

# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

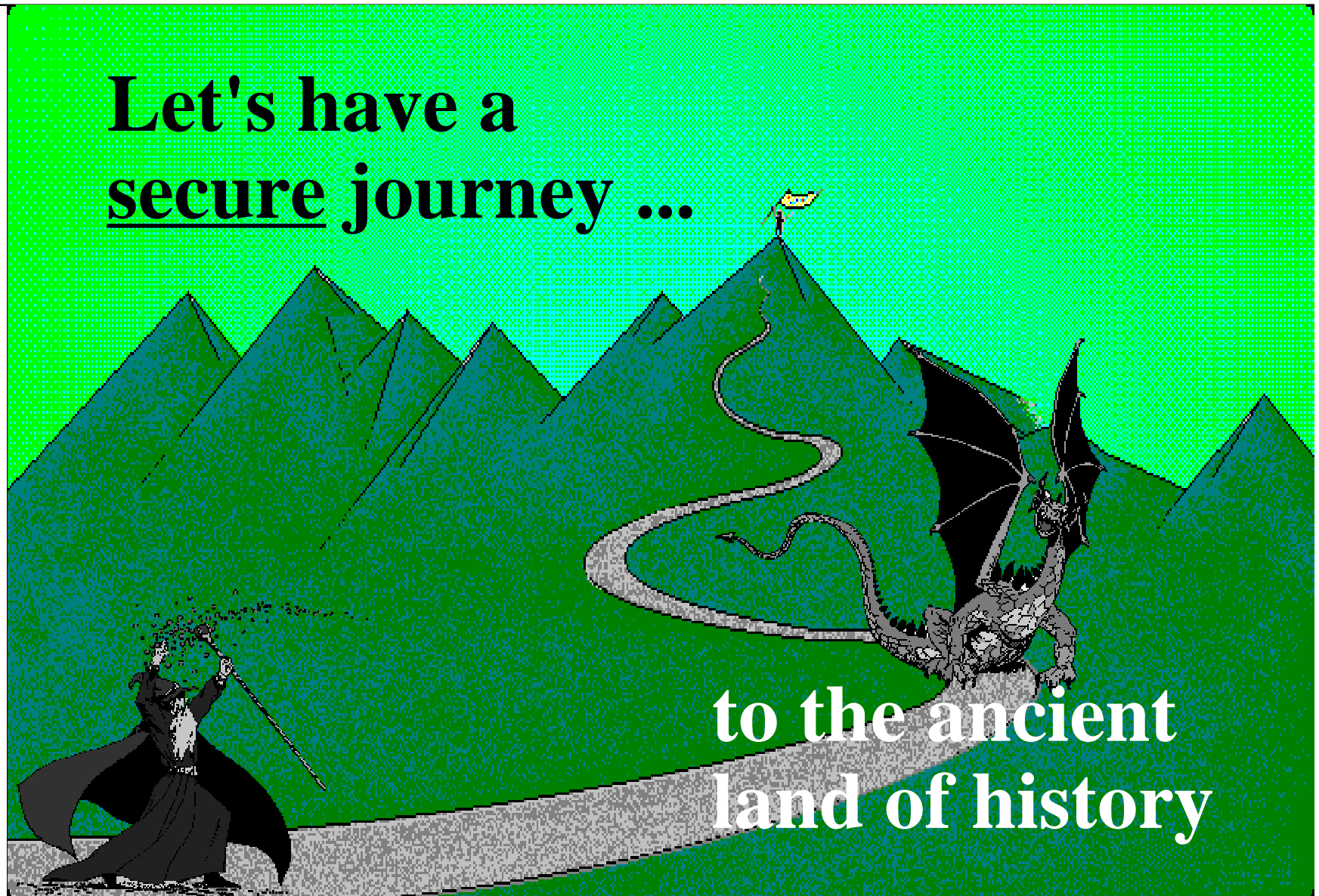
Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion



# Efficient Implementation of Historical Data in DB2

## Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

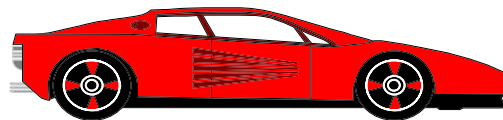
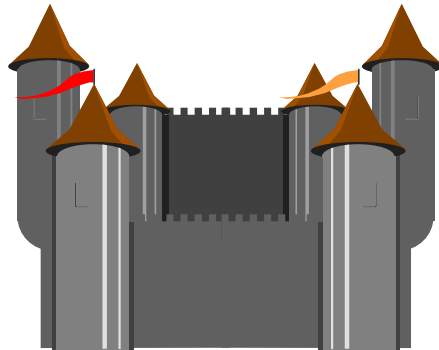
Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion



- **Architecture for historization is essential**

- to meet future demands
- otherwise application development tasks cannot be automatized
- advantages for maintenance

- **Efficiency is required**

- volume of stored data will increase at least by a factor of 10 in the next years
- we must be able to backup/restore huge tables

# Efficient Implementation of Historical Data in DB2

## Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

- **We shall learn ...**

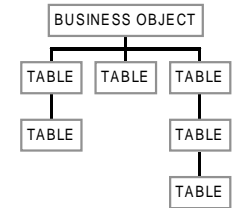
- about standardized maintenance of historical data
- about implementation of data-access-modules
- about the maintenance of a logical-unit-of-work (LUOW)
- how to organize things efficiently



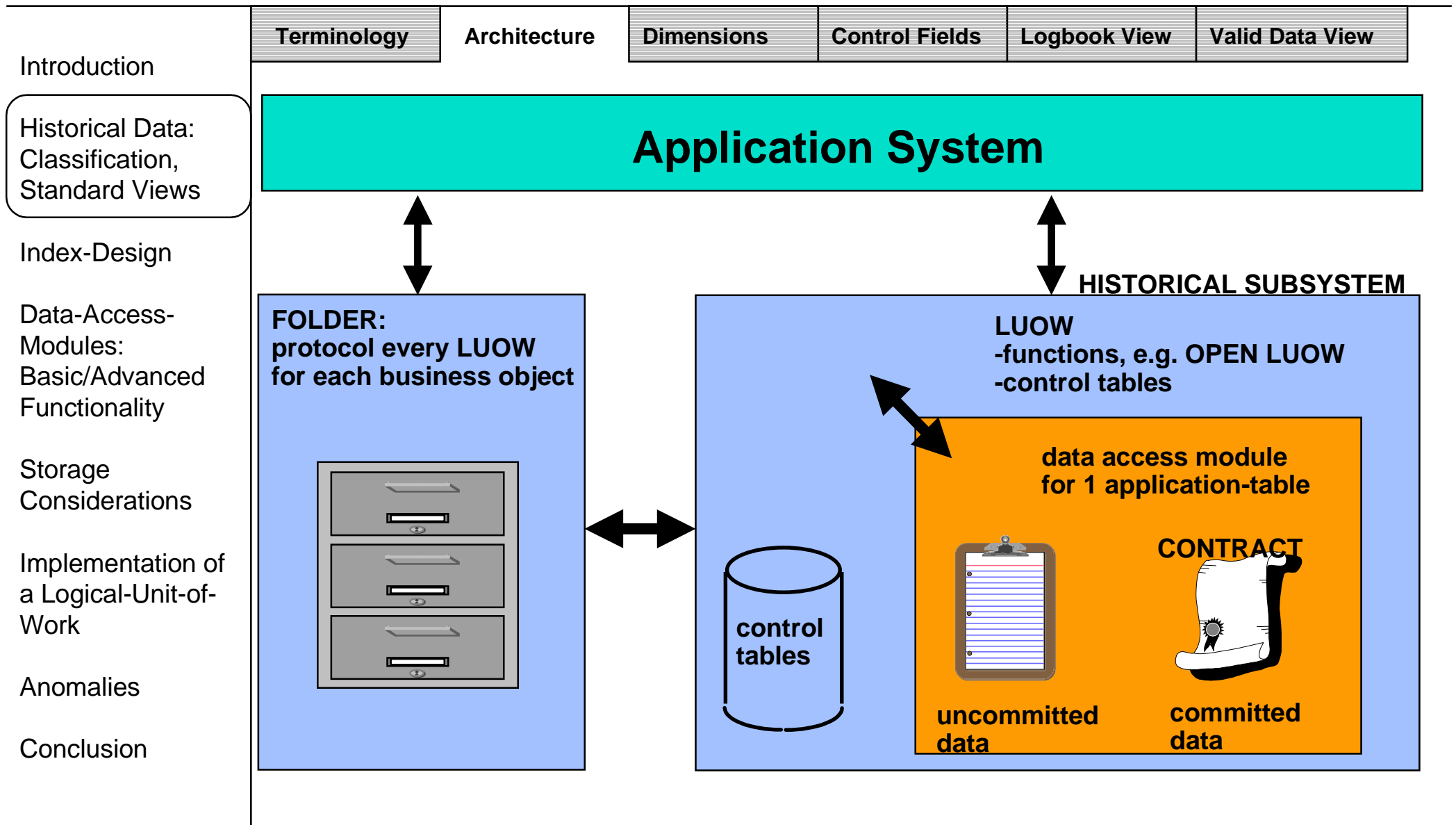
# Efficient Implementation of Historical Data in DB2

	Terminology	Architecture	Dimensions	Control Fields	Logbook View	Valid Data View
Introduction						
Historical Data: Classification, Standard Views						
Index-Design						
Data-Access-Modules: Basic/Advanced Functionality						
Storage Considerations						
Implementation of a Logical-Unit-of-Work						
Anomalies						
Conclusion						

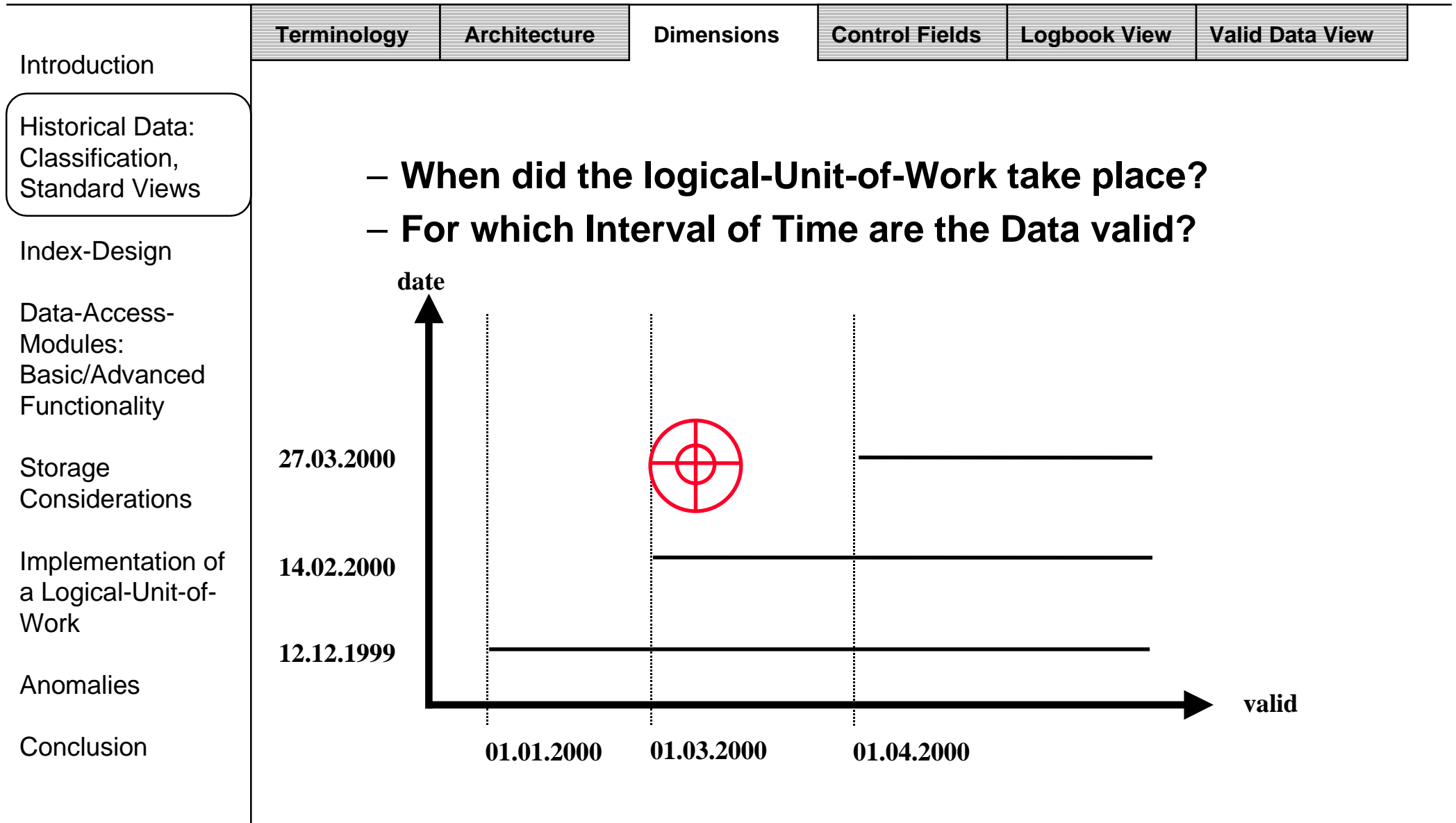
- **Business Object:**
  - » group of related data structures (=tables)
  
- **logical Unit of Work (LUOW):**
  - » summary of changes to one or more business-objects
  - » generalized UOW
  
- **Control field:**
  - » used to maintain relevant views on historical data in tables
  
- **Status:**
  - » control field to support transaction-independent UOW's



# Efficient Implementation of Historical Data in DB2



# Efficient Implementation of Historical Data in DB2



# Efficient Implementation of Historical Data in DB2

	Terminology	Architecture	Dimensions	Control Fields	Logbook View	Valid Data View
Introduction						
Historical Data: Classification, Standard Views						
Index-Design						
Data-Access- Modules: Basic/Advanced Functionality						
Storage Considerations						
Implementation of a Logical-Unit-of- Work						
Anomalies						
Conclusion						

## – Implementation:

- » no separate table(s) for "work data" ,  
i.e. integration into the original application tables
- » usage of status-field
- » row-wise historization

## – Benefits:

- » high efficiency for retrieval
  - only one table is affected
  - "simple" SQL-statements
- » less expensive at commit / rollback
  - COMMIT:  
UPDATE instead of DELETE/INSERT
  - ROLLBACK: DELETE



# Efficient Implementation of Historical Data in DB2

	Terminology	Architecture	Dimensions	Control Fields	Logbook View	Valid Data View
Introduction						
Historical Data: Classification, Standard Views						
Index-Design						
Data-Access-Modules: Basic/Advanced Functionality						
Storage Considerations						
Implementation of a Logical-Unit-of-Work						
Anomalies						
Conclusion						
	– <b>LU_START:</b>		start of a logical-unit-of-work			<b>TIMESTAMP</b>
	– <b>VALID_FROM:</b>		data is valid from ...			<b>DATE</b>
	– <b>VALID_TO:</b>		data is valid until ...			<b>DATE</b>
	– <b>INVALIDATED_BY:</b>		data is made invalid by the logical-unit-of-work			<b>TIMESTAMP</b>
	– <b>LU_COMMIT:</b>		commit-date of the logical-unit-of-work			<b>DATE</b>
	– <b>STATUS:</b>		status of the logical-unit-of-work			<b>CHAR(1)</b>

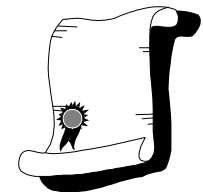
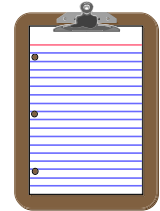


# Efficient Implementation of Historical Data in DB2

	Terminology	Architecture	Dimensions	Control Fields	Logbook View	Valid Data View
Introduction						
Historical Data: Classification, Standard Views						
Index-Design						
Data-Access- Modules: Basic/Advanced Functionality						
Storage Considerations						
Implementation of a Logical-Unit-of- Work						
Anomalies						
Conclusion						

- **Status**

- **A:** work data,  
rows can be inserted, updated or deleted  
"uncommitted" from the business point-of-view
- **B:** made permanent,  
can't be changed any more  
"committed" from the business point-of-view
- **X:** for information only,  
not relevant for the "valid data"-view



# Efficient Implementation of Historical Data in DB2

	Terminology	Architecture	Dimensions	Control Fields	Logbook View	Valid Data View
Introduction						
Historical Data: Classification, Standard Views						
Index-Design						
Data-Access- Modules: Basic/Advanced Functionality						
Storage Considerations						
Implementation of a Logical-Unit-of- Work						
Anomalies						
Conclusion						

– **development of data, from now back to the beginning (descending time order)**

– **"work data" is included**

– **Cursor from the business point of view**

```
SELECT    ...
FROM      table
WHERE     id = :hv-id
ORDER BY LU_start DESC
          ,valid_from DESC
```

# Efficient Implementation of Historical Data in DB2

	Terminology	Architecture	Dimensions	Control Fields	Logbook Cursor	Valid Data View
Introduction						
<div style="border: 1px solid black; border-radius: 10px; padding: 5px; width: fit-content;">           Historical Data:            Classification,            Standard Views         </div>						<ul style="list-style-type: none"> <li>– <b>(valid) state of business data at a given time</b></li> <li>– <b>exclusion of "work data ", i.e. "uncommitted" data from the business point of view</b></li> <li>– <b>only one "document" from the business point of view</b></li> </ul>
Index-Design						
Data-Access- Modules: Basic/Advanced Functionality						<pre> SELECT    ... FROM      table WHERE     id                = :hv-id AND       LU_start          &lt;= :hv-sight AND       valid_from        &lt;= :hv-valid AND       valid_to          &gt;= :hv-valid AND       invalidated_by    &gt;= :hv-sight AND       LU_commit         &lt;= :hv-sight AND       status            = 'B'  ORDER BY  LU_start DESC           ,valid_from DESC         </pre>
Storage Considerations						
Implementation of a Logical-Unit-of- Work						
Anomalies						
Conclusion						

# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

- **Choosing the Clustering Index**

- **"Valid data" view**

- » business point of view: singleton select
    - » SQL point of view: no singleton select  
no subselect  
but: ordered cursor with only one fetch

- **"Logbook" view**

- » business point of view: ordered cursor
    - » SQL point of view: ordered cursor

# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

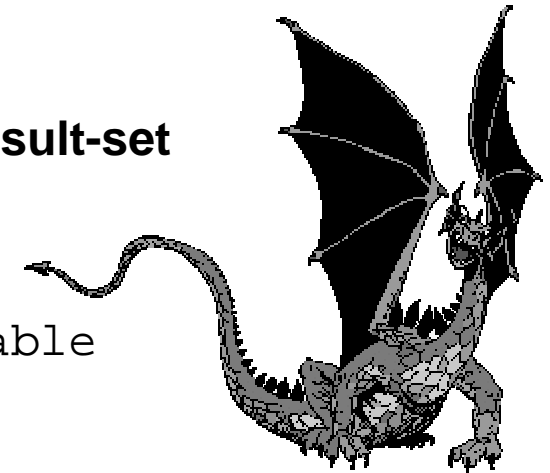
– **Avoid SORT:**  
**You never know the cardinality of the result-set**

– **Clustering Index for both views:**

```
» CREATE UNIQUE INDEX ... ON table  
   (id asc  
    ,LU_start desc  
    ,valid_from desc)  
   CLUSTER
```

– **Clustering Index satisfies:**

- » uniqueness
- » logbook view,       no sort required !
- » valid data view,    no sort required !



# Efficient Implementation of Historical Data in DB2

<p>Introduction</p> <p>Historical Data: Classification, Standard Views</p> <p>Index-Design</p> <p>Data-Access- Modules: Basic/Advanced Functionality</p> <p>Storage Considerations</p> <p>Implementation of a Logical-Unit-of- Work</p> <p>Anomalies</p> <p>Conclusion</p>	<p>Data Access Functions for Single Table</p> <ul style="list-style-type: none"><li>• <b>Basic Functions</b><ul style="list-style-type: none"><li>– INSERT, UPDATE, DELETE one row with status "A"</li><li>– UPDATE all rows for a given <i>id</i> and <i>LU_start</i> with status "A", set the status to "B" or "X"</li><li>– DELETE all rows for a given <i>id</i> and <i>LU_start</i> with status "A"</li><li>– logbook view</li><li>– valid data view</li><li>– for every (technical) cursor supply the following functions:<ul style="list-style-type: none"><li>» OPEN</li><li>» OPEN, FETCH n ROWS</li><li>» OPEN, FETCH n ROWS, CLOSE</li><li>» FETCH n ROWS</li><li>» FETCH n ROWS, CLOSE</li><li>» CLOSE</li></ul></li></ul></li></ul>	<p>Links between Tables</p>
--	--	-----------------------------

# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

Data Access Functions for Single Table

Links between Tables

- **Advanced Functions**

- **id consists of more than one column**

```
CREATE UNIQUE INDEX ... ON table
  (id_a asc
  ,id_b asc
  ,id_c asc
  ,LU_start desc
  ,valid_from desc)
CLUSTER
```

- **advanced functions, but often needed:**

- » return all latest=newest data for id\_a (all statuses)
- » return all current valid/committed data for id\_a (status "B" only)

# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

Data Access Functions for Single Table

Links between Tables

- **Advanced Functions**

id_a	id_b	id_c	bu_start	valid_from	valid_to	invalidated_by	bu_commit	status	user data
4711	1	1	1995-12-20-00.00.00	1996-01-01	9999-12-31	0001-01-01-00.00.00	1995-12-20-00	A	doesn't matter here
4711	1	1	1995-09-26-00.00.00	1995-10-01	9999-12-31	0001-01-01-00.00.00	1995-09-26-00	B	doesn't matter here
4711	1	1	1995-07-03-00.00.00	1995-08-01	9999-12-31	0001-01-01-00.00.00	1995-07-03-00	B	doesn't matter here
4711	1	1	1994-12-20-00.00.00	1995-01-01	9999-12-31	0001-01-01-00.00.00	1994-12-20-00	X	doesn't matter here
4711	1	1	1994-10-20-00.00.00	1994-11-01	9999-12-31	0001-01-01-00.00.00	1994-10-20-00	B	doesn't matter here
4711	1	2	1995-12-20-00.00.00	1996-01-01	9999-12-31	0001-01-01-00.00.00	1995-12-20-00	B	doesn't matter here
4711	1	2	1994-12-20-00.00.00	1995-01-01	9999-12-31	0001-01-01-00.00.00	1994-12-20-00	B	doesn't matter here
4711	1	2	1994-10-20-00.00.00	1994-11-01	9999-12-31	0001-01-01-00.00.00	1994-10-20-00	B	doesn't matter here
4711	2	1	1995-12-20-00.00.00	1996-02-01	9999-12-31	0001-01-01-00.00.00	1995-12-20-00	A	doesn't matter here
4711	2	1	1995-08-15-00.00.00	1995-09-01	9999-12-31	0001-01-01-00.00.00	1995-09-01	B	doesn't matter here
4711	2	1	1995-06-15-00.00.00	1995-07-01	9999-12-31	0001-01-01-00.00.00	1995-07-01	B	doesn't matter here



# Efficient Implementation of Historical Data in DB2

Introduction	Data Access Functions for Single Table	Links between Tables
Historical Data: Classification, Standard Views	<ul style="list-style-type: none"><li>• <b>Advanced Functions</b><ul style="list-style-type: none"><li>– return the latest data for id_a:<ul style="list-style-type: none"><li>» DECLARE cursor</li><li>» only id_a is qualified</li><li>» filter the desired rows in the application program</li></ul></li></ul></li></ul>	
Index-Design		
Data-Access- Modules: Basic/Advanced Functionality	<pre>SELECT    ... FROM      table t WHERE     t1.id_a      = :hv-id-a ORDER BY  t1.id_b           ,t1.id_c           ,t1.LU_start DESC           ,t1.valid_from DESC</pre>	
Storage Considerations	<ul style="list-style-type: none"><li>» simple SQL-statement</li><li>» prefetch may be used if desired</li></ul>	
Implementation of a Logical-Unit-of- Work		
Anomalies		
Conclusion		

# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

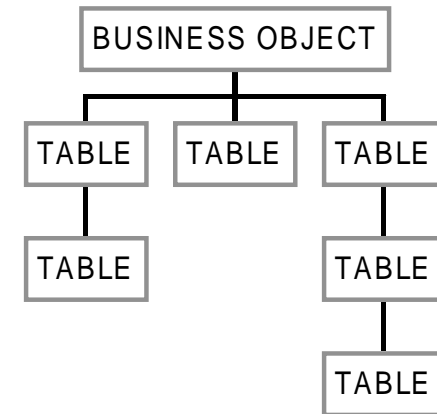
Anomalies

Conclusion

## Data Access Functions for Single Table

## Links between Tables

- **business object:**  
group of related data structures (=tables)
- **Examples:**
  - » orders and order-positions
  - » insurance contracts and risks
  - » claim and associated payments
  - » banking-account and positions
- **Efficiency**
  - » new rows are inserted only in those tables with changed data
  - » no complete new copy of the business object



# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

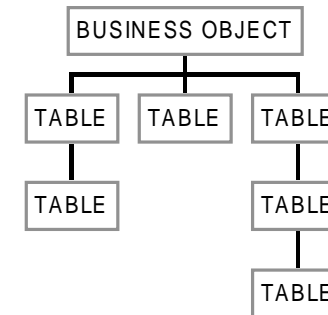
Anomalies

Conclusion

## Data Access Functions for Single Table

## Links between Tables

- **business object:** group of related data structures (=tables)
- **this fact should be visible in the key-definition**
- **consequence:**
  - » use of generic keys
  - » "root"-table: 1-column key=index  
if possible: good distribution of key
  - » "child"-tables inherit the parent key
  - » child-key consists of one additional column  
for uniqueness
- **benefits:**
  - » easier control of a logical-unit-of-work
  - » easier control of a business object,  
only one generic key req'd
  - » less physical I/O for advanced functions



# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

## Storing a Table on Different Devices

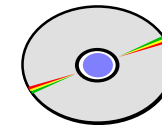
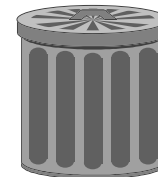
## Arguments for particular Devices

– **High-speed devices:** 3390-3  
RAMACS

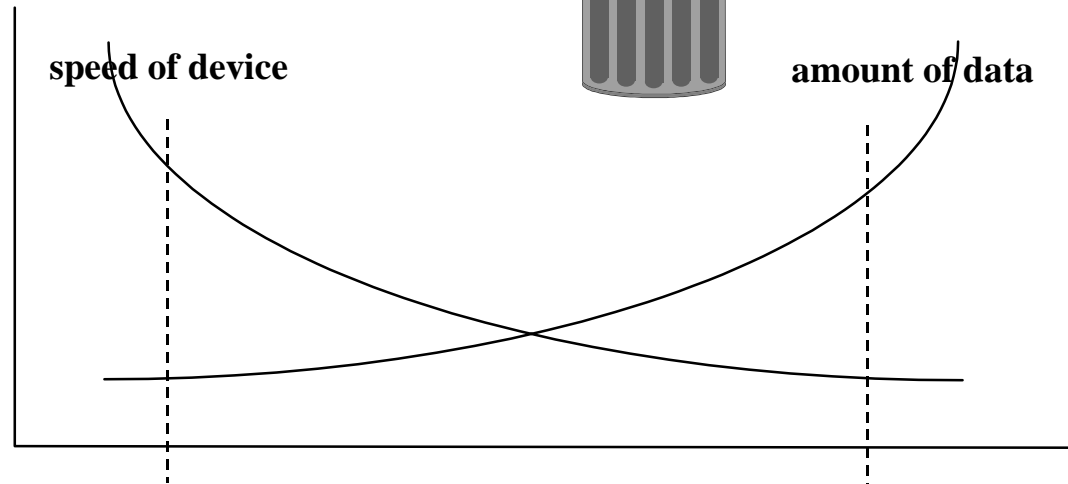
– **Medium-speed devices:** 3390-9

– **Low-speed devices:** Optical disc

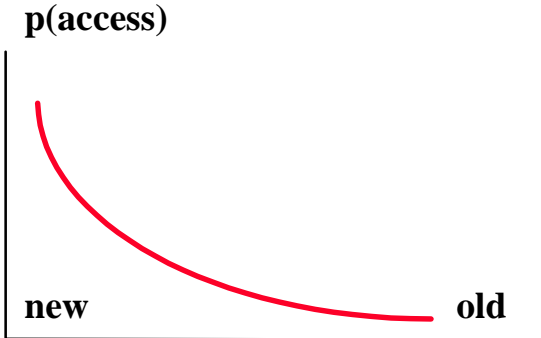
– **No-speed-device:**



*are data still required?*



# Efficient Implementation of Historical Data in DB2

Introduction	<b>Storing a Table on Different Devices</b>	Arguments for particular Devices
Historical Data: Classification, Standard Views	<ul style="list-style-type: none"><li>– <b>the 80/20-rule also applies here:</b><ul style="list-style-type: none"><li>» 80% old data = 20% access probability</li><li>» 20% new data = 80% access probability</li></ul></li><li>– <b>Rule-of-Thumb:</b><ul style="list-style-type: none"><li>» 20% of data on the fastest (= most expensive) devices always status = "A", the newest status = "B" lowest possible index-level at the fastest device</li><li>» 50 % of data on medium-speed-devices consider compression, low probability of changes</li><li>» 30 % of data on low-speed-devices</li></ul></li><li>• <b>Devices must be transparent to the application</b><ul style="list-style-type: none"><li>– another reason for data-access-modules</li></ul></li></ul>	
Index-Design		
Data-Access- Modules: Basic/Advanced Functionality		
<b>Storage Considerations</b>		
Implementation of a Logical-Unit-of- Work		
Anomalies		
Conclusion		

# Efficient Implementation of Historical Data in DB2

Introduction	<b>Storing a Table on Different Devices</b>	Arguments for particular Devices
Historical Data: Classification, Standard Views	<ul style="list-style-type: none"><li>• <b>Code-Example: logbook view on a table</b></li></ul>	
Index-Design		
Data-Access- Modules: Basic/Advanced Functionality		
<b>Storage Considerations</b>		
Implementation of a Logical-Unit-of- Work		
Anomalies		
Conclusion		

## Storing a Table on Different Devices

## Arguments for particular Devices

- **Code-Example: logbook view on a table**

```
OPEN c1-fast
FETCH c1-fast
PERFORM UNTIL SQLCODE <> ZERO
    process data
    FETCH c1-fast
END-PERFORM
IF SQLCODE = 100
    CLOSE c1-fast
END-IF
```

```
IF count-c1-medium > zero
    OPEN c1-optical
    FETCH c1-optical
    PERFORM UNTIL SQLCODE <> ZERO
        process data
        FETCH c1-optical
    END-PERFORM
    IF SQLCODE = 100
        CLOSE c1-optical
    END-IF
END-IF
```

```
move zero to count-c1-medium
OPEN c1-medium
FETCH c1-medium
PERFORM UNTIL SQLCODE <> ZERO
    add 1 to count-c1-medium
    process data
    FETCH c1-medium
END-PERFORM
IF SQLCODE = 100
    CLOSE c1-medium
END-IF
```

# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

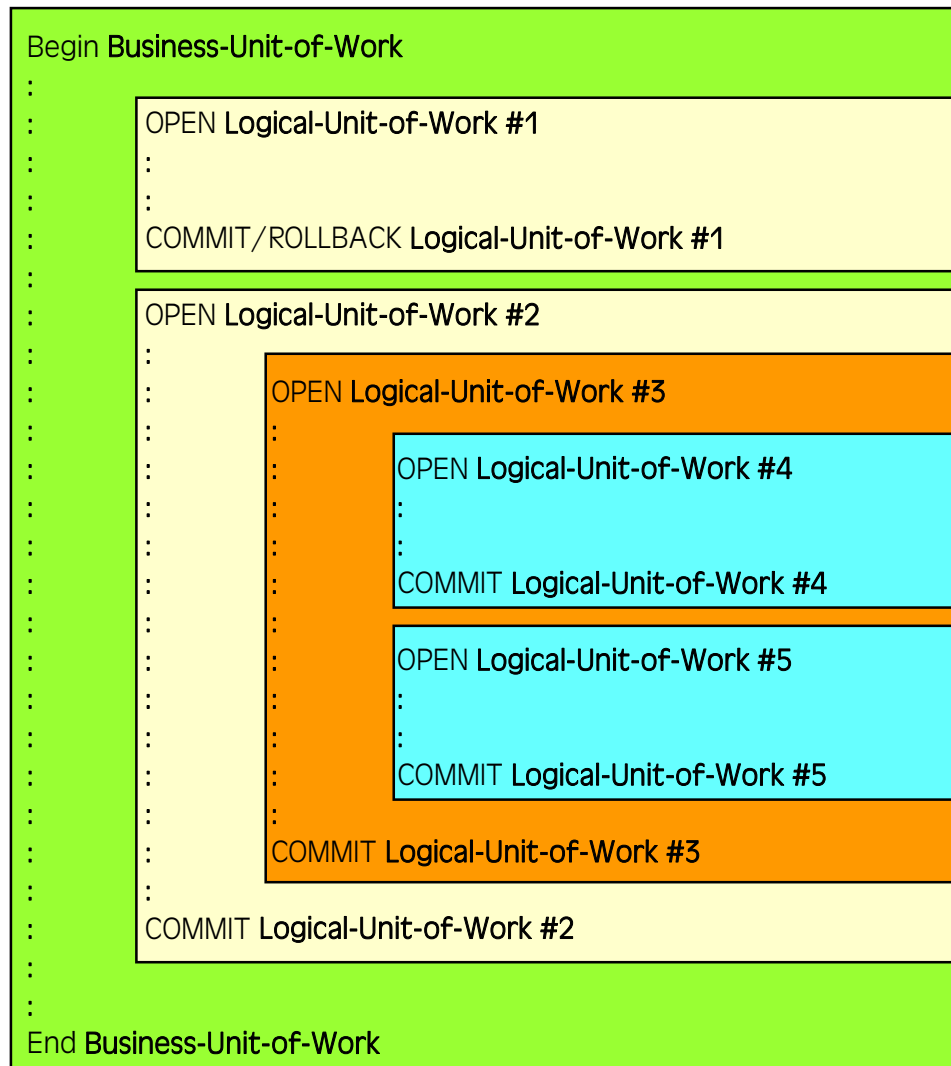
Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

## Properties

## Functions for a Business-Unit-of-Work



## Business-Unit-of-Work:

- **multiple** Logical-Unit-of-Work
- **nested** Logical-Unit-of-Work
- **suspend/resume at every point**
- **transaction independent**
- **common identifier, i.e. LU\_start**
- **modification of multiple BO's**
- **Exactly one status at every time**

# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

## Properties

```

OPEN logical-Unit-of-Work
:
:   GET Global-Change-Log Business-Objekt #1
:
:   GET Business-Objekt #1
:   LOCK Business-Objekt #1
:
:   GET Global-Change-Log-Business-Objekt #2
:   GET Global-Change-Log-Business-Objekt #3
:
:   LOCK Business-Objekt #2
:   LOCK Business-Objekt #3
:   GET Business-Objekt #2
:
:   CHANGE Business-Objekt #1
:
:   U N T E R B R E C H U N G
:
:   CHANGE Business-Objekt #2
:
:
COMMIT/ROLLBACK Logical-Unit-of-Work
    
```

## Functions for a Business-Unit-of-Work

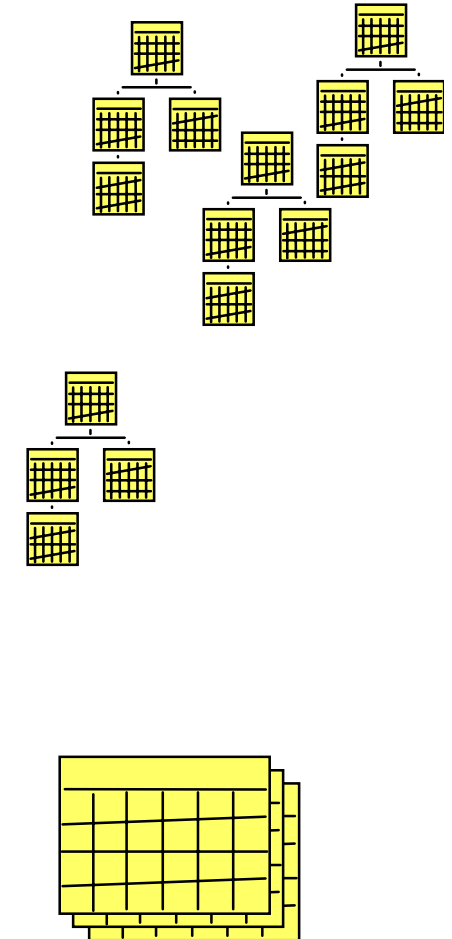
Logical-  
Unit-of-Work

read/  
modify

Business-Objects

built of

Tables





# Efficient Implementation of Historical Data in DB2

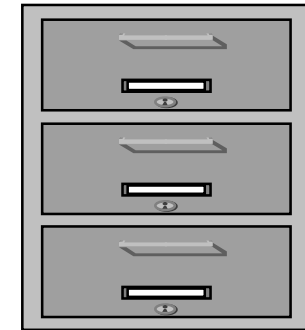
	Properties	Functions for a Business-Unit-of-Work
Introduction		
Historical Data: Classification, Standard Views	<h2 data-bbox="555 459 1684 513">Control Functions for a Logical-Unit-of-Work</h2>	
Index-Design		
Data-Access- Modules: Basic/Advanced Functionality	<ul data-bbox="555 619 1930 810" style="list-style-type: none"><li data-bbox="555 619 936 667">– <b>OPEN LUOW</b><ul data-bbox="645 699 1706 810" style="list-style-type: none"><li data-bbox="645 699 1550 746">» Reservation of a system-wide timestamp</li><li data-bbox="645 762 1706 810">» timestamp is made permanent in a control-table</li></ul></li></ul>	
Storage Considerations		
Implementation of a Logical-Unit-of- Work	<ul data-bbox="555 906 1930 1098" style="list-style-type: none"><li data-bbox="555 906 990 954">– <b>COMMIT LUOW</b><ul data-bbox="645 986 1930 1098" style="list-style-type: none"><li data-bbox="645 986 1930 1034">» status of all rows belonging to the LUOW is changed to "B"</li><li data-bbox="645 1050 1281 1098">» previous status must be "A"</li></ul></li></ul>	
Anomalies		
Conclusion	<ul data-bbox="555 1193 1617 1385" style="list-style-type: none"><li data-bbox="555 1193 1079 1241">– <b>ROLLBACK LUOW</b><ul data-bbox="645 1273 1617 1385" style="list-style-type: none"><li data-bbox="645 1273 1617 1321">» all rows belonging to the LUOW are deleted</li><li data-bbox="645 1337 1370 1385">» status of the LUOW must be "A"</li></ul></li></ul>	

# Efficient Implementation of Historical Data in DB2

Properties	Functions for a Business-Unit-of-Work
Introduction	
Historical Data: Classification, Standard Views	<h2>Logical locking of Business Objects (BO) in a LUOW</h2>
Index-Design	<ul style="list-style-type: none"><li>– <b>LOCK BO</b><ul style="list-style-type: none"><li>» explicit lock issued by application</li><li>» no data changes required</li><li>» with data changes BO is locked implicitly</li></ul></li></ul>
Data-Access- Modules: Basic/Advanced Functionality	<ul style="list-style-type: none"><li>– <b>INQUIRE BO</b><ul style="list-style-type: none"><li>» function returns if a logical lock is held on a BO</li></ul></li></ul>
Storage Considerations	<ul style="list-style-type: none"><li>– <b>RELEASE BO</b><ul style="list-style-type: none"><li>» reset of an explicit logical lock</li><li>» no changes to BO allowed</li></ul></li></ul>
Implementation of a Logical-Unit-of- Work	
Anomalies	
Conclusion	

# Efficient Implementation of Historical Data in DB2

Properties	Functions for a Business-Unit-of-Work
Introduction	
Historical Data: Classification, Standard Views	<ul style="list-style-type: none"><li>• <b>BO-"folder" for better efficiency</b><ul style="list-style-type: none"><li>– recommended, but not a must</li><li>– contains complete history of BO since beginning Reason: no complete copy of BO available</li></ul></li></ul>
Index-Design	
Data-Access- Modules: Basic/Advanced Functionality	<ul style="list-style-type: none"><li>• <b>Control-table for OPEN LUOW</b><ul style="list-style-type: none"><li>– one column only: LU_start_iv CHAR(10), contains inverted timestamp</li><li>– avoids a hot-spot</li><li>– ensures system-wide uniqueness of LUOW identifier</li></ul></li></ul>
Storage Considerations	
Implementation of a Logical-Unit-of- Work	<ul style="list-style-type: none"><li>• <b>Control-table for LUOW "logging"</b><ul style="list-style-type: none"><li>– (id-container, business_object, LU_start_iv, status, table-container)</li><li>– no hot-spot</li><li>– protocol of every change on business objects, system wide logging</li><li>– table is needed for efficient commit/rollback entries can be deleted after commit/rollback</li><li>– also used to lock business objects</li></ul></li></ul>
Anomalies	
Conclusion	



# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

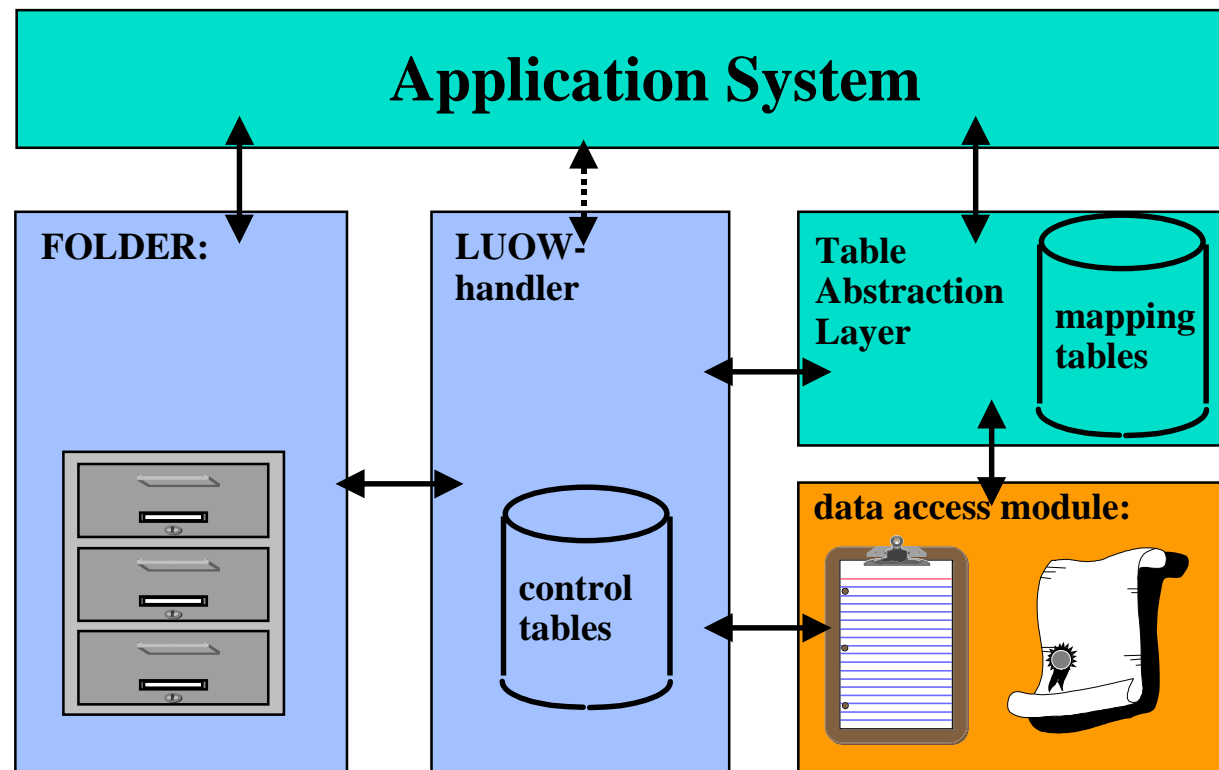
Anomalies

Conclusion

ALTER TABLE

Structure Change of Business-Object

*Architecture makes things easier*



# Efficient Implementation of Historical Data in DB2

Introduction	ALTER TABLE	Structure Change of Business-Object
Historical Data: Classification, Standard Views	<ul style="list-style-type: none"><li>• <b>ALTER TABLE</b><ul style="list-style-type: none"><li>– e.g. add column</li></ul></li></ul>	
Index-Design	<ul style="list-style-type: none"><li>– ALTER table on the fastest device</li></ul>	
Data-Access- Modules: Basic/Advanced Functionality	<ul style="list-style-type: none"><li>– new views for the other tables</li><li>– req'd changes:<ul style="list-style-type: none"><li>» table abstraction layer: extended mapping</li><li>» data-access-module: extended interface</li></ul></li></ul>	
Storage Considerations		
Implementation of a Logical-Unit-of- Work		
Anomalies		
Conclusion		

# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

ALTER TABLE

Structure Change of Business-Object

## – Table Abstraction Layer in detail:

» mapping between business-objects and tables

» *recommendation: keep the mapping as simple as possible*

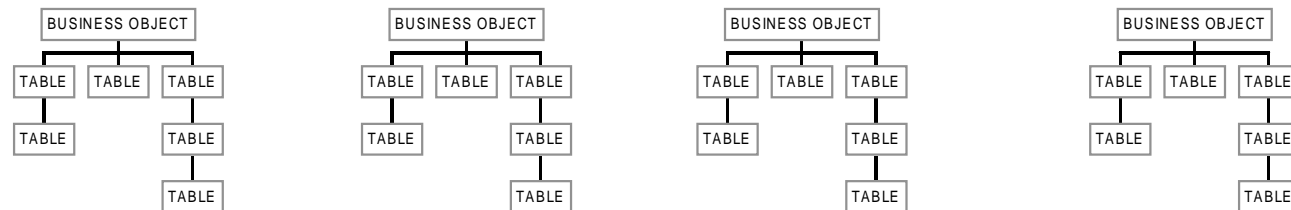
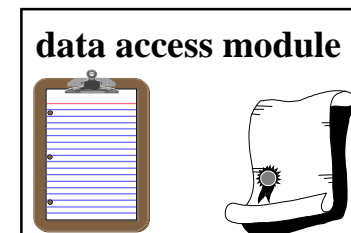
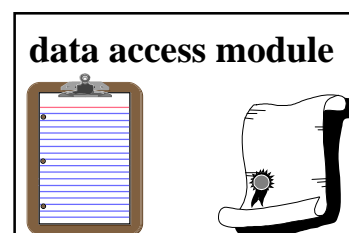
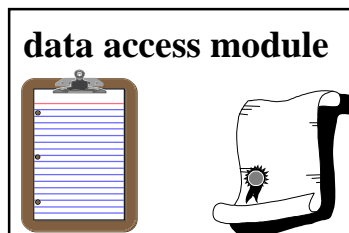


Table Abstraction Layer



# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

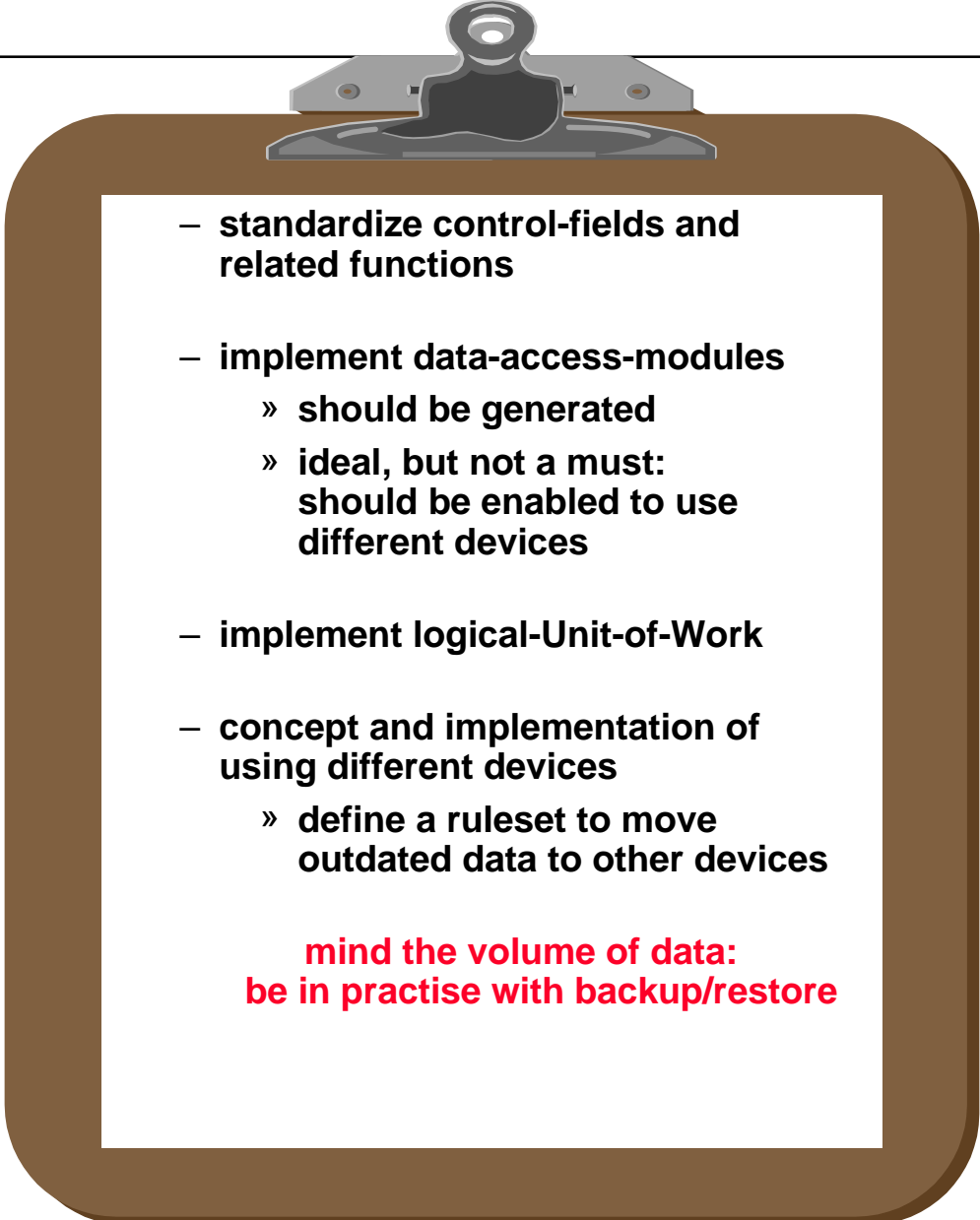
Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

- 
- **standardize control-fields and related functions**
  - **implement data-access-modules**
    - » **should be generated**
    - » **ideal, but not a must: should be enabled to use different devices**
  - **implement logical-Unit-of-Work**
  - **concept and implementation of using different devices**
    - » **define a ruleset to move outdated data to other devices**

**mind the volume of data:  
be in practise with backup/restore**

# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion

- **We wanted to learn ...**

- about standardized maintenance of historical data

➔ ***control-fields***

- about implementation of data-access-modules

➔ ***flexibility, control***

- about the maintenance of a logical-unit-of-work (LUOW)

➔ ***functions  
control data***

- how to do organize efficiently





# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion



# Efficient Implementation of Historical Data in DB2

Introduction

Historical Data:  
Classification,  
Standard Views

Index-Design

Data-Access-  
Modules:  
Basic/Advanced  
Functionality

Storage  
Considerations

Implementation of  
a Logical-Unit-of-  
Work

Anomalies

Conclusion



## Questions?

**Jürgen Glag Consulting**  
Brucknerstrasse 38  
D-40822 Mettmann  
Germany

phone: +49 172 2420393  
fax: +49 2104 916075  
email: [jglag@glag-consult.de](mailto:jglag@glag-consult.de)  
<http://www.glag-consult.de>