This presentation is designed for application developers by an application developer. Topics covered: We usually get the problem when it is too late or too expensive to fix it. How you can use your business knowledge to write efficient SQL. What do you know about the data to choose the best path for your query? How you can get all necessary information from the plan table and DB2 catalog in a format you can read and understand.

Boris Ioffe – Programmer Analyst

- 33 years of IS experience
  - 13 years in application development, 7 years of DB2 experience
  - 20 years as Mainframe system programmer

DB2 Application Development from the Programmer's Point of View

- Common data related problems in DB2 application
- How to avoid data corruption in the first place
- Starting SQL tuning before writing anything
- Predicting and avoiding performance degradation before it happens
- How to write and use customized EXPLAIN for your applications

This presentation is designed for application developers by an application developer and is a result of analyzing 5 years worth of historical data related problems found in large scale DB2 applications.

What is the root of the data integrity corruption and the source of the data with broken business rules?

How to use your knowledge of data and business rules to write correct and efficient SQL.

What are the benefits of writing the “pilot” processing using SQL only, not Cobol.

How to predict and avoid performance degradation before it happens.

How to write customized EXPLAIN for your application.

Common Data Related Problem in DB2 Applications

- Traditionally not thought as “Application development”, but who fixes bad data? (Answer: Application developer)
- Production fixes require logging and documentation. Why not do it in a DB2 table?
- Audit table:
  - Date/time of the ‘Fix’
  - Application table affected
  - Key of affected table
  - Problem description or reference to description
  - Approved by
  - Report name if any
- As history accumulates it can be analyzed and summarized resulting in a report card on data and application health.

During past 7 years my primary job function was to check and support data integrity of a large scale membership database, perform data conversions for new business and SQL tuning for ‘slow running’ batch jobs.

About 5 years ago during routine audit I was asked to record all “manual” intervention of production data for auditors. I designed a DB2 table with simple structure: date/time of intervention, affected table, key data, brief description of the problem, approval name, and reference to reports related to this intervention.

Couple of years ago I was asked to provide to the management some statistics summarizing the data fixes. When I began to analyze data in this table, it reminded me of patient’s medical history.

(Some of the statistic:

Number of records to analyze: 13,439,069 (about 3,000,000 a year)

By Years: 2003 - 2,842,981
2004 - 4,749,145
2005 - 2,381,891
2006 - 3,258,523
2007 - 206,603

So I decided to classify these data in a manner similar to a medical record.

The following is a result of such classification:

Audit Table as a Medical Record

- Conversions
  - Large number of records is related to conversions. Not a data problem.
- Accidents
  - Sporadic random problems caused by bad input, operator errors, abends.
  - Usually isolated and easy to fix.
- Minor data problems (Common cold)
  - Do not significantly impact processing. Difficult to trace to the source.
  - Happen rather frequently.
- Major data problems (Emergency room visits)
  - Significantly impact processing. Often caused by major software releases.
  - Most extensive to fix.
- Reoccurring data problems (Chronic illness)
  - Start as minor data problems but keep reappearing. Lead to permanents reports and automated or manual interventions on regular basis.
  - Can be light or severe.

Here is what I get:

Conversions : Usually very big volume affected, happens 2-3 times a year – not an illness at all. I will skip it.

Accidents : Happen sporadically. Data broken by abends, operators or client errors. Smaller volume of corrupted data. Easy to identify , usually easy to fix.

Example – some fields are not populated in the record or were updated incorrectly, wrong data came from the client feed.

Common cold: Happen pretty often – sometimes monthly or weekly. More than one process creates the same data problem. Not easy to identify all corrupted data – problem requires large volume of data scans to isolate damaged data.

Usually not easy to identify the source – sometimes require continuous data fixes. Do not significantly impact processing.

Typical example – orphan child records appear in some tables.

Emergency room visit: Often caused by major software releases. Most extensive to fix.

Usually fix needs to be applied directly to the production data without sufficient testing.

Chronic illness : Can be light or severe. Usually a consequence of the flu or common cold. Usually caused by design flaws or application issues that cannot be fixed without major rewrite. Leads to a daily or weekly reports and automated or manual interventions on a regular basis.

Lets Talk About Preventive Care

- Most of the conversions and accidents are caused by outside world and can not be controlled or prevented by application developers.
- Common cold, emergency room visits, and chronic illnesses often could and should be prevented.
- The rest of the presentation will focus on preventive measures that application developers can take.

Most of the conversions and accidents are caused by outside world and can not be controlled or prevented by application developers, but common cold, emergency room visits and chronic illnesses often could and should be prevented.

Causes of Data Health Problems

- Inconsistencies in existing data and wrong assumptions about data integrity
- The “IF” problem
- Programmer’s errors not found during testing

Lets start from the root cause of the problem.

Three major sources can be identified:

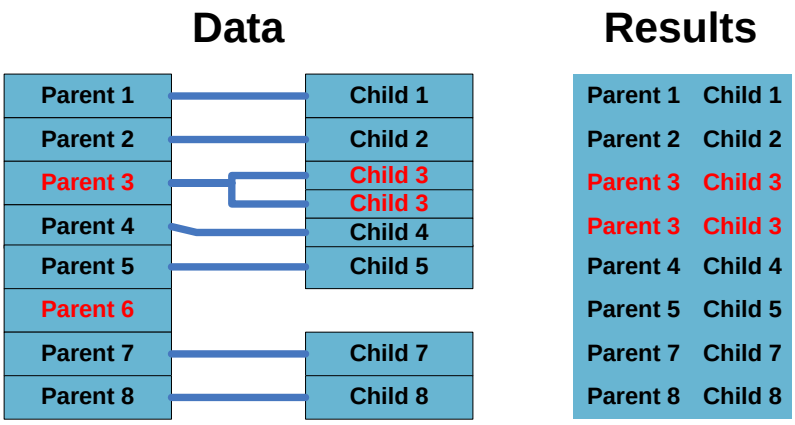
Initial inconsistencies in existing data.

“IF” problem.

Programmer’s errors not found during testing

Any one of the above could lead to (directly or indirectly) any of the data problems – depending of the type of the processing.

Inconsistencies in existing Data and Wrong Assumptions About Data Integrity



Results

Parent 1	Child 1
Parent 2	Child 2
Parent 3	Child 3
Parent 3	Child 3
Parent 4	Child 4
Parent 5	Child 5
Parent 7	Child 7
Parent 8	Child 8

Here is an example of the first problem:

Typical business assumption: We know that if we have a record in a parent table we should have a child record in another table with record type 1. In real life – more than one record exists for some parents (no full primary key was used in child table) and no child record can be found at all for other parents. Why does this happen – this rule was never strictly enforced. Why? No processing was based on this rules before.

If we are lucky we just abend on these exceptions without creating any damage to the data or spreading wrong business data relation across other tables.

But we have good programmers and they always included checking for -811 SQL code. As a result, processing simply picks up a random row or skips some records with inner join of the parent's and the child tables.

The "IF" Problem

- Business requirements:

Create a weekly email report to identify those members who are …….
1.1 Gather all members loaded to ……
1.1.1 Exclude … members by :
1.1.1.1 If the 'Group Type' equals 'D' bypass the record and do not display the member on the report.
1.1.1.2 Exclude ….. Products and DP HMO members by reading the ….. table.
1.1.1.3 If members are found in the clients listed on the ….I table, bypass the record and do not display the member on the report.
1.1.1.4 Exclude the members enrolled in ….and … groups by criteria described in …..requirements for….
1.1.1.5 If there is a 'R', 'S', 'A', or 'C' in the …….
1.1.1.5.1 If indicator is 'S' display the member with ….
1.1.1.5.2 If indicator is 'A' display the member with ….
1.1.1.5.3 If indicator is 'C' display the member with ….
1.1.1.5.4 If indicator is ' ' display the member with …
1.1.2 For the remaining members, gather the maximum coverage effective date in HST for each member.
1.1.2.1 If there is a coverage effective date with ….., this member should appear on the report
1.1.2.2 If there is a coverage effective date with a …., this member should be bypassed.
…….
1.1.2 For…
1.1.2.1…
…..
1.1.11 For…
1.1.11.1……

What I call the "IF" problem – is the five pages of IF statements in requirements repeated in 50 screens of Cobol program or in worst case scenario – spread across 50 programs.

After debugging and testing it is very difficult to find where and what condition is missing. If you add the cost of new testing you get into real problems.

Here is an example of the "IF" problem:

The “IF” Problem

Cobol

```
EXEC SQL
    FETCH MAINCURSOR  INTO
          :WS-….
……….
    :WS-….
END-EXEC.
IF WS: …    and   WS-…
   PERFORM 3000-GET-CORRECT-DATESD THRU 3000-EXIT
 ELSE
,,,,,,,,,,,
    IF (WS-…=. AND  WS-HOLD-…= OR (WS-.. = AND WS-.. <>  )
            PERFORM 1020-OPEN-COVMIN1-CURSOR
            IF WS-GOT-.. = 'N‘
                MOVE 'N' TO WS-…
            ELSE
                PERFORM 1030-OPEN-COVMIN2-CURSOR
                IF WS-COVMIN2-FOUND
                     PERFORM 1050-CALL-GET-…
                           ……..
                              MOVE WS_GET-GRP' TO WS-CHECK_GRP
                      PERFORM 1191-CHECK-GRP-OPEN-CURSR
                       IF WS-CHECKRET
                            PERFORM 1194-CHECK-CLIENT-OPEN-CURSR
                       ELSE
            ……..
```

50 more pages of code like that…

Now try to find the right place to insert missing rule or change an existing one.

What about testing? All test cases must be ran and analyzed.

The “IF” Problem

SQL

```
--                      BEGIN RULE 1.2 SELECT
SELECT CASE WHEN FL1 IS NULL THEN FL3 ELSE
            CASE WHEN FL1 > CHECK1  THEN FL4  ELSE FL1  END
           END          ,
……..
--                      END  RULE 1.2 SELECT
        ……..
FROM TBL1          --
--                      BEGIN RULE 1,2 NESTED SELECT
LEFT OUTER JOIN
(SELECT FLN, FLM, ….
--                                  BEGIN RULE 1,2.1.1
    CASE WHEN FL4 > CURRENT_DATE – 30 DAYS THEN … ELSE .. END
--                                  END   RULE 1,2.1.1
……
FROM TBL2 , TBL3
WHERE T2.FL1=T3.FL1 ……
--                      END RULE 1,2 NESTED SELECT
) NTBL
……..
```


All rules can be clearly located, updated or checked.

Most importantly – you can see it usually in no more than 5 screens, not 50.

Testing – you can run the test for whole database with old code and new code and compare results by comparing two flat files.

Finally, programmer errors are simply things not found in testing.

Corrupted Data Have to be Fixed

- Impacted records have to be identified
- Once records are identified:
  - Fix has to be done quickly
  - Fix has to be done correctly
  - Fix has to be done in production environment
  - Fix has to be done on production data
- There is no room for error and testing opportunities are very limited

Now lets start to look at how the problems are fixed.

First step – to identify all problems created in the database by wrong processing.

To do that we need to apply almost the same rules as production process but we need to do it fast, correctly and with production data in production environment. We can not rely on testing – we just have no time.

The only language that I have at my disposal is dynamic SQL.

First we write the business rules in SQL statement to find all the exceptions, and after that we create another SQL that updates data the right way according to business rules.

Next step - if business assumptions are not represented in real data we need to find all the exceptions before we get the next set of corrupted data.

We need to write yet another SQL to find all the future problems.

To my big surprise it was much easer to write all business rules in SQL rather than in Cobol. In addition, this SQL runs in a fraction of production processing run time.

DB2 Application Development from the Programmer's Point of View

- Common data related problems in DB2 application
- **How to avoid data corruption in the first place**
- Starting SQL tuning before writing anything
- Predicting and avoiding performance degradation before it happens
- How to write and use customized EXPLAIN for your applications

How to Avoid Data Corruption in the First Place

- We can use SQL queries to check business assumptions about existing data and identify problems **before corrupted data propagates** in production so we include data cleanup in project requirements.
- We can write pilot SQL for data processing and identify potential performance problems so we can ask for additional business level filtering **before we start programming**.
- We can write pilot queries to get data that will end up in reports or sample images of updated/generated data for the entire database so it can be validated by business **before we start programming**.

My next thought:
Why not start the project simply from "pilot" SQL?

I did it with a couple of projects.

I decided to start with providing business analysts with results of almost all requirements just as spufi results.

The first problem I found in the beginning of writing SQL – before you do any join you need to know what join you need – Inner or Outer. Lets do outer for all new rules – this way we can eliminate problems for not processed data.
In a couple of hours I was able to produce reports listing wrong assumptions about existing data.
Results:
– new stage was added to the requirement – data cleanup before implementation
- added exception report if wrong data relations are found.

Next step – Pilot SQL for real processing.
Even from the first try I could see that this SQL will run forever. Now I was prepared to discuss with business what other filtering can be added to the requirements.
Result:
– things that were viewed unimportant and redundant by business ended up being extremely helpful
– other table can be used as a driver for the process, instead of checking 40 million records we can start with other table and scan only 50 thousand.

After updating the requirements we get 'first draft' report.
This report includes images of the updated records without any update in the actual database.
More than that – instead of limited number of test cases I was able to run the process for the entire database!
More than that – I put almost all "IFs" in one SQL statement – driver for the process.
More than that – why not to put this SQL instead of Cobol program to fast unload utility?
Not always, but sometimes it works perfectly.

How to Avoid Data Corruption in the First Place

- Before you do any join you need to know what join you need – Inner or Outer.
- Data cleanup may be needed before program implementation.
- Use exception report for wrong data relations found in processing.
- Run “pilot” SQL for all qualified data to find any violations of data assumptions.

Never Abend on SQL code -811 or 100 – try to rollback and record the error

Lets do outer for all new rules – this way we can eliminate not processed data – just by simple count of the qualified rows from the left table.

How to Avoid Data Corruption in the First Place

- Pilot SQL benefits
  - Provides general idea of performance (long before implementation)
  - Identifies valid data relations not accounted for in the requirements
  - Identifies data corruption that was unknown
  - Could help to determine the best design approach
- Results: Data corruption can be avoided or at least minimized.

DB2 Application Development from the Programmer's Point of View

- Common data related problems in DB2 application.
- How to avoid data corruption in the first place.
- **Starting SQL tuning before writing anything.**
- Predicting and avoiding performance degradation before it happens.
- How to write and use customized EXPLAIN for your applications.

Starting SQL Tuning Before Writing Anything

- You know your data better than DBAs and see things that do not get passed to DBAs
- Early SQL tuning is critical for good DB2 application design (so much is driven by the SQL design – good and bad)
  - Too many open and close cursors cannot be easily fixed – usually program needs to be rewritten.
  - Wrong table might be used as a driver in the FROM clause.
  - Poor choice of variables in the WHERE clause could lead to wrong results or to confuse the query optimizer.

Why am I switching to performance problems? Because look at this as a sickness of the application.

Like any other sickness it can hurt application and in most cases it is related to the understanding of the application data.

Pilot SQL some times frightens programmers – it is big and seems difficult to understand.

Now a little more about SQL tuning before writing it.

The worst performance problems usually happen with "perfect" SQL where all access paths are perfect.

For example, access of the data by primary index only. Only business and application knowledge can help with this performance problem.

In my experience there are two major problems with perfect SQL:

1.Too many open and close cursors.

This problem cannot be easily fixed – usually program needs to be rewritten or processing needs to be redesigned.

2.The problem located in FROM statement – wrong table is used as a driver.

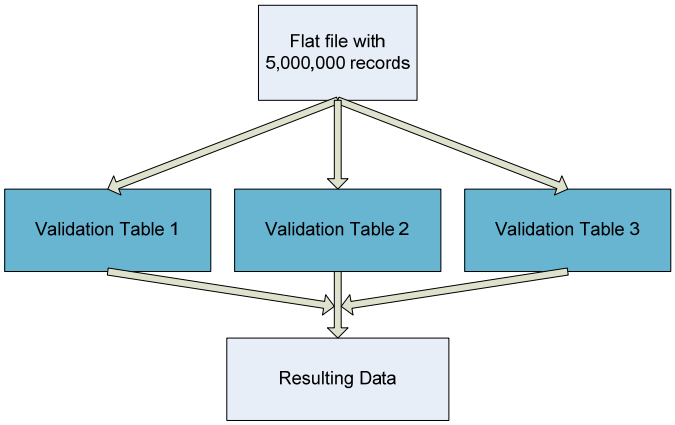
You cannot expect that DBA or any DB2 expert will help unless you ask the right questions.

You have to know and provide in your question all possible places to fetch the data.

The best you can do is to provide the full set of alternative SQL statements.

In Version 8 optimizer using all available statistic to choose the access path and it may create a big problem for static SQL with variables.

Some Examples



I'm not a DBA and cannot create generic examples for illustrating my presentation – I can only use the tables that I have access to.

Because of that I will be using examples from my industry. I'll try to make them as simple and as generic as possible.

1.New process was created.

Requirements: Find all people from input file who has valid medical insurance as of today and satisfy some additional conditions by checking specific data in database.

Process design: Read a record from the file and open cursor to check if person has valid medical coverage.

If yes, check another 15 cursors to make sure that you need this person.

If yes – write output file.

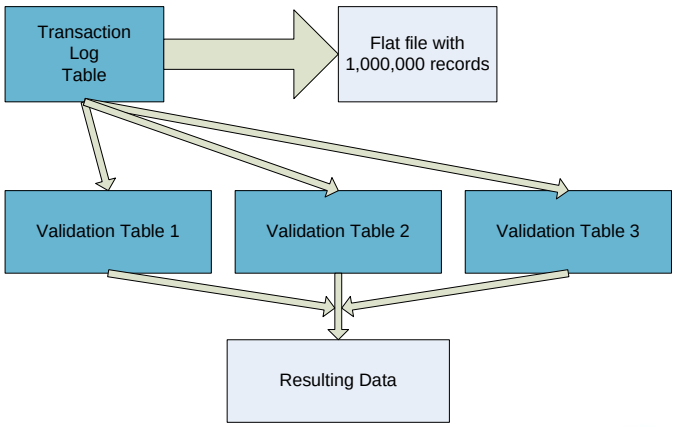
Input file – 5,000,000 records…

Every cursor in the program uses only clustering index.

Run time exceeds 4 hours

DBA recommendation – try to sort file in clustering order of the first table – results – 2 hours run time.

Some Examples



Application approach:

What is in the input file? Input file was created by previous job and is the strip from transaction log for days range. But transaction log is a DB2 table and have much more information than input file.

It can be joined with other tables to filter data before it hits the other cursors or even combine almost all of these cursors into one.

Next step – test the cursor and use SQL with fast unload utility.

To do that I need to generate one line in the “WHERE” clause that has hard coded date range and makes concatenated DD statement in JCL.

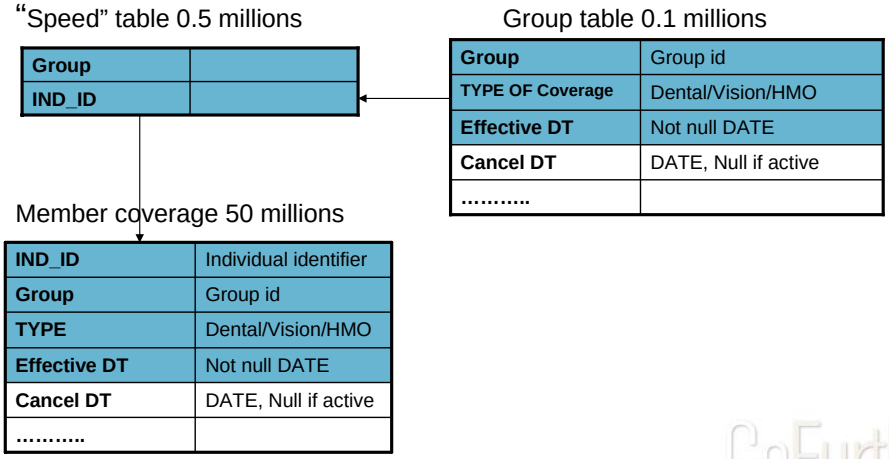
Results:

Utility run time - 2 min.

Data set to process in the program - 2,000 records.

Program (what is left after SQL-coded logic plus generating the SQL string for the next run) – 1 min.

Some Examples



Another example:

Requirements: Provide all ids with active dental insurance.

System specs: Scan member coverage table to find all necessary data.

Table size 50,000,000 records (10 years of operation).

No index can help – cancel date is not included in any index.

Type is the 3rd column of the existing index and has only 5 distinct values – not a big help for new index request.

Results: query scans 50 million records and runs for hours.

Question to business - do we need to include any individual if group was canceled or has no dental at all?

Answer – no, and you should not find any active members in canceled groups or dental coverage for any member if group does not have it.

New approach – joining 3 tables and filtering for only active groups with active dental type and accessing biggest table by index.

Results: run time gets down to 10 minutes.

DB2 Application Development from the Programmer's Point of View

- Common data related problems in DB2 application
- How to avoid data corruption in the first place
- Starting SQL tuning before writing anything
- **Predicting and avoiding performance degradation before it happens**
- How to write and use customized EXPLAIN for your applications

Predicting and Avoiding Performance Degradation Before it Happens

- Signs of danger:
  - Expected significant changes in table size and/or data patterns
  - New index might be grabbed by existing programs during unrelated recompile/rebind. This could cause old performance problems to reappear
  - Data purge from a table could make queries with List prefetch to run significantly longer

Very briefly about typical signs of danger that application programmer can see:

Nightmare for every application developer:

1. New release involves new tables.

You run in the test region all initial loads with full production data.

You test all possible scenarios and the process finishes within minutes.

On release day your Cobol packages were promoted to production in the evening, at night conversions process ran and loaded the tables. Your job started and …. newer ended.

2. Another example - your batch process ran 5 minutes daily for two months and suddenly tonight it's running for hours and hours.

Each scenario was created by the same problem – wrong bind time.

On release scenario – production bind happened exactly when new tables were empty.

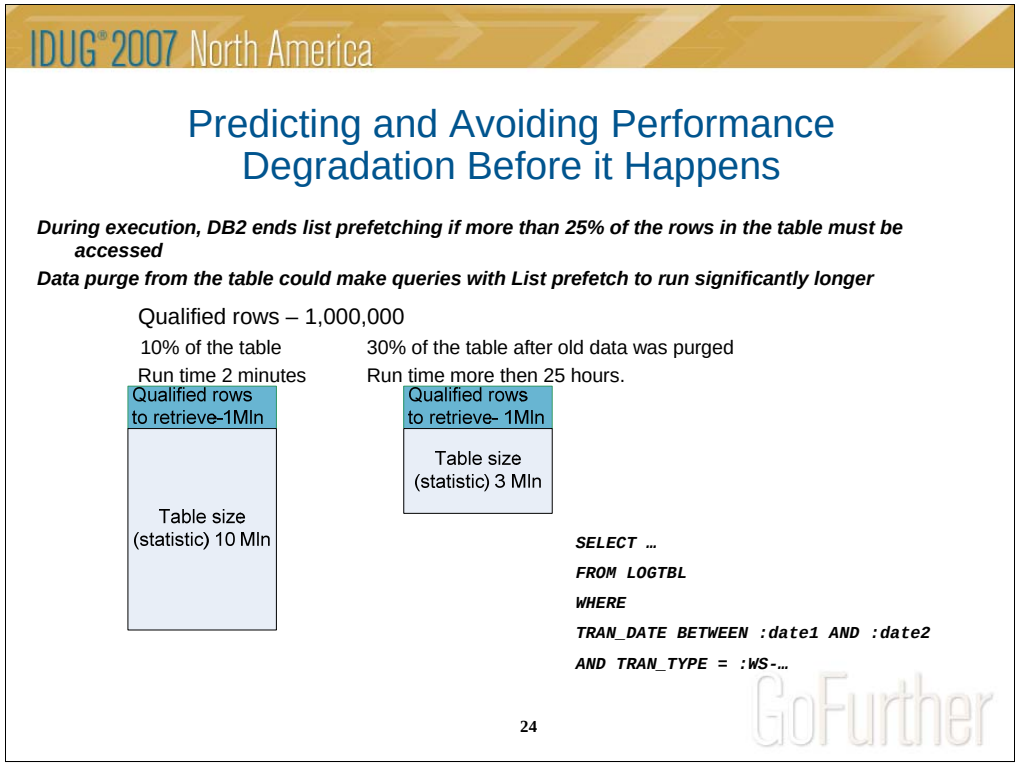
On second one you get significant increase in data volume or just significant data pattern changes and what was the cheapest and the fastest access became the worst one.

Others signs of danger:

1.Program needs to be changed (not SQL at all) and you remember that there was a significant effort to make it work fast in the past.

Check if optimizer kept the path you need. This is true even if the data (statistic) is the same but a new index was added to the table.

Little known fact about list prefetch.



2.List prefetch is used for fetching data.

According to IBM web page, DB2 V8 for z/OS does not consider list prefetch if the estimated number of RIDs to be processed would take more than 50% of the RID pool when the query is executed. During execution, DB2 ends list prefetching if more than 25% of the rows in the table must be accessed.

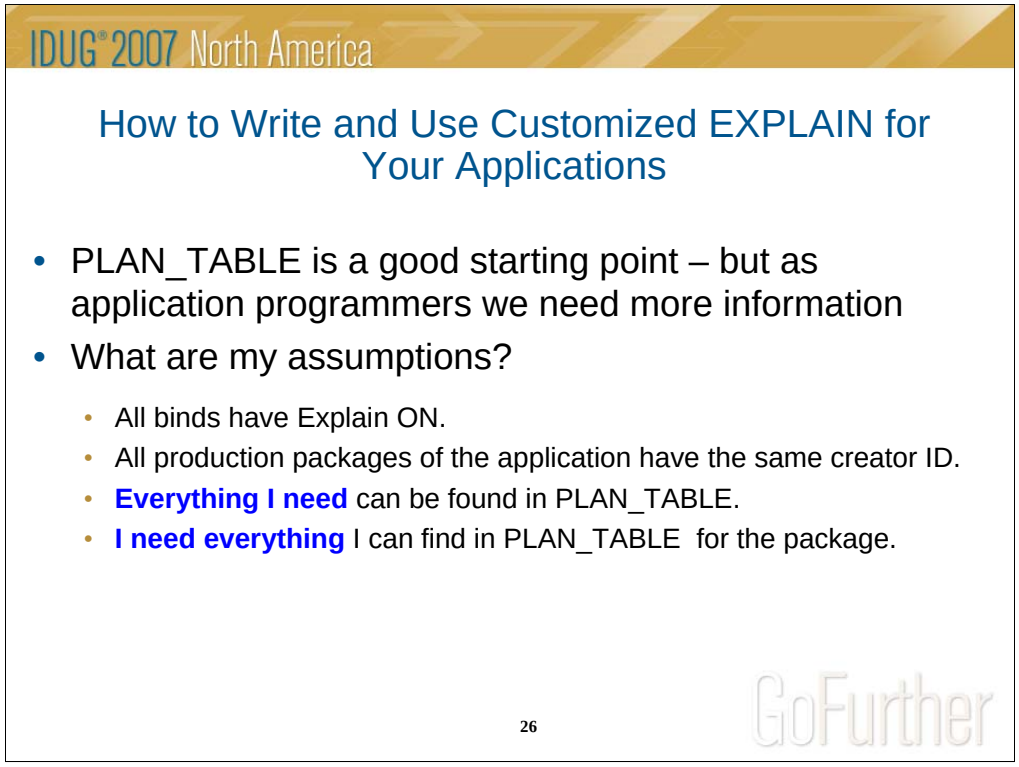
Processing continues by a table space scan IF single index was used.

The maximum size of a single RID list is approximately 26 million RIDs.

Scenario: The program opens the cursor with list prefetch to check if a key from input record exists in transaction log for a weekly date interval. You purged old data from the log and DBA ran statistics. Now your interval is 1/3 of the table. The program is checking the log for 1,000,000 records – every open cursor instruction starts list prefetch and aborts it. Actual size of the table decreased, but the process runs forever. The same is true if the record set increases significantly for requested data interval – one needs to be extremely careful with list prefetch if table or RID data set is volatile and may hit high volume.

DB2 Application Development from the Programmer's Point of View

- Common data related problems in DB2 application.
- How to avoid data corruption in the first place.
- Starting SQL tuning before writing anything.
- Predicting and avoiding performance degradation before it happens.
- **How to write and use customized EXPLAIN for your applications.**



Again, I'm an application developer and my main job is application development. If application database has 200 tables and 2-3 indexes for every table – it is not possible to remember all indexes and all data patterns needed for developing good SQL.

Now I'll try to show how we can use pilot SQL approach to develop customized EXPLAIN for providing information to application developers in format that we, developers, can read and understand.

The goal of this SQL program is to make a tool that will show all the necessary data about existing processing to help us to predict and prevent some of the problems.

I am not an expert in DB2 catalog structure. My DB2 knowledge is comparable with the knowledge of a business analyst who knows what needs to be done but not sure what is the best place to find data and how to make it fast and right.

My knowledge:

PLAN_TABLE is the table where I can find all access paths for my SQL after I run EXPLAIN.

How to Write and Use Customized EXPLAIN for Your Applications

- **Programmer's view of EXPLAIN**

```
DELETE FROM PLAN_TABLE ;
EXPLAIN ALL SET QUERYNO = 01  FOR …

Your SQL here

SELECT QBLOCKNO                    BL
        ,PLANNO                    PL
        ,METHOD                    MT
        ,SUBSTR(TNAME,1,10)        TNAME
        ,ACCESSTYPE                ACCT
        ,MATCHCOLS                 MCOL
        ,SUBSTR(ACCESSNAME,1,10)   INDXNM
        ,PREFETCH                  PREF
        ,INDEXONLY                 INDXO
        ,MIXOPSEQ                  MIXOPS
        ,QUERYNO                   QNM
  FROM PLAN_TABLE
  ORDER BY QUERYNO,QBLOCKNO,PLANNO,MIXOPSEQ
```


Put in any SQL and...

How to Write and Use Customized EXPLAIN for Your Applications

Typical EXPLAIN

```
SELECT QBLOCKNO                    BL
      ,PLANNO                      PL
      ,METHOD                      MT
      ,SUBSTR(TNAME,1,10)          TNAME
      ,ACCESSTYPE                  ACCT
      ,MATCHCOLS                   MCOL
      ,SUBSTR(ACCESSNAME,1,10)     INDXNM
      ,PREFETCH                    PREF
      ,INDEXONLY                   INDXO
      ,MIXOPSEQ                    MIXOPS
      ,QUERYNO                     QNM
  FROM PLAN_TABLE
  ORDER BY QUERYNO,QBLOCKNO,PLANNO,MIXOPSEQ
```

BL	PL	MT	TNAME	ACCT	MCOL	INDXNM	PREF	INDXO	MIXOPS	QNM
1	1	0	Table A	R	0		S	N	0	1
1	1	0	Table B	I	0	XMBD0333	S	N	0	1

This is what I get. Now I need to know what the index XMBD0 is, what are the sizes of tables A and B and is it a good or a bad access path?

How to Write and Use Customized EXPLAIN for Your Applications

What about static SQL?

```
SELECT A.*
FROM
DB2PROD.PLAN_TABLE A
WHERE A.PROGNAME = 'mypackagename'
AND COLLID ='myappcollid'
                WITH UR
```


DB2PROD is qualifier in our shop – can be any in yours – check with your DBA – from now on DBA is my "application expert".

Looking at the result I see 10 Query numbers and every query path repeated 5 times– but I know that my program has only 3 SQL statements.

I can see that Plan table have all Bind history of the program. DBA told me that no cleanup is done on the table – I need the latest Bind only.

Now we can check our assumptions:

How to Write and Use Customized EXPLAIN for Your Applications

- Validating assumptions.
  - All bind has Explain ON.
    - Correct.
  - All production packages have the same creator id.
    - Correct. In our shop it's DB2PROD.
  - Everything I need can be found in PLAN_TABLE.
    - Not at all. I need more information.
  - I need everything I can find in PLAN_TABLE for the package.
    - Not true – I do not need the history I need today's picture.

How to Write and Use Customized EXPLAIN for Your Applications

New SQL

```
SELECT A.*
   FROM DB2PROD.PLAN_TABLE A ,
            ( SELECT  MAX(BIND_TIME)  MAXBIND, PROGNAME,CREATOR
                  FROM DB2PROD.PLAN_TABLE
              GROUP BY PROGNAME, CREATOR
               HAVING PROGNAME = 'mypackagename'
                         AND CREATOR='DB2PROD'  )  B
               WHERE A.PROGNAME=B.PROGNAME
                AND MAXBIND = A.BIND_TIME
                AND A.PROGNAME  = 'mypackagename'
                AND A.CREATOR='DB2PROD'
        ;
```


Do not try to use MAX of TIMESTAMP field – this is not a TIMESTAMP – it is a character field – looks like IBM does not have our data modeling department.

The field to use is BIND_TIME. Here is new SQL.

How to Write and Use Customized EXPLAIN for Your Applications

```
DELETE FROM PLAN_TABLE ;
INSERT INTO      PLAN_TABLE
       SELECT A.*
          FROM DB2PROD.PLAN_TABLE A ,
         ( SELECT  MAX(BIND_TIME)  MAXBIND,
PROGNAME,CREATOR
              FROM DB2PROD.PLAN_TABLE
               GROUP BY PROGNAME, CREATOR
                 HAVING PROGNAME  = 'mypackagename'
                           AND CREATOR='DB2PROD'  )  B
            WHERE A.PROGNAME=B.PROGNAME
                AND MAXDAY   = A.BIND_TIME
                AND A.PROGNAME  = 'mypackagename'
               AND A.CREATOR='DB2PROD'  ;
           COMMIT ;
```


Add Insert statement to this SQL and you can start playing with your plan table. Now I can see what optimizer did with all my SQLs from the PLAN_TABLE.

How to Write and Use Customized EXPLAIN for Your Applications

QN	QUERY #
BL	BLOCK #
PL	PLAN #
METHD	METHOD
TBL NAME	TABLE NAME
TY	ACCESS TYPE
MC	NUMBER OF MATCHED COLUMN
ACCESSNM	ACCESS NAME - INDEX NAME
P	PREFETCH
I	INDEX ONLY
T	QUERY BLOCK TYPE
STATROWS	NUMBER OF ROWS IN THE TABLE ACCORDING TO THE LAST STATISTIC RUN
CL	CLUSTERING RATIO
IDISTROWS	NUMBER OF ROWS IN THE INDEX ACCORDING TO THE LAST STATISTIC RUN
INDCOLNM	INDEX COLUMN NAMES
FIRST KEY	NUMBER OF UNIQUE VALUES FOR THE FIRST FIELD IN THE INDEX FROM THE LAST STATISTIC RUN
STAT DATE	DATE OF THE LAST STATISTIC RUN

But can I interpret it? Hardly.

According to what I learned, to evaluate access path I need to know a lot of details - index column names, size of the table, size of the index, clustering ratio, cardinality, first key distinct values..

Yes, I can find all this information but again, my job is to create good running program, not to spend all the time analyzing every SQL statement.

I want to take a brief look and find the place where performance problem could be located.

Let’s write an explain report with all the information that I want to see, not with the information that tool wants to show me.

Here is what I want to see for my package:

How to Write and Use Customized EXPLAIN for Your Applications

```
FROM
(PLAN_TABLE A
 LEFT OUTER JOIN
SYSIBM.SYSKEYS B
 ON  IXNAME=ACCESSNAME
    AND IXCREATOR=A.CREATOR )
LEFT OUTER JOIN
SYSIBM.SYSTABLES C
ON C.NAME=TNAME
   AND A.CREATOR=C.CREATOR
  LEFT OUTER JOIN
SYSIBM.SYSINDEXES I
ON I.NAME=ACCESSNAME
   AND A.CREATOR=I.CREATOR
```


Let's do it.

Start from "FROM" statement. Index column number, not name, we can find in SYSIBMKEYS table – again, according to "application expert" DBA this is the way to get the correct join – Left need to be used because we may have access path without an index. To convert column number to name we need to go to SYSTABLES. To get statistical information about the index we need to go to SYSINDEXES.

Let's write select statement now.

How to Write and Use Customized EXPLAIN for Your Applications

```
SELECT
    SUBSTR(CHAR(QUERYNO),1,4)                                  || '|' ||
    SUBSTR(CHAR(QBLOCKNO),1,2)                                 || '|' ||
    SUBSTR(CHAR(PLANNO),1,2)                                   || '|' ||
    SUBSTR(CHAR(METHOD),1,2)                                   || '|' ||
    SUBSTR(TNAME,1,12)                                         || '|' ||
    ACCESSTYPE                                                 || '|' ||
    SUBSTR(CHAR(MATCHCOLS),1,2)                                || '|' ||
    SUBSTR(ACCESSNAME,1,08)                                    || '|' ||
    PREFETCH                                                   || '|' ||
    INDEXONLY                                                  || '|' ||
    SUBSTR(QBLOCK_TYPE,1,1)                                    || '|' ||
    SUBSTR(CHAR(DEC(CARDF)),8,09)                              || '|' ||
    SUBSTR(CHAR(DEC(CLUSTERRATIOF*100)),14,3)                  || '|' ||
    SUBSTR(CHAR(DEC(FULLKEYCARDF)),8,09)                       || '|' ||
    SUBSTR(COLNAME,1,15)                                       || '|' ||
    SUBSTR(CHAR(DEC(FIRSTKEYCARDF)),8,09)                      || '|' ||
    CHAR(DATE(C.STATSTIME))
```


Result of this query is not what we expected –

How to Write and Use Customized EXPLAIN for Your Applications

Initial results:

172	1	2	1	RETRNA	I	1	XMBD142C		Y	S	000000034	100	000000034	CI_ID
172	1	2	1	RETRNA	I	1	XMBD142C		Y	S	000000034	100	000000034	AGR_ID
172	1	2	1	RETRNA	I	1	XMBD142C		Y	S	000000034	100	000000034	TRNS_ID

Last row indicates that we need to handle null values– we have left outer join here

And let's make it more readable…

First we do not want to repeat the same field for every key field of the index:

How to Write and Use Customized EXPLAIN for Your Applications

```
.....                                                                  || '|'  ||
PREFETCH                                                               || '|'  ||
  INDEXONLY                                                            || '|'  ||
  SUBSTR(QBLOCK_TYPE,1,1)                                              || '|'  ||
--******* BEGIN: IF NO INDEX USED **************
  CASE
    WHEN IXNAME IS NOT NULL THEN
     SUBSTR(CHAR(DEC(CARDF)),8,09)                || '|' ||
     SUBSTR(CHAR(DEC(CLUSTERRATIOF*100)),14,3)    || '|' ||
     SUBSTR(CHAR(DEC(FULLKEYCARDF)),8,09)         || '|' ||
     SUBSTR(COLNAME,1,15)                         || '|' ||
     SUBSTR(CHAR(DEC(FIRSTKEYCARDF)),8,09)
    ELSE
     SPACE(9)                                     || '|' ||
     SPACE(3)                                     || '|' ||
     SPACE(9)                                     || '|' ||
     SPACE(15)                                    || '|' ||
     SPACE(9)
                  END                                                  || '|'  ||
--******* END                    **************
  CASE WHEN C.STATSTIME IS NULL THEN SPACE(10) ELSE CHAR(DATE(C.STATSTIME)) END
……..
```


If there is no index in access path, all fields from SYSKEYS will be NULL. I'm writing it this way to make it easier to remove or to add fields to the results.

Look better already…

How to Write and Use Customized EXPLAIN for Your Applications

New results:

172	1	1	0	TABLE2	R	0	S	N	S					08/14/2006
172	1	2	1	TABLE1	1	XMBD142C		Y	S	000000034	100	000000034	TRNS_ID	0000000034	05/03/2006
172	1	2	1	TABLE1	1	XMBD142C		Y	S	000000034	100	000000034	AGR_ID	0000000034	05/03/2006
172	1	2	1	TABLE1	1	XMBD142C		Y	S	000000034	100	000000034	CI_ID	0000000034	05/03/2006

We do not want to see the same information for every column name in the index.

How to Write and Use Customized EXPLAIN for Your Applications

```
SELECT
--** BEGIN: Skipping redundant columns if index access (show for the first row only)
 CASE WHEN COLSEQ = 1 OR COLSEQ IS NULL OR ACCESSNAME=' '
    THEN  SUBSTR(CHAR(QUERYNO),1,4)                                  || '|'  ||
……..
ELSE
  SPACE(4)                                                           || '|' ||
  SPACE(2)                                                           || '|' ||
  SPACE(2)                                                           || '|' ||
……..
…….
  SPACE(9)                                                           || '|' ||
  SUBSTR(COLNAME,1,15)                                               || '|' ||
   SPACE(9)                                                          || '|' ||
  SPACE(10)
  END
--*** END: Skipping redundant columns if index access ( show for the first row only)
```


If we have an index and sequence of column is not null, then we need to replace every field with a space.

Now we add order by statement and here is what we get:

How to Write and Use Customized EXPLAIN for Your Applications

```
SELECT
----******* BEGIN: Skipping redundant columns if index access ( show for the first
    row only) CASE WHEN COLSEQ = 1 OR COLSEQ IS NULL OR ACCESSNAME=' '
     THEN  SUBSTR(CHAR(QUERYNO),1,4)                                        || '|'
     ||
……….
ELSE
  SPACE(4)                                                                  || '|' ||
  SPACE(2)                                                                  || '|' ||
……….
  SPACE(9)                                                                  || '|' ||
  SUBSTR(COLNAME,1,15)                                                      || '|' ||
   SPACE(9)                                                                 || '|' ||
  SPACE(10)
  END
----******* END: Skipping redundant columns if index access ( show for the first row
    only) FROM  (  PLAN_TABLE A LEFT OUTER JOIN SYSIBM.SYSKEYS B
 ON  IXNAME=ACCESSNAME
    AND IXCREATOR=A.CREATOR )
  LEFT OUTER JOIN SYSIBM.SYSTABLES C ON C.NAME=TNAME
   AND A.CREATOR=C.CREATOR
  LEFT OUTER JOIN SYSIBM.SYSINDEXES I  ON I.NAME=ACCESSNAME
 AND A.CREATOR=I.CREATOR --WHERE QUERYNO IN (0)
 ORDER BY PROGNAME,QUERYNO ,QBLOCKNO,PLANNO
```


And this is the results:

How to Write and Use Customized EXPLAIN for Your Applications

Here is what we get:

751	1	1	0	CBA_CNTR	I	1	XMBD0351		N	S	292240	83	292240	CNTR_ID	292240	01/15/2007
														CLI_ID		
751	1	2	1	CBA_CI	I	1	XMBD0304	L	N	S	11487563	44	5005734	CH_AGR_ID	5005734	01/15/2007
751	1	3	1	CBA_CI_CTCOV	N	3	XMBD033C		Y	S	34514851	100	34514851	CI_ID	11487551	01/15/2007
														INT_CNTR_ID		
														INS_LIN_C		
														CTCOV_EFF_DT		
835	1	1	0	CBA_CI	I	1	XMBD0304	L	N	S	11487563	44	5005734	CH_AGR_ID	5005734	01/15/2007
835	1	2	1	CBA_CI_CTCOV	I	1	XMBD033C		N	S	34514851	100	34514851	CI_ID	11487551	01/15/2007
														INT_CNTR_ID		
														INS_LIN_C		
														CTCOV_EFF_DT		
835	1	3	1	CBA_CNTR	I	1	XMBD035C		N	S	292240	100	292240	INT_CNTR_ID	292240	01/15/2007
914	1	1	0	CBA_CI	I	1	XMBD0304	L	N	S	11487563	44	5005734	CH_AGR_ID	5005734	01/15/2007
914	1	2	1	CBA_CI_CTCOV	I	1	XMBD033C		N	S	34514851	100	34514851	CI_ID	11487551	01/15/2007
														INT_CNTR_ID		
														INS_LIN_C		
														CTCOV_EFF_DT		
914	1	3	1	CBA_CNTR	I	1	XMBD035C		N	S	292240	100	292240	INT_CNTR_ID	292240	01/15/2007

Ok, look like we got the data we wanted,

Let’s make it look like a report

How to Write and Use Customized EXPLAIN for Your Applications

Next SQL:

```
UNION ALL
SELECT    'QN  '              || '|' ||
          'BL'                || '|' ||
          'PL'                || '|' ||
          'METHD'             || '|' ||
          'TBL NAME    '      || '|' ||
          'TY'                || '|' ||
          'MC'                || '|' ||
          'ACCESSNM'          || '|' ||
          'P'                 || '|' ||
          'I'                 || '|' ||
          'T'                 || '|' ||
          'STATROWS   '       || '|' ||
          'CL '               || '|' ||
          'IDISTROWS'         || '|' ||
          'INDCOLNM       '   || '|' ||
          'FIRST KEY'         || '|' ||
          'STAT DATE '        || '|'
 ,0 COLSEQ,-1 QUERYNO, 0 QBLOCKNO, 0 PLANNO, '  ' ACCESSNAME, '  ' PROGNAME, '  '
    TNAME
FROM SYSIBM.SYSDUMMY1
```


To add header – just **union all** with something like that..

We need to add separate sorting field to main cursor and to the header to put header on the top of the report.

How to Write and Use Customized EXPLAIN for Your Applications

- **What else can be done?**
  Making it even easier to read.

```
……..
CASE WHEN SUBSTR(CHAR(METHOD),1,2) = '0' THEN  '0-TBA'
          WHEN SUBSTR(CHAR(METHOD),1,2) = '1' THEN  '1-NLP'
          WHEN SUBSTR(CHAR(METHOD),1,2) = '2' THEN  '2-MSC'
          WHEN SUBSTR(CHAR(METHOD),1,2) = '3' THEN  '3-S' ||
               CASE WHEN SORTC_UNIQ    =  'Y' THEN  'UQ'
                    WHEN SORTC_JOIN    =  'Y' THEN  'JN'
                    WHEN SORTC_ORDERBY =  'Y' THEN  'OB'
                    WHEN SORTC_GROUPBY =  'Y' THEN  'GR'
                    ELSE ' '
               END
          WHEN SUBSTR(CHAR(METHOD),1,2) = '4' THEN  '4-HJN'
          ELSE '    '
     END                                                    ||'|'||
……..
```


Lets make it even easier to read.

You may remember what method numbers mean – I usually do not. Let's put it in the explain report.

How to Write and Use Customized EXPLAIN for Your Applications

New results:

QN	BL	PL	METHD	TBL NAME	TY	MC	ACCESSNM	P	I	T	STATROWS	CL	IDISTROWS	INDCOLNM	FIRSTKEY	STAT DATE
751	1	1	0-TBA	CBA_CNTR	I	1	XMBD0351		N	S	0000029224	083	0000292240	CNTR_ID	292240	2007-01-15
														CLI_ID		
751	1	3	1-NLP	CBA_CI_CTCOV	N	3	XMBD033C		Y	S	0034514851	100	0034514851	CI_ID	11487551	2007-01-15
														INT_CNTR_ID		
														INS_LIN_C		
														CTCOV_EFF_DT		

I know that TBA is table space scan and NLP mean nested loop.

What else can we do?

How to Write and Use Customized EXPLAIN for Your Applications

More customization:

```
CASE WHEN (COLSEQ=1 OR COLSEQ IS NULL)
        THEN CASE WHEN TSLOCKMODE = ' IS' THEN 'INT SHR LOCK'
                  WHEN TSLOCKMODE = ' IX' THEN 'INT EXL LOCK'
                  WHEN TSLOCKMODE = '  S' THEN 'SHR LOCK    '
                  WHEN TSLOCKMODE = '  U' THEN 'UPDT LOCK   '
                  WHEN TSLOCKMODE = '  X' THEN 'EXL LOCK    '
                  WHEN TSLOCKMODE = 'SIX' THEN 'SHR INT EXL '
                  WHEN TSLOCKMODE = '  N' THEN 'NO LOCK     '
                  WHEN TSLOCKMODE = ' NS' THEN 'CANT DTR  NS'
                  WHEN TSLOCKMODE = 'NIS' THEN 'CANT DTR NIS'
                  WHEN TSLOCKMODE = 'NSS' THEN 'CANT DTR NSS'
                  WHEN TSLOCKMODE = ' SS' THEN 'CANT DTR  SS'
                  ELSE SPACE(12)
             END
```


Now, I want to check locking – lets look for locking

All we need to do is to add another concatenated element to select and add the title to the header.

How to Write and Use Customized EXPLAIN for Your Applications

More customization:

Old line in select statement:

```
ACCESSTYPE                                                    || 'I'  ||
```

New lines:

```
-----Begin ACCESS TYPE  customization – show "!!"  if R and table is physical DB2
    table  -----
CASE WHEN ACCESSTYPE = 'R' AND TABLE_TYPE <> 'W'
            THEN 'R!'                                                 ||'!'
          ELSE ACCESSTYPE                                             ||'|'
      END                                                             ||
---Continue ACCESS TYPE  customization – show "!!"  if " I " and matching column = 0

      CASE WHEN SUBSTR(CHAR(MATCHCOLS),1,2) = '0' AND METHOD = 3
            THEN '  |'
           WHEN SUBSTR(CHAR(MATCHCOLS),1,2) = '0' AND ACCESSTYPE<>' '
               AND ACCESSTYPE<>'R'
            THEN SUBSTR(CHAR(MATCHCOLS),1,1)                          ||'!!'
           WHEN MATCHCOLS IS  NULL OR ACCESSTYPE='R'
            THEN '  |'
           ELSE SUBSTR(CHAR(MATCHCOLS),1,2)                           ||'|'
      END                                                             ||
-----End ACCESSTYPE customization -----
```


Now let's show when we can suspect bad access path:

For example:

We want to see "!!" in report if:

Access type is R and we are accessing physical DB2 table or access type is I and we are do not have any matching column.

If we want to exclude any !! if table size is less then a certain number of rows, or exclude any table by name it can be done in one place and in 5 minute.

What we got is an example of the pilot SQL that we discuss earlier.

How to Write and Use Customized EXPLAIN for Your Applications

- Results of customized EXPLAIN I use:

QN	BL	PL	METHD	TBL_NAME	TY	MC	ACCESSNM	P	I	T	STAT RWS	CL	IDXROWS	INDCOLNM	FRST KEY	STAT DT	LOCKING	NSU	NSJ	NSO	NSG	JTY	TTY	C NM	BIND DT	PROGRAM
969	1	1	0-TBA	CBA_CI_CTCOV	I	1	XMBD0332		N	S	34670631	49	4846947	CTHLD_SSN_ID	4846947	02/12/07	INT SHR LOCK	N	N	N	N		T	A	05/20/06	ECOSB171
969	1	2	1-NLP	CBA_CI	I	1	XMBD030C		N	S	11547815	100	11547815	CI_ID	1154781 5	02/12/07	INT SHR LOCK	N	N	N	N		T	B	05/20/06	ECOSB171
969	1	3	1-NLP	CBA_DERV_CTH	I	1	XMBD1213		Y	S	18326400	99	18854813	CI_ID	11547805	12/11/06	INT SHR LOCK	N	N	N	N		T	C	05/20/06	ECOSB171
														INT_CNTR_ID												ECOSB171
														CI_REL_C												ECOSB171
969	2	1	0-TBA	CBA_CI_CTCOV	I	1	XMBD033C		Y	C	34670631	100	34670631	CI_ID	11547805	02/12/07	INT SHR LOCK	N	N	N	N		T	D	05/20/06	ECOSB171
														INT_CNTR_ID												ECOSB171
														INS_LIN_C												ECOSB171
														CTCOV_EFF_DT												ECOSB171
2311	1	1	0-TBA	CBA_CTCOV	I	0!!	XMBD037C	S	Y	S	652547	98	897673	INT_CNTR_ID	275057	08/04/03	INT SHR LOCK	N	N	N	N		T	C	05/20/06	ECOSB172
														INS_LIN_C												ECOSB172
2311	1	2	2-MSC	CBA_CI_CTCOV	R!!			S	N	S	34670631					02/12/07	INT SHR LOCK	N	Y	N	N		T	A	05/20/06	ECOSB172
2334	1	1	0-TBA	CBA_CI_CTCOV	R!!			S	N	S	34670631					02/12/07	INT SHR LOCK	N	N	N	N		T	A	05/20/06	ECOSB172
2334	1	2	1-NLP	CBA_CTCOV	I	1	XMBD037C		Y	S	652547	98	897673	INT_CNTR_ID	275057	08/04/03	INT SHR LOCK	N	N	N	N		T	C	05/20/06	ECOSB172
														INS_LIN_C												ECOSB172
2360	1	1	0-TBA	CBA_CI_CTCOV	R!!			S	N	S	34670631					02/12/07	INT SHR LOCK	N	N	N	N		T	A	05/20/06	ECOSB172
2360	2	1	0-TBA	CBA_CTCOV	N	2	XMBD037C		Y	C	652547	98	897673	INT_CNTR_ID	275057	08/04/03	INT SHR LOCK	N	N	N	N		T	C	05/20/06	ECOSB172
														INS_LIN_C												ECOSB172
2383	1	1	0-TBA	CBA_CI_CTCOV	I	0!!	XMBD0333	S	N	S	34670631	99	34347234	CI_ID	11326211	02/12/07	INT SHR LOCK	N	N	N	N		T		05/20/06	ECOSB172
														INT_CNTR_ID												ECOSB172
														INS_LIN_C												ECOSB172
														CTCOV_EFF_DT												ECOSB172
														UPDT_SEQ_N												ECOSB172
														CTCOV_CAN_DT												ECOSB172
2412	1	1	0-TBA	CBA_CLI	I	1	XMBD034C		Y	S	99005	60	99005	CLI_ID	99005	12/12/05	INT SHR LOCK	N	N	N	N		T	CLI	05/20/06	ECOSB172
2412	1	2	1-NLP	CBA_CNTR	I	1	XMBD0352		N	S	292240	69	292240	CLI_ID	92712	01/15/07	INT SHR LOCK	N	N	N	N		T	CNTR	05/20/06	ECOSB172
														CNTR_ID												ECOSB172
2423	1	1	0-TBA	CBA_CI_INT_C	I	1	XMBD098C		Y	S	18013090	99	18854813	INT_CNTR_ID	195645	11/13/06	INT SHR LOCK	N	N	N	N		T	CIC	05/20/06	ECOSB172
														CI_ID												ECOSB172

Conclusions

- The most valuable part of your application is your data. Interfaces change often, data stay the same.
- Understanding data structure and underlying business rules is the key to application health.
- Your data are stored in a very powerful database, DB2, use SQL to maintain and access the data.
- “Pilot SQL” is a very efficient and reliable way of retrieving, validating, and maintaining the data.

Acknowledgment:

A special thank you to Joseph Burns for inspiring me to give this presentation, and for all his help.

Session: G07

DB2 Application Development from the Programmer's Point of View

Boris Ioffe

Highmark Blue Cross Blue Shield

boris.ioffe@highmark.com