Access Path Selection and Physical DB Design in Relational Database Management Systems:
An Overview

Padua University – 10 June 2005

Angelo Sironi
angelo_sironi@it.ibm.com

Contents

- RDBMS and SQL Language
  - Static SQL and Dynamic SQL
- Mapping the Physical Design
- Query Compiler Overview
- Query Optimization
  - Access Path Selection
  - Query Rewrite
  - Examples
- The Optimizer Explained
- The Role of the DBA

Relational Model & Relational Database Management Systems

- The relational model is the application of
  - **predicate logic**

    and
  - **set theory**

  to database management
- Theory: **predicate logic** and **set mathematics**
  - Structure : **R-tables**
  - Integrity : **domain**, **column**, **table** and **database** integrity
  - Manipulation: **R-operations** (restrict, project, join, etc.)

a. sironi

R-operations: The SQL Language

```
SELECT *
  FROM Today_Orders
```

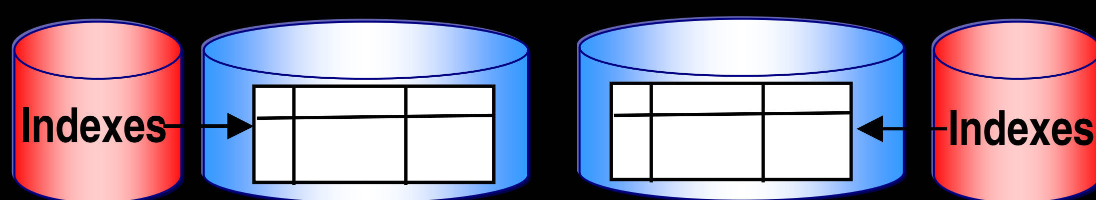
- Query constructs
  - Set Oriented
  - Logical “expressions”
    - NO reference to any physical structures
    - Functionally independent from physical structures
- Query performance
  - Highly (but not fully) independent from language constructs
  - Highly dependent on DB physical design

a. sironi

R-operations: Mapping the Physical Design

```
SELECT *
  FROM Today_Orders
```

```
CREATE VIEW Today_Orders AS
SELECT O.CUST#, O.ORD#, RO.PROD#, RO.QTY
  FROM Orders       O
     , Order_Lines  RO
 WHERE O.ODR# = RO.ORD#
   AND O.ORDER_DATE = current date
ORDER BY O.CUST#, O.ORD#, RO.PROD#
```



a. sironi

Static SQL

- Pre-defined syntax, hard-coded into a host language (e.g., Cobol, C, C++, etc.)
  - It may reference (usually does) host-program variables
- Example

```
EXEC SQL DECLARE CRS1 CURSOR FOR
         SELECT C1, C2, C3
           FROM T1
          WHERE C4 = :c4
OPEN CRS1
FETCH CRS1 INTO :c1, :c2, :c3
…….
CLOSE CRS1
```

- Mostly used for OLTP and Batch processing, where data needs are known in advance

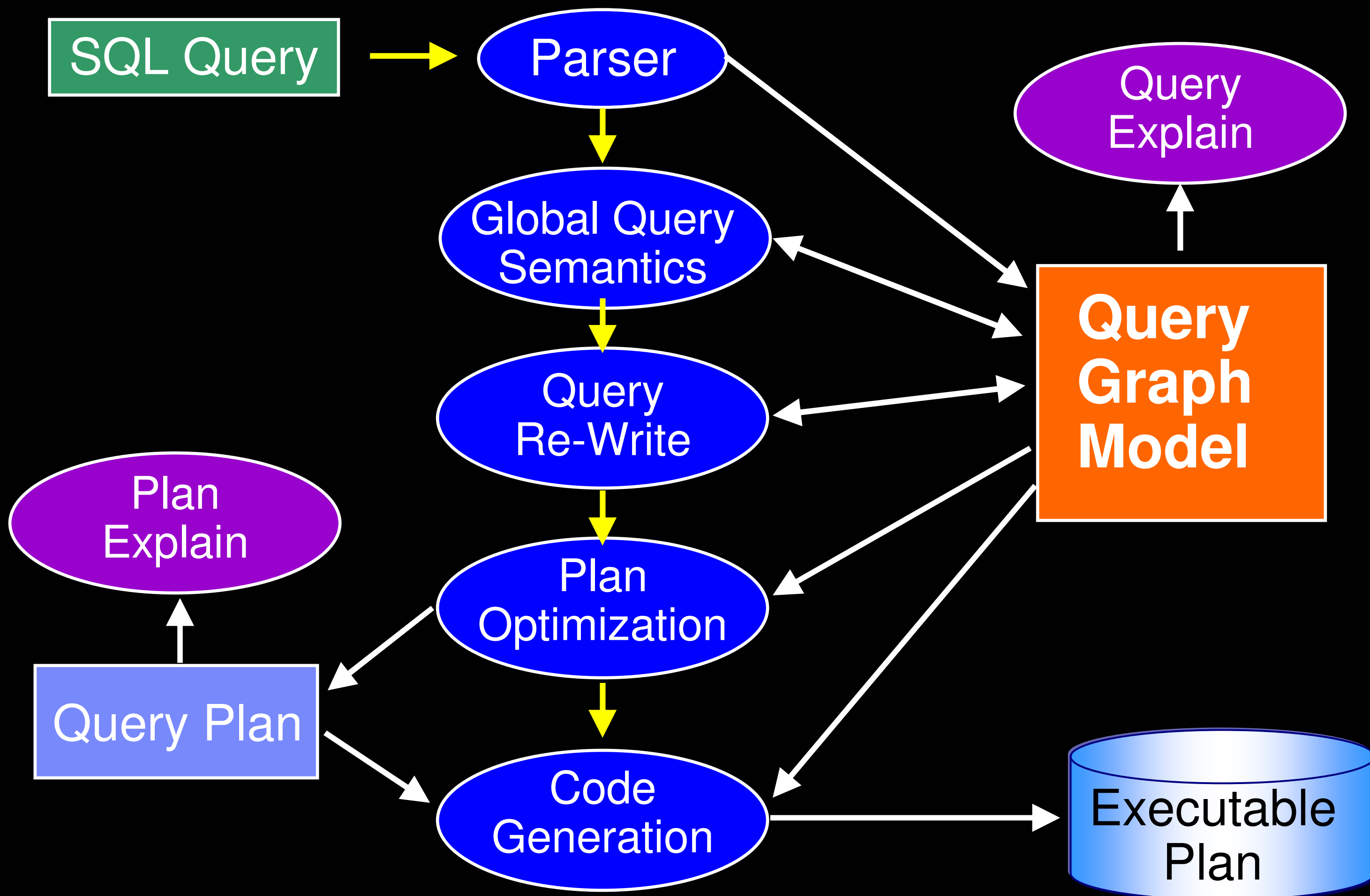
Dynamic SQL

- Data needs totally or partially unknown: e.g.
  - Query tools
- Query typed by end-user on a browser page and buffered by application program
- Example

```
EXEC SQL PREPARE STMT INTO :MYSQLDA FROM :SQLBUFFER
EXEC SQL DECLARE CRS1 CURSOR FOR STMT
EXEC SQL OPEN CRS1 USING :C
EXEC SQL FETCH CRS1 INTO :C1, :C2, :C3
….
EXEC SQL CLOSE CRS1

SQLBUFFER = 'SELECT C1, C2, C3 FROM T1 WHERE C4 = ?'
```

Query Compiler Overview



a. sironi

Steps in Query Compilation

- Parsing
  - Analyze "text" of SQL query
  - Detect syntax errors
  - Create internal query representation
- Semantic Checking
  - Validate SQL statement
  - View analysis
  - Incorporate constraints, triggers, etc.
- Query Optimization
  - Modify query to improve performance (Query Rewrite)
  - Choose the most efficient "access plan" (Query Optimization)
- Code Generation: generate code that is
  - Executable
  - Efficient

Query Compilation executed during

- Application Program BIND process (Static SQL)
- Execution of SQL Prepare (Dynamic SQL)

a. sironi

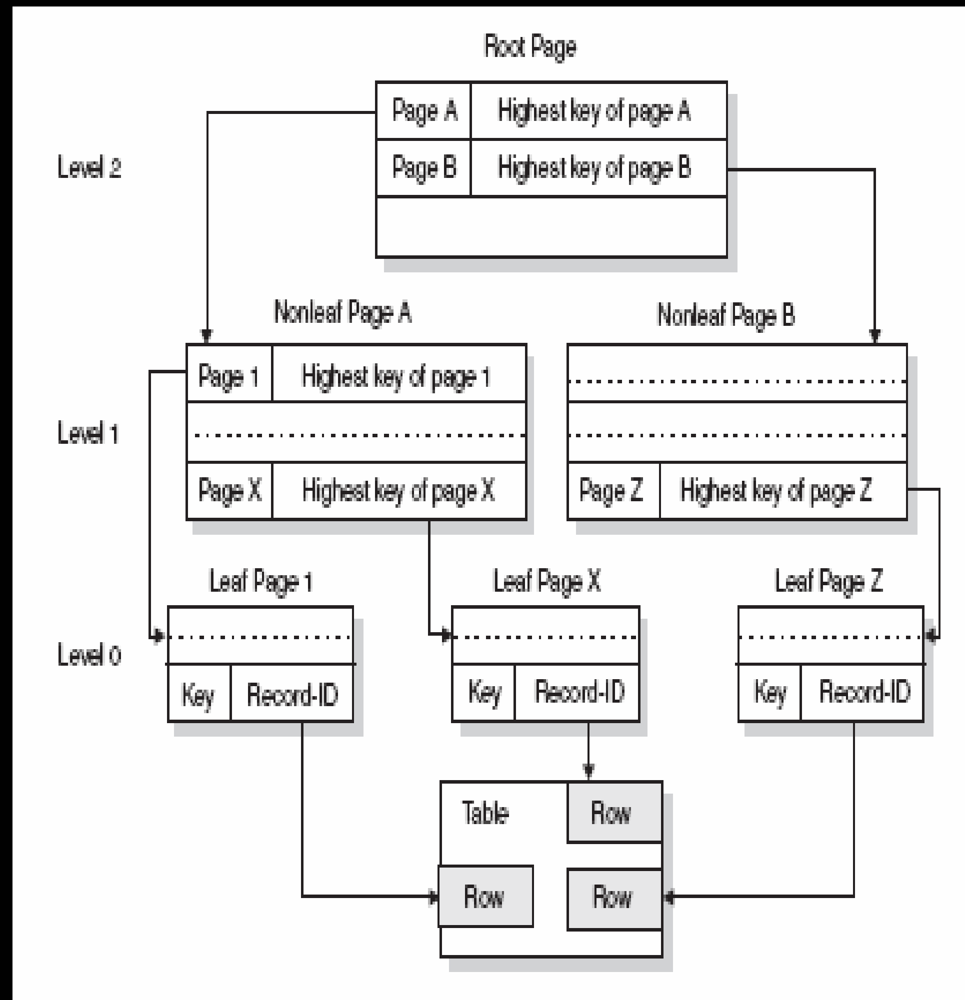
Query Optimization: Access Path Selection

- Based on estimating Query execution cost …
  - CPU
  - I/O
  - Elapsed Time
- … of Access Path variations based on
  - Using different index(es) / no index at all
  - Using different join sequence
  - Using different join method
  - Using query parallelism
  - ….
- Less expensive Access Path chosen

a. sironi

What’s an Index

- An Index is a physical structure allowing direct (and ordered) access to records matching the values of the Index Key or a portion of it
- An Index key can be
  - single-column or multi-column
  - Unique or non-unique
  - Additional options might apply (e.g. Cluster)
- Most common Indexes have a B-tree structure (see figure)
- Indexes provide performance advantage for data retrieval, but increase the cost of Delete / Insert / Update operations



a. sironi

Access Path Quiz – 1: Single Table Access

```
SELECT  C1, C2, C3, C4
  FROM  T1
 WHERE  C1 = ?
   AND  C2 = ?
ORDER BY C3
```

- T1 = Relational Table (not a View)
  - Single table in single physical file
  - TCARD = 1,000,000
  - Index IX1 on (C1), with Duplicates
  - Index IX2 on (C2), with Duplicates
  - Index IX3 on (C4), Unique
- Which Index might provide the best Access Path ?
- Any suggestions for improving the Access Path?

Access Path Quiz 1: Filter Factors

```
SELECT  C1, C2, C3, C4
  FROM  T1
 WHERE  C1 = ?
   AND  C2 = ?
ORDER BY C3
```

- FF(Col = literal) = 1 / Col-CARD
  - Col-CARD = Number of distinct values for Col
  - TCARD = Number of tuples in Table
- Example:
  - C1-CARD = 10,000; C2-CARD = 100,000
  - FF(C1 = literal) = 1 / 10,000 = 0,0001 (i.e. 0,01%)
  - FF(C2 = literal) = 1 / 100,000 = 0,00001 (i.e. 0,001%)

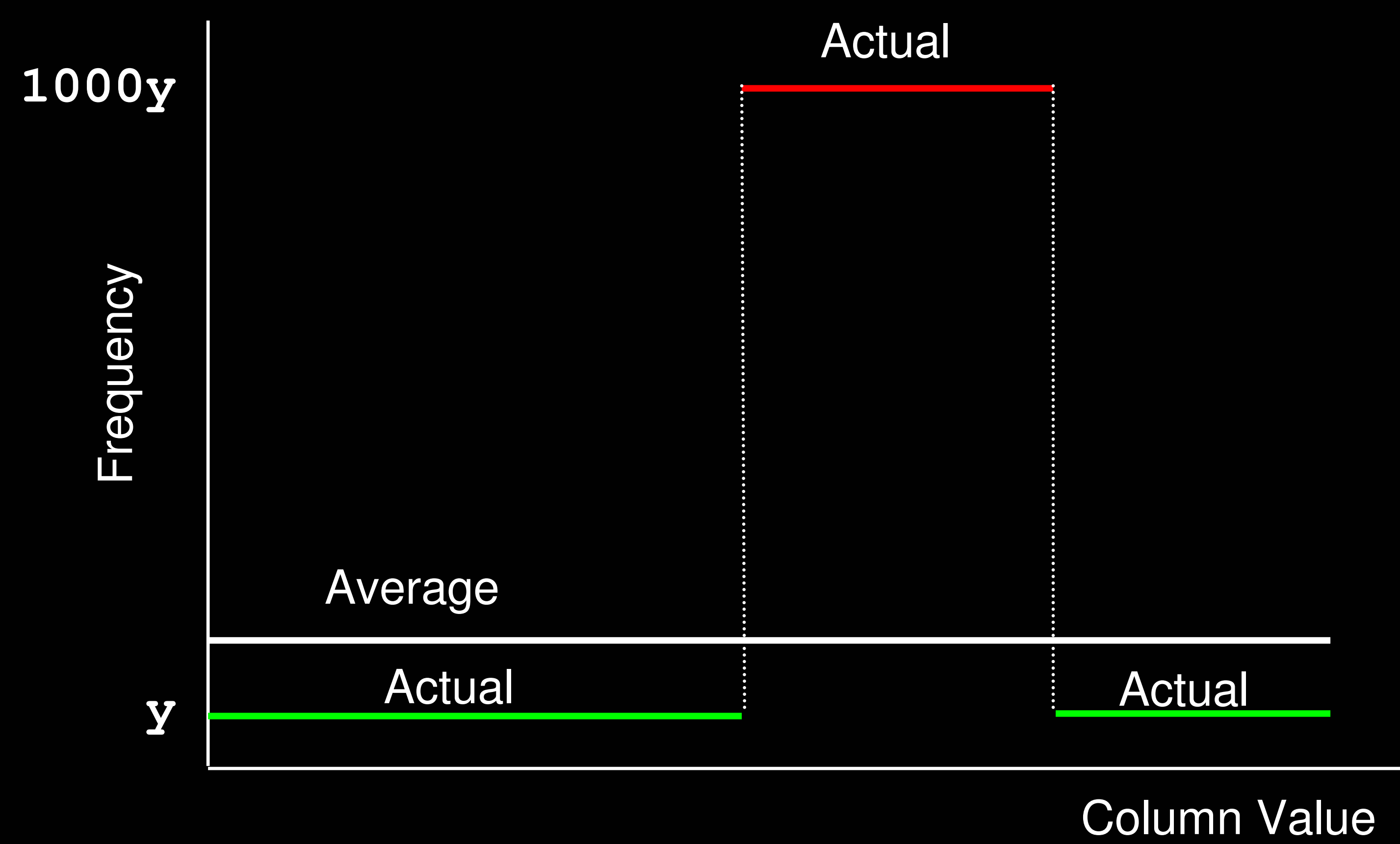
a. sironi

Access Path Quiz 1: Comparing Index Access

- Using IX1 (C1)
  - Qualifying Tuples (estimate) = $10^{**6} * 10^{**(-4)}$ = 100
  - Retrieve all 100 qualifying Tuples, apply predicate on C2, sort (ORDER BY C3)
- Using IX2 (C2)
  - Qualifying Tuples (estimate) = $10^{**6} * 10^{**(-5)}$ = 10
  - Retrieve all 10 qualifying Tuples, apply predicate on C1, sort
- What about Skewing?
- What if IX4(C1,C2)?
  - Suppose number of Distinct Keys in IX4 = 500,000
  - FF(C1 = literal AND C2 = literal) = $4 * 10^{**(-6)}$
  - Can the RDBMS derive FF(C1 = literal, C2 = literal) by its knowledge of FF(C1 = literal) and FF(C2 = literal) ?
- Any advantage, if above holds true, with IX4(C1,C2,C3) ?

a. sironi

Access Path Quiz 1: Skewing



a. sironi

Skewing: Notes

- Filter Factor assumes Uniform Distribution of Values Frequency
- That’s usually not the case in real life situations
  - RDBMS Optimizer needs to know more on values frequency (e.g. distribution statistics)
- Query performance will be highly dependent on values specified in predicates
  - For better optimization, RDBMS Optimizer must know comparison values in predicate at compile time
    - Use Dynamic SQL without parameter markers; or ..
    - Re-optimize SQL query at execution time

Access Path Quiz – 2: Join

```
SELECT   T1.C1, T1.C2, T2.C3
  FROM   T1
     ,   T2
 WHERE   T1.C1 = T2.C1
   AND   T1.C2 = ?
   AND   T2.C3 = ?
ORDER BY T1.C1
```

- T1, T2 = Relational Tables (not a Views)
  - T1-CARD = 500,000
  - T2-CARD = 5,000,000
- Filter factors
  - FF(T1.C2 = literal) = $10^{**(-4)}$
  - FF(T2.C3 = literal) = $10^{**(-6)}$
- No Indexes: How would you manage the join?
- Which Indexes would you recommend?

a. sironi

Access Path Quiz – 2: Most Common Join Methods

- Nested Loop Join (NLJ)
  - For each qualifying tuple of Tx, select matching tuples from Ty
  - Usually, an Index on Ty.JCs used
- Merge Scan Join (MSJ)
  - Sort qualifying tuples from Tx on join columns
  - Sort qualifying tuples from Ty on join columns
  - Merge tuples matching join predicates
- Hash Join (HJ)
  - Similar to NLJ
  - Hashing used to speed up retrieval of matching tuples from Ty

a. sironi

Query Rewrite

- **Rewriting a given SQL query into a semantically equivalent form that**
  - may be processed more efficiently
  - gives the Optimizer more latitude
- **Why**
  - Same query may have multiple representations in SQL
  - Complex queries often result in redundancy, especially with views
  - Query generators
    - **often produce suboptimal queries that don't perform well**
    - **don't permit "hand optimization"**
- **DB2 capabilities is based on Starburst Query Rewrite**
  - Rule-based query rewrite engine
  - Transforms legal QGM into more efficient QGM
  - Terminates when no rules eligible or budget exceded

Query Rewrite: Examples

- Equivalent predicates
  - NAME LIKE ‘a%’
  - SUBSTR(NAME,1,1) = ‘a’
- Transform Subselect into Join

```
SELECT   T1.*
  FROM   T1
 WHERE   EXISTS
  (SELECT *
     FROM T2
    WHERE T1.K = T2.K)
```

```
SELECT   DISTINCT T1.*
  FROM   T1
     ,   T2
 WHERE   T1.K = T2.K
```

- View Merge
  - Allows additional joins order
  - Can eliminate redundant joins
- Redundant join elimination
  - Satisfies multiple references to the same table with a single scan

a. sironi

Query Rewrite: Subquery Madness!

```
SELECT *
   FROM ( SELECT   FLAG, TO_NUMBER ( NUM ) NUM
              FROM   SUBTEST
            WHERE   FLAG = 'N' )
 WHERE  NUM > 0 ;
```

- ERROR: ORA-01722: invalid number
- Reason: query rewrite

```
SELECT   FLAG, TO_NUMBER ( NUM ) NUM
  FROM   SUBTEST
WHERE   FLAG = 'N'
    AND  TO_NUMBER ( NUM ) > 0 ;
```

and predicate sequence changed!!!

- “Model *vs*. implementation is one of the great **logical differences” (C. Date)**

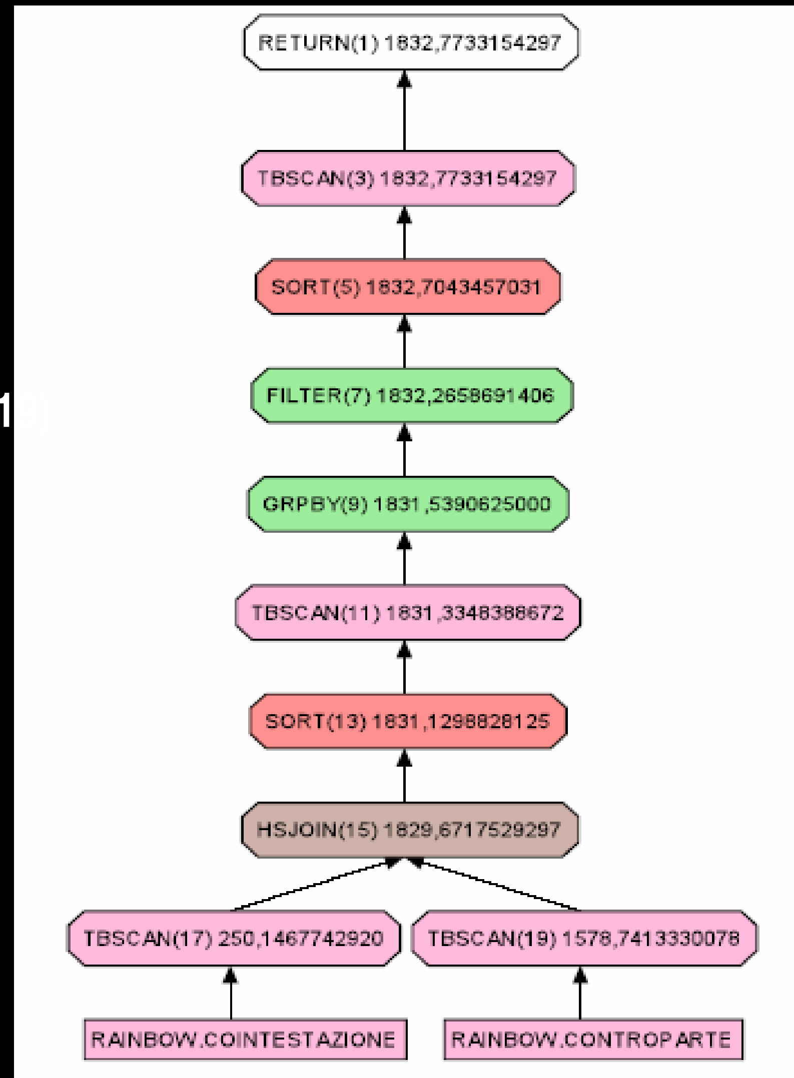
FLAG	NUM
N	100
C	Abc
N	23
N	428
C	Zdv

a. sironi

Optimizer Explained – 1a

```
select   cod_ndg_controparte, count(*)
  from rainbow.cointestazione ct
where exists (select *
                   from rainbow.controparte cp
                 where cp.idn_controparte =
                                       ct.idn_cointestatario
                   and  cp.idn_cr_tipo_controparte = 421
group by cod_ndg_controparte
having count(*) > 10
order by 2 desc
```

```
SELECT Q4.$C1 AS "COD_NDG_CONTROPARTE", Q4.$C0
FROM  (SELECT  COUNT(* ), Q3.$C0
          FROM  (SELECT  Q2.COD_NDG_CONTROPARTE
                     FROM  RAINBOW.CONTROPARTE       AS Q1
                         ,  RAINBOW.COINTESTAZIONE   AS Q2
                  WHERE (Q1.IDN_CONTROPARTE =
                                  Q2.IDN_COINTESTATARIO)
                     AND (Q1.IDN_CR_TIPO_CONTROPARTE
                                                   = 4219)) AS Q3
             GROUP BY Q3.$C0) AS Q4
      WHERE (10 < Q4.$C0)
ORDER BY Q4.$C0 DESC
```



a. sironi

Optimizer Explained – 1b (after Reorg)

```
select   cod_ndg_controparte, count(*)
  from rainbow.cointestazione ct
where exists (select *
                  from rainbow.controparte cp
                where cp.idn_controparte =
                                      ct.idn_cointestatario
                  and  cp.idn_cr_tipo_controparte = 4219)
group by cod_ndg_controparte
having count(*) > 10
order by 2 desc
```

```
SELECT Q4.$C1 AS "COD_NDG_CONTROPARTE", Q4.$C0
FROM  (SELECT  COUNT(* ), Q3.$C0
          FROM  (SELECT  Q2.COD_NDG_CONTROPARTE
                     FROM  RAINBOW.CONTROPARTE        AS Q1
                           ,  RAINBOW.COINTESTAZIONE   AS Q2
                  WHERE (Q1.IDN_CONTROPARTE =
                                  Q2.IDN_COINTESTATARIO)
                     AND (Q1.IDN_CR_TIPO_CONTROPARTE
                                                 = 4219)) AS Q3
             GROUP BY Q3.$C0) AS Q4
      WHERE (10 < Q4.$C0)
ORDER BY Q4.$C0 DESC
```

a. sironi

The Optimizer: Summary

- Very clever piece of code
- Behavior heavily dependent on logical and physical design
  - Number of joins (logical design)
  - Indexes and their characteristics
- Also dependent on
  - RDBMS configuration (e.g. buffer size, sort heap size, etc.)
  - Number of CPU and CPU power
  - I/O configuration

DBA Role: Most relevant performance knobs

- RDBMS configuration settings
  - Highly dependent on named RDBMS
- Logical design
  - Be aware of cost of over-normalization
    - Higher number of joins
    - Higher number of joined tables
- Physical design
  - Number of indexes
  - Index key
    - Be aware of the maintenance cost of each index
    - Goal: balance throughput vs. single query performance

a. sironi

DBA Role: A Method Approach to SQL Optimization

- Ensure the entire Data Access Logic is part of the SQL statement
  - Avoid handling predicates or relational operators (e.g. join) in application code, unless required by a known product limitation
- Run Optimizer Explain
- Understand the provided information
- Understand whether there is room for improvement
  - At the SQL syntax level
  - At the DB design level
- Understand the cost of implementation and potential impact on existing applications.
  - E.g. adding indexes
  - Re-ordering index columns
- Implement changes

The Impact of Poor Optimization

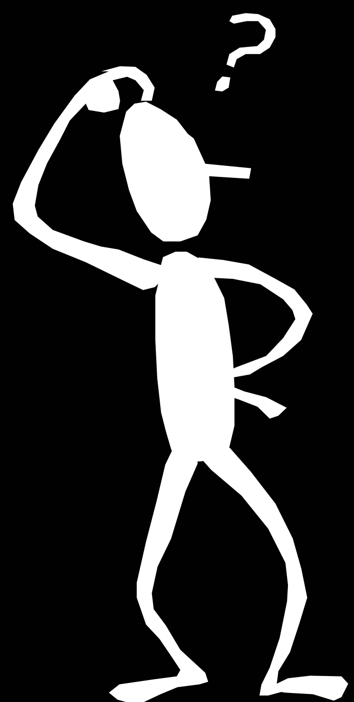
- In the past, due to limited memory size
  - I/O bottlenecks
  - Long elapsed times
- Today, with very powerful CPUs and very large memory size
  - High CPU utilization
  - Throughput lower than expected
  - Locking contention

We all tend to overlook the impact of shared resources, due to the incredible amount of resources available on our own single-user PC

a. sironi

Contents Review

- RDBMS and SQL Language
  - Static SQL and Dynamic SQL
- Mapping the Physical Design
- Query Compiler Overview
- Query Optimization
  - Access Path Selection
  - Query Rewrite
  - Examples
- The Optimizer Explained
- The Role of the DBA



Questions?

a. sironi

Bibliography - 1

1. Selinger et al., “Access Path Selection in a Relational Database Management System”, Sigmod 1979, http://www.cs.berkeley.edu/~brewer/cs262/AccessPath.pdf
2. Ioannidis and Kang, “Randomized Algorithms for Optimizing Large Join Queries”, Sigmod 1990, http://www.cc.gatech.edu/computing/Database/readinggroup/articles/p312-ioannidis.pdf
3. Hamid Pirahesh, T. Y. Cliff Leung, Waqar Hasan, "A Rule Engine for Query Transformation in Starburst and IBM DB2 C/S DBMS", ICDE 1997, pp. 391-400, http://www.diku.dk/undervisning/2003f/729/papers/rule_engine.pdf
4. Peter Gassner, Guy M. Lohman, K. Bernhard Schiefer, Yun Wang, "Query Optimization in the IBM DB2 Family", Data Engineering Bulletin 16(4): 4-18 (1993), ftp://ftp.research.microsoft.com/pub/debull/93DEC-CD.pdf

Bibliography - 2

1. Subquery Madness! at http://five.pairlist.net/pipermail/oracle-article/2004/000012.html ; http://dba-oracle.com/oracle_news/2004_9_14a_2004.htm ; http://www.dbdebunk.com/page/page/1351381.htm
2. Cost Control: Inside the Oracle Optimizer, http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/burleson_cbo_pt1.html
3. DB2 Redbooks: http://www.redbooks.ibm.com/
4. Oracle White Papers:

   http://oracle.ittoolbox.com/documents/default.asp?Section=White+Papers