SQL Rocks!

Iggy Fernandez

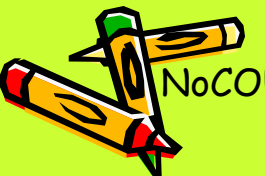
THE FINE PRINT

Focus on the important bits!

In addition to historical information, this Quarterly Report on Form 10-Q (this "Quarterly Report") contains forward-looking statements within the meaning of Section 27A of the Securities Act of 1933, as amended, and Section 21E of the Securities Exchange Act of 1934, as amended. Forward-looking statements are those that predict or describe future events or trends and that do not relate solely to historical matters. ***My employer does not endorse this presentation.*** You can generally identify forward-looking statements as statements containing the words "believe," "expect," "will," "anticipate," "intend," "estimate," "project," "plan," "may," "predict," "assume" or other similar expressions, although not all forward-looking statements contain these identifying words. ***Please don't blindly believe anything I say.*** All statements in this Quarterly Report regarding our future strategy, future operations, projected financial position, estimated future revenues, projected costs, future prospects, and results that might be obtained by pursuing management's current plans and objectives are forward-looking statements. You should not place undue reliance on our forward-looking statements because the matters they describe are subject to known and unknown risks, uncertainties and other unpredictable factors, many of which are beyond our control. ***Please conduct your own tests and investigations before choosing a plan of action.*** Our forward-looking statements are based on the information currently available to us and speak only as of the date on which this Quarterly Report was filed with the Securities and Exchange Commission ("SEC"). We expressly disclaim any obligation to issue any updates or revisions to our forward-looking statements, even if subsequent events cause our expectations to change regarding the matters discussed in those statements. ***Every situation is different and any one rule may not apply to every situation.*** Over time, our actual results, performance or achievements will likely differ from the anticipated results, performance or achievements that are expressed or implied by our forward-looking statements, and such difference might be significant and materially adverse to our stockholders. ***I do not guarantee the accuracy of statements made in this presentation.*** Many important factors that could cause such a difference are described in this Quarterly Report under the caption "Risk Factors" which you should review carefully. Please consider our forward-looking statements in light of those risks as you read this Quarterly Report. ***Thumb rules, unorthodox opinions, and statements unsupported by theoretical or empirical evidence should be treated especially cautiously.***

Subliminal Message

SQL performance requires conscious effort on the part of the Developer!

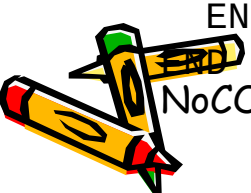


"RATTY" PROGRAMMING

```
CREATE TABLE tabledum (
  tabledum_ID                 INTEGER           NOT NULL,
  tabledum_version_ID         INTEGER           NOT NULL,
  tabledum_name               VARCHAR(32)       NOT NULL,
  constraint tabledum_PK PRIMARY KEY (tabledum_ID,
   tabledum_version_ID)
);

CREATE TABLE tabledee (
  tabledee_ID                 INTEGER           NOT NULL,
  tabledum_ID                 INTEGER           NOT NULL,
  tabledum_version_ID         INTEGER           NOT NULL,
  tabledum_name               VARCHAR(32)       NOT NULL,
  constraint tabledee_PK PRIMARY KEY (tabledee_ID),
  constraint tabledee_FK FOREIGN KEY (tabledum_ID,
   tabledum_version_ID) REFERENCES tabledum
);
```

"RATTY" PROGRAMMING

```
DECLARE
  tabledum_name_v         tabledum.tabledum_name%TYPE ;
  tabledum_version_ID_v   tabledum.tabledum_version_ID%TYPE ;
  tabledum_ID_v           tabledum.tabledum_ID%TYPE ;
CURSOR C1 IS
  SELECT tabledum_ID, tabledum_version_ID, tabledum_name
  FROM tabledum
  ORDER BY tabledum_ID, tabledum_version_ID;
  -- This will ensure that the last update is with the most recent
version.
BEGIN
  OPEN C1 ;
  LOOP
    FETCH C1 INTO tabledum_ID_v, tabledum_version_ID_v, tabledum_name_v ;
    EXIT WHEN C1%NOTFOUND ;
    UPDATE tabledee
    SET tabledum_name = tabledum_name_v,
        tabledum_version_ID = tabledum_version_ID_v
    WHERE tabledum_ID = tabledum_ID_v ;
    COMMIT ;
  END LOOP ;
END ;
```



SQL ROCKS!

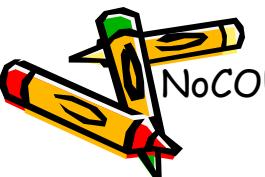
```
CREATE TABLE son_of_tabledum AS
  SELECT tabledum_ID          AS
son_of_tabledum_ID,
         tabledum_version_ID  AS
son_of_tabledum_version_ID,
         tabledum_name        AS
son_of_tabledum_name
    FROM (
      SELECT tabledum_ID,
             tabledum_version_ID,
             tabledum_name,
             MAX(tabledum_version_ID)
OVER (PARTITION BY tabledum_ID) AS
tabledum_max_version_ID
        FROM tabledum
    )
  WHERE tabledum_version_ID =
tabledum_max_version_ID;

ALTER TABLE son_of_tabledum
ADD CONSTRAINT son_of_tabledum_PK
PRIMARY KEY (son_of_tabledum_ID);
```

```
UPDATE (
  /* updateable view */
  SELECT tabledee.tabledee_ID,
         tabledee.tabledum_ID,
         tabledee.tabledum_version_ID,
         tabledee.tabledum_name,
son_of_tabledum.son_of_tabledum_ID,
son_of_tabledum.son_of_tabledum_name,
son_of_tabledum.son_of_tabledum_version_ID
    FROM tabledee,
         son_of_tabledum
   WHERE tabledee.tabledum_ID =
son_of_tabledum.son_of_tabledum_ID
     AND tabledee.tabledum_version_ID !=
son_of_tabledum.son_of_tabledum_version_ID
)
   SET tabledum_version_ID =
son_of_tabledum_version_ID,
       tabledum_name =
son_of_tabledum_name;
```

Quotable Quotes

"The other terror that scares us from self-trust is our *consistency*, a reverence for our past act or word, because the eyes of others have no other data for computing our orbit than our past acts, and we are loath to disappoint them..... Bring the past for judgment into the thousand-eyed present, and live ever in a new day.... *A foolish consistency is the hobgoblin of little minds, adored by little statesmen and philosophers and divines*.... Speak what you think now in hard words, and tomorrow speak what tomorrow thinks in hard words again, though it contradict every thing you said today."

—Ralph Waldo Emerson —

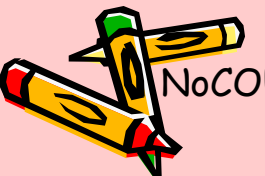


SQL Sucks!

Iggy Fernandez

Subliminal Message

SQL performance requires conscious effort on the part of the Developer!

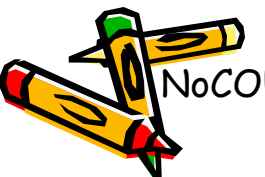


More Quotable Quotes!

(Do you spot the irony?)

"Man is timid and apologetic; he is no longer upright; he dares not say "I think," "I am," but quotes some saint or sage… We are like children who repeat by rote the sentences of grandames and tutors, and, as they grow older, of the men of talents and character they chance to see,— painfully recollecting the exact words they spoke; afterwards, when they come into the point of view which those had who uttered these sayings, they understand them and are willing to let the words go; for at any time they can use words as good when occasion comes."

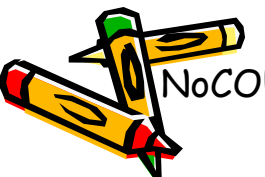
—Ralph Waldo Emerson—



Sagely Saying!
(with apologies to Emerson)

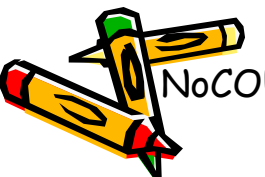
"Some people can perform seeming miracles with straight SQL, but the statements end up looking like *pretzels created by somebody who is experimenting with hallucinogens.*"

—Steven Feuerstein—



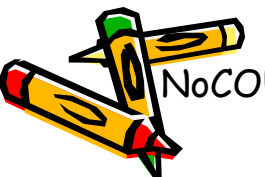
Why SQL Sucks?
—Logic Problems—

- NULL values—Angels fear them
- Duplicate rows—Always SELECT DISTINCT?



Why SQL Sucks?
—Performance Problems—

- Redundancy—Can the Query Optimizer recognize equivalent queries and use the same query plan each time?
- Non-procedural Nature—But isn't that a good thing?



REDUNDANCY

—SEVEN VARIATIONS FOR SEVEN PROGRAMMERS

```
CREATE TABLE personnel (
  empid INTEGER,
  lname VARCHAR(256) NOT NULL,
  CONSTRAINT personnel_PK
     PRIMARY KEY (empid)
);

CREATE TABLE payroll (
  empid INTEGER NOT NULL,
  salary NUMBER NOT NULL,
  CONSTRAINT payroll_PK PRIMARY
     KEY (empid),
  CONSTRAINT payroll_FK1
     FOREIGN KEY (empid)
     REFERENCES iggy123.personnel
);
```

```
INSERT INTO personnel
SELECT
  object_id,
  owner||object_type||object_name
FROM dba_objects
WHERE object_id IS NOT NULL;

INSERT INTO payroll
SELECT
  empid,
  dbms_random.value (1, 250000)
FROM personnel;

INSERT INTO personnel VALUES (-
    1, 'IGGY');
INSERT INTO payroll VALUES (-1,
    199170);
```

REDUNDANCY—JOIN

SELECT lname
FROM personnel, payroll
WHERE personnel.empid = payroll.empid
AND salary = 199170;

http://www.dbdebunk.com/page/page/1317920.htm—Fabian Pascal circa 1988

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      1  NESTED LOOPS  (cr=249 r=5 w=0 time=16701 us)
      1   TABLE ACCESS FULL PAYROLL (cr=246 r=4 w=0 time=16559 us)
      1   TABLE ACCESS BY INDEX ROWID PERSONNEL (cr=3 r=1 w=0 time=123 us)
      1    INDEX UNIQUE SCAN PERSONNEL_PK (cr=2 r=1 w=0 time=96 us)(object id
142621)
```

REDUNDANCY—IN1

```
SELECT lname
FROM personnel
WHERE empid in (SELECT empid
  FROM payroll
  WHERE salary = 199170);
```

http://www.dbdebunk.com/page/page/1317920.htm—Fabian Pascal circa 1988

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      1  NESTED LOOPS  (cr=26642 r=241 w=0 time=395917 us)
  13181   TABLE ACCESS FULL PERSONNEL (cr=278 r=241 w=0 time=38170 us)
      1   TABLE ACCESS BY INDEX ROWID PAYROLL (cr=26364 r=0 w=0 time=289811 us)
  13181    INDEX UNIQUE SCAN PAYROLL_PK (cr=13183 r=0 w=0 time=147241
us)(object id 142623)
```

REDUNDANCY—ANY1

```
SELECT lname
FROM personnel
WHERE empid = ANY (SELECT empid
  FROM payroll
  WHERE salary = 199170);
```

http://www.dbdebunk.com/page/page/1317920.htm—Fabian Pascal circa 1988

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      1  NESTED LOOPS  (cr=26642 r=251 w=0 time=390740 us)
  13181   TABLE ACCESS FULL PERSONNEL (cr=278 r=251 w=0 time=38613 us)
      1   TABLE ACCESS BY INDEX ROWID PAYROLL (cr=26364 r=0 w=0 time=284813 us)
  13181    INDEX UNIQUE SCAN PAYROLL_PK (cr=13183 r=0 w=0 time=143925
us)(object id 142623)
```

REDUNDANCY—IN2

```
SELECT lname
FROM personnel
WHERE 199170 IN (SELECT salary
  FROM payroll
  WHERE personnel.empid = payroll.empid);
```

http://www.dbdebunk.com/page/page/1317920.htm—Fabian Pascal circa 1988

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      1  NESTED LOOPS  (cr=26642 r=251 w=0 time=411057 us)
  13181   TABLE ACCESS FULL PERSONNEL (cr=278 r=251 w=0 time=37503 us)
      1   TABLE ACCESS BY INDEX ROWID PAYROLL (cr=26364 r=0 w=0 time=284932 us)
  13181    INDEX UNIQUE SCAN PAYROLL_PK (cr=13183 r=0 w=0 time=145010
us)(object id 142623)
```

REDUNDANCY—ANY2

```
SELECT lname
FROM personnel
WHERE 199170 = ANY (SELECT salary
  FROM payroll
  WHERE personnel.empid = payroll.empid);
```

http://www.dbdebunk.com/page/page/1317920.htm—Fabian Pascal circa 1988

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      1  NESTED LOOPS  (cr=26642 r=250 w=0 time=414041 us)
  13181   TABLE ACCESS FULL PERSONNEL (cr=278 r=250 w=0 time=40327 us)
      1   TABLE ACCESS BY INDEX ROWID PAYROLL (cr=26364 r=0 w=0 time=303447 us)
  13181    INDEX UNIQUE SCAN PAYROLL_PK (cr=13183 r=0 w=0 time=147278
us)(object id 142623)
```

REDUNDANCY—EXISTS

```
SELECT lname
FROM personnel
WHERE EXISTS (SELECT *
  FROM payroll
  WHERE personnel.empid = payroll.empid
  AND salary = 199170);
```

http://www.dbdebunk.com/page/page/1317920.htm—Fabian Pascal circa 1988

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      1  FILTER  (cr=39821 r=259 w=0 time=729194 us)
  13181   TABLE ACCESS FULL PERSONNEL (cr=278 r=259 w=0 time=43087 us)
      1   TABLE ACCESS BY INDEX ROWID PAYROLL (cr=39543 r=0 w=0 time=452283 us)
  13181    INDEX UNIQUE SCAN PAYROLL_PK (cr=26362 r=0 w=0 time=296877 us)(object
id 142623)
```

REDUNDANCY—EXISTS2

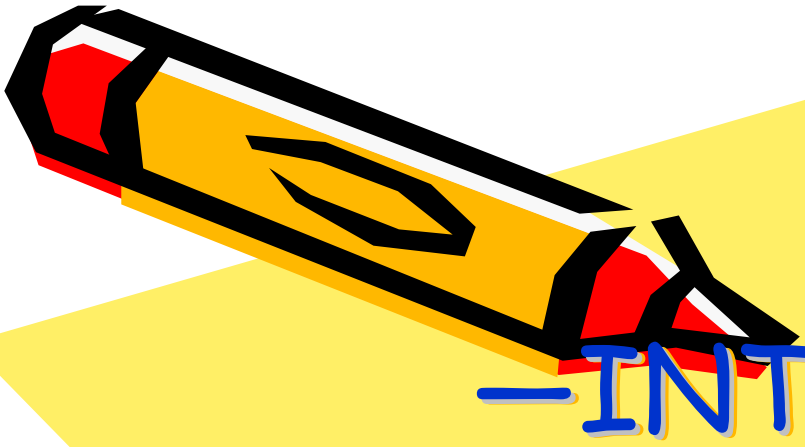
```
SELECT lname
FROM personnel
WHERE 0 < (SELECT COUNT(*)
  FROM payroll
  WHERE personnel.empid = payroll.empid
  AND salary = 199170);
```

http://www.dbdebunk.com/page/page/1317920.htm—Fabian Pascal circa 1988

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      1  FILTER  (cr=39821 r=277 w=0 time=699178 us)
  13181   TABLE ACCESS FULL PERSONNEL (cr=278 r=277 w=0 time=44223 us)
      1   TABLE ACCESS BY INDEX ROWID PAYROLL (cr=39543 r=0 w=0 time=429133 us)
  13181    INDEX UNIQUE SCAN PAYROLL_PK (cr=26362 r=0 w=0 time=272885 us)(object
id 142623)
```

TIME CAPSULE

	ORACLE —1988—	ORACLE —2006—	INGRES —1988—
JOIN	19	0.0167	35
IN1	22	0.3959 (25X)	34
ANY1	20	0.3907 (25X)	33
IN2	525 (25X)	0.4110 (25X)	33
ANY2	525 (25X)	0.4140 (25X)	33
EXISTS	525 (25X)	0.7291 (40X)	33
COUNT	1818 (10X)	0.6991 (40X)	N/A

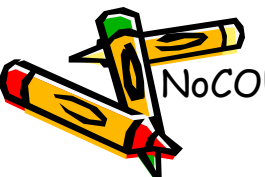


—INTERLUDE—
A Funny Thing Happened at Work!

The Curious Case Of The Cartesian Product

Who's Responsible for Performance?

- Application Developer?
- Database Administrator?
- Query Optimizer?



Subliminal Message

SQL performance requires conscious effort on the part of the Developer!

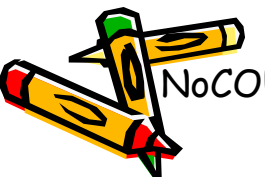
Does the Query Optimizer Have a Clue?

—UNDERESTIMATION—

```
select * from CarSales
where Manufacturer = 'Toyota'
  and Model = 'Celica';
```

—OVERESTIMATION—

```
select * from CarSales
where Manufacturer = 'Toyota'
  and ModelYear < 1975;
```



BACK TO SCHOOL

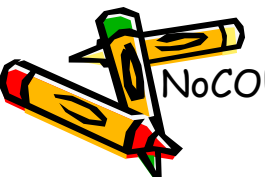
Probability(X and Y) =

Probability(X) x Probability(Y given X)

Probability(X or Y) =

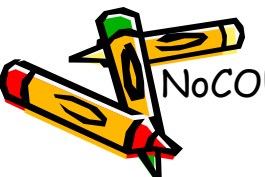
Probability(X) + Probability(Y)

- Probability(X and Y)



JOIN ASSUMPTIONS

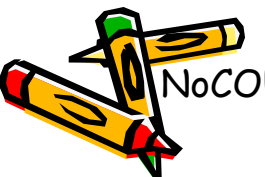
- Every row in Table A will match exactly NR_B/NDV_B rows from Table B
- Every row in Table B will match exactly NR_A/NDV_A rows from Table A



Heard in the "Real World"

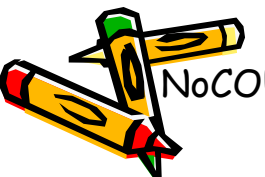
"I became interested in the CBO's selectivity calculations trying to understand why it comes up with some of the ridiculously low cardinality estimates (like 1 when in reality there are 80,000+) which then lead to disastrous access plans that take hours, provided they finish at all, instead of minutes or seconds."

http://asktom.oracle.com/pls/ask/f?p=4950:8:::::F4950_P8_DISPLAYID:4344365159075



MORE COMPLICATIONS

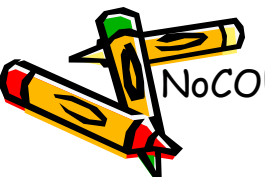
- Cursor Sharing
- Bind Variable Peeking



QUOTABLE QUOTES

"It's the best possible time to be alive, when almost everything you thought you knew is wrong."

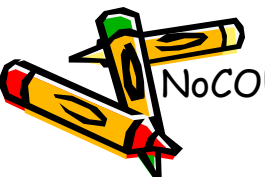
"Then felt I like some watcher of the skies
When a new planet swims into his ken;
Or like stout Cortez, when with eagle eyes
He stared at the Pacific—and all his men
Looked at each other with a wild surmise—
Silent, upon a peak in Darien."



QUESTIONABLE BELIEFS

#1

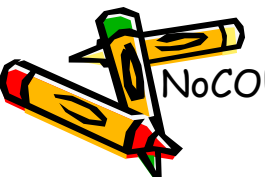
DBAs bear chief responsibility for the performance of SQL statements.



QUESTIONABLE BELIEFS

#2

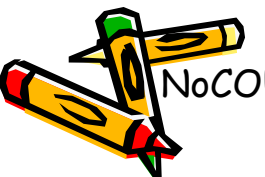
Applications should be designed without reference to the way data is stored, e.g., index-organized tables, hash clusters, partitions, etc.



QUESTIONABLE BELIEFS

#3

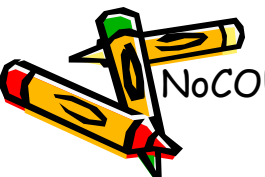
Application programmers should not tailor their SQL statements to make use of existing indexes. DBAs should instead create traps to catch badly performing SQL at runtime and create new indexes as necessary to make them perform better.



QUESTIONABLE BELIEFS

#4

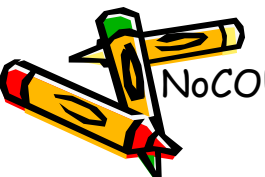
It is not necessary to review the Query Execution Plan of an SQL statement before releasing it into a production environment. It is further not necessary to *freeze* the Query Execution Plan of an SQL statement before releasing it into a production environment. It is desirable that Query Execution Plans change in response to changes in the statistical information that the query optimizer relies upon. Such changes are always for the better.



QUESTIONABLE BELIEFS

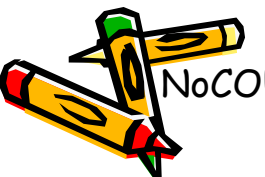
#5

The most common cause of poorly performing SQL is the failure of the DBA to collect statistical information on the distribution of data for the use of the query optimizer. This statistical information should be refreshed frequently.



SAGELY SAYING

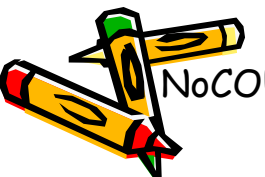
"It astonishes me how many shops prohibit any un-approved production changes and yet re-analyze schema stats weekly. Evidently, they do not understand that the purpose of schema re-analysis is to change their production SQL execution plans, and they act surprised when performance changes!"—Don Burleson



Hinting to the Max!

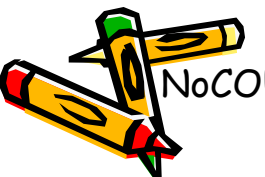
```
create table Suppliers (
  SupplierName varchar(64)
not null,
  constraint SupplierName_PK
primary key (SupplierName)
);

create table Parts (
  PartName varchar(64) not
null,
  constraint PartName_PK
primary key (PartName)
);
```

```
create table SuppliedParts (
  SupplierName varchar(64)
not null,
  PartName varchar(64) not
null,
  constraint
SuppliedParts_PK primary
key (SupplierName,
PartName),
  constraint
SuppliedParts_FK1 foreign
key (SupplierName)
references Suppliers,
  constraint
SuppliedParts_FK2 foreign
key (PartName) references
Parts
);
```



Hinting to the Max!

```
with
  AllCombinations as (
    select SupplierName,
PartName
    from Suppliers, Parts
  ),

  InvalidCombinations as (
    select SupplierName,
PartName
    from AllCombinations
    where (SupplierName,
PartName) not in (
      select SupplierName,
PartName
      from SuppliedParts
    )
  ),

 UnwantedSuppliers as (
   select SupplierName
   from InvalidCombinations
 ),

 WantedSuppliers as (
   select SupplierName
   from Suppliers
   where SupplierName not in (
     select SupplierName
     from UnwantedSuppliers
   )
 )

select *
from WantedSuppliers;
```

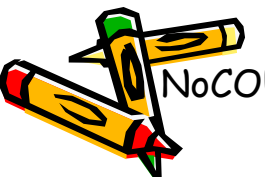


Hinting to the Max!

```
with
  AllCombinations as (
    select /*+ NO_MERGE
ORDERED FULL(Suppliers)
FULL(Parts) USE_NL(Parts) */
SupplierName, PartName
    from Suppliers, Parts
  ),

  InvalidCombinations as (
    select /*+ NO_MERGE */
SupplierName, PartName
    from AllCombinations
    where (SupplierName,
PartName) not in (
      select /*+
INDEX(SuppliedParts
SuppliedParts_PK) NL_AJ */
SupplierName, PartName
      from SuppliedParts
    )
  ),

 UnwantedSuppliers as (
    select /*+ NO_MERGE */
SupplierName
    from InvalidCombinations
  ),

  WantedSuppliers as (
    select /*+ NO_MERGE */
SupplierName
    from Suppliers
    where SupplierName not in (
      select /*+ HASH_AJ */
SupplierName
      from UnwantedSuppliers
    )
  )

select *
from WantedSuppliers;
```

Other Options

- Stored Outlines and Plan Stability
- SQL Profiles
- Manual Statistics
- Dynamic Sampling
- RBO?



Takeaway Message

SQL performance requires conscious effort on the part of the Developer!

Q & A

iggy_fernandez@hotmail.com