Exploit Db2's Log

Disclaimer

- I do not work for IBM
- I did not develop any part of Db2
- All information in this presentation is based on publicly available API documentation, examination of the behavior of Db2 for z/OS, and long hours of trial and error
- This presentation has been made to the best of my knowledge but I cannot guarantee correctness
- Some aspects are simplified because it either makes things easier to understand or I just don't know any better

Purpose of the DB2 log

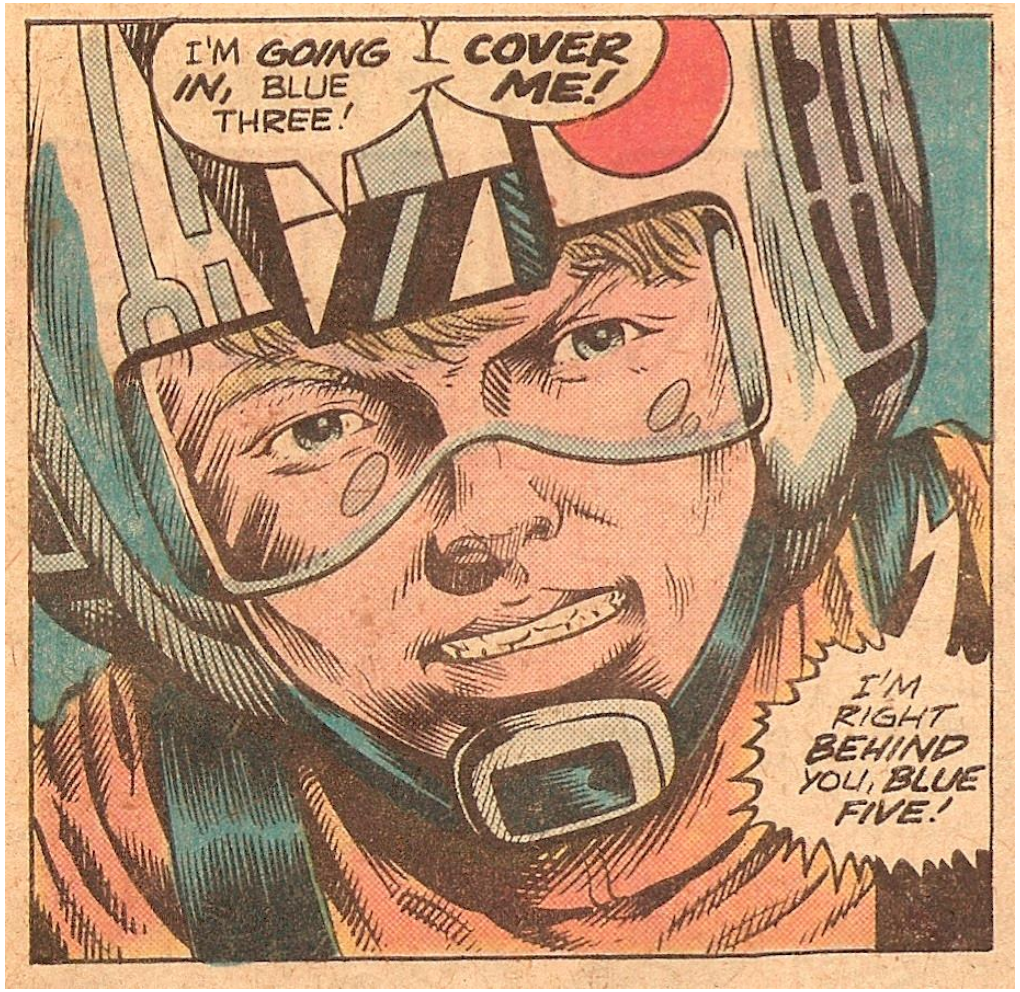
- Essential for maintaining the consistency of the database
- Also essential for recovering objects
- DB2 considers the log so important that it has the option to keep two identical copies
- Log records are only added, existing log records are never changed or removed
- Bottom line: The log is a protocol of every event that modified data

Role of the log in ACID properties

- Allows Db2 to roll back a transaction
  - Undo all changes made by a transaction
  - After explicit ROLLBACK of if the transaction fails for any other reason
  - Key element in guaranteeing **atomicity** of transactions
- Allows DB2 to achieve consistency after a crash
  - Write-ahead-log: Changes are written to the log first, then to the table space
  - Key element in guaranteeing **durability** of transactions

How does Db2 write to the log?

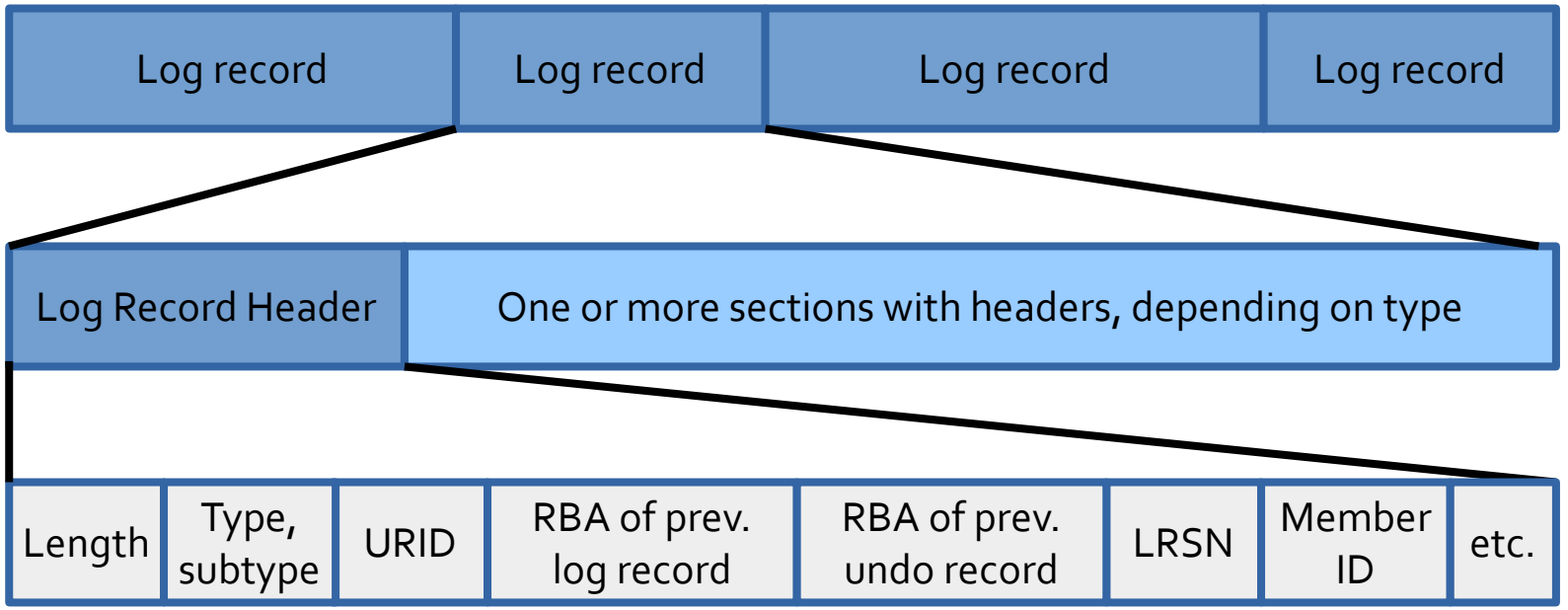
- Log records are written into the log output buffer (fixed in real storage), flushed to DASD when full
- At COMMIT time, the log buffer is synchronously written to the active log (on DASD)
  - Unlike the modified pages, which stay in the buffer pool
  - COMMIT is not confirmed until log records are on DASD
- Current active log data set is copied into an archive log (on DASD or tape) when it is full, or when the ARCHIVE LOG command is invoked



Enough warm up. Let's look at the gory details.

Structure of the log

(not to any scale)



All log records have a fixed length log record header (**LRH**) that always contains the same fields. Data after the log record header depends on what the log record represents.

Log record types

- **Unit of recovery** log records
  - Begin of UR, Commit, Rollback
- **Data change** log records
  - Describe physical changes to a page
  - Can represent insert, update, delete in a table space / index
  - Can also represent space map changes or other changes
- **Checkpoint** log records
  - Created whenever DB2 creates a checkpoint
  - Contain list of open transactions, modified page sets

Data change log records

- Written whenever something on a page changes
- Always contains:
  - LRH
  - LGDBHEAD (has fields for DBID, PSID, page#)
- Mostly insert / update / delete, in which case it also contains:
  - LGBENTRY (has fields for OBID, slot# in page)

Making log records visible

- DSN1LOGP
  - Specify start and end RBA/LRSN
  - Optionally specify filters (DBID, PSID, URID, log record type)
  - Output is a hex dump of all matching log records
  - Some header fields are formatted, but the rest is hard to read

Let’s dissect a log record

Image: Wikipedia, License: Public Domain

Example

- Db2 V12 Log record as printed by DSN1LOGP
- INSERT of a row into a tablespace
- DSN1LOGP formats parts of LRH, LGDBHEAD
- DSECT with log record structure: SDSNMACS(DSNDQJ00)

```
0000000000243FFB00A6  TYPE( UNDO  REDO )  URID(0000000000243FFAF8BA)
            LRSN(00D3F3E7C96C60520000)  DBID(0125)  OBID(0002)  PAGE(00000002)                    10:25:08 18.058
            SUBTYPE(INSERT IN A DATA PAGE)  CLR(NO)  PROCNAME(DSNISGRT)

     *LRH* 000000D1 00A60009 0EA00000 00000000 00000000 00243FFA F8BA0000 00000000  *   J w                  8
           00000000 00243FFB 00005000 06000001 00000000 00000024 3FFB0000 000000D3  *          &                    L
           F3E7C96C 60520000 00000000 00000000                                     *3XI%-
     *LG** 08012500 02000000 02000000 00000024 3FFB0000 4C400000 00000000 00000000  *                     <
           00000000                                                                *
     0000  005D4001 00030011 00000000 00000000 00004D00 03018000 00011858 10270012  * )                (
     0020  001A0023 0038003D E3C8C5D6 C4D6D9C5 D9D6D6E2 C5E5C5D3 E3F1F6F0 F040D7C5  *        THEODOREROOSEVELT1600 PE
     0040  D5D5E2C9 D3E5C1D5 C9C140C1 E5C5F2F0 F0F0F6E6 C1E2C8C9 D5C7E3D6 D5        *NNSILVANIA AVE20006WASHINGTON
```

```
*LRH*  000000D1 00A60009 0EA00000 00000000 00000000 00243FFA F8BA0000 00000000
       00000000 00243FFB 00005000 06000001 00000000 00000024 3FFB0000 0000 00D3
       F3E7C96C 60520000 00000000 00000000
*LG**  08012500 02000000 02000000 00000024 3FFB0000 4C400000 00000000 00000000
       00000000
0000   005D4001 00030011 00000000 00000000 00004D00 03018000 00011858 10270012
0020   001A0023 0038003D E3C8C5D6 C4D6D9C5 D9D6D6E2 C5E5C5D3 E3F1F6F0 F040D7C5
0040   D5D5E2C9 D3E5C1D5 C9C140C1 E5C5F2F0 F0F0F6E6 C1E2C8C9 D5C7E3D6 D5
```

- URID of the log record. Identical for all log records that belong to this transaction.
- RBA of previous log record within this transaction
- Indicates that this is a data change record with redo and undo information
- Indicates that this log record represents a basic data page change
- LRSN of the log record
- DBID and PSID of the modified table space (DBID X'0125', PSID X'0002')
- Page number of the page that was modified (Page X'00000002'
- If compensation record: RBA of the log record that is compensated by this log record
- Slot number ("ID map entry") inside the page
- OBID of the table to which the new row belongs (OBID X'0003')
- The new row exactly as it appears in the table space including 6 byte row header

UBS HAINER

INSERT, UPDATE, DELETE

- INSERT and DELETE are simple
  - Contain the entire row
- UPDATE is more complex
  - Contains before image and after image
  - Can be split to more than one log record
  - Roughly a dozen different update variations
  - With or without data capture changes
  - In-place or non in-place
  - Can change normal records to pointer records

UPDATE

- Non-DCC UPDATE records log a partial row
  - Only the bytes that changed
  - That's all DB2 needs to apply the log record
- Tricky to restore the full row:
  - Identify page and slot number
  - Find an older full image of the row (where?)
  - Look for additional updates since the identified full image
  - Possible, but can take a long time

Redoing and undoing changes

- REDO: This log record contains information required to **apply** the change
  - In this case: DB2 inserts the row found in the log record into the table space
  - Example: Recovery process
    - Restore an image copy
    - Then apply all log records up to the desired point in time

Redoing and undoing changes

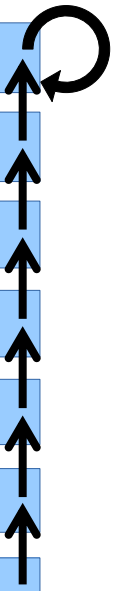
- UNDO: This log record contains information required to **reverse** the change
  - In this case: DB2 removes the row found in the log record from the table space
  - Example: Canceling a transaction
    - Reverse the effects of all changes that were made in the transaction that is being canceled
    - While DB2 reverses the changes, it writes log records to protocol what it is doing (compensation records)

Transaction / COMMIT

- Transaction start log record:
  - BEGIN UR (the RBA of this log record becomes the URID)
- Log records describing data changes
- Transaction end log records:
  - BEGIN COMMIT PHASE1
  - SWITCH PHASE 1 TO 2
  - END COMMIT PHASE 2

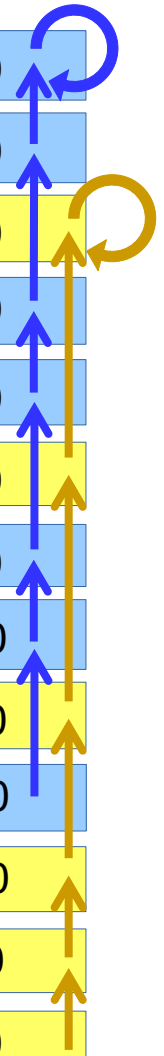
Transaction / COMMIT

RBA	Type	URID	Compens.	Comp.RBA	Undo Next
000500	BEGIN UR	000500			000500
000600	INSERT	000500	N		000500
000700	INSERT	000500	N		000600
000800	UPDATE	000500	N		000700
000900	BEGIN COMMIT1	000500			000800
000A00	PHASE 1 TO 2	000500			000900
000B00	END COMMIT2	000500			000A00



Multiple Parallel Transactions

RBA	Type	URID	Compens.	Comp.RBA	Undo Next
000500	BEGIN UR	000500			000500
000600	INSERT	000500	N		000500
000700	BEGIN UR	000700			000700
000800	INSERT	000500	N		000600
000900	UPDATE	000500	N		000800
000A00	INSERT	000700	N		000700
000B00	BEGIN COMMIT1	000500			000900
000C00	PHASE 1 TO 2	000500			000B00
000D00	INSERT	000700	N		000A00
000E00	END COMMIT2	000500			000C00
000F00	BEGIN COMMIT1	000700			000D00
001000	PHASE 1 TO 2	000700			000F00
001100	END COMMIT2	000700			001000

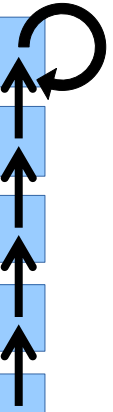


Transaction / ROLLBACK

- Transaction start log record:
  - BEGIN UR
- Log records describing data changes
- Transaction end log records (for commit):
  - BEGIN ABORT
  - Log records describing how all changes are undone
  - END ABORT

Transaction / ROLLBACK

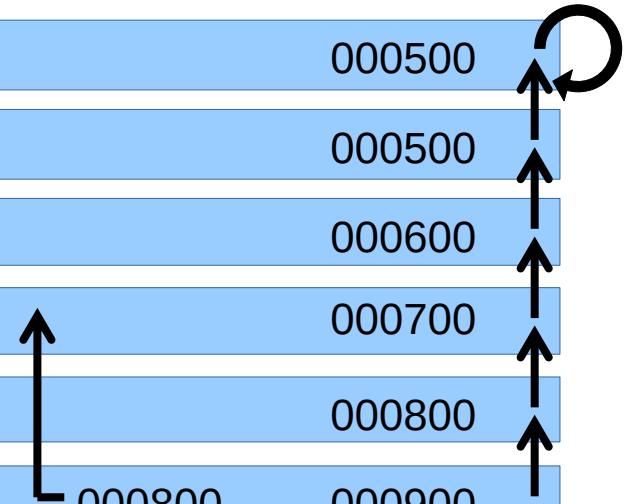
RBA	Type	URID	Compens.	Comp.RBA	Undo Next
000500	BEGIN UR	000500			000500
000600	INSERT	000500	N		000500
000700	INSERT	000500	N		000600
000800	UPDATE	000500	N		000700
000900	BEGIN ABORT	000500			000800



- DB2 must undo all changes from this transaction
- It follows the "undo next" chain
- For each log record that carries UNDO information:
  - The change that this log record describes is reverted
  - A new log record is written, documenting what was done

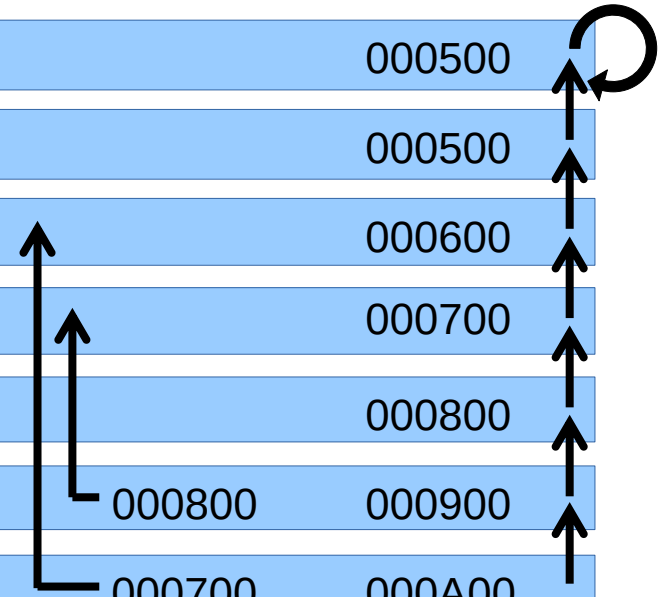
Transaction / ROLLBACK

RBA	Type	URID	Compens.	Comp.RBA	Undo Next
000500	BEGIN UR	000500			000500
000600	INSERT	000500	N		000500
000700	INSERT	000500	N		000600
000800	UPDATE	000500	N		000700
000900	BEGIN ABORT	000500			000800
000A00	UPDATE	000500	Y	000800	000900



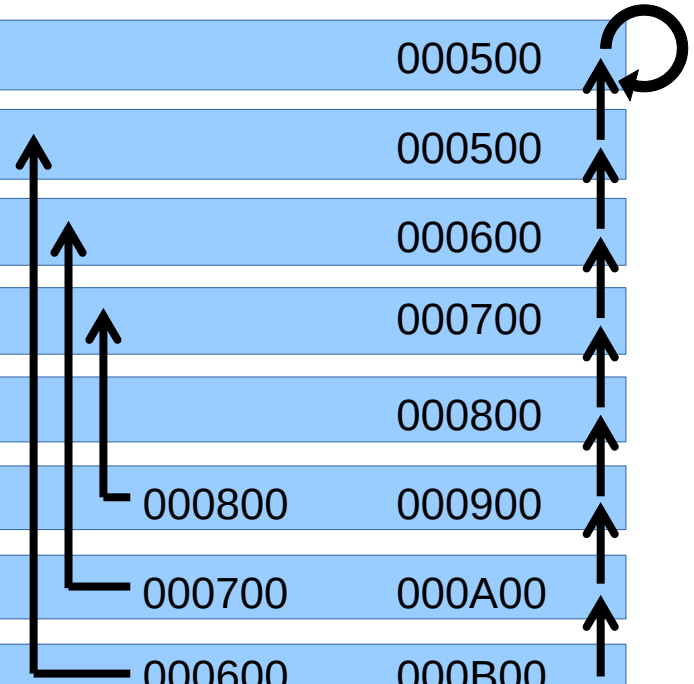
Transaction / ROLLBACK

RBA	Type	URID	Compens.	Comp.RBA	Undo Next
000500	BEGIN UR	000500			000500
000600	INSERT	000500	N		000500
000700	INSERT	000500	N		000600
000800	UPDATE	000500	N		000700
000900	BEGIN ABORT	000500			000800
000A00	UPDATE	000500	Y	000800	000900
000B00	DELETE	000500	Y	000700	000A00



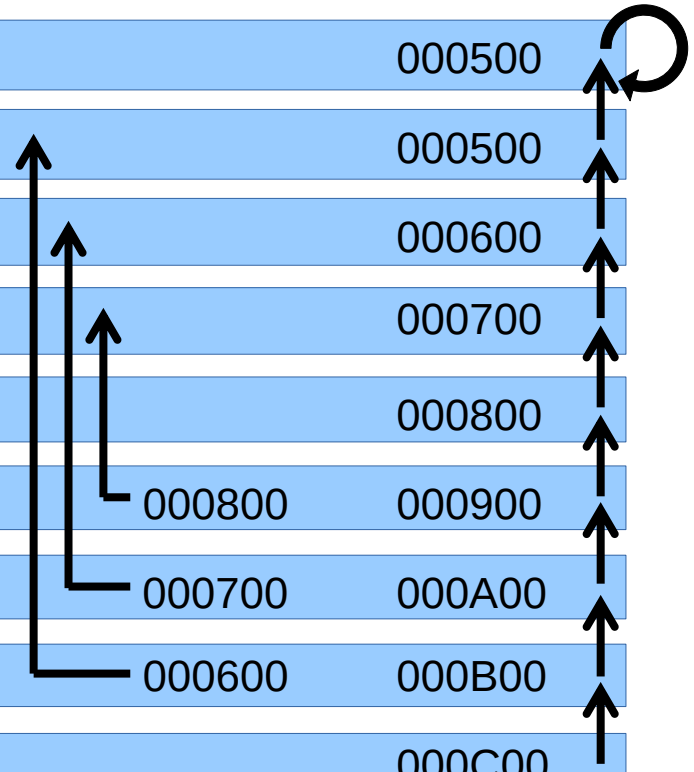
Transaction / ROLLBACK

RBA	Type	URID	Compens.	Comp.RBA	Undo Next
000500	BEGIN UR	000500			000500
000600	INSERT	000500	N		000500
000700	INSERT	000500	N		000600
000800	UPDATE	000500	N		000700
000900	BEGIN ABORT	000500			000800
000A00	UPDATE	000500	Y	000800	000900
000B00	DELETE	000500	Y	000700	000A00
000C00	DELETE	000500	Y	000600	000B00



Transaction / ROLLBACK

RBA	Type	URID	Compens.	Comp.RBA	Undo Next
000500	BEGIN UR	000500			000500
000600	INSERT	000500	N		000500
000700	INSERT	000500	N		000600
000800	UPDATE	000500	N		000700
000900	BEGIN ABORT	000500			000800
000A00	UPDATE	000500	Y	000800	000900
000B00	DELETE	000500	Y	000700	000A00
000C00	DELETE	000500	Y	000600	000B00
000D00	END ABORT	000500			000C00



- This is the result

Applying the log

- When DB2 applies log records (for example, when running RECOVER to do a point-in-time recovery), it will:
  - Start at a "baseline" point in time, such as a full copy
  - Identify the latest checkpoint before the baseline
  - Apply log records in forward direction using "REDO" information.
  - Including records from aborted transactions (both the regular and the compensation records) – also using the "REDO" information
  - Keep track of when transactions open and close
  - Use information from checkpoint records to learn about transactions that may be idle, but still open
  - After reaching the target PIT, undo changes from all records that belong to transactions that are still open, using "UNDO" information

Indexes

- DB2 also writes log records for all indexes (including COPY NO indexes)
- Index log records describe:
  - Addition / Deletion of Keys
  - Addition / Deletion of RIDs
  - Index structure changes (e.g., page splits)
  - And more

LOBs

- LOBs with LOG YES
  - Log records for space map changes
  - Log records for data
- LOBs with LOG NO
  - Only log records for space map changes
- LOB updates are never in-place
  - Therefore, DB2 can always rollback a transaction, even if the LOB is LOG NO

Table spaces with LOG NO

- DB2 does not write any log records about data changes
- Improves performance
- No ROLLBACK possible
  - ROLLBACK results in RECP state
  - Programs may cancel a transaction when a SQL error occurs: Also results in RECP state
  - Need to recover to an image copy

Checkpoint records

- Written whenever a checkpoint is created. Contain information about:
  - all transactions in progress at the time of checkpoint
  - all objects that were modified by these transactions
  - and more information about the current status
- Essentially all the information about the state of all transactions, collected in one place
- Which is why DB2 looks for the last checkpoint on the log when it is restarted

- How to find out who changed something

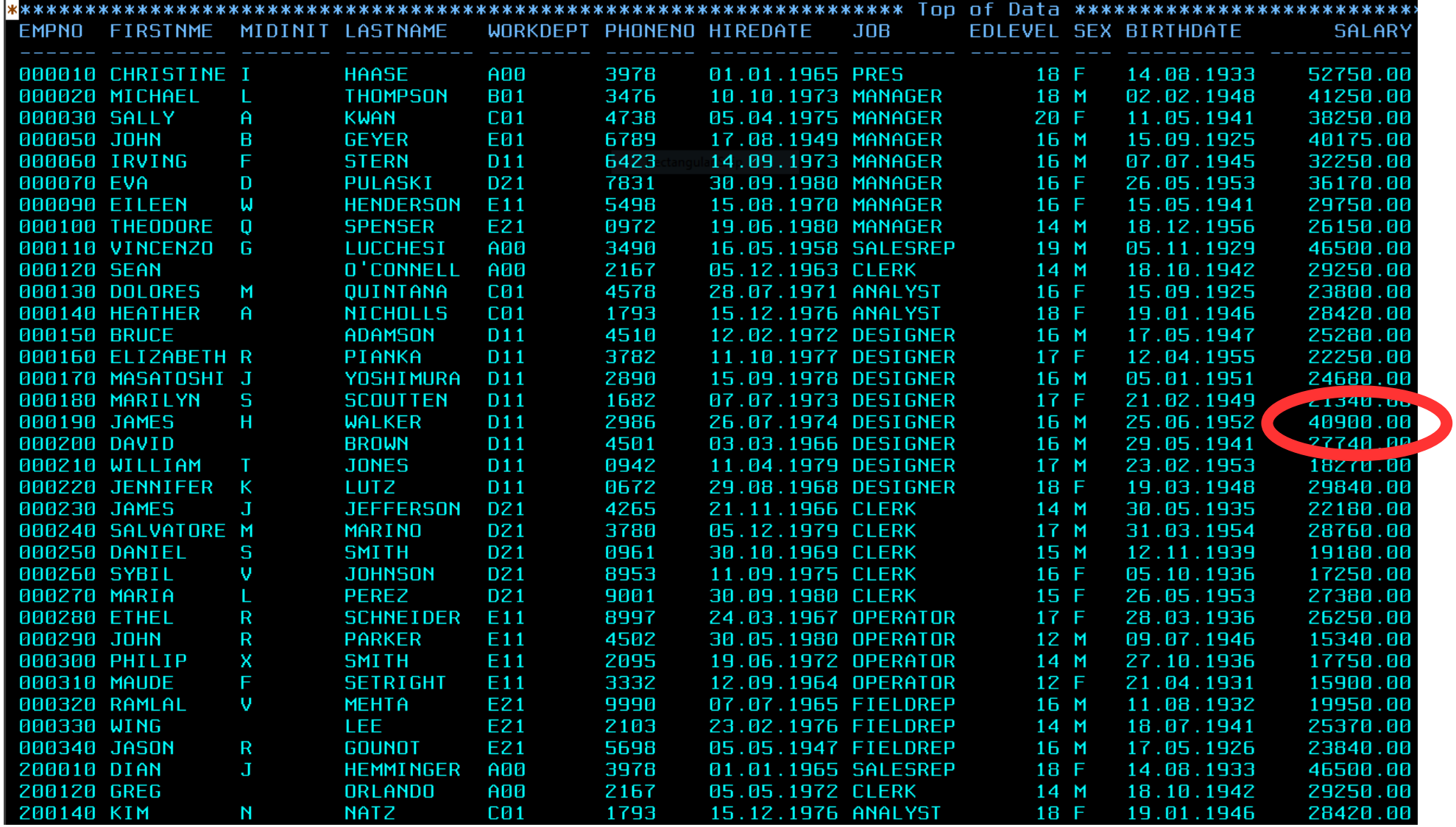
The problem

- In example table from DB2: DSN81010.EMP, the salary of one of the employees looks fishy
- DSN1LOGP cannot really filter by column contents
- Data change records do not tell us who is responsible for the change
- Everything is binary data, not human readable (EBCDIC text is readable, though)

Some assumptions

- We are looking for a row that still exists
- The table space is not compressed
- The row has not moved since the change (e.g. because of a REORG)

ULT



```
********************************************************************** Top of Data **********************************
EMPNO  FIRSTNME  MIDINIT LASTNAME   WORKDEPT PHONENO HIREDATE   JOB      EDLEVEL SEX BIRTHDATE  SALARY
------ --------- ------- ---------- -------- ------- ---------- -------- ------- --- ---------- -----------
000010 CHRISTINE I       HAASE      A00      3978    01.01.1965 PRES          18 F   14.08.1933    52750.00
000020 MICHAEL   L       THOMPSON   B01      3476    10.10.1973 MANAGER       18 M   02.02.1948    41250.00
000030 SALLY     A       KWAN       C01      4738    05.04.1975 MANAGER       20 F   11.05.1941    38250.00
000050 JOHN      B       GEYER      E01      6789    17.08.1949 MANAGER       16 M   15.09.1925    40175.00
000060 IRVING    F       STERN      D11      6423    14.09.1973 MANAGER       16 M   07.07.1945    32250.00
000070 EVA       D       PULASKI    D21      7831    30.09.1980 MANAGER       16 F   26.05.1953    36170.00
000090 EILEEN    W       HENDERSON  E11      5498    15.08.1970 MANAGER       16 F   15.05.1941    29750.00
000100 THEODORE  Q       SPENSER    E21      0972    19.06.1980 MANAGER       14 M   18.12.1956    26150.00
000110 VINCENZO  G       LUCCHESI   A00      3490    16.05.1958 SALESREP      19 M   05.11.1929    46500.00
000120 SEAN              O'CONNELL  A00      2167    05.12.1963 CLERK         14 M   18.10.1942    29250.00
000130 DOLORES   M       QUINTANA   C01      4578    28.07.1971 ANALYST       16 F   15.09.1925    23800.00
000140 HEATHER   A       NICHOLLS   C01      1793    15.12.1976 ANALYST       18 F   19.01.1946    28420.00
000150 BRUCE             ADAMSON    D11      4510    12.02.1972 DESIGNER      16 M   17.05.1947    25280.00
000160 ELIZABETH R       PIANKA     D11      3782    11.10.1977 DESIGNER      17 F   12.04.1955    22250.00
000170 MASATOSHI J       YOSHIMURA  D11      2890    15.09.1978 DESIGNER      16 M   05.01.1951    24680.00
000180 MARILYN   S       SCOUTTEN   D11      1682    07.07.1973 DESIGNER      17 F   21.02.1949    21340.00
000190 JAMES     H       WALKER     D11      2986    26.07.1974 DESIGNER      16 M   25.06.1952    40900.00
000200 DAVID             BROWN      D11      4501    03.03.1966 DESIGNER      16 M   29.05.1941    27740.00
000210 WILLIAM   T       JONES      D11      0942    11.04.1979 DESIGNER      17 M   23.02.1953    18270.00
000220 JENNIFER  K       LUTZ       D11      0672    29.08.1968 DESIGNER      18 F   19.03.1948    29840.00
000230 JAMES     J       JEFFERSON  D21      4265    21.11.1966 CLERK         14 M   30.05.1935    22180.00
000240 SALVATORE M       MARINO     D21      3780    05.12.1979 CLERK         17 M   31.03.1954    28760.00
000250 DANIEL    S       SMITH      D21      0961    30.10.1969 CLERK         15 M   12.11.1939    19180.00
000260 SYBIL     V       JOHNSON    D21      8953    11.09.1975 CLERK         16 F   05.10.1936    17250.00
000270 MARIA     L       PEREZ      D21      9001    30.09.1980 CLERK         15 F   26.05.1953    27380.00
000280 ETHEL     R       SCHNEIDER  E11      8997    24.03.1967 OPERATOR      17 F   28.03.1936    26250.00
000290 JOHN      R       PARKER     E11      4502    30.05.1980 OPERATOR      12 M   09.07.1946    15340.00
000300 PHILIP    X       SMITH      E11      2095    19.06.1972 OPERATOR      14 M   27.10.1936    17750.00
000310 MAUDE     F       SETRIGHT   E11      3332    12.09.1964 OPERATOR      12 F   21.04.1931    15900.00
000320 RAMLAL    V       MEHTA      E21      9990    07.07.1965 FIELDREP      16 M   11.08.1932    19950.00
000330 WING              LEE        E21      2103    23.02.1976 FIELDREP      14 M   18.07.1941    25370.00
000340 JASON     R       GOUNOT     E21      5698    05.05.1947 FIELDREP      16 M   17.05.1926    23840.00
200010 DIAN      J       HEMMINGER  A00      3978    01.01.1965 SALESREP      18 F   14.08.1933    46500.00
200120 GREG              ORLANDO    A00      2167    05.05.1972 CLERK         14 M   18.10.1942    29250.00
200140 KIM       N       NATZ       C01      1793    15.12.1976 ANALYST       18 F   19.01.1946    28420.00
```

- Employee James Walker is making twice as much as the average designer

Step 1

- Determine internal IDs of the affected object
  - In our case: DBID 0x0106, PSID 0x0004
- Determine which row is affected
  - SELECT HEX(RID(DSN81010.EMP))
    FROM DSN81010.EMP
    WHERE EMPNO = '000190'
  - Result: 000000**00000000**2**11**
  - Red = page number 0x00000002, blue = slot number 0x11
- Determine approximate time of change (smaller time frame = better)

Step 2

- Run DSN1LOGP

```
//DSN1LOGP EXEC PGM=DSN1LOGP
//STEPLIB  DD DISP=SHR,DSN=DSNA10.SDSNLOAD
//BSDS     DD DISP=SHR,DSN=DSNA10.DBAG.BSDS01
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//SYSIN    DD *
 RBASTART (00ECD81ECF8F)
 RBAEND   (00ECD81FAD0B)
 DATAONLY (YES)
 DBID     (0106)
 OBID     (0004)
 RID      (0000000211)
 SUBTYPE  (1)
/*
```

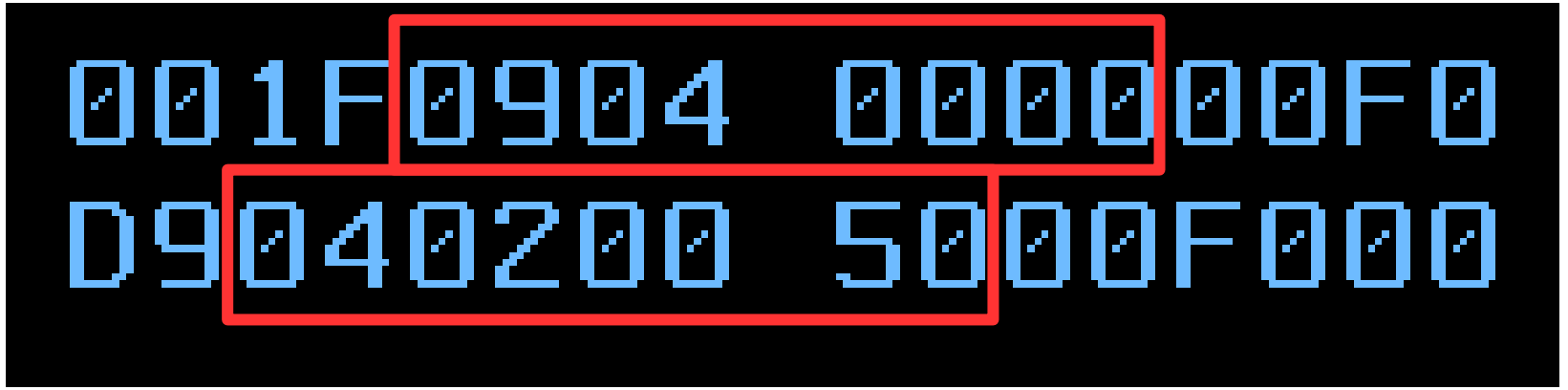
Subtype 1 means Undo/Redo records, basic data page change

ULT

```
=======================================SEARCH CRITERIA=======================================
  STARTRBA(00ECD81ECF8F)  ENDRBA(00ECD81FAD0B)
  SUMMARY(NO)
  DATAONLY(YES)
  SYSCOPY(NO)
  ALL URIDS     === YOU MAY SPECIFY URID(XXXXXXXXXXXX)
  ALL LUWIDS    === YOU MAY SPECIFY LUWID(NNNNNNNN.LLLLLLLL.XXXXXXXXXXXX.XXXX)
  DBID(0106)    THE FOLLOWING OBIDS SPECIFIED:    0004
  THE FOLLOWING PAGES SPECIFIED:    00000002
  ALL TYPES     === YOU MAY SPECIFY TYPE(XX)
  SUBTYPE(0001)

  THE FOLLOWING OFFSETS, VALUES SPECIFIED:
  OFFSET(0000003B)
  VALUE(00000000000000000000000000000000000000000000000000000011)

=============================================================================================

 DSN1212I DSN1LGRD FIRST LOG  RBA ENCOUNTERED 00ECD81ECF8F

 00ECD81FABB5  TYPE( UNDO  REDO )  URID(00ECD81FAAB9)
               LRSN(D12B4FAD76E7)  DB
               SUBTYPE(UPDATE NOT IN-PLACE , DATA PART ONLY IN A DATA PAGE)
               CLR(NO)  PROCNAME(DSNIREPR)

      *LRH* 0084003D 06000001 0E8000EC D81FAAB9 00ECD81F AB780726 00ECD81F AB78D12B  * d          Q     Q       Q   J
            4FAD76E7 0000                                                          *|  X
      *LG** 80010600 04000000 02000000 00000000 2D00                               *
      0000  004C6111 00128218 0030001F 001F0904 000000F0 00040000 00F00016 3600003E  *
      0020  0043D1C1 D4C5E2E6 C1D3D2C5 D9040200 5000F000 04000000 F0001636 00003E00    JAMESWALKER   & 0     0
      0040  43D1C1D4 C5E2E6C1 D3D2C5D9                                               * JAMESWALKER

 DSN1213I DSN1LGRD LAST LOG  RBA ENCOUNTERED 00ECD81FAD0B
```

- DSN1LOGP converts the LRSN into a timestamp
- The EBCDIC text confirms it is the correct row
- The URID tells us in which transaction the change was made

UBS HAINER



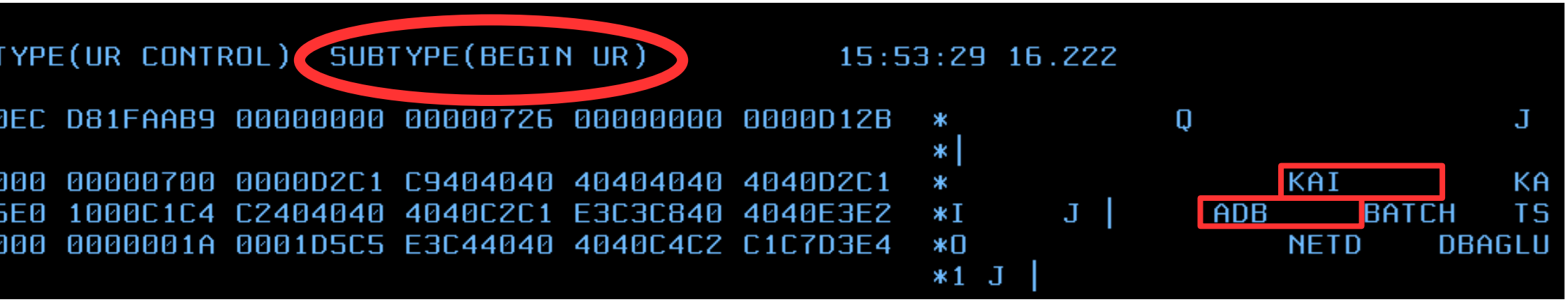
- A field was changed from 20450.00 to 40900.00
- Standard representation would be oxF002045000 and oxF004090000
- DB2 starts logging after …Fo because everything up to and including this byte has not changed
- Also, this column has an editproc…

UBS HAINER

Step 3

- Run DSN1LOGP again

```
//DSN1LOGP EXEC PGM=DSN1LOGP
//STEPLIB  DD DISP=SHR,DSN=DSNA10.SDSNLOAD
//BSDS     DD DISP=SHR,DSN=DSNA10.DBAG.BSDS01
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//SYSIN    DD *
 RBASTART (00ECD81ECF8F)
 RBAEND   (00ECD81FAD0B)
 DATAONLY (YES)
 URID     (00ECD81FAAB9) <-------- Limit output to the transaction we have identified
/*
```



- The BEGIN UR record for this URID shows, among other things, the user name and the plan name

Revisit our assumptions

- We are looking for a row that still exists
  - So we can use the RID function
  - If the row has been deleted: no practical way to determine old row position

Revisit our assumptions

- The table space is not compressed
  - Log records contain raw binary data, therefore they are also compressed / encrypted
  - Compression: Need to decompress row using the correct decompression dictionary
  - Decompression dictionary from VSAM may not be the correct one

Revisit our assumptions

- The row has not moved since the change (e.g. because of a REORG)
  - If it has moved, the RID function will return the new position
  - But the log record refers to the old position

Thank you for your attention

Contact us:
info@ubs-hainersoftware.com