise because XES is aware of only two lock modes, share and exclusive. IRLM locking supports many additional lock modes (S, X, U, IX ... ). When the MVS XES component detects a contention because of incompatible lock modes for the same resource, that contention is not necessarily a real contention by IRLM standards.

When a contention occurs, execution of the requester’s SQL statement is suspended until the contention is resolved. If the contention is real, the requester remains suspended until the incompatible lock is released or the program abends because of timeout or deadlock.

DB2 PM Accounting and DB2 PM Statistics reports show locking information in a data sharing environment.

The DB2 PM Accounting report provides information about the number of lock requests for physical locks, the number of lock requests propagated to MVS XES, and the number of suspensions because of IRLM global resource contentions and MVS XES global resource contentions.

The DB2 PM Statistics report provides information about the global number of lock requests for physical locks, number of lock requests propagated to MVS XES synchronously and asynchronously, and the total number of suspensions because of IRLM global resource contentions.

For more details, refer to *Data Sharing: Planning and Administration* (SC26-3269-01) and *DB2 for MVS/ESA Version 4 Data Sharing Performance Topics* (SG24-4611-00).

**Note**

The ITSO redbook on DB2 for MVS/ESA Version 4 data sharing implementation (SG24-4791), scheduled to be released before the end of 1996, contains information about locking in a data sharing environment.
6.4 Claims and Drains

A drain lock is a real IRLM lock at a page-set level. In a data sharing environment, a drain lock is a global logical lock on the page set. Global logical locks always propagate to the CF, even if the page sets are not dependent on the GBP.

Claimers must request a shared logical lock on the page set to wait for the drain to finish. Claim counters, once a claim is acquired by one of the members of the data sharing group, are maintained locally in each member. No additional shared locks need be acquired for the sharing DB2 members.

6.5 Lock Avoidance

Lock avoidance is important primarily for concurrency, but it is also used for CPU reduction in a data sharing environment. In terms of the CPU, the main saving occurs because IRLM does not propagate the lock request to MVS XES and to the CF.

Lock avoidance in a non-data-sharing environment (all data is local) depends on the lowest uncommitted log record sequence number of the single DB2 for a page set or partition. In a data sharing environment, each DB2 member still maintains the CLSN for every page set or partition. However, these CLSNs are used only for page sets or partitions that are not shared. For shared page sets or partitions, each member uses a global CLSN (GCLSN) value. Thus, for lock avoidance to be effective, all applications in the data sharing group must commit frequently.

Lock avoidance uses the same techniques in a data sharing environment as in a non-data-sharing environment. The only new concept added in this environment is the GCLSN.

For nonshared page sets or partitions, each member uses a local CLSN (kept at page set or partition level), which is the lowest uncommitted log sequence number for all active transactions against that page set or partition.

In a data sharing environment, members cannot rely on the local CLSN because other members may have transactions with a lower CLSN for these table spaces. For shared page sets, a single GCLSN value is maintained and used for lock avoidance for the entire data sharing group. The single GCLSN value is derived from each member’s CLSN values. The member’s CLSN value is maintained at the DB2 subsystem level, not at the page set or partition level.

The GCLSN is the earliest CLSN value across all members across all page sets or partitions, regardless of whether the page sets or partitions are shared or not.

DB2 data sharing needs to interchange the GCLSN between members in order to apply lock avoidance. Periodically, each member obtains the GCLSN from the shared communication area (SCA).

One long-running update process without intermediate commit points can effectively stop the GCLSN value from moving forward, regardless of whether the process is updating shared or nonshared page sets. This has a negative impact on lock-avoidance mechanisms for the data sharing group.

The following actions are recommended to maximize lock avoidance in a data sharing environment:
6.6 Application Design Considerations

DB2 data sharing provides more flexible ways to configure your environment, increasing the global availability of the database system and lowering the cost for performance because of the use of inexpensive complementary metal oxide semiconductor (CMOS) processors.

DB2 data sharing implementation does not require any changes to your existing application programs; it is completely transparent to programs.

Now that you have a certain number of DB2 subsystems accessing the same data, logical locks must be propagated to the CF structures to verify access compatibility between different processes in different DB2 subsystems. That basically means additional locks and logic, which may affect the performance of your existing applications.

Also, the propagation of pages to the group buffer pool in case of inter-DB2 read-write interest increases the CPU cost of the application programs, independently of the lock processing cost.

Additionally, in a data sharing system, DB2 uses physical locks (P-locks) to ensure inter-DB2 data coherency, which is an overhead for the application programs.

The propagation of these locks to the CF lock structure is more expensive than a lock in a non-data-sharing environment. The cost rises even higher if there is lock negotiation between the members (an order of magnitude more expensive than a local lock).

The obvious conclusion is that you need to try to minimize the number of lock requests in a data sharing environment. Two basic recommendations to follow when planning a data sharing environment are:

- Make sure your applications use **lock avoidance** when the application program requirements allow it:
  - Use CURRENTDATA(NO) when data currency is not needed.
  - Use uncommitted read isolation if possible.
  - Avoid using RS or RR isolation when not needed.

- Try to avoid the use of **row locking** in these environments, as the total number of locks may increase, raising the CPU consumption, especially if there is a lot of contention and negotiation, as:
  - Contention on data pages when inserting an ever-increasing ascending key
  - Contention on small tables.
Appendix A. Special Notices

This publication is intended to help technical professionals understand how locking affects the performance of DB2 applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by DB2 for MVS/ESA Version 4. See the PUBLICATIONS section of the IBM Programming Announcement for the DB2 product for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM’s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM’s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

- CICS
- DB2
- DFSMS/MVS
- DFSORT
- IBM
- OS/2
- RMF
- Sysplex Timer
- VTAM
- DATABASE 2
- DFSMS
- DFSMSdss
- DRDA
- MVS/ESA
- RACF
- S/390
- System/390
- 3090

The following terms are trademarks of other companies:

- C-bus is a trademark of Corollary, Inc.
- PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.
UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Windows is a trademark of Microsoft Corporation.

Other trademarks are trademarks of their respective companies.
Appendix B. Related Publications

- IBM DATABASE 2 for MVS/ESA Version 4 Data Sharing: Planning and Administration, SC26-3269-01
- IBM DATABASE 2 for MVS/ESA Version 4 Command Reference, SC26-3267
- IBM DATABASE 2 for MVS/ESA Version 4 Utility Guide and Reference, SC26-3395
- IBM DATABASE 2 for MVS/ESA Version 4 SQL Reference, SC26-3270
- IBM DATABASE 2 for MVS/ESA Version 4 Messages and Codes, SC26-3268

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 171.

- DB2 for MVS/ESA Version 4 Data Sharing Performance Topics, SG24-4611
- DB2 for MVS/ESA Version 4 Non-Data-Sharing Performance Topics, SG24-4562
- Distributed Relational Database: Cross Platform Connectivity and Application, SG24-4311-01
- DB2 Quick Upper Bound Estimate: An Application Design Methodology, SG24-2549
- DB2 PM Usage Guide Update, SG24-2584
- DB2 V3 Performance Topics, GG24-4284
- DB2 V2.3 Nondistributed Performance Topics, GG24-3823
- Capacity Planning for DB2 Applications, GG24-3512
- DB2 V2.2 Design Guidelines for High Performance, GG24-3383
- DB2 Referential Integrity Usage Guide, GG24-3312
- System/390 MVS Parallel Sysplex Performance, SG24-4356-01

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.
B.2 Other Publications

These publications are also relevant as further information sources:

- IBM DATABASE 2 Performance Monitor for MVS (DB2 PM) Version 4 Report Reference, Volume 1, SH12-6163
How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com/redbooks.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**
  To get LIST3820s of redbooks, type one of the following commands:
  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```
  To get lists of redbooks:
  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTOOLS MKTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTOOLS MKTOOLS GET LISTSERV PACKAGE
  ```
  To register for information on workshops, residencies, and redbooks:
  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```
  For a list of product area specialists in the ITSO:
  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```
- **Redbooks Home Page on the World Wide Web**
- **IBM Direct Publications Catalog on the World Wide Web**
  IBM employees may obtain LIST3820s of redbooks from this page.
- **ITSO4USA category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**
  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.
How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:
  
<table>
<thead>
<tr>
<th>Country</th>
<th>IBMMAIL</th>
<th>Internet</th>
</tr>
</thead>
<tbody>
<tr>
<td>In United States:</td>
<td>usib66pl at ibmmail</td>
<td><a href="mailto:usib66pl@ibmmail.com">usib66pl@ibmmail.com</a></td>
</tr>
<tr>
<td>In Canada:</td>
<td>calibmzk at ibmmail</td>
<td><a href="mailto:lmannix@vnet.ibm.com">lmannix@vnet.ibm.com</a></td>
</tr>
<tr>
<td>Outside North America:</td>
<td>bookshop at dkibmsh at ibmmail</td>
<td><a href="mailto:bookshop@dk.ibm.com">bookshop@dk.ibm.com</a></td>
</tr>
</tbody>
</table>

- **Telephone orders**
  
  United States (toll free) | 1-800-879-2755
  Canada (toll free)        | 1-800-IBM-4YOU
  Outside North America:    | (long distance charges apply)
  (+45) 4810-1320 - Danish  | (+45) 4810-1020 - German
  (+45) 4810-1420 - Dutch   | (+45) 4810-1620 - Italian
  (+45) 4810-1540 - English | (+45) 4810-1270 - Norwegian
  (+45) 4810-1670 - Finnish | (+45) 4810-1120 - Spanish
  (+45) 4810-1220 - French  | (+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

  IBM Publications
  Publications Customer Support
  P.O. Box 29554
  Raleigh, NC 27626-0570
  USA
  IBM Direct Services
  144-4th Avenue, S.W.
  Calgary, Alberta T2P 3N5
  Canada
  DK-3450 AllerT3d
  Denmark

- **Fax** — send orders to:

  United States (toll free) | 1-800-445-9269
  Canada (toll free)        | 1-800-267-4455
  Outside North America:    | (+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:

  - Index # 4421 Abstracts of new redbooks
  - Index # 4422 IBM redbooks
  - Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com

- **On the World Wide Web**

  - Redbooks Home Page http://www.redbooks.ibm.com/redbooks

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).
IBM Redbook Order Form

Please send me the following:

<table>
<thead>
<tr>
<th>Title</th>
<th>Order Number</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Please put me on the mailing list for updated versions of the IBM Redbook Catalog.

First name                             Last name

Company

Address

City                                      Postal code          Country

Telephone number                        Telefax number       VAT number

- Invoice to customer number

- Credit card number

Credit card expiration date                          Card issued to     Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AET</td>
<td>average elapsed time</td>
</tr>
<tr>
<td>APA</td>
<td>all points addressable</td>
</tr>
<tr>
<td>AR</td>
<td>application requester</td>
</tr>
<tr>
<td>AS</td>
<td>application server</td>
</tr>
<tr>
<td>CAF</td>
<td>call attachment facility</td>
</tr>
<tr>
<td>CC</td>
<td>claim class</td>
</tr>
<tr>
<td>CD</td>
<td>claim descriptor</td>
</tr>
<tr>
<td>CICS</td>
<td>customer information control system</td>
</tr>
<tr>
<td>CLSN</td>
<td>commit log sequence number</td>
</tr>
<tr>
<td>CP</td>
<td>central processor</td>
</tr>
<tr>
<td>CPU</td>
<td>central processing unit</td>
</tr>
<tr>
<td>CS</td>
<td>cursor stability</td>
</tr>
<tr>
<td>CSA</td>
<td>common storage area</td>
</tr>
<tr>
<td>DASD</td>
<td>direct access storage device</td>
</tr>
<tr>
<td>DB2 PM</td>
<td>DB2 performance monitor</td>
</tr>
<tr>
<td>DBD</td>
<td>database descriptor</td>
</tr>
<tr>
<td>DBMS</td>
<td>database management system</td>
</tr>
<tr>
<td>DBRM</td>
<td>database request module</td>
</tr>
<tr>
<td>DCL</td>
<td>data control language</td>
</tr>
<tr>
<td>DDL</td>
<td>data definition language</td>
</tr>
<tr>
<td>DML</td>
<td>data manipulation language</td>
</tr>
<tr>
<td>ECSA</td>
<td>extended common storage area</td>
</tr>
<tr>
<td>EDM</td>
<td>environment descriptor management</td>
</tr>
<tr>
<td>EOF</td>
<td>end of file</td>
</tr>
<tr>
<td>GCLSN</td>
<td>global commit log sequence number</td>
</tr>
<tr>
<td>GTF</td>
<td>generalized trace facility</td>
</tr>
<tr>
<td>I/O</td>
<td>input/output</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines Corporation</td>
</tr>
<tr>
<td>ID</td>
<td>identification</td>
</tr>
<tr>
<td>IFCID</td>
<td>instrumentation facility component identifier</td>
</tr>
<tr>
<td>IFI</td>
<td>instrumentation facility interface</td>
</tr>
<tr>
<td>IFP</td>
<td>IMS/VS fast path</td>
</tr>
<tr>
<td>IMS</td>
<td>Information management system</td>
</tr>
<tr>
<td>IRLM</td>
<td>internal resource lock manager</td>
</tr>
<tr>
<td>IS</td>
<td>intent share</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical Support Organization</td>
</tr>
<tr>
<td>IX</td>
<td>intent exclusive</td>
</tr>
<tr>
<td>LPL</td>
<td>logical page list</td>
</tr>
<tr>
<td>LUW</td>
<td>logical unit of work</td>
</tr>
<tr>
<td>MVS</td>
<td>multiple virtual storage</td>
</tr>
<tr>
<td>OBID</td>
<td>object identifier</td>
</tr>
<tr>
<td>OPX</td>
<td>online performance monitor destination</td>
</tr>
<tr>
<td>PROFS</td>
<td>Professional Office System</td>
</tr>
<tr>
<td>PUNC</td>
<td>possibly uncommitted</td>
</tr>
<tr>
<td>QMF</td>
<td>query management facility</td>
</tr>
<tr>
<td>RBA</td>
<td>relative byte address</td>
</tr>
<tr>
<td>RID</td>
<td>record (row) identifier</td>
</tr>
<tr>
<td>RMF</td>
<td>resource measurement facility</td>
</tr>
<tr>
<td>RO</td>
<td>read-only</td>
</tr>
<tr>
<td>RR</td>
<td>repeatable read</td>
</tr>
<tr>
<td>RS</td>
<td>read stability</td>
</tr>
<tr>
<td>RW</td>
<td>read and write</td>
</tr>
<tr>
<td>S</td>
<td>share</td>
</tr>
<tr>
<td>SIX</td>
<td>share with intent exclusive</td>
</tr>
<tr>
<td>SKCT</td>
<td>skeleton cursor table</td>
</tr>
<tr>
<td>SKPT</td>
<td>skeleton package table</td>
</tr>
<tr>
<td>SMF</td>
<td>system management facility</td>
</tr>
<tr>
<td>SQL</td>
<td>structured query language</td>
</tr>
<tr>
<td>TS</td>
<td>table space</td>
</tr>
<tr>
<td>TSO</td>
<td>time-sharing option</td>
</tr>
<tr>
<td>U</td>
<td>update</td>
</tr>
<tr>
<td>UR</td>
<td>unit of recovery</td>
</tr>
<tr>
<td>UT</td>
<td>utility</td>
</tr>
<tr>
<td>UW</td>
<td>unit of work</td>
</tr>
<tr>
<td>VSAM</td>
<td>virtual storage access method</td>
</tr>
<tr>
<td>VTAM</td>
<td>virtual telecommunications access method</td>
</tr>
<tr>
<td>WFI</td>
<td>wait for input</td>
</tr>
<tr>
<td>WFI</td>
<td>exclusive</td>
</tr>
</tbody>
</table>
Index

A
ACQUIRE
option of BIND PLAN subcommand locking tables and table spaces 13, 67
ALTER TABLESPACE statement
LOCKMAX option 20
LOCKSIZE option 12
application
design
application access strategy 84
distributed application considerations 84
hot pages 82
read-only considerations 74
SQL design 77
unit of recovery 73
unit of work 73
problems
DB2 PM Accounting report 118
DISPLAY DATABASE LOCKS command 114
Explain statement 116
application system
See AS
AS 12

B
bind option
CURRENTDATA 27
ISOLATION 27
bind option considerations
ACQUIRE 67
CURRENTDATA 72
ISOLATION 69
RELEASE 67

C
CAF 1, 7, 133
CICS 1, 6, 30, 74, 113, 134
claim class
CS 43
RR 43
write 43
claims 42
claims and drains
compatibility rules 44
interaction 43
CLSN 22
commit frequency
CPU time and maximum pages locked analysis 141
elapsed and CPU time analysis 140
when to commit 142
commit log sequence number
See CLSN
concurrency
coeexistence of transactions and utilities 44
identifying problems
DB2 PM Statistics report 124
DISPLAY DATABASE USE command 123
DISPLAY THREAD command 121
LOCK TABLE statement 12
LOCKSIZE option 12
type 2 index 54
cooling hot pages and hot rows 97
coolers for the hot pages 101
Swiss solution 98
UK solution 99
CREATE TABLESPACE statement
LOCKMAX option 20
LOCKSIZE option 12
CS claim class 43
CS drain class 44
CS isolation 71
CURRENTDATA
block fetch enablement 27
lock avoidance 27
parallelism 27
cursor stability
See CS isolation 71

data sharing 159
application design considerations 165
claims 164
contention 163
drains 164
GCLSN 164
lock avoidance 164
logical locks 160
physical locks 160
data sharing contention
false 163
real 163
XES 163
database descriptor
See DBD
database design
index considerations 66
table considerations 66
table space considerations 64
DB2 command
DISPLAY DATABASE CLAIMERS 128
DISPLAY DATABASE LOCKS 113, 128
DISPLAY DATABASE USE 121
DISPLAY THREAD 121
START DATABASE 5
DB2 PM Accounting report 118, 163
DB2 PM Lock suspension report 130
DB2 PM Lock suspension trace 132
DB2 PM Lockout trace 134
DB2 PM Statistics report 126, 163
DB2 trace records
  accounting 111
  performance 112
  statistics 111
DBD 41
deadlock 21, 133
  DB2 PM Lockout trace 134
DELETE scenario
  case A 147
  case B 149
  case C 151
  case D 153
  case E 155
  summary 157
DISPLAY DATABASE CLAIMERS command 129
DISPLAY DATABASE LOCKS command 114, 128
DISPLAY DATABASE USE command 123
DISPLAY THREAD command 121
distributed application considerations
  prefetch 86
  reducing network traffic 85
  stored procedures 86
drain class
  CS 44
  RR 44
  write 44
drains 42, 43
duration of lock 12

E
end-of-file lock 58
exclusive
  See X-lock mode

G
GCLSN 164
global CLSN
  See GCLSN
global problems
  DB2 PM I/O activity report 127
  DB2 PM Statistics report 126
  RMF reports 127
gross mode
  S 17
  U 17
  X 18

H
hybrid state 18

I
IMS 1, 7, 14, 33, 74, 113, 133
  intent exclusive
    See IX-lock mode
  intent mode
    IS 17
    IX 17
  intent share
    See IS-lock mode
IRLM 28
  lock processing flow 32
  locking implementation 34
IRLM and DSNZPARM options 29
  cross-address space program call 31
  deadlock cycle 32
  deadlock time 29
  IRLM resource wait timeout 29
  NUMLKTS 31
  NUMLKUS 31
  U lock for RR/RS isolation 30
  utility timeout 30
IS-lock mode 17
ISOLATION 27
  considerations 71
  CS 71
  RR 69
  RS 69
  uncommitted read 71
IX-lock mode 17

L
latches 50
lock avoidance 22
  CLSN 22
  control 26
  flow process 25
  PUNC 23
  recommendations 164
lock duration 12
  page lock 14
  row lock 14
  table lock 13
  table space lock 13
lock escalation 19
lock hierarchy 11
lock management 1
  how locking works 1
  reasons for locking 2
  unit of recovery 6
  unit of work 6
lock mode
  compatibility rules 18
  page lock 18
  row lock 18
  table lock 17
  table space lock 17
lock negotiation 161
lock processing flow 32
lock size ANY 12
lock size PAGE 12
lock size TABLE 12
lock size TABLESPACE 12
lock suspension 21
lock suspension analysis:
  DB2 PM Lock suspension report 130
  DB2 PM Lock suspension trace 132
  DISPLAY DATABASE CLAIMERS command 129
  DISPLAY DATABASE LOCKS command 128
LOCK TABLE statement 12
locking control options 21
locking problems:
  application problems 113
  concurrency problems 121
  console messages 111
  deadlock 133
  global problems 126
  identifying 109
  periodic monitoring 111
  preventing 63
  recommendations 136
  suspensions 128
  timeout 132
  user warning 109
locking scenario:
  commit frequency test 139
  CPU cost of row locking 144
  DELETE 146
LOCKMAX 19

M
mass-delete lock 60
maximum locks per table space
  See NUMLKTS
maximum locks per user
  See NUMLKUS
mode of lock 17
  See lock mode

N
NUMLKTS 31
NUMLKUS 31

P
page lock
  read and write 16
  read-only 14
page lock mode
  S 18
  U 18
  X 18
page physical locks 161
page-set physical locks 161
partition independence 48
periodic monitoring 111
  DB2 commands 111
  DB2 Explain 111
  DB2 PM Exception reports 111
  DB2 trace records 111
  user reports 112
phantom problem 57
physical locks
  lock negotiation 161
  page physical locks 161
  page-set physical locks 161
possibly uncommitted
  See PUNC
prevent locking problems:
  application design 72
  bind option considerations 67
  database design 63
  programming considerations 72
  quality card analysis 87
  recommendations 107
PUNC 23

Q
QMF 12
quality card analysis 87
  background 87
  cooling hot pages and hot rows 97
  INSERT 88
  quality card for NEXTORDER table 92
  quality card for ORDER table 91
  quality card for type 1 index 106
  simplified P(PL) estimate 88
  using quality card 90
query management facility
  See QMF

R
read stability
  See RS isolation 69
read-only
  lock avoidance 76
  temporary result tables 76
  uncommitted read considerations 75
RELEASE
  option of BIND PLAN subcommand
    combining with other options 13, 67
repeatable read
  See RR isolation 69
resource lock manager
  See IRLM
restricted state
  illustration 39
  level 1 37
  level 2 37
row lock
  read and write 16
  read-only 14
row lock mode
  S 18
  U 18
  X 18
RR claim class 43
RR drain class 44
RR isolation 69
RS isolation 69

S
S-lock mode 17, 18
serialization mechanisms 9
  claims and drains 42
  latches 50
  partition independence 48
  restricted states 36
  SQL, DB2 utilities, and DB2 commands 9
  subsystem object locking 39
  transaction locking 9
  type 1 and type 2 index considerations 51
share
  See S-lock mode
share with intent exclusive
  See SIX-lock mode
SIX-lock mode 18
SKCT 41
skeleton cursor table
  See SKCT
skeleton package table
  See SKPT
SKPT 41
SQL design
  Australian way 79
  normal solution 79
  surprising results 77
  Swiss solution 83
  UK solution 81
SQLCODE
  -911 33, 133, 134
  -913 33, 133, 134
stored procedures 86
  DSNZPARM parameters 86
subsystem object locking
  catalog and directory 40
  DBD 41
  SKCT 41
  SKPT 41
sysplex

T
table lock mode (continued)
  SIX 18
  U 17
  X 18
table space considerations
  data compression 65
  free space 65
  lock size 64
  maximum number of page or row locks 64
  page size 64
  partitioned 66
  segmented 65
  simple 65
  type of table space 65
table space lock mode
  IS 17
  IX 17
  S 17
  SIX 18
  U 17
  X 18
table space logical locks 160
timeout 132
  DB2 PM Lockout trace 134
transaction locking
  deadlock 21
  duration of lock 12
  IRLM 28
  lock avoidance 22
  lock duration 12
  lock escalation 19
  lock hierarchy 11
  lock suspension 21
  locking control options 21
  two-phase commit 35
TSO 7, 74, 111, 136
  online 30
  TSO batch 7, 30
two-phase commit 35
type 2 index
  advantages 51
  concurrent structure modification 52
  DASD allocation 52
  logical partition independence 53
  row locking 51
  uncommitted read 51

U
U-lock mode 17, 18
uncommitted read 71, 75
unit of recovery 6
  batch environment 7
  conversation integrity 73
  distributed environment 7
  interaction integrity 74
  intermediate commit points 74
  online transaction processing 7
unit of work 6
    batch environment 7
    distributed environment 7
    online transaction processing 7
update
    See U-lock mode

V
VSAM 1
VTAM 86

W
write claim class 43
write drain class 44

X
X-lock mode 18
### Artwork Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>File</th>
<th>Page</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITLOGO</td>
<td>4725SU</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>ITLOGOS</td>
<td>4725SU</td>
<td>i</td>
<td></td>
</tr>
</tbody>
</table>

### Table Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>File</th>
<th>Page</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLOCK</td>
<td>4725CH01</td>
<td>2</td>
<td>2, 2</td>
</tr>
<tr>
<td>THD</td>
<td>4725CH01</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>PLOCK</td>
<td>4725CH02</td>
<td>19</td>
<td>19, 19</td>
</tr>
<tr>
<td>THD0</td>
<td>4725CH02</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>TLOCK</td>
<td>4725CH02</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>THD1</td>
<td>4725CH02</td>
<td>19</td>
<td>19, 19</td>
</tr>
<tr>
<td>T11</td>
<td>4725CH02</td>
<td>28</td>
<td>28, 28</td>
</tr>
<tr>
<td>T12</td>
<td>4725CH02</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>SQLCODE</td>
<td>4725CH02</td>
<td>33</td>
<td>33, 33</td>
</tr>
<tr>
<td>THD2</td>
<td>4725CH02</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>472515A</td>
<td>4725CH02</td>
<td>42</td>
<td>42, 42</td>
</tr>
<tr>
<td>THD3</td>
<td>4725CH02</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>COOL</td>
<td>4725CH03</td>
<td>101</td>
<td>101</td>
</tr>
</tbody>
</table>

### Example Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>File</th>
<th>Page</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>4725XMP</td>
<td>4725VARS</td>
<td>i</td>
<td>125, 131, 135, 148, 150, 152, 154, 156</td>
</tr>
</tbody>
</table>

### Figures

<table>
<thead>
<tr>
<th>id</th>
<th>File</th>
<th>Page</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKU15</td>
<td>4725CH01</td>
<td>3</td>
<td>1, 2, 4</td>
</tr>
<tr>
<td>472511E</td>
<td>4725CH01</td>
<td>6</td>
<td>2, 6</td>
</tr>
<tr>
<td>LOCKU10</td>
<td>4725CH02</td>
<td>10</td>
<td>3, 9</td>
</tr>
<tr>
<td>LOCKU14</td>
<td>4725CH02</td>
<td>11</td>
<td>4, 11</td>
</tr>
<tr>
<td>LOCKU11</td>
<td>4725CH02</td>
<td>13</td>
<td>5, 13</td>
</tr>
<tr>
<td>LOCKU12</td>
<td>4725CH02</td>
<td>15</td>
<td>6, 14</td>
</tr>
<tr>
<td>LOCKU13</td>
<td>4725CH02</td>
<td>16</td>
<td>7, 16</td>
</tr>
<tr>
<td>LOCK6</td>
<td>4725CH02</td>
<td>23</td>
<td>8, 22</td>
</tr>
<tr>
<td>LOCK7</td>
<td>4725CH02</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Page</td>
<td>Line</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>QUAL6</td>
<td>4725Q6</td>
<td>104</td>
<td>39</td>
</tr>
<tr>
<td>QUAL7</td>
<td>4725Q7</td>
<td>105</td>
<td>40</td>
</tr>
<tr>
<td>QUAL8</td>
<td>4725Q8</td>
<td>106</td>
<td>41</td>
</tr>
<tr>
<td>LOCKE</td>
<td>4725CH04</td>
<td>110</td>
<td>42</td>
</tr>
<tr>
<td>LOCKF</td>
<td>4725CH04</td>
<td>112</td>
<td>43</td>
</tr>
<tr>
<td>LOCKG</td>
<td>4725CH04</td>
<td>114</td>
<td>44</td>
</tr>
<tr>
<td>LOCKH</td>
<td>4725CH04</td>
<td>117</td>
<td>45</td>
</tr>
<tr>
<td>LOCKI</td>
<td>4725CH04</td>
<td>118</td>
<td>46</td>
</tr>
<tr>
<td>LOCKJ</td>
<td>4725CH04</td>
<td>120</td>
<td>47</td>
</tr>
<tr>
<td>LOCKL</td>
<td>4725CH04</td>
<td>122</td>
<td>48</td>
</tr>
<tr>
<td>LOCKM</td>
<td>4725CH04</td>
<td>124</td>
<td>49</td>
</tr>
<tr>
<td>LOCKN</td>
<td>4725CH04</td>
<td>125</td>
<td>50</td>
</tr>
<tr>
<td>LOCKO</td>
<td>4725CH04</td>
<td>129</td>
<td>51</td>
</tr>
<tr>
<td>LOCKP</td>
<td>4725CH04</td>
<td>131</td>
<td>52</td>
</tr>
<tr>
<td>LOCKR</td>
<td>4725CH04</td>
<td>135</td>
<td>53</td>
</tr>
<tr>
<td>SCEN3</td>
<td>4725CH05</td>
<td>139</td>
<td>54</td>
</tr>
<tr>
<td>SCEN1</td>
<td>4725CH05</td>
<td>141</td>
<td>55</td>
</tr>
<tr>
<td>SCEN2</td>
<td>4725CH05</td>
<td>142</td>
<td>56</td>
</tr>
<tr>
<td>SCEN4</td>
<td>4725CH05</td>
<td>143</td>
<td>57</td>
</tr>
<tr>
<td>SCEN5</td>
<td>4725CH05</td>
<td>146</td>
<td>58</td>
</tr>
<tr>
<td>CASEA11</td>
<td>4725CH05</td>
<td>148</td>
<td>59</td>
</tr>
<tr>
<td>CASEB11</td>
<td>4725CH05</td>
<td>150</td>
<td>60</td>
</tr>
<tr>
<td>CASEC11</td>
<td>4725CH05</td>
<td>152</td>
<td>61</td>
</tr>
<tr>
<td>CASED11</td>
<td>4725CH05</td>
<td>154</td>
<td>62</td>
</tr>
<tr>
<td>CASEE11</td>
<td>4725CH05</td>
<td>156</td>
<td>63</td>
</tr>
<tr>
<td>LOCKD</td>
<td>4725CH06</td>
<td>159</td>
<td>64</td>
</tr>
<tr>
<td>PLCKNEG</td>
<td>4725CH06</td>
<td>162</td>
<td>65</td>
</tr>
<tr>
<td>id</td>
<td>File</td>
<td>Page</td>
<td>References</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>CH01</td>
<td>4725CH01</td>
<td>1</td>
<td>Chapter 1, DB2 Lock Management xi</td>
</tr>
<tr>
<td>THREECA</td>
<td>4725CH01</td>
<td>2</td>
<td>1.3, Three Basic Reasons for Locking 2</td>
</tr>
<tr>
<td>CH02</td>
<td>4725CH02</td>
<td>9</td>
<td>Chapter 2, DB2 Serialization Mechanisms xi</td>
</tr>
<tr>
<td>CH01S5</td>
<td>4725CH02</td>
<td>9</td>
<td>2.1, Transaction Locking 1</td>
</tr>
<tr>
<td>CH01S9</td>
<td>4725CH02</td>
<td>22</td>
<td>2.1.9, Lock Avoidance 28, 76</td>
</tr>
<tr>
<td>CH01S10</td>
<td>4725CH02</td>
<td>39</td>
<td>2.3, DB2 Subsystem Object Locking 10</td>
</tr>
<tr>
<td>CH03</td>
<td>4725CH03</td>
<td>63</td>
<td>Chapter 3, How to Prevent Locking Problems xi</td>
</tr>
<tr>
<td>PARM</td>
<td>4725CH03</td>
<td>67</td>
<td>3.2, Bind Option Considerations 77</td>
</tr>
<tr>
<td>CH02S2</td>
<td>4725CH03</td>
<td>72</td>
<td>3.3, Application Design and Programming Considerations 35</td>
</tr>
<tr>
<td>RO</td>
<td>4725CH03</td>
<td>74</td>
<td>3.3.2, Read-Only Considerations 76</td>
</tr>
<tr>
<td>UR</td>
<td>4725CH03</td>
<td>75</td>
<td>3.3.2.1, Uncommitted Read Considerations 71</td>
</tr>
<tr>
<td>LOCKNS</td>
<td>4725CH03</td>
<td>79</td>
<td>3.3.3.2, The Normal Solution: U-Lock Instead of S-Lock 79</td>
</tr>
<tr>
<td>AUSTRAL</td>
<td>4725CH03</td>
<td>79</td>
<td>3.3.3.3, The Australian Way: X-Lock Instead of S-Lock 97</td>
</tr>
<tr>
<td>MILLS</td>
<td>4725CH03</td>
<td>81</td>
<td>3.3.3.4, UK Solution</td>
</tr>
<tr>
<td>HOT</td>
<td>4725CH03</td>
<td>82</td>
<td>3.3.4, Hot pages 77, 87</td>
</tr>
<tr>
<td>LOCKSW</td>
<td>4725CH03</td>
<td>83</td>
<td>3.3.4.1, Swiss Solution</td>
</tr>
<tr>
<td>QUAL</td>
<td>4725CH03</td>
<td>87</td>
<td>3.4, Quality Card Analysis to Determine the Probability of Locking 64</td>
</tr>
<tr>
<td>CH02S27</td>
<td>4725CH03</td>
<td>107</td>
<td>3.5, Summary of Recommendations 139</td>
</tr>
<tr>
<td>CH04</td>
<td>4725CH04</td>
<td>109</td>
<td>Chapter 4, How to Identify and Analyze Locking Problems xi</td>
</tr>
<tr>
<td>DDLC</td>
<td>4725CH04</td>
<td>114</td>
<td>4.2.1, DISPLAY DATABASE LOCKS command 128</td>
</tr>
<tr>
<td>CH05</td>
<td>4725CH05</td>
<td>139</td>
<td>Chapter 5, Locking Scenarios xi</td>
</tr>
<tr>
<td>CH06</td>
<td>4725CH06</td>
<td>159</td>
<td>Chapter 6, Data Sharing Considerations xi</td>
</tr>
<tr>
<td>NOTICES</td>
<td>SG244725 SCRIPT</td>
<td>167</td>
<td>Appendix A, Special Notices ii</td>
</tr>
<tr>
<td>BIBL</td>
<td>4725BIBL</td>
<td>169</td>
<td>Appendix B, Related Publications</td>
</tr>
<tr>
<td>ORDER</td>
<td>REDB$ORD</td>
<td>171</td>
<td>How To Get ITSO Redbooks 169</td>
</tr>
<tr>
<td>id</td>
<td>File</td>
<td>Page</td>
<td>References</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>RLOCK</td>
<td>4725CH01</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>SERI</td>
<td>4725CH02</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9, 9</td>
</tr>
<tr>
<td>PLOCK</td>
<td>4725CH02</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>TLOCK</td>
<td>4725CH02</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>CH08T03</td>
<td>4725CH02</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>CH08T04</td>
<td>4725CH02</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>CH08T01</td>
<td>4725CH02</td>
<td>28</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>472516A</td>
<td>4725CH02</td>
<td>30</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>SQLCODE</td>
<td>4725CH02</td>
<td>33</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>RESS</td>
<td>4725CH02</td>
<td>37</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>RESP</td>
<td>4725CH02</td>
<td>38</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>36, 37</td>
</tr>
<tr>
<td>472515A</td>
<td>4725CH02</td>
<td>42</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>472514A</td>
<td>4725CH02</td>
<td>50</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>COOLERS</td>
<td>4725CH03</td>
<td>101</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>101, 101</td>
</tr>
<tr>
<td>472550A</td>
<td>4725CH05</td>
<td>140</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>140, 140, 140</td>
</tr>
<tr>
<td>472550B</td>
<td>4725CH05</td>
<td>144</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>143, 143</td>
</tr>
<tr>
<td>472551A</td>
<td>4725CH05</td>
<td>145</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>144, 145, 145</td>
</tr>
<tr>
<td>472552A</td>
<td>4725CH05</td>
<td>147</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>147</td>
</tr>
<tr>
<td>472552B</td>
<td>4725CH05</td>
<td>149</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>149</td>
</tr>
<tr>
<td>472552C</td>
<td>4725CH05</td>
<td>158</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>157</td>
</tr>
</tbody>
</table>