Oracle Rdb Technical Forums

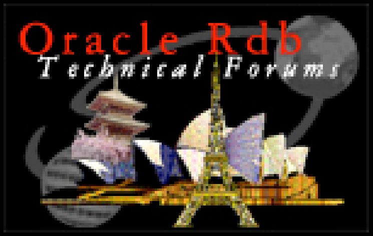
Optimizer Update

Jim Murray

Oracle New England Development Centre

Agenda

- Query Timeout.
- Index Prefetch.
- Peephole Optimization.
- Bitmapped Scan Performance Enhancements.
  - Improved processing of unique keys.
  - Enhanced Fast First tactic.
- Dynamic Optimizer Fast First Shortcut.
- Bitmapped Scan For OR Index Retrieval.



Query Governor Enhancement Execution Timeouts

- Abort long running queries
  - Elapsed time
  - CPU time
- SQL Interface
  - SET QUERY EXECUTION LIMIT {CPU|ELAPSED} n {SECONDS|MINUTES}
- RMU/SHOW STATISTICS
  - “Terminate Request” option on “Tools” menu
- Available in 7.1.2.4 and later

Query Timeout

- Timeout execution limit:

```
%RDB-E-EXQUOTA, Oracle Rdb runtime quota exceeded
-RDMS-E-MAXTIMLIM, query governor maximum timeout
has been reached
```

- Timeout forced from RMU:

```
%RDB-E-EXQUOTA, Oracle Rdb runtime quota exceeded
-RDMS-E-REQCANCELED, request canceled
```

New in 7.2
Index Prefetch

- Range retrievals only
  - SELECT C1 FROM T1 ORDER BY C2;

    ```
    Get     Retrieval by index of relation T1
    Index name  I1 [0:0]
    ```
- Prefetch done for:
  - Index nodes
  - Data pages pointed to by entries in index nodes
- Prefetch not done for:
  - Index nodes in a cached index
  - Data rows in a cached table
  - DDL (metadata) lookups
  - Dynamic optimizer strategies

New in 7.2
Index Prefetch

- Test results for a “worst case” index showed about a 40-60% improvement in elapsed execution time, depending on disk speed
- CPU consumption difference is essentially nil
- Underlying caches (like XFC) have big impact on actual improvement
  – No improvement if no disk I/O performed

Peephole Optimization for Hidden Key Retrieval

- The problem:
  - Column references inside functions can prevent optimal retrieval strategies.
  - Note the full [0:0] index scan.

```
SQL> select last_name from employees where cast (last_name
as varchar (31)) = 'Smith`;
Tables:
  0 = EMPLOYEES
Conjunct: CAST (0.LAST_NAME AS VARCHAR (31)) = 'Smith'
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_NAME [0:0]
 LAST_NAME
 Smith
 Smith
2 rows selected
```

Peephole Optimization for Hidden Key Retrieval

- Optimzer “peeks” into the predicate to see if these operators are present (may be deeply nested), and to use the hidden base column, if detected, as the index retrieval.
- Note that the notation “Hidden Key” is displayed to indicate that the key is retrieved by the index lookup scan performed on the index key (ikey) of the base column hidden under the function.
- Disabled with set flags ‘NOHIDDEN_KEY’.

Peephole Optimization for Hidden Key Retrieval

- Works for:
  - equals (=)
  - greater than (>)
  - less than (<)
  - greater or equal (>=)
  - less than or equal (<=)
  - IS NULL
  - IS NOT NULL
- On the functions:
  - CAST(? AS CHAR(n))
  - TRIM(TRAILING FROM ?)
  - STARTING WITH and LIKE

Peephole Optimization for Hidden Key Retrieval

- Note the “Hidden Key” and [1:1] scan.

```
SQL> select last_name from employees where cast
(last_name as varchar(31)) starting with 'Smith';
Tables:
  0 = EMPLOYEES
Conjunct: CAST (0.LAST_NAME AS VARCHAR(31))
STARTING WITH 'Smith'
Index only retrieval of relation 0:EMPLOYEES
  Index name  EMP_LAST_NAME [1:1]  Hidden Key
     Keys: 0.LAST_NAME STARTING WITH 'Smith'
 LAST_NAME
 Smith
 Smith
2 rows selected
```

Bitmapped Scan

- Processing of unique keys and non-ranked indexes.
  - Used to add one Dbkey at a time into the BBC.
  - Very CPU intensive.
  - Now uses a 1024 Dbkey buffer.
  - Used when any key/index does not have a BBC duplicates.
  - Sorts and rolls dbkeys into a BBC:
    - At end of index scan.
    - When Dbkey buffer fills.
  - Execution trace shows Bld_Map trace.
  - Up to 75% CPU reduction for some queries.

Bitmapped Scan

- Old Dbkey at a time processing of an index

```
~E#0001.01(1) Estim   Index/Estimate 1_1 2/29 3/29
~E#0001.01(1) BgrNdx1 FillMap2  DBKeys=1 Fetches=1+0
~E#0001.01(1) BgrNdx1 FillMap2  DBKeys=1 Fetches=0+0
~E#0001.01(1) BgrNdx1 Or__Map2  DBKeys=2 Fetches=0+0
~E#0001.01(1) BgrNdx1 FillMap2  DBKeys=1 Fetches=0+0
~E#0001.01(1) BgrNdx1 Or__Map2  DBKeys=3 Fetches=0+0
...
~E#0001.01(1) BgrNdx1 Or__Map2  DBKeys=19 Fetches=0+0
~E#0001.01(1) BgrNdx1 EofData   DBKeys=19 Fetches=1+0
```

- New Behavior

```
~E#0001.01(1) Estim   Index/Estimate 1_1 2/29 3/29
~E#0001.01(1) BgrNdx1 EofData   DBKeys=19 Fetches=1+0
~E#0001.01(1) BgrNdx1 Bld_Map2  DBKeys=19 Fetches=0+0
```

ORACLE

Bitmapped Scan

- Enhanced Fast First
  - Old Behaviour
    - Background (Bgr) scans an index passing dbkeys for Foreground (Fgr).
    - Fgr fetches the rows, filters, and possibly delivers.
    - Fgr has 1024 dbkey list of delivered rows.
    - Bgr has dbkey BBC and must add dbkeys one at a time.
    - Final (Fin) uses Bgr dbkey BBC to fetch and deliver rows not already delivered by Fgr.

Bitmapped Scan

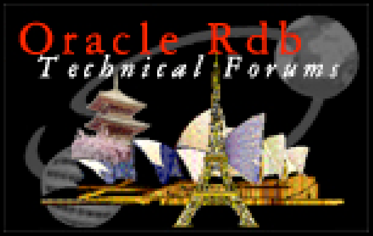
- Enhanced Fast First
  - Limitations:
    - If many rows are not delivered Bgr BBC can contain many dbkeys for unwanted rows that have already been fetched and filtered by Fgr.
    - Fin will have to fetch and re-test any dbkey not in Fgr dbkey list.
    - Maintaining BBC costly.
    - It is known that Fgr has processed these rows, so simply wasted effort.
    - Most costly if many rows are rejected (Bgr Dbkey BBC much bigger than Fgr).

Bitmapped Scan

- Enhanced Fast First
  - New behaviour:
    - Bgr dbkey BBC not maintained while Fast First is running.
    - If Fast First is abandoned, new dbkeys are recorded (buffer and or BBC).
    - Fin only has to process dbkeys fetched after Fast First is terminated.
  - When First Bgr index is an OR index list:
    - The same dbkey could be read from two different OR indexes.
    - Must maintain Dbkeys BBC to prevent duplicates.
    - Discarded when Fast First finishes.
  - Saves CPU time during Fast First.
  - Saves IO and CPU during Fin.

Fast First Shortcut

- Old Behaviour:
  - Bgr Scans 1st Bgr index passing dbkeys to Fgr.
  - Fgr fetches, filters and potentially delivers row.
  - If 1st Bgr index scan completes scan 2nd Bgr index.
  - Fast First only turned on for 1st Bgr index.
  - Fin uses Bgr dbkey list to fetch filter and deliver rows not in Fgr dbkey list.
- The problem:
  - If the first Bgr index scan completes then we know that all possible rows have been delivered.
  - Scanning additional indexes is redundant
  - E.G “where last_name=‘Smith’ and first_name=‘John’
  - If we have processed all ‘Smith’ rows in FFirst then we don’t need to read the first_name index.
- Now abandon Fin if fast first is still running.



Fast First Shortcut

- In the following execution trace, the Fin phase does not execute because Fast First was still running when the first index scan completed.

```
~E#0003.01(1) Estim   Index/Estimate 1/17 2/46
~E#0003.01(1) BgrNdx1 EofData  DBKeys=63  Fetches=0+0  RecsOut=0
#Bufs=24
~E#0003.01(1) FgrNdx  FFirst   DBKeys=0  Fetches=0+23  RecsOut=0`ABA
~E#0003.01(1) Fin     Buf_Ini  DBKeys=0  Fetches=0+0  RecsOut=0`ABA
```

Special Image 02 for V7.1.3

- Fast First shortcut exposed a problem where the 1025th row could fail to be delivered.

```
SQL> select * from t1 where f1=1;
~E#0000.00(2) Estim   Index/Estimate 1/2
~E#0000.00(2) BgrNdx1 EofData  DBKeys=2  Fetches=0+0  RecsOut=0 #Bufs=1
~E#0000.00(2) Fin     Buf      DBKeys=2  Fetches=0+1  RecsOut=2
~S#0001
Leaf#01 FFirst T1 Card=1025
  BgrNdx1 I1 [1:1] Fan=17
~E#0001.01(1) Estim   Index/Estimate 1/2050
          F1             F2
           1              1
...
           1              1
~E#0001.01(1) BgrNdx1 EofBuf   DBKeys=1024  Fetches=0+3  RecsOut=1024
~E#0001.01(1) BgrNdx1 EofData  DBKeys=1025* Fetches=0+0  RecsOut=1024
#Bufs=10
~E#0001.01(1) FgrNdx  FFirst   DBKeys=1024  Fetches=0+12  RecsOut=1024`ABA
~E#0001.01(1) Fin     TTblIni  DBKeys=0  Fetches=0+0  RecsOut=1024`ABA
           1              1
1024 rows selected
```

Bitmapped Scan Or Index Retrieval

- In the past, would only used bitmapped scan if there was at least one AND index.
- Would not use bitmapped scan for a simple OR query.
- E.g. select * from employees where last_name=‘Smith’ or first_name=‘John’;
- Would use a ‘static’ OR tactic instead.
- Now uses bitmapped scan for simple OR queries if bitmapped scan is enabled.

Bitmapped Scan OR Index Retrieval

- This is a simple example of Bitmapped OR:

```
set flags 'strategy,detail,bitmap'
select count(*) from employees
  where (employee_id>'00300‘ or last_name>'L');
Tables:
  0 = EMPLOYEES
Aggregate: 0:COUNT (*)
Leaf#01 BgrOnly 0:EMPLOYEES Card=100     Bitmapped scan
  Bool: (0.EMPLOYEE_ID > '00300') OR (0.LAST_NAME > 'L')
  BgrNdx1 EMP_EMPLOYEE_ID [1:0] Fan=17
    Keys: 0.EMPLOYEE_ID > '00300'
      OrNdx1 EMP_LAST_NAME [1:0] Fan=12
        Keys: 0.LAST_NAME > 'L'

          56
1 row selected
```

For More Information

- www.oracle.com/rdb
- metalink.oracle.com
- www.hp.com/products/openvms
- mark.bradley@oracle.com
- jim.murray@oracle.com
- paul.mead@oracle.com
- michael.ong@oracle.com

QUESTIONS
&
ANSWERS

ORACLE®