**The University of British Columbia**
**Department of Computer Science**

# CPSC 404: Assignment #2
# Topics: SQL Catalog Queries for DB2; B+ Trees; Extendible and Linear Hashing

Last Updated on Feb. 4, 2013 @ 19:00

```
Feb. 4  @ 19:00:  Part A: examples of output (but your column names can differ)
Jan. 31 @ 17:25:  Part A, Q4: include the database name, but not the DB creator
Jan. 30 @ 13:55:  Created
```

**Due Date:** Thursday, February 7, 2013 at 11:00 AM

**Where to Hand it in:** In the metal hand-in box #10 in room ICCS X235 where the other new, metal hand-in boxes are located. This is also the place to hand in late assignments.

**Late Penalty:** 20% (of the assignment's maximum value) per school day, where 11:00 AM to (next day's) 11:00 AM is the extent of one school day for this assignment. For example, if you submit your late assignment to the hand-in box by Friday, February 8 @ 11:00, then that's a 20% deduction (so, if you would have gotten 90% if you were on-time, you'll now get 90-20=70%). Weekends count as one day. **We won't accept late assignments after 11:00 AM on Monday** (I know that Feb. 11th is a holiday, but you can still get into the X-Wing building with your fob or student card, or e-mail me your late assignment) … because we want to post the solutions before Wednesday, Feb. 13th's midterm. I plan to post the solutions around noon on Monday.

The assignment you submit should be done on your own, not in pairs or groups, and not copied from any other source. You can discuss questions with each other as long as you don't copy answers. Please refer to the UBC CPSC rules on plagiarism if you have any questions.

NOTE: **For all of the calculation-based questions in this assignment, show your work**; otherwise, you may get no marks. In case of ambiguity, write down any (reasonable) assumptions that you make.

## Part A: DB2 Catalog Queries using SQL

For this question, we will use DB2 catalog data for IBM's enterprise-class servers running DB2 on the z/OS operating system. One of the outcomes of this question is to give you some experience in actually exploring the contents of what's in an RDBMS catalog. This will also give us a head start in thinking about query optimization—a major part of this course.

Go to this Web site: **http://publib.boulder.ibm.com/epubs/pdf/dsnsqk18.pdf**
(Note that the second last character of "dsnsqk18" is the number one. "dsn" is the prefix that IBM uses for its DB2 program products. This manual is the *SQL Reference* for DB2 Version 9.1 for z/OS, published in June 2010.)

Once you open this URL, expand the Appendix (in the PDF index on the left hand side), and then expand "DB2 Catalog Tables". There are 80+ tables, but to limit your search, we are only interested in the schemas, including descriptions, for some (but not necessarily all) of the following DB2 catalog tables:

- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSCOPY
- SYSIBM.SYSDATABASE
- SYSIBM.SYSDBRM
- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART
- SYSIBM.SYSKEYS
- SYSIBM.SYSTABLES
- SYSIBM.SYSTABLESPACE

Why is this exercise useful? In this course, we will frequently encounter a program that is a crucial component of an RDBMS. That program is called an *optimizer*. It tries to determine the optimal access path when accessing data in a database. An optimizer obtains statistics or *metadata* (i.e., data about data) from a data repository in the DBMS; that repository is called a *catalog*. The catalog will have information about the columns of tables and indices; the number of rows in a table; the numbers, types, and (possibly) distribution of values for columns in the tables; and many, many other pieces of information.

As we've discussed in class, DB2's RUNSTATS utility is a DBMS program that is used to update the data in the DB2 catalog. For example, a DBA might execute the following command to capture statistics for the AB2005TS tablespace (including all tables) and any associated indexes—in the HEAVYOIL database:

RUNSTATS TABLESPACE HEAVYOIL.AB2005TS INDEX(ALL)

The catalog tables are part of IBM's DSNDB06 database. In case you come across the term "partitioned" or "partitioning" while reading the IBM documentation: none of the tables, tablespaces, or indexes that we will be seeking information about are partitioned. Similarly, you can assume that none are "segmented".

For the questions below, unless told otherwise, a table or index should always be qualified with its creator ID (sometimes called the "owner" or an authorization ID ("authID")). Why? Within a given database, *there may be more than one table* called PAYROLL or ACCOUNTS_PAYABLE. For instance, in the DB2TEST subsystem, we might have an HR_TEST_DB database. Within that DB, there may be these tables: TEST.PAYROLL (e.g., integration testing), W145.PAYROLL (e.g., the developer with authID W145 is using his/her own copy of the PAYROLL table), etc. The qualifiers TEST, W145, etc. refer to the owners of these versions of the PAYROLL table; in other words, they are examples of creator IDs or authIDs for the PAYROLL table. Note that there could be 10-20 "PAYROLL" tables out there, most of which will be for developers' testing purposes. (The production version of PAYROLL with the live data might be called PROD.PAYROLL running on the DB2PROD subsystem).

Note: The IBM manual calls some of the creator ID (or authID) fields, "The schema of the table, view, or alias", in the description of the attribute CREATOR. This is an unfortunate choice of words, but what they really mean that it's "the authorization ID of the owner of the table, view, or

alias". CREATOR differs from the CREATEDBY attribute (also present in some of the same catalog tables), as the following example shows:

> PROD.PAYROLL is the production version of the payroll table, and PROD is the owning authID (or the qualified table name). The person who actually ran the job to create this table might have been the DBA with authid ZEMK06. What this means is that DBA ZEMK06 submitted the job with the proper permissions (users typically don't get to create tables or other objects) ... but all the users are going to call the table by its qualified/owning name: PROD.PAYROLL ... rather than ZEMK06.PAYROLL. Thus, we use the CREATOR field to qualify the table name because we're really interested in the PROD.PAYROLL table, and there is no ZEMK06.PAYROLL table. (Actually, DBAs can share a "PROD" group userid, and therefore create tables with the PROD qualifier.)
>
> The bottom line is: for some questions, you may need to use the CREATOR field to list the fully qualified table name: PROD.PAYROLL, rather than the CREATEDBY field. Don't worry about specifying the DB2 subsystem that the query will run on (e.g., ignore DB2PROD vs. DB2TEST).

**What is your task for this part of the assignment?** **Create the following SQL queries** based on the information that you see in the relevant DB2 catalog tables. You may need to consult your textbook to refresh your SQL knowledge from CPSC 304. Note that there is a find feature in IBM's DB2 catalog tables because the manual is actually a PDF file (which is easily string-searchable). Note that there are some examples in the practice questions/solutions online.

1) Write an SQL statement to list all tables that have userid "C3PO" as the creator. List their database name, tablespace creator ID, and tablespace names, too, as well as the number of pages in the table that actually contain data (as of the last time that RUNSTATS was run). Sort the output by database name first, and then within database name, by tablespace name, and then by table name.

   Example: Your headings may differ (the default column names are fine, I just don't want to give away the column names):

| DB | TS AuthID | TS Name | Table | Pages of Data |
|---|---|---|---|---|
| PAYROLL | ABC | TS14068 | PAYCALC | 1500 |
| PAYROLL | ABC | TS14077 | SCHED | 381 |
| PAYROLL | DEF | TS4IGEN | PAYCALC | 30 |

2) You're interested in a table, but you can't remember its name. You do, however, happen to remember that it has an attribute called "BONUS". Write an SQL statement to list the names of all tables (with their creator) that have a column with that name. Include the data type (e.g., integer) of the BONUS column in your answer, too.

| AuthID | Table | Data Type |
|---|---|---|
| PROD | PAYROLL | Decimal |
| PROD | BENEFITS | Decimal |
| PROD | UNIONHRS | Integer |
| PROD | MANAGERS | Integer |

3) Write an SQL statement to list the last time that RUNSTATS was run on each of the *tables* in the OILSANDS database. Order then from least recently to most recently run. Be sure to include tables which have never had RUNSTATs run on them (they'll appear first in the list). Be sure to list the database name (but not its creator), the tablespace creator, and the tablespace name for these tables, in addition to the table's name and creator.

| DB | TS AuthID | TS Name | Tab AuthID | Table | RunstatsTime |
|---|---|---|---|---|---|
| … | | | | | … |
| OILSANDS | ABC | TS14068 | ABC | BITUMIN2 | 2012-07-22.04.01.13.258810 |
| OILSANDS | AAX | TS14X68 | ABC | COLDLAKE | 2012-07-27.05.22.56.233842 |

4) Which indexes have the highest number of leaf pages that are located physically far away from their previous leaf pages (i.e., when the leaf pages are accessed from left to right at the leaf level). Recall from our B+ tree lecture slides that the leaf pages are all connected by pointers, when going from left to right. The pictures in the lecture slides are in logical order, but those pages don't have to be beside each other consecutively on disk—they could be a long way away from each other. This would obviously affect performance. Provide the table creator and table name and database name (but not the DB creator).

   Order the results by the highest number of leaf pages that are located far away (i.e., descending order), and break ties by ordering ties according to the lowest percentage of freespace on a leaf page (in ascending order)—within the same query. For example, if two rows have 2500 far away pages, and one has 8 percent free space and the other has 4 percent freespace (with appropriate additional fields):

| Ind AuthID | Index | Tab AuthID | Table | DB | #Dist Pages | %Free |
|---|---|---|---|---|---|---|
| D6Z | CUSTIX6 | D6Z | CUSTOMERS | CUSTDB | 2500 | 4 |
| B64R | ACUSTX2 | B64R | ACPAYABLE | FINANCE | 2500 | 8 |
| D6Z | CUSTIX9 | D6Z | CUSTOMERS | CUSTDB | 2483 | 2 |

5) A table is clustered according to the key(s) found in a particular clustering index. Write an SQL statement to tell us which indexes that allow duplicate index keys are supposed to be clustering indexes, but for which less than 90% of the corresponding rows in the data table are actually in clustering order at the time RUNSTATS was last run. This will give us an indicator of which tables need reorganization. Determine an appropriate list of attributes in your query, including the index, table, and database name. Feel free to make appropriate assumptions. Ignore those indexes which have never had RUNSTATS run against them.

| Ind AuthID | Index | Tab AuthID | Table | DB | % in Clust. Order |
|---|---|---|---|---|---|
| D6Z | CUSTIX6 | D6Z | CUSTOMERS | CUSTDB | 86.5 |
| B64R | ACUSTX2 | B64R | ACPAYABLE | FINANCE | 81.3 |
| TST | IGEN99 | TST | ACRECV | FINANCE | 80.9 |

## Part B: B+ Trees

1) Starting with an empty B+ tree root node, of order 2, insert the following keys in the given sequence into the B+ tree, showing the tree structure at the time each node split occurred. You can continue to fill the nodes with subsequent keys, until once again, a split occurs—in which case, you'd show the B+ tree structure again.

12, 14, 16, 18, 15, 26, 20, 50, 9, 13, 51, 1, 56, 60

2) Ignoring the keys in part (a), what is the *minimum* number of keys that can be added to the empty root node to force the tree to go to 3 levels? Provide a sequence of such keys.

3) How many Alt. 2, 4K pages are needed if we plan to build a B+ tree index on a 24-byte character field for a table that has 2,000,000 records of size 130 bytes each? Show your calculations. Use a 10-byte rid. No record or index/data entry can be split across 2 pages. Also, for each level of the tree, compute the number of nodes that would be at that level, assuming we attempt to fill nodes to capacity.

4) How many 4K data pages are needed for the *data table* in Question (3)?

## Part C: Extendible Hashing

1) Start with an *empty* extendible hashing index directory with directory entries 0 and 1, each pointing to one of two distinct, empty buckets. The capacity of each bucket will be 3 data entries.

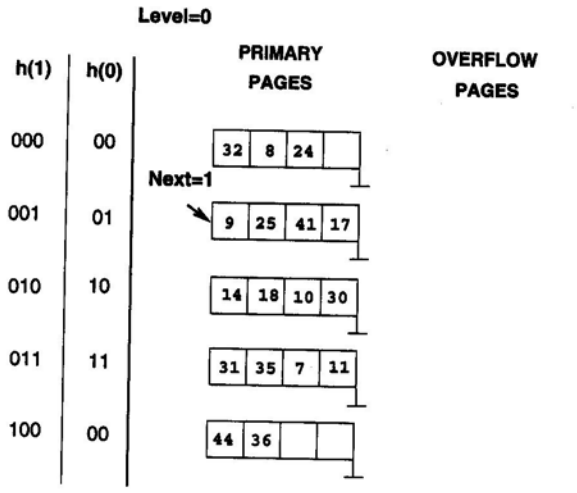
Show the contents of the index (directory, buckets, entries, nubs, etc.) after each change occurs to any of the nubs of the index structure, assuming we insert the following sequence of keys, in this order:

1, 7, 12, 0, 6, 13, 8, 4, 9

Use the same hash function as shown in class (i.e., last *k* bits of the binary value of the key).

## Part D: Linear Hashing

Consider the following linear hashing index taken from the textbook:




1) Show the index after inserting the key values 0, 4, 15, and 12 into the above (existing) hash structure, in that order.

2) Find a sequence of entries whose insertion into the original index would lead to a bucket with 3 overflow pages. Use as few entries as possible to accomplish this. Explain your answer.