SERIALIZABILITY
There is no free lunch!

By Iggy Fernandez

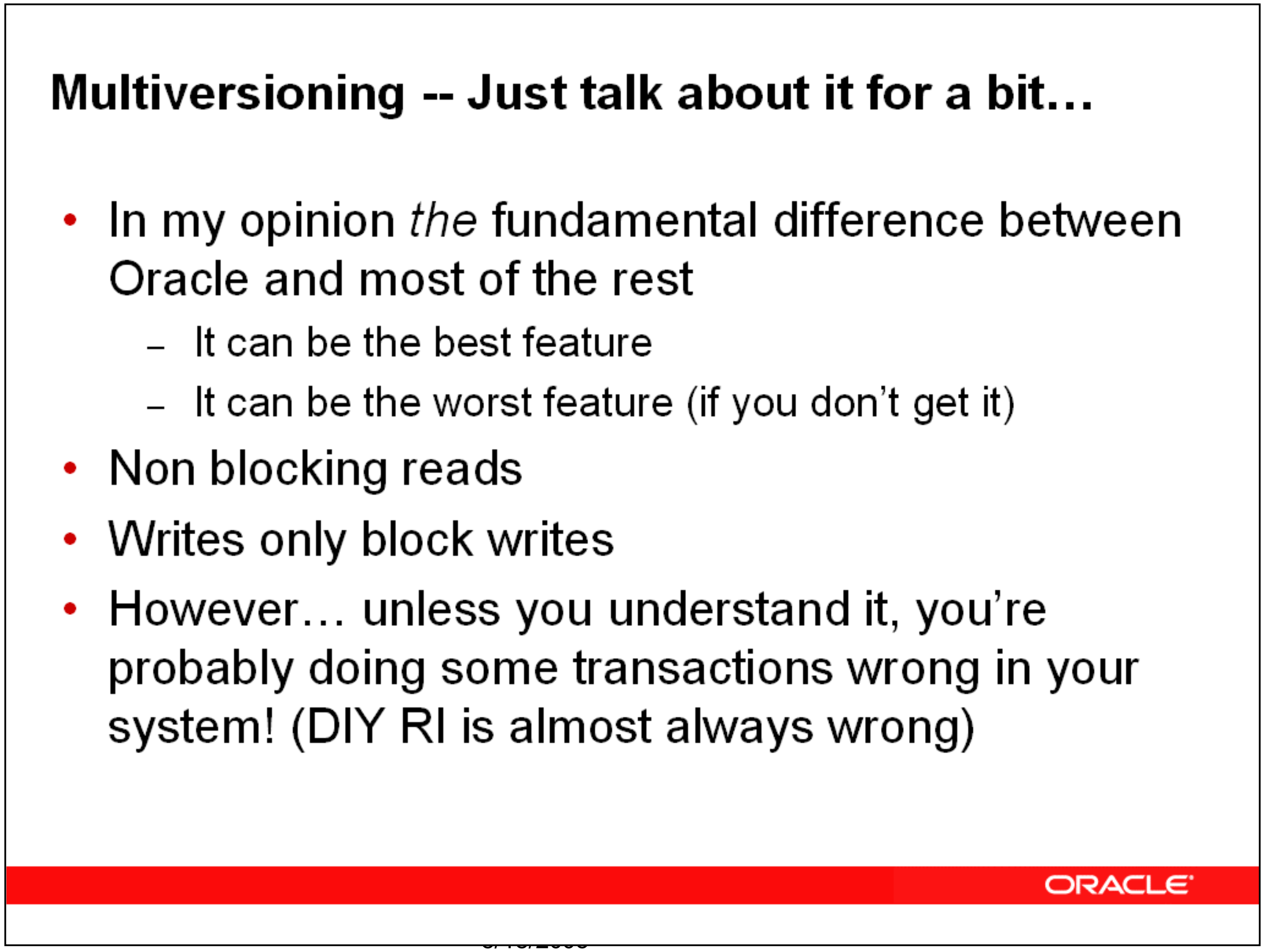
NoCOUG Summer Conference

Aug 18, 2005

CAVEAT LECTOR!

- Don’t believe (blindly) anything I say ☺
- What you are about to see and hear may be false or deficient in some way ☺
- Oracle Corporation has not reviewed this presentation for accuracy ☺

Slide from Tom Kyte's presentation to NoCOUG



Multiversioning -- Just talk about it for a bit…

- In my opinion *the* fundamental difference between Oracle and most of the rest
  - It can be the best feature
  - It can be the worst feature (if you don't get it)
- Non blocking reads
- Writes only block writes
- However… unless you understand it, you're probably doing some transactions wrong in your system! (DIY RI is almost always wrong)

ORACLE

Lions and Tigers and Bears – Oh My!

- Dirty Reads
- Non-repeatable Reads
- Phantoms

Oracle says … (Patent Documents)

- “To describe fully consistent transaction behavior when transactions execute concurrently, database researchers have defined a transaction isolation level called serializability.”
- “In the serializable isolation level, transactions must execute in such a way that they appear to be executed one at a time (serially), rather than concurrently.”
- “In other words, concurrent transactions executing in serializable mode are only permitted to make database changes they could have made if the transactions had been scheduled to execute one after another, in some specific order, rather than concurrently.”

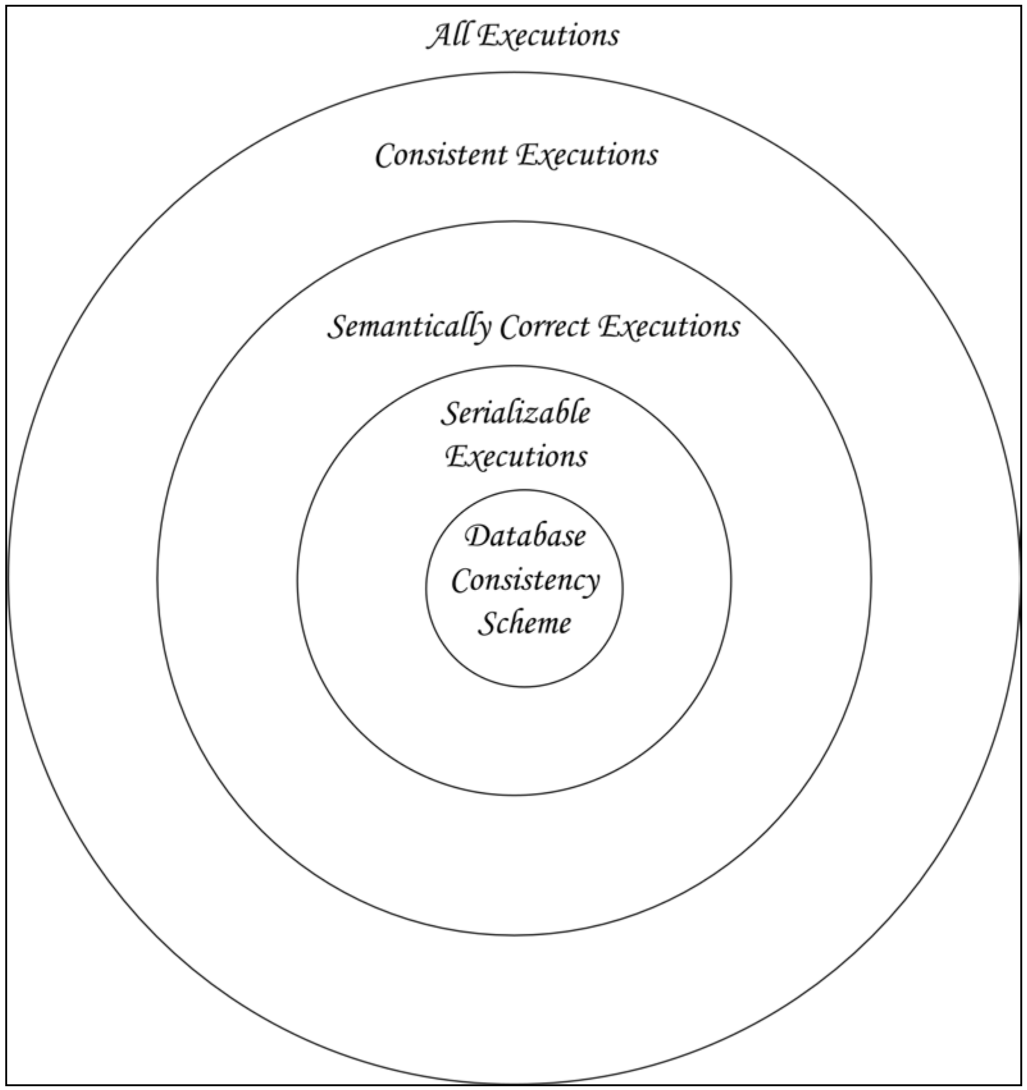
The ANSI/ISO SQL standard says …

“A serializable execution is defined to be an execution of the operations of concurrently executing SQL-transactions that produces the same effect as some serial execution of those same SQL-transactions.”

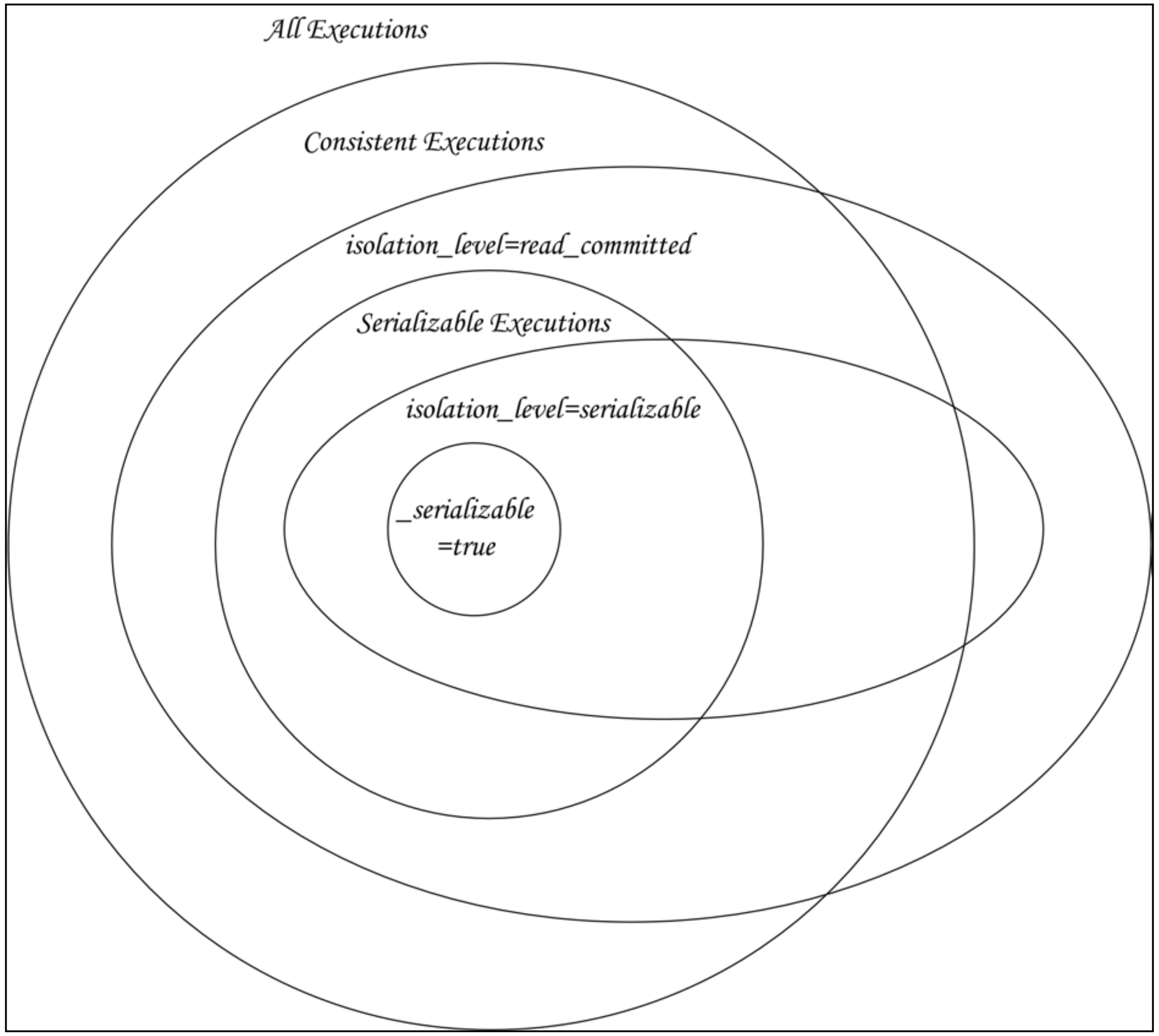
Tom Kyte says …
(One-On-One Oracle)

“SERIALIZABLE does **not** mean that all transactions executed by the users are the same as if they were executed one right after another in a serial fashion.” ☺

Onion Shell Diagram – I



Onion Shell Diagram – II



Examples

- The Case of the Disappearing Dollars
- The Second case of the Disappearing Dollars
- Poor Orphan Annie!
- The Case of the Popular Projector
- The Case of the Dangling DBA
- The Case of the Troublesome Tables
- Those Crazy ORA-8177 Blues
- More ORA-8177 Blues

Example 1 – Setup
The Case of the Disappearing Dollars

```
create table bank_account (
  account# integer,
  balance number
);

insert into bank_account values (1,10);
insert into bank_account values (2,10);
insert into bank_account values (3,10);
```

Example 1 – Setup (Contd.)

```
create or replace procedure
   debit_credit(
 debit_account in integer,
 credit_account in integer,
 debit_amount in integer
)
is
 debit_account_balance number;
 credit_account_balance number;
begin
 select balance
 into debit_account_balance
 from bank_account
 where account#=debit_account;
 --
 select balance
 into credit_account_balance
 from bank_account
 where account#=credit_account;
```

```
 debit_account_balance :=
 debit_account_balance - debit_amount;
 --
 credit_account_balance :=
 credit_account_balance + debit_amount;
 --
 user_lock.sleep(600);
 --
 update bank_account
 set balance = debit_account_balance
 where account# = debit_account;
 --
 update bank_account
 set balance = credit_account_balance
 where account# = credit_account;
 --
 commit;
end;
```

Example 1 – Execution History

18:09:14 SQL> execute debit_credit(1,2,5); PL/SQL procedure successfully completed.	
	18:09:15 SQL> execute debit_credit(3,2,5); PL/SQL procedure successfully completed.

Example 1 – Results

```
18:09:21 SQL> select * from bank_account;
  ACCOUNT#    BALANCE
---------- ----------
         1          5
         2         15
         3          5
3 rows selected.
```

Example 1a – Execution History

18:10:41 SQL> alter session set isolation_level=serializable; Session altered. 18:10:41 SQL> execute debit_credit(1,2,5); PL/SQL procedure successfully completed.	
	18:10:42 SQL> alter session set isolation_level=serializable; Session altered. 18:10:42 SQL> execute debit_credit(3,2,5); BEGIN debit_credit(3,2,5); END; * ERROR at line 1: **ORA-08177: can't serialize access for this transaction**

Example 1a – Results

```
18:10:49 SQL> select * from bank_account;
  ACCOUNT#    BALANCE
---------- ----------
         1          5
         2         15
         3         10
```

Example 2 – Setup
The Second Case of the Disappearing Dollars

```
create table bank_account (
  account# integer,
  account_type varchar(1),
  balance number
);

insert into bank_account values (1,'C',70);
insert into bank_account values (1,'S',80);
```

Example 2 – Setup (Contd.)

```
create or replace procedure withdrawal(
  in_account# in integer,
  in_account_type in varchar,
  in_withdrawal_amount in number
)
is
  checking_account_balance number;
  savings_account_balance number;
begin
  select balance
  into checking_account_balance
  from bank_account
  where account#=in_account#
  and account_type='C';
select balance
into savings_account_balance
from bank_account
where account#=in_account#
and account_type='S';
--
user_lock.sleep(600);
--
if (checking_account_balance +
    savings_account_balance >=
    in_withdrawal_amount) then
  update bank_account
  set balance = balance -
    in_withdrawal_amount
  where account# = in_account#
  and account_type = in_account_type;
end if;
commit;
end;
```

Example 2 – Execution History

09:39:58 SQL> alter session set isolation_level=**serializable**; Session altered. 09:39:58 SQL> **execute withdrawal(1,'C',100);** PL/SQL procedure successfully completed.	
	09:40:01 SQL> alter session set isolation_level=**serializable**; Session altered. 09:40:01 SQL> **execute withdrawal(1,'S',100);** PL/SQL procedure successfully completed.

Example 2 – Results

```
09:40:07 SQL> select * from bank_account;

  ACCOUNT# A     BALANCE
---------- - ----------
         1 C        -30
         1 S        -20

2 rows selected.
```

Example 3 – Setup (Poor Orphan Annie!)

```
create table parent (
  parent_name varchar(8)
);

create table child (
  child_name varchar(8),
  parent_name varchar(8)
);

insert into parent values(‘Warbucks');
```

Example 3 – Execution History

18:25:07 SQL> alter session set isolation_level=**serializable**;
Session altered.
18:25:07 SQL> **select * from parent** where parent_name='Warbucks';
`PARENT_N`
`---------`
`Warbucks`
1 row selected.	

	18:25:16 SQL> alter session set isolation_level=**serializable**; Session altered. 18:25:16 SQL> **select * from child** where parent_name='Warbucks'; **no rows selected**

Example 3 – Execution History

18:25:19 SQL> **insert into child** values ('Annie','Warbucks'); **1 row created.**	
	18:25:21 SQL> **delete from parent** where parent_name='Warbucks'; **1 row deleted.**
18:25:23 SQL> **commit**; Commit complete.	
	18:25:25 SQL> **commit**; Commit complete.

Example 3 – Results

```
18:25:28 SQL> select * from parent;
no rows selected

18:25:28 SQL> select * from child;
CHILD_NA PARENT_N
-------- --------
Annie    Warbucks
1 row selected.
```

Example 4 – Setup
The Case of the Popular Projector

```
create table schedules(
  resource_name varchar(25),
  start_time date,
  end_time date
);

create or replace procedure
resource_scheduler(
  room_name in varchar,
  new_start_time in date,
  new_end_time in date
)
is
  already_reserved integer;
```

```
begin
  already_reserved := 0;
  --
  select count(*) into already_reserved
  from schedules
  where resource_name = room_name
  and (start_time between new_start_time and
     new_end_time)
  or (end_time between new_start_time and
     new_end_time);
  --
  user_lock.sleep(600);
  --
  if (already_reserved = 0) then
   insert into schedules values
    (room_name,new_start_time,new_end_time);
  end if;
  --
  commit;
end;
```

Example 4 – Execution History

18:19:08 SQL> alter session set isolation_level=**serializable**; Session altered. 18:19:08 SQL> exec resource_scheduler('Projector', '2005/08/31 **09:00**', '2005/08/31 **10:00**'); PL/SQL procedure successfully completed.	
	18:19:10 SQL> alter session set isolation_level=**serializable**; Session altered. 18:19:10 SQL> exec resource_scheduler('Projector', '2005/08/31 **09:30**', '2005/08/31 **10:30**'); PL/SQL procedure successfully completed.

Example 4 – Results

```
18:19:17 SQL> select * from schedules;

RESOURCE_NAME             START_TIME       END_TIME
------------------------- ---------------- ----------------
Projector                 2005/08/31 09:00 2005/08/31 10:00
Projector                 2005/08/31 09:30 2005/08/31 10:30

2 rows selected.
```

Example 5 – Setup The Case of the Dangling DBA

```
create table schedules (
  course_name varchar(32),
  student_name varchar(32)
);

declare
  i integer;
begin
  for i in 1..99 loop
    insert into schedules values ('DBA 101',i);
  end loop;
  commit;
end;
```

Example 5 – Setup (Contd.)

```
create or replace procedure
signup(
  in_course_name in
    varchar,
  in_student_name in
    varchar
)
is
  signups integer;
begin
  select count(*) into signups
  from schedules
  where course_name =
    in_course_name;
  --
  user_lock.sleep(600);
  --
  if (signups < 100) then
    insert into schedules
    values(in_course_name,
    in_student_name);
  end if;
  commit;
end;
```

Example 5 – Execution History

19:05:08 SQL> alter session set isolation_level=**serializable**; Session altered. 19:05:08 SQL> exec signup('DBA 101','**Iggy**'); PL/SQL procedure successfully completed.	
	19:05:10 SQL> alter session set isolation_level=**serializable**; Session altered. 19:05:10 SQL> exec signup('DBA 101','**Ziggy**'); PL/SQL procedure successfully completed.

Example 5 – Results

```
19:05:16 SQL> select count(*) from schedules where course_name='DBA
   101';
  COUNT(*)
----------
       101

1 row selected.
```

Example 6 – Setup
The Case of the Troublesome Tables

```
create table a (x int);

create table b (x int);
```

Example 6 – Execution History

18:19:18 SQL> alter session set isolation_level=**serializable**; Session altered. 18:19:18 SQL> **insert into a select count(*) from b**; 1 row created.	
	18:19:26 SQL> alter session set isolation_level=**serializable**; Session altered. 18:19:26 SQL> **insert into b select count(*) from a**; 1 row created.
18:19:27 SQL> commit; Commit complete.	
	18:19:31 SQL> commit; Commit complete.

Example 6 – Results

```
18:19:33 SQL> select * from a;
          X
----------
          0
1 row selected.

18:19:33 SQL> select * from b;
          X
----------
          0
1 row selected.
```

Example 7 – Setup
Those Crazy ORA-8177 Blues

```
create table ora8177 (col1 integer);

insert into ora8177 values(1);
insert into ora8177 values(2);
insert into ora8177 values(3);
```

Example 7 – Execution History

18:15:09 SQL> alter session set isolation_level=**serializable**;
Session altered.
18:15:09 SQL> select sysdate from dual;
`SYSDATE`
`----------`
`08/14/2005`
1 row selected.	

	18:15:12 SQL> update ora8177 set col1=col1 where **col1=2**; 1 row updated. 18:15:12 SQL> commit; Commit complete.

Example 7 – Execution History

	18:15:12 SQL> update ora8177 set col1=col1 where **col1=3**; 1 row updated. 18:15:12 SQL> commit; Commit complete.
18:15:21 SQL> update ora8177 set col1=col1 where **col1=1**; update ora8177 set col1=col1 where col1=1 * ERROR at line 1: **ORA-08177: can't serialize access for this transaction**	

More ORA-8177 Blues (Single Transaction!)

```
SQL> create table ora8177(col1 varchar(4000));
Table created.

SQL> create index ix_ora8177 on ora8177(col1);
Index created.

SQL> alter session set isolation_level=serializable;
Session altered.

SQL> insert into ora8177 select rpad('x',3999,'x') from dba_objects;
insert into ora8177 select rpad('x',3999,'x') from dba_objects
        *
ERROR at line 1:
```
ORA-08177: can't serialize access for this transaction

A VERY Big Hammer!

- "The INIT.ORA file parameter SERIALIZABLE=TRUE or _SERIALIZABLE implies automatic table-level locking ON READ in Release 8.0.3. It should not be used in any Oracle8 installation and will probably be permanently deleted from the maintenance release after 8.1." – Oracle 8.0.3 Manual
- Lives on in Oracle 10g as a hidden parameter (_SERIALIZABLE)

Little Hammers!

- Use “alter session set isolation_level=serializable”
- Handle ORA-8177 errors in your program code
- Avoid “DIY RI”
- Materialize your constraints
- Use “SELECT FOR UPDATE”

The Moral of the Story

- Concurrency has a price
- There is no free lunch

Tom Kyte has the Last Word! (One-On-One Oracle)

“Unless you know how it works, you **will** write programs that corrupt data. **It is that simple**.” – Closing sentence of the chapter on Locking and Concurrency

References

- A. **Fekete**, D. Liarokapis, E. O'Neil, P. O'Neil, and D. Shasha. Making snapshot isolation serializable. 1996. Available at *http://www.cs.umb.edu/~isotest/snaptest/snaptest.pdf*.
- S. **Elnikety**, F. Pedone, and W. Zwaenepoel. Generalized Snapshot Isolation and a Prefix-Consistent Implementation. 2004. Available at *http://icwww.epfl.ch/publications/documents/IC_TECH_REPORT_200421.pdf*.

Q & A

- Send your questions to iggy_fernandez@hotmail.com
- A zipped file containing the scripts used in this presentation can be obtained by sending a request to iggy_fernandez@hotmail.com
- A follow-up article with detailed analyses and solutions to the examples is planned for a forthcoming issue of the NoCOUG journal