TRACK VI: DESIGN
Session 640

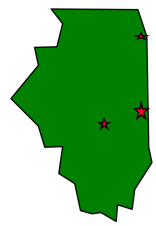
An Audit Trail for Your DB2 System

Patricia A. Tuchman
University of Illinois - AISS

Hi. My name is Pat Tuchman. I would like to describe a general purpose audit trail system that we've implemented at the University of Illinois. This system audits changes to DB2 data, regardless of how the changes were initiated. I am the primary analyst and programmer for this system.

Where is U of I?

- Chicago
- Champaign-Urbana
- Springfield
- NOT the University of Chicago



The University of Illinois is a state funded University. It once again has three main campuses. At the UIUC campus, we have approximately 35,000 students and about the same number of faculty and support staff. The Chicago campus has both the medical school (UIMC) and the old Circle (UICC). Springfield was added to the University last summer and is much smaller.

Who is AISS?

- Administrative Information Systems and Services
- Administrative computing for the University of Illinois
- Staff both in Chicago and in Champaign-Urbana
- NOT a teaching department

AISS is a general University department - we don't belong to a specific campus. We are responsible for administrative computing including the traditional business applications of payroll and human resources as well as university specific applications such as student records. AISS is not a teaching department.

AISS has been using IEF/Composer since 1987.

Topics

- Overview
  - Data flow
  - Database design
- Specific Example
- Specifics
  - Problems encountered
  - User reactions
  - System statistics
  - Good things to know

In my talk, I'd like to first give an overview of our audit trail system, then show a specific example of data being audited, and follow that with some detailed technical information. I hope that all of this is on your CD-ROM, along with reports showing the details of the data model we used.

Why did we do this project?

- The University MUST be able to account for all changes made to a student’s record by authorized staff
- The University SHOULD be able to explain how a student’s record got that way
- The University MUST be able to detect unauthorized changes to a student’s record

Over the past several years, we re-engineered one of our existing student systems. We had a 10+year old IMS system that needed work. The UIUC campus also wanted to provide students with the ability to register for classes directly, without going through the cycle of filling out forms and waiting to see what classes were already filled. The soon-to-be obsolete system had a rudimentary audit capability whose function needed to be carried over into the new system. In the old system, selected online programs would write before and after images to an audit database. This provided only limited audit capability and was very inflexible. None of the batch programs did any auditing. On the positive side, very few people were allowed to update the data, so the audits were sufficient.

When we switched to the new student system, not only could many more people update records (including the 35,000 students themselves!), but we also had the QMF table editor and SPUFI which don’t lend themselves to self-auditing. We knew we needed a different approach.

We are currently using the audit system for student records; we have several other applications that may use this in the future.

What we're NOT using this for

- Problems occurring right then
- Paper trail of all changes done

The resulting system is not used for resolving problems that just happened. There is a time delay before the data shows up in the database. Also, we are not producing a paper trail of all the changes that happened any where in the DB2 region.

Vocabulary

- Audit Trail - a record of (all) changes made to key data
- Log Analyzer - utility from Platinum Technology
- BSDS - DB2's Boot Strap Data Set
- Unit of Recovery - changes done between commits

Some basic vocabulary...

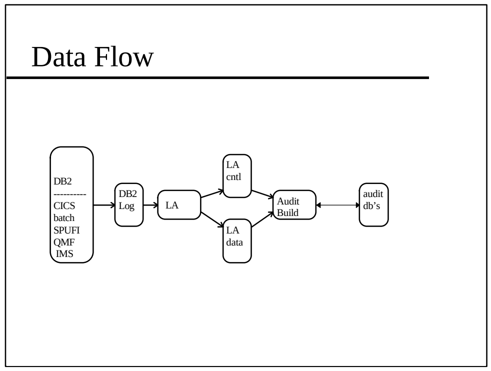
The Boot Strap Data Set holds the index to the DB2 log information. The data here can direct you to the exact DB2 log for either a date/time range or an RBA range.

Unit of Recovery/commit: For an online transaction, this is usually the interval between enter and/or function keys being hit. For a batch transaction, this is the interval between DBCOMMIT commands. In our audit system, we group together all changes done in a unit of recovery.

Design Requirements

- Minimal changes to existing applications
  We really made none
- Easy to report from
- Easy to control what data is being audited
- Capture changes made anywhere
  Not just by IEF generated applications

Here are our design requirements. We decided that we needed to audit from the DB2 logs directly. When we started the re-engineering process, there were no products available to help with this. By the time we started working on this part of the system, Platinum Technology (PT) had their Log Analyzer (LA) product in late beta; we decided to go with this.



DB2 writes all changes to its log, regardless of how the changes were made. We then run PT's LA over the log, culling out changes to tables that we are interested in. LA produces two files; the control file describes the layout of the tables that have been found in the log; the data file has the changed data. (I have samples of these coming up.) These are input to our build/populate program which parses the control file and uses this information to decipher the data file. The interesting changes are written to the audit database.

There is a second batch job which purges data from the audit database when we no longer need it.

We keep the LA control and data information for 2 years. This makes the auditors happy.

Data is Filtered Out

- DB2 table definition must be DATA CAPTURE ON
- LA job must have table name in list
- Audit definition must include table
- Audit definition must include column
- Audit definition must not exclude plan

We don't record all changes made within the DB2 region. The changed data is filtered in these 5 ways.

Data capture on - if this isn't set for a table, then the log doesn't have the entire row that changed. We need the entire row to get the identifiers and foreign keys so we know what changed. After all, it's no good knowing that a grade changed from “D” to “A” if we can't derive for what student and for what course.

LA table list - the LA job has a list of tables that it will extract changes for. If we're not interested in auditing a table, we don't include it on this list.

Audit definition - we define within the audit system what tables we're interested in.

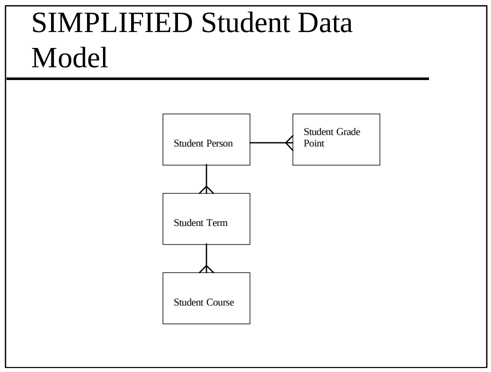
Audit definition - we further define what columns of what tables we're interested in. For example, we rarely audit create_date or update_time. We also only record columns that have been changed.

Audit exclusion - we can ignore changes to a table based on the plan that did the change.

Databases Used

- Audit
- Audit definition
- Audit control
- Audit report
- “Live”

The audit system has four databases in it. The first three on the list are operational; the fourth is pending. We also do reads of the “live” (non-audit) data to resolve identifying information (remember the grade change!) I’ll describe the first three audit databases in more detail.



But first, let me give you a little background in the data model design philosophy we use. In all but a few of our entity types, we have a generic "identifier" that is the unique identifier for the entity type. We do not include relationships in unique identifiers.

So, in this very small portion of the student data model, we have 4 entity types.

SP has the basic information about a student. This is related to the enterprise-wide PERSON entity type, which isn't shown.

ST is the term based information for a student.

SC is the actual course that a student is or was enrolled in.

SGP is where grade point information is stored. We don't actually store a student's grade point average; we save the number of earned hours and the number of grade points and do the calculation when the GPA is needed. The campus has defined a number of different GPA's, so this seemed a better approach.

These entity types will be used in the rest of my examples.

Note to the reader: I never type table names; I use acronyms instead.

Student Data Columns

Student Person
- Identifier
- DN_Personal_ID
- DN_Name
- Sex
- FK_person identifie

Student Term
- Identifier
- Year/term
- Term hours graded
- FK_student_perside

Student Course
- Identifier
- Rubric
- Number
- Title
- Grade
- FK_student_termide

Student Grade Point
- Identifier
- Graded Credit
- Graded Points
- Credit Type
- Detail Type
- FK_student_perside

Just to show you what can be found in the tables.

In every table, there is a column named “IDENTIFIER”. This is a text(16) field that contains a timestamp that has been shuffled.

This makes reading a specific row very easy and keeps the length of the key short and the length of the foreign key short. Users can change any column they want, without us needing to worry about not changing part of a unique key. However, it means that there is no intrinsic identifying information to be found in a row. Again, just because you have read a row in student_course, you don’t know who took it and when. You must do additional reads for this information, if you need it.

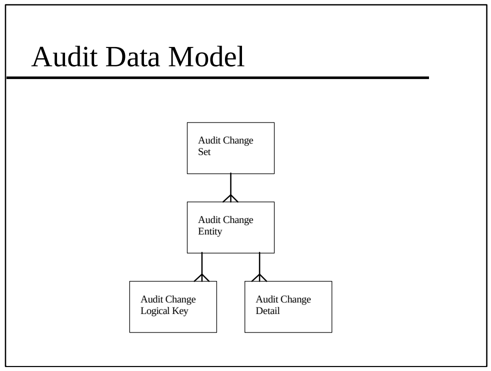
Audit Database

- Where the changed data goes
- HUGE!!
- Tried to keep design simple for reporting purposes
- All before and after images are varying character data

Now, back to the audit system databases.

The main audit database is conceptually very simple. This is where the changes are recorded. We tried to keep it simple so that reporting would be easy and could be done by anyone using any tool - and not limited to IEF/Composer.

There is a LOT of data here. I'll show some statistics later.



ACS - there will be one of these for each unit of recovery - it has basic information about the associated changes - who did them, with what plan, and when. For an online update, this is usually not very much information. For a batch update, this can be associated with a LOT of changed data, depending on the commit interval for the program.

ACE - describes what table was changed; this records table name, table creator, entity identifier, and type of change. The entity identifier is the generic identifier that describes the exact row within the table that changed.

ACD - is for each audited, changed column of the table; this shows the before and after images which have been converted to varchar. Defaults based on data type are used for the before image for inserts and the after image for deletes.

ACLK - each row of a table that has a changed column or columns needs to be identified for the user - to have logical keys; this data describes completely what changed, which is different from what the changes were. Since our physical "key" (identifier) doesn't have any inherent meaning, we need to associate the changes with other data that will describe what changed, e.g. the student's ID number, name, term, course name. A person verifying a change would be helpless if they were presented with one of our identifier's to match against the paper trail that validated the changed data. We refer to this set of information as "the logical keys" or "the identifying information".

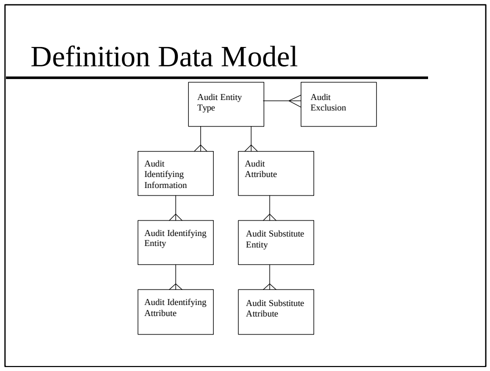
Definition Database

- Maintained by programming staff
- Date sensitive
- Defines what columns (attributes) of which tables (entity types) will be audited
- Defines how to fully identify a change (logical key)
- Substitution not being used

The definition database defines what columns of what tables we’re actually interested in. This is maintained by the programmers, since the DB2 names are used and must match exactly what is in the DB2 catalog. Everything in this database is date sensitive, so we can have different data being audited depending on the time of year. It turns out we’re not using the date feature very much. We don’t audit based on type of change (insert, update, delete).

We define here the identifying information - this is used to build the logical keys in the audit database.

We also allow for the definition of a substitute for a column. We thought originally that instead of recording that a coded field changed, we might want to record the decoded values. We’re not using this either.



AET - DB2 table creator and name.

AA - DB2 column name, Y/N audit indicator; these first two tables define what changed fields will be in the audit database.

ASE - DB2 table creator and name for where to find the substitute value(s).

ASA - DB2 column name(s) for the substitute

AII - start and stop date for identifying information; this allows for the identifying information to change over time. We actually haven't needed to do so.

AIE - DB2 table creator and name for where to find a part of the identifying information.

AIA - DB2 column name for a part of the identifying information; also known as the logical key. This table and the preceding table define the logical key pieces that will be built for a particular table.

AE - what plan or plan mask to exclude changes for.

Audit Entity Type Example

```
AU05              List Audit Entity Types          02-13-96 11:26:04

Start at Table Creator AEZCASE  Table Name TIMETABLE_II_OPEN
Select Table Creator Table Name                    Start Date   Stop Date
       AEZCASE      STUDENT_COURSE
       AEZCASE      STUDENT_COURSE_XM
       AEZCASE      STUDENT_CRSE_X_LOC
       AEZCASE      STUDENT_CURRICULUM
       AEZCASE      STUDENT_DEGREE
       AEZCASE      STUDENT_DEGREE_CUR
       AEZCASE      STUDENT_DEGREE_HON
       AEZCASE      STUDENT_DISTINCTIO
       AEZCASE      STUDENT_ENCUMBRANC
       AEZCASE      STUDENT_EXTERNAL_D
       AEZCASE      STUDENT_FERPA
       AEZCASE      STUDENT_GAS
       AEZCASE      STUDENT_GDOL
       AEZCASE      STUDENT_GRAD_DATA
       AEZCASE      STUDENT_GRAD_EXAM
       AEZCASE      STUDENT_GRADE_POIN
       AEZCASE      STUDENT_HS_SUBJECT
```

This shows a partial list of the tables that are defined within the audit definition database. A table must be on this list for any changes to a row from the table to be audited.

Audit Attribute Example

AU04 List Attributes for Audit Entity Type 02-13-96 11:29:35

Table Creator AEZCASE Table Name STUDENT_GRADE_POIN
Start Date 00000000 Stop Date 00000000

Select	Column Name	Start Date	Stop Date	Audit?	Substitute?
	CREDIT_TYPE	00000000	00000000	Y	N
	DETAIL_TYPE	00000000	00000000	Y	Y
	EARNED_CREDIT	00000000	00000000	Y	N
	FK_STUDENT_PERSIDE	00000000	00000000	N	N
	GRADED_CREDIT	00000000	00000000	Y	N
	GRADED_POINTS	00000000	00000000	Y	N
	IDENTIFIER	00000000	00000000	Y	N
	NOMINAL CREDIT	00000000	00000000	Y	N

For one of the defined tables, here is an example showing the defined columns. You can see that we are not auditing changes to the foreign key; this will only change on inserts and deletes since this is not a transferrable relationship. I’ve included it on the list so that I remember that we’ve made a deliberate decision to NOT audit the field.

We are also doing a substitution for one of the coded fields.

Again, if a field is not in the definition with an audit flag = Y, changes to that field will not be audited.

Audit Substitution Example

```
AU19              List Substitutes for Audit Attribute          02-13-96 11:41:53

Table Creator AEZCASE  Table Name STUDENT_GRADE_POIN 00000000 00000000
Column Name   DETAIL_TYPE         00000000 00000000 Audit? Y Substitute? Y
Select Table Creator Table Name          Column Name
         AEZCASE      TABLE_VALUE        LONG_VALUE
         AEZCASE      TABLE_NAME         DESCRIPTION
```

This is a fairly simple substitution.... for the encoded field, go to the code tables and look up the decoding.

Audit Logical Key Example

AU12 List Entities/Attributes for Identifying Information 02-01-96 12:18:26

Table Creator AEZCASE Table Name STUDENT_COURSE
Identifying Information Start Date 00000000 Stop Date 00000000

Order	Table Creator	Table Name	Attribute Name
1	AEZCASE	STUDENT_PERSON	DN_PERSONAL_ID
2	AEZCASE	STUDENT_PERSON	DN_CURRENT_FULL_NA
3	AEZCASE	STUDENT_COURSE	YEAR_TERM
4	AEZCASE	TERM	TYPE
5	AEZCASE	STUDENT_COURSE	RUBRIC
6	AEZCASE	STUDENT_COURSE	NUMBER_PREFIX
7	AEZCASE	STUDENT_COURSE	NUMBER
8	AEZCASE	STUDENT_COURSE	SECTION_ID

This shows the actual definition of the logical key (identifying information) for one of the tables. In order to fully identify a row in student_course, you need to know the student's ID number, the student's name, the year/term (e.g. FA96), the term type (e.g. traditional, extramural, or correspondence), the course rubric (e.g. MATH), the course number (e.g. 120), the course number prefix, and the section. Some of this information comes from student_course itself. Since we only record the CHANGED columns in the audit_change_detail table, we need to have this data in the key. After all, the course rubric rarely changes; the course grade often does.

We actually don't need both the student ID and name, but having both as part of the logical key makes reporting easier.

Audit Exclusion Screen

AU17 List Audit Exclusions 02-01-96 12:10:2

Table Creator AEZCASE Table Name STUDENT_GRADE_POIN

Select	Plan Name	Start Date	Stop Date	Include Or Exclude
	AREGADD2	00000000	00000000	E
	R*	00000000	00000000	E
	REGAT04T	00000000	00000000	I
	REGAT04U	00000000	00000000	I
	REGAT04V	00000000	00000000	I
	REGAT04W	00000000	00000000	I

Here's an example of excluding data changes based on the plan. GP is the Grade Point data. It has the numbers we need to compute a grade point average. Since lots of things go into a grade point average, lots of changes all over the system can cause "invisible" changes to the GPA. The users have said they don't care about these indirect updates. We exclude GP changes done by AREGADD2 (a batch update) and R* (all online student & staff plans). However, we can change a student's GP numbers directly on 4 screens. We want any changes done on those 4 screens to be audited. So, we've explicitly included those 4 plans. Implicit includes override excludes. During the end of semester mass grade update, EVERYONE's grade point information will be updated; noone wants to look at that volume of information. Another reason for the excludes is that a lot of the batch updates have validated input data that is saved. If there is a suspected problem, checking the input might be faster.

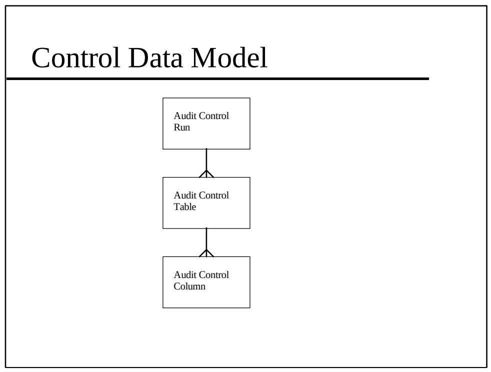
This has cut down on a vast amount of data that noone would ever want to see anyway.

Note: the excluded data is still being extracted by LA and is on the LA data tapes. If we wanted, we could go look at the tapes for a particular change. We can also run the tape in at a later time, having changed the exclusion to now allow this audit.

Control Database

- Used while adding data to audit database
- Information from LA control data
- Not saved beyond build job

The control database is where the LA control file is parsed into just before the LA data is read and deciphered. We don't save this beyond the life of the job.



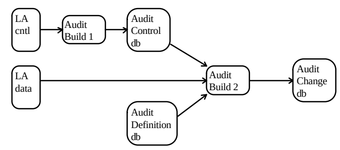
ACR - Contains information that is constant for all records within the run. This includes the start position and length of: plan name, update date, update time, etc.

ACT - Has information for each requested DB2 table that was found in the log; contains DB2 creator and table name.

ACC - For each column within a table, records the start position, length, update flag position, column domain, and decimal position.

We rebuild this information for every LA extract run. Since table definitions change over time, this seems the safest. Also, if a requested table has no updates for a execution of the LA, LA doesn't include that table's control information in the control file.

Putting the Pieces Together



We have our two datasets that LA created for us. The first batch procedure step of the build job reads the control data, parses it, and creates a DB2 version of that information in the Audit Control database.

The second batch job step reads the LA data dataset. For each record here, the table name is extracted, the audit control information is found for that table, and then the rest of the record is broken into individual fields based on the information in audit_control_column. For each field that has been flagged as updated, the audit definition database is read to see if this is an update to record. If so, the data is converted to character, the before and after images are created, the logical key is built, and all of this is added to the audit database.

Log Analyzer Control Raw Data

```
INTO TABLE AEZCASE.STUDENT_GRADE_POIN
WHEN (1:26) = 'AEZCASE.STUDENT_GRADE_POIN’
(PLA_STMT_TYPE      POSITION(  27:  28)       CHAR(  2)
,PLA_URID           POSITION(  29:  34)       CHAR(  6)
,PLA_UPDT_DATE      POSITION(  35:  44)       DATE EXTERNAL(10)
,PLA_UPDT_TIME      POSITION(  45:  52)       TIME EXTERNAL(8)
,PLA_PLAN           POSITION(  53:  60)       CHAR(  8)
,PLA_AUTHID         POSITION(  61:  68)       CHAR(  8)
,PLA_CORRID         POSITION(  69:  76)       CHAR(  8)
,PLA_CONNID         POSITION(  77:  84)       CHAR(  8)
,PLA_CONN_TYPE      POSITION(  85:  85)       CHAR(  1)
,PLA_STATUS         POSITION(  86 :  86)      CHAR(  1)
,PLA_LOGRBA         POSITION(  87:  92)       CHAR(  6)
,PLA_UPDT_001       POSITION(  93:  93)       CHAR(  1)
,IDENTIFIER         POSITION(  94: 109)       CHAR( 16)
,PLA_UPDT_002       POSITION( 110: 110)       CHAR(  1)
,CREDIT_TYPE        POSITION( 112: 112)       CHAR(  1)
                    NULLIF( 111) = ‘?’
,PLA_UPDT_003       POSITION( 113: 113)       CHAR(  1)
,NOMINAL_CREDIT     POSITION( 115: 119)       DECIMAL
                    NULLIF( 114) = ‘?’
```

For one of our audited tables, here’s part of the information in the control file. Columns beginning “PLA” are LA control fields. The fields are shown in file position order.

For instance, for this run of the LA, the field named nominal credit will be found in positions 115 - 119 of the output record. It is a decimal field. If the field contains a null, this will be signalled by a ‘?’ in column 114. If this field changed, this will be signalled by a ‘U’ in column 113.

Log Analyzer Control Parsed Data

AU01 Audit Control Run Display/Delete 01-22-96 16:29:33

Run Date 01-13-96 Run Time 11:29:55

	Stmt Type	UR ID	Date	Time	Plan	Auth ID	Corr ID	Conn ID	Status
Start Pos:	027	029	035	045	053	061	069	077	086
Length:	002	006	010	008	008	008	008	008	001

Table Creator AEZCASE Name STUDENT_GRADE_POIN

Column	Type	Start Pos	Length	Update Pos	Decimal Pos
CREDIT_TYPE	CHAR	112	001	110	
DETAIL_TYPE	CHAR	161	002	159	
EARNED_CREDIT	DECIMAL	122	005	120	005
FK_STUDENT_PERSIDE	CHAR	143	016	141	
GRADED_CREDIT	DECIMAL	129	005	127	005
GRADED_POINTS	DECIMAL	136	005	134	005
IDENTIFIER	CHAR	094	016	093	
NOMINAL_CREDIT	DECIMAL	115	005	113	005
PLA_CONN_TYPE	CHAR	085	001		
PLA_LOGRBA	CHAR	087	006		

Here’s the same information after it’s been parsed and inserted into the audit control database. The columns are sorted into ascending order by name.

The top part of the screen shows the fields that are in all the LA records: for example, we see that the plan name begins in column 53 and is of length 8.

The bottom part of the screen shows the fields that are specific to the table being displayed. We see here that nominal_credit is a decimal field, with a starting position of 115, a length of 5 in the record, and that the resulting number should be interpreted as having 5 decimal positions.

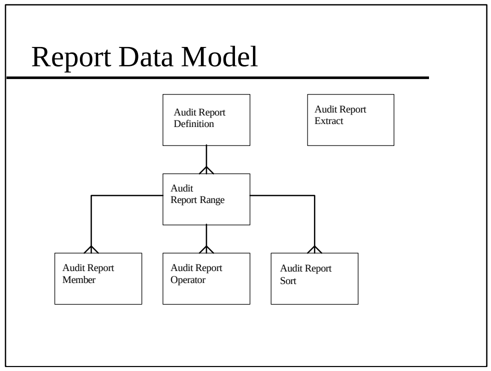
Given some clever substringing, we now have enough information to decipher the LA data!

Proposed Report Database

- Used to define reports
- Can specify updates not only by table but by plan
- Generic output format
- Can specify
  - Data to include
  - Sort order
  - Page breaks

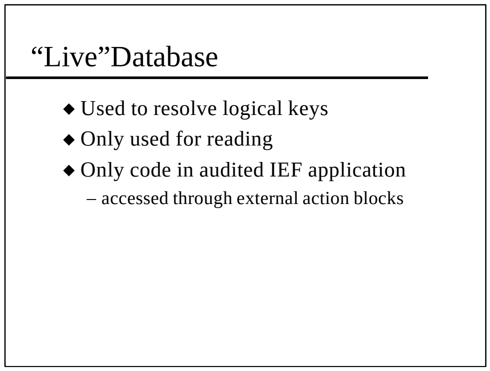
As an add-on to the system, we are currently (this is written in February) designing a report database. This will give the users complete control over what is on a report. It turns out that they want to see updates to a column done only on certain screens by certain users for some of their auditing. We continue to put all of the changes in the database and will only report on some of them.

We haven't done anything beyond the data model work on this subsystem yet.



The preliminary design for the report database allows the user to define multiple reports. Each report definition is valid for a range of dates. For each range, the user will be able to specify the exact fields for each update screen to include on the final report, the operator id's to include or exclude, and a sort order. This information will be used to read from the audit trail database and information will be put into the report extract table. The actual report will be written from the extract table.

This approach will allow us to optimize the reads of the data in the main database without regard to the final appearance of data on the report.



Since we need to build the logical keys, we will not have all of the necessary information in the audit database itself or in the input LA data. We have written a series of reads against the "live" (non-audit) database to retrieve that information. For example, if we need a student_course logical key, we will read for student_term and student_person. Since the audit system is in its own model, the information is passed by way of external action blocks. We have two external action blocks: one accepts a table name and identifier; it returns identifiers of interesting related tables. The second accepts a table name, identifier, and column name. It reads that row in that table and returns the value in that column.

This was the only code written in the non-audit model that had anything to do with auditing. If the system being audited was not written with Composer, we would be able to access the information through external action blocks also.

If we had fully concatenated logical keys (as we did in the IMS system), we wouldn't have needed to access the live data at all.

Example - Retrieve Identifiers From “Live” Database

Table name: STUDENT_COURSE   Creator: AEZCASE Id: 9999999947454209

Creator	Table Name	Identifier	Length
AEZCASE	STUDENT_TERM	9999999996901471	016
AEZCASE	TERM	0195993190539890	016
AEZCASE	STUDENT_PERSON	9999999994415970	016
AEZCASE	PERSON	9999999990000000	016

This shows an example from our test screen for retrieving from the live database. For a given table creator, name, and identifier, it returns the identifiers of all of the associated tables that would help identify this row in this table.

Example - Retrieve Data Value from “Live” Database

```
RG6M                    Audit Get Attribute Value            03-13-96 12:28:09

 Table name: STUDENT_TERM        Attribute name:    YEAR_TERM
 Creator:    AEZCASE              Entity Identifier: 9999999996901471


Attribute value:
  96SP
```

This shows taking one of the identifiers found on the previous test screen and requesting a specific data value to be returned.

Example - Change a Student's Grade

- show update screen
- show entity/attributes changed
- show logical keys

Now, I'd like to show an example that will (hopefully) tie all of this together. I'll change a student's grade and follow the data flow from my fingertips into the audit database.

Before Screen

```
                 Maintain Course Enrollment           02-96 13:41:13

 ID: 999 99 9999  Term/Year: SP96     Type: TR   Campus: 1
 Name: Tuchman Patricia Amy
 Hours Earned:          Graded:        Points:          Nom:   13.00  Gpa:
 Units Earned:          Graded:        Points:          Nom:          Gpa:
   College:  32   Curriculum Code: 1423   Class Code: 4  Subspecialty:   Val: 6
 Residence: R           New Status:  0                JS:   Chanc Honors:
Enr Status: R   Registration Date:    011196   Reg Term Date:
RBM Fee Code:
                                                                      Cr/Nc
Act Call#  Rubric   Num Title                    Sect. I-Type G   Credit  Gd #  Ind   H V
    00236 E E E    105 ENVIRONMENTAL BIO    A    LECT Y   3.00 H
    00010 MATH     120 CALC & ANAL GEOM I   F    QUIZ  Y   5.00 H
    07075 SCAN     102 ELEMENTARY SCAND     D    LECD  Y   4.00 H
```

Here's the "student courses for a term" update screen, before the grade is changed. This screen shows student_person, student_grade_point, and student_term data at the top and the student_courses listed at the bottom. This student is enrolled for three classes this term. Don't worry - this isn't real data!

After Screen

```
                    Maintain Course Enrollment              02-96 13:43:02

 ID: 999 99 9999  Term/Year: SP96     Type: TR   Campus: 1
 Name: Tuchman Patricia Amy
 Hours Earned:   5.00     Graded:   5.00   Points:    25.00   Nom:   13.00  Gpa: 5.000
 Units Earned:            Graded:          Points:            Nom:          Gpa:
   College:  32    Curriculum Code: 1423   Class Code: 4  Subspecialty:    Val: 6
 Residence: R            New Status:  0                JS:   Chanc Honors:
Enr Status: R   Registration Date:    011196   Reg Term Date:
RBM Fee Code:
                                                                        Cr/Nc
Act Call#  Rubric   Num Title                       Sect. I-Type G   Credit  Gd #  Ind   H V
      00236 E E E    105 ENVIRONMENTAL BIO    A    LECT Y   3.00 H
      00010 MATH    120 CALC & ANAL GEOM I    F    QUIZ   Y   5.00 H  A
      07075 SCAN     102 ELEMENTARY SCAND     D    LECD  Y   4.00 H
```

The student’s grade for Math 120 has been changed from spaces (not yet graded) to an A. At the same time, some of the GP data has been updated automatically.

In case you’re wondering - UIUC is currently on a A=5.0 grade point scale. We’ll be switching to a 4.0 scale Labor Day weekend.

LA JCL generated

```
STRATEGY    = (DB2D,AEZZPAT,AUDIT1)
START       = (DATE(1996-02-02),TIME(13:40:00.00))
END         = (DATE(1996-02-02),TIME(13:45:00.00))
LOGSRC      = (BSDS)
OBJSRC      = (CATALOG)
DYNSORT     = (DSNUM(2),SPACE(10,10),MAINSIZE(1000))
GENUNIT     = (SYSDA)
RPTLINES    = (60)
DMLREPT     = (LEVEL      (DETAIL)
               ,ROLLBACK  (EXCLUDE)
               ,CATALOG   (EXCLUDE)
               ,ORDERBY   (URID)
               ,INCLUDE   (AND
                          ,TABLE (AEZCASE.PERSON,
                                  AEZCASE.STUDENT_ GRADE_POIN,
                                  AEZCASE.STUDENT_COURSE,
                                  AEZCASE.STUDENT_TERM,
```

I went into LA and requested all changes made to student related tables for the timeframe in question.

The production version of this job is set up to do “resume” processing. It will pick up where the previous job left off and will process 24 hours worth of data. This is great since it means we don’t need to make JCL changes constantly.

LA Data (partial)

```
STUDENT_GRADE_POINUA  @ @m1996-02-0213.43.01REGAT008AEZZPAT PT00RG08
STUDENT_GRADE_POINUB  @ @m1996-02-0213.43.01REGAT008AEZZPAT PT00RG08
STUDENT_GRADE_POIND    @ @m1996-02-0213.43.01REGAT008AEZZPAT PT00RG08
STUDENT_TERM         UA  @ @m1996-02-0213.43.01REGAT008AEZZPAT PT00RG08
STUDENT_TERM         UB  @ @m1996-02-0213.43.01REGAT008AEZZPAT PT00RG08
STUDENT_COURSE       UA  @ @m1996-02-0213.43.01REGAT008AEZZPAT PT00RG08
STUDENT_COURSE       UB   @ @m1996-02-0213.43.01REGAT008AEZZPAT PT00RG08
```

and the data continues...

```
      U 13300 U 10700 U  496    3414099141590767    A
      U 12800 U 10200 U  471    3414099141590767    A
      U 00000 U 00000 U  000 U 3414099141590767U  D
P R     0   4    S    C 1996-01-08         1996-01-11    0001-01-01   R
P R     0   4    S    C 1996-01-08         1996-01-11    0001-01-01   R
P   MATH          120    CALC & ANAL GEOM I      HU A          0001-01-01   0001-01-01
P   MATH          120    CALC & ANAL GEOM I      HU            0001-01-01   0001-01-01
```

There are 7 records pulled from the DB2 log for the grade change. We updated one student_grade_point, deleted a different student_grade_point, updated the student_term, and updated the student_course. This overhead shows the start of the records where the standard information is (table name, type of change, date, time) and then a portion of the data part of the records. Every field that is changed will have a “U” in front of it. Some of the fields shown are binary numbers which don’t display very well; I’ve retyped them to be more interesting.

We can see in the data the plan name (REGAT008), the userid (AEZZPAT), and other information. The positions of these fields can be found in the control dataset.

So, for the user making a one character change on the screen, quite a lot of background updating happened.

Audit Build Statistics

Insert log records read:	0
Delete log records read:	1
Update pair log records read:	3
Invalid log records read:	0
Uncommitted log records ignored:	0
Log records not in control db:	0
Log records not in definition db:	0
Log records excluded:	2
URID's processed:	1
Control entities created:	1
Logical key pieces created:	7
Unknown data type found:	0
Change details created:	1
Commits initiated:	1

After running the audit build (populate) program, we get these statistics. The records excluded were the student_cad_detail’s which have audit_exclusions written for them.

We created:

1 ACS

1 ACE

1 ACD

and 7 logical key pieces!

Audit Online Display

List Student Audit Details 02-02-96 15:54:58

ID: 999 99 9999 Network Id: PTUCHMAN Term/Year: SP96 Term Type: TR
Name: Tuchman Patricia Amy
Plan Name: REGAT008 Change Date: 02-02-96 Time: 13:43:01 Agent: AEZZPAT
Program Name:

Table Name	Column Name	Before Value	After Value
STUDENT_COURSE	GRADE		A

We have a few online screens to look at the information. These are not ones we've given to the users, so they have minimal information on them. This screen shows the ACS, ACE, and ACD information for this change. It doesn't show all of the logical keys; some are in the screen's header.

Audit Report

1996-02-02 13.43.01 DONE BY AEZZPAT

TABLE NAME	ACT	COLUMN NAME/ KEY TABLE NAME	BEFORE VALUE/ KEY COLUMN NAME	AFTER VALUE/ KEY VALUE	REP LINE
STUDENT_COURSE	U	GRADE		A	C
		STUDENT_PERSON	DN_PERSONAL_ID	999999999	L
		STUDENT_PERSON	DN_CURRENT_FULL_NA	Tuchman Patricia Amy	L
		STUDENT_COURSE	YEAR_TERM	96SP	L
		TERM	TYPE	TR	L
		STUDENT_COURSE	RUBRIC	MATH	L
		STUDENT_COURSE	NUMBER_PREFIX		L
		STUDENT_COURSE	NUMBER	120	L

Here's the audit information formatted by a QMF report. In the REP LINE (report line) column, a C indicates that the line has Change data; an L indicates that the line contains Logical key information.

We use the denormalized versions of personal ID and name for the logical keys whenever possible; this means one less table to read in the live database to build the keys.

Specifics!

Hopefully, this all makes sense. Now, I'll go onto some of the reality we encountered.

Problems Encountered

- **Too much data!**
- **Not enough disks**

We were AMAZED at the amount of data that we were gathering. Since the old IMS system had only audited limited online transactions, we knew that we had no good way to estimate the amount of data we would be getting based on the old approach. When you add in the reality that we have many more people making changes, we had a problem...

Solutions Found

- Eliminate updates from large batch jobs
- Data partitioned by year/month
- Unload partitions that aren't being used
- Reload partitions when needed for reports
- Lost a lot of time implementing these

We've decided to exclude updates done by certain of the larger batch jobs. Every report that the users have asked for so far has had us exclude this information. We still are extracting the data from the DB2 logs and we could add it to the database if there was a need. Most of the batch jobs that we're excluding have input data that was verified and saved, so the users have that as an alternative source of information about changes to the student records. End of term grade updates is one example of this.

We've partitioned the 4 audit change tables (in the audit database); the partition field is YYYYMM. Our plan is to add data into a partition; then, at the end of the month, reorg it to set freespace to 0, report it to death, and unload the partition. We will be able to reload a partition if there is a need.

We're currently (in February) running a specialized purge job to selectively delete from a partition the data from the large batch jobs that we're no longer auditing in the newer partitions. Once a partition has been cleansed in this way, we'll do the unload.

We actually lost a LOT of time (person and machine) because we added all the data originally and then deleted it out. If we had been able to estimate the volume of data more accurately before we started adding data, we would have saved a lot of agony. Also, we had VERY explicit instructions from the users to save everything; since we didn't have any estimates, we couldn't reasonably say "no".

User Reactions

- Reports didn't look like they used to
- Too much information!
- A delete shows all the fields as being changed
- An insert shows all the fields as being changed

Needless to say, the users thought that there was too much data on the initial reports. A row delete is shown with all of the after images as defaults; a row insert is shown with all of the before images as defaults. This really bulks up a report. It was very disconcerting to the user when a record was on the report as an insert and both before and after values were the same, which happens everytime a column is defaulted.

The users discovered that the volume of data they were seeing on their reports was masking what they were really interested in. If you're really looking for unauthorized grade changes, you don't want to see the legitimate end-of-term grade processing for 35,000 students.

At one point, it was taking 1 FTE 1 week to work though 1 week's worth of audit reports.

System Statistics - DB2

- Number of tables audited - 131
- Number of rows added in month
  - audit_change_set: 31,000
  - audit_change_entity: 234,000
  - audit_change_detail: 1,822,000
  - audit_change_logical_key: 1,189,000
- Peak disk usage
  - 31 disk packs before starting to reduce
  - getting more than 60% reduction

We're currently auditing 131 DB2 tables. Since we've partitioned the audit db by month, we can easily break-out monthly statistics. These statistics are for months that were done AFTER we started excluding the big batch updates.

At one point, we were consuming 31 complete disk packs; we had not yet added all of the data we had extracted. This is when we decided to start to both stop adding selected batch updates and to delete the data from those updates that had already been added.

By the way, we do have data compression turned on. The numbers for that are:

ACS - 99%

ACE - 78%

ACD - 80%

ACLK - 75%

System Statistics - Programs

- Number of batch jobs
  - 2
- Number of online screens for definition db
  - 20
- Number of display screens for audit db
  - 3
- Number of external action blocks
  - 9

The batch jobs are the add data and the delete data jobs. We have two versions of the delete - one for regular and one to get rid of the data we decided we never should have put in the database in the first place.

There are 20 screens (display and update) for the tables in the audit definition db. These are only used by the programmers and aren't "user friendly".

We have 3 different displays for the data in the audit db - again, these are only used by the programmers. We have some QMF queries that we've used for problem resolution also.

We have 9 external action blocks. In addition to the 2 that read the "live" data, we have 2 to read the LA files, and the rest do various conversions (eg packed data to text.)

The current reports the users are getting are written in DB2/Natural, by a different part ofmy department. I don't have information on them.

Requirements

- DATA CAPTURE ON for selected tables
- LOTS of tapes at data center
- DB2 installed
- Log Analyzer installed
- BSDS as large as possible

Here are some of the requirements for using the audit system:

Lots of tapes - since we're saving the output from LA for 2 years, we're using a lot of tapes. Also, setting Data Capture On increased the size of the rows being written to the DB2 log, which increased the number of tapes being used for the log.

BSDS - LA can retrieve information from DB2 log tapes that are no longer in the BSDS, but it's a pain. You need to hard-code the log dataset name in the JCL and you can't do resume processing. On several occasions, we got behind in the daily processing (when we were doing tuning) and found out the BSDS had rolled tapes of interest out.

Errata

- Should associate during create, not afterwards
- Cascade delete shows as delete of “parent” row followed by deletes of “children” rows

We have some entries without logical keys built. Upon investigation, I found that the create process was doing the associate as a separate update. Because of this, the foreign key was null for the create. Each of these inserts is followed by an update, but that doesn’t help. By changing the create PAD, we not only made the audit data more usable, but we got rid of a uneccessary update.

Cascade deletes confused the users since they didn’t understand the data model structure. If they deleted a student’s term, they really didn’t want to see all of the associated student_courses.

Log Analyzer Overview

- ISPF dialog to define “strategy”
  - Define tables to extract from DB2 log
  - Define output format
  - Specify “resume” processing
  - Generate JCL
- Need JCL changes for production
- Save run information (optional)

LA by itself is quite useful. We’ve used it to track down application problems before the data was available in the audit database. It has an ISPF dialog that steps you through defining your log extraction criteria, defining your output format, and specifying what date/time period you’re interested in. You can run a LA either online or batch; if batch, it generates JCL for you.

Once I had the generated JCL, I needed to modify it for production; it had some dataset sizes that were inappropriate, some dataset names that didn’t meet standards, etc.

For now, I’ve enabled the option to save information about the LA runs. I can easily see what data has been extracted from the logs and when. This has helped resolve some production turnover problems.

Log Analyzer Benefits

- Use during development to easily see what is happening
- Use for tuning
- Use for problem resolution

Although we bought LA for the audit project, we’ve actually been using it for a number of other reasons. For those of us with an IMS background and accustomed to running BTS, this is a wonderful way to get a trace of all database updates done during a trial execution of a program in development. (“You mean it updated THAT table?”)

As I mentioned before, we’ve used it for tuning - we’ve caught optional relationships that were never being updated or being updated to the wrong entity, or an ASSOCIATE not being done within the CREATE.

We’ve also run LA in production to help resolve problems before the audit data is available in the audit database or for data that isn’t being audited.

Log Analyzer Quirks

- Alter side effects
- Very awkward dialog

LA is a wonderful product, but it has a few quirks.

If you alter a table definition, LA doesn’t always remember how to decode the table layout. Even if what you did is add a nullable column on the end of a row. Recycling DB2 or image copying the table usually fixes this problem. PT says they have a fix they’re about to ship. When LA encounters a changed row for a table that it can’t format, it prints the hex version of the row(s). My program can’t add this to the audit database.

The ISPF dialog is one of the most awkward I’ve ever used. For example, in most ISPF dialog’s (including Composer’s), F3 means to go back to the previous screen. Within LA, F3 sometimes means to go back to the previous screen; sometimes it means to go forward to the next screen. Sometimes the enter key means to go forward and the command “BACK” means to go back. Sometimes you can’t go back but can only go forward.

Summary

- We now have a working audit system
- We now know all sorts of tuning techniques
- We can use this for any DB2 system we have or will develop

In conclusion, I hope this overview of our working audit system has been of interest. We are currently using it for our “mission critical system” and we are looking into expanding it to other systems, including systems that were not developed with IEF/Composer.

Thank you for your time and attention!

Are there any questions?