The Hobgoblin of Little Minds

A foolish consistency is the hobgoblin of little minds,

Adored by little statesmen and philosophers and divines.

RALPH WALDO EMERSON

Introduction

The above quote appears in the chapter on Data Consistency and Concurrency in older editions of the Oracle Concepts Manual. Tom Kyte, Vice President of Core Technologies at Oracle, characterizes differences in approaches to Data Consistency and Concurrency as *the* fundamental difference between Oracle and other database vendors, saying that it can be Oracle's best feature or it's worst feature (*if* you don't understand it) (Reference [8]). He also says that, if you don't understand it, you are probably doing some transactions *wrong* in your system and that "Do-It-Yourself" Referential Integrity is *almost always wrong*!

Caveat Lector!

Oracle Corporation has *not* reviewed this essay for accuracy.

The Data Consistency and Concurrency Challenge

Oracle has patented the techniques it uses for concurrency control. One of the names on the patent filing is that of Dr. Kenneth Jacobs, a.k.a. "Dr. DBA", currently the Vice President of Product Strategy at Oracle. Here is a quote from the patent documents (Reference [2]).

> *"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,[1] rather than concurrently."*

The serializability criterion for database consistency is very well known and is even mentioned in the ANSI SQL standard (Reference [1]). Here is an excerpt.

> *"The isolation level of an SQL-transaction defines the degree to which the operations on SQL-data or schemas in that SQL-transaction are affected by the effects of and can affect operations on SQL-data or schemas in concurrent SQL-transactions. […] The execution of concurrent SQL-transactions at isolation level SERIALIZABLE is guaranteed to be serializable.*
>
> *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. A serial execution is one in which each SQL-transaction executes to completion before the next SQL-transaction begins."*

[1] Note that different serial orderings of transactions can conceivably produce different results. (For example, multiplying a number by 2 and then adding 3 will produce a different result if the operations are reversed.) Since each such result is permissible when the transactions are executed in serial fashion, they are all permissible when the transactions are executed in concurrent fashion.

However, it is not very well understood that serializability is *not* a necessary condition for database consistency even though it is certainly a *sufficient* condition. In other words, serializability is more restrictive than strictly necessary. A Microsoft researcher (Reference [9]) recently described *another* "sufficient condition" for database consistency called "semantic correctness" which is less restrictive than serializability. The following example is provided in Reference [9].

> *"For example, a stock trading application might have a buy transaction type that takes as parameters the identity of a stock and the number of shares, n, to be purchased and a result that states 'when each share was purchased no cheaper unbought shares of the stock existed in the database'.*
>
> *In a semantically correct schedule, two concurrent transactions, T1 and T2, could each buy some shares at $30 and some at $31 per share, even though initially there are n shares available at $30.*
>
> *First, T1 buys n/2 shares at $30; then, T2 buys n/2 shares at $30; then, since there are no more shares available at $30, T1 buys n/2 shares at $31; and, finally, T2 buys n/2 shares at $31.*
>
> *When each transaction terminates, its result is true since, when each share was bought, no cheaper unbought shares existed in the database.*
>
> *The final state could not have been produced by a serializable schedule since the purchase price of all shares bought by one or the other of the two transactions would have been $30."*

It is also not very well understood that, as a general rule, database consistency schemes that guarantee serializability are also more restrictive than strictly necessary i.e. they enforce restrictions that are *sufficient* but not absolutely *necessary*.

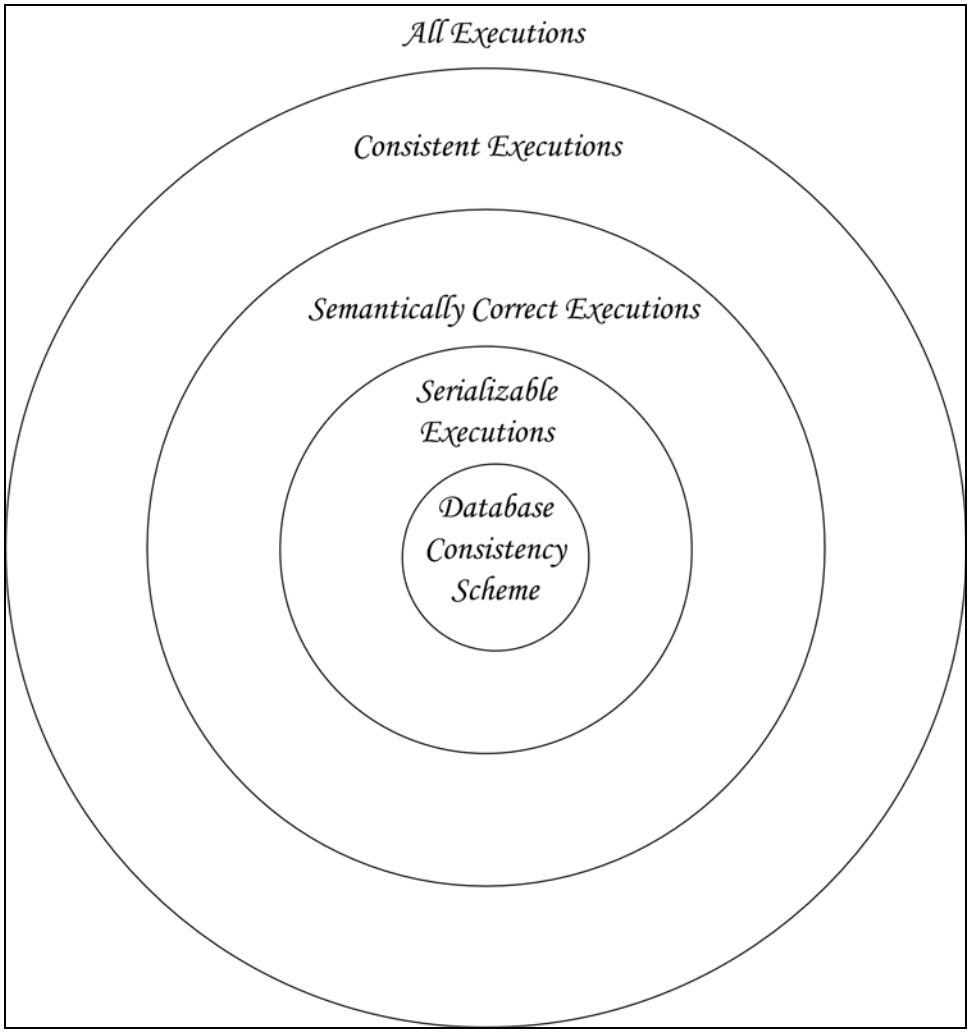

Fig. 1: The relationship between different classes of executions.

Houston, We Have a Problem!

The following quote is from an academic paper by researchers at the University of Massachusetts at Boston (Reference [5]).

> *"All major database system products are delivered with default non-serializable isolation levels, often ones that encounter serialization anomalies more commonly than [Oracle's Snapshot Isolation], and we suspect that numerous isolation errors occur each day at many large sites because of this, leading to corrupt data sometimes noted in data warehouse applications."*

The following quote is from the chapter on Data Consistency and Concurrency in the Oracle 10*g* Concepts Manual (Reference [10]). The language has not changed since the days of Oracle 7 and can be traced to an Oracle white paper written by Dr. Kenneth Jacobs ("Dr. DBA") in 1995 (Reference [6]).

> *"Although Oracle serializable mode […] offers many benefits compared with read locking implementations, it does not provide semantics identical to such systems. Application designers must take into account the fact that reads in Oracle do not block writes as they do in other systems.*
>
> ***Transactions that check for database consistency at the application level can require coding techniques such as the use of SELECT FOR UPDATE*** *[editorial emphasis added]. This issue should be considered when applications using serializable mode are ported to Oracle from other environments."*

In another place in the same chapter, the caution is repeated, once again using language from the 1995 paper by Dr. Jacobs.

> *"Because Oracle does not use read locks […], data read by one transaction can be overwritten by another. Transactions that perform database consistency checks at the application level cannot assume that the data they read will remain unchanged during the execution of the transaction even though such changes are not visible to the transaction.*
>
> ***Database inconsistencies can result*** *[editorial emphasis added] unless such application-level consistency checks are coded with this in mind,* ***even when using serializable transactions*** *[editorial emphasis added]."*

The next quote is from Reference [5].

> *"The classical justification for lower isolation levels is that applications can be run under such levels to improve efficiency* ***when they can be shown not to result in serious errors*** *[editorial emphasis added], but little or no guidance has been offered to application programmers and DBAs by vendors as to how to avoid such errors."*

Part of the problem lies in the fact that the necessary academic research has only recently been completed. Here is another quote from Reference [5].

> *"When two official auditors for the TPC-C benchmark were asked to certify that the Oracle SERIALIZABLE isolation level acted in a serializable fashion on the TPC-C application, they did so by "thinking hard about it" […]. It is noteworthy that there was no theoretical means to certify such a fact […]."*

To summarize, application developers must take into account that the default Oracle isolation level does *not* guarantee consistent results and that *program modifications* may be necessary to guarantee consistent results (even when using stricter isolation levels). This is a good time to repeat Tom Kyte's words of warning (Reference [8]).

> *"Unless you understand it, you're probably doing some transactions wrong in your system! ([Do-It-Yourself Referential Integrity] is almost always wrong)"*

Isolation Levels … And All That Jazz!

Concurrency control duties put a heavy burden on the DBMS. For example, if a write-transaction modifies a data item, it is advisable that other transactions not be allowed to read the modified value until the write-transaction commits. "Pessimistic" concurrency control schemes such as those used by Microsoft and IBM achieve this by forcing read-transactions to acquire "read-locks" on the data items they want to read[2]. A read-transaction will not be able to acquire a read-lock if a write-transaction has modified the data item in question and has not yet saved its modifications.

[2] Microsoft SQL Server 2005 will partially follow Oracle's lead and provide a non-locking concurrency scheme similar to Oracle's transaction-level read consistency scheme. However, it will be limited to read-only transactions.

The "READ UNCOMMITTED" isolation level[3] provides the application developers with the ability to signal to the DBMS that read-locks are not necessary because no write-transactions are anticipated (as in the case of a Data Warehouse). The DBMS then no longer has to expend effort in acquiring read-locks and efficiency is thereby improved.

Reference [5] summarizes the situation perfectly.

> *"The classical justification for lower isolation levels is that applications can be run under such levels to improve efficiency* ***when they can be shown not to result in serious errors*** *[editorial emphasis added] …"*

Oracle offers *three* isolation levels, one of which is *not* documented in the Oracle 10*g* manuals.

The default isolation level corresponds to the transaction setting "isolation_level=read_committed" and provides statement-level consistency.

A second, stricter, isolation level, activated using the transaction setting "isolation_level=serializable", provides transaction-wide consistency and is referred to as "snapshot isolation with the first-updater-wins rule" in the academic literature (Reference [5]).

A third, very strict isolation level, activated using the database setting "serializable=true" (Oracle 9*i* and prior versions) or "_serializable=true" (Oracle 10*g*), guarantees serializability, but only at the expense of *table-level* read-locks on *all* tables accessed by the transaction. It is not documented in the Oracle 10*g* manuals.

[3] The READ UNCOMMITTED isolation level is not supported by Oracle.

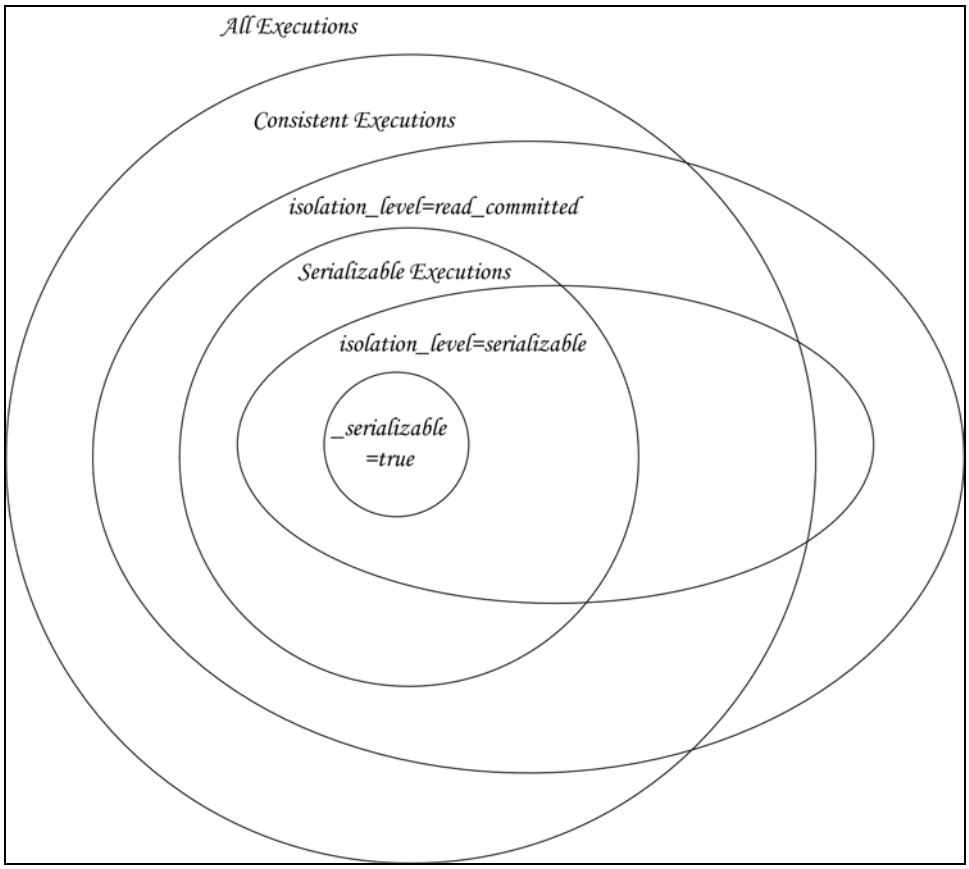

Fig. 2: The relationships between Oracle isolation levels.

Statement-level Consistency

This is the *default* isolation level provided by Oracle and corresponds to the transaction setting "isolation_level=read_committed". Every SQL statement operates on a database snapshot containing *only* data values that were committed *before* the statement began. Every new statement within the same transaction operates on a *different* snapshot and, therefore, this isolation level only provides *statement-level* consistency.

Readers do *not* acquire "read-locks" on rows satisfying their selection criteria and, therefore, do *not* block writers. *Writers* acquire exclusive locks on rows that they modify and, therefore, they block other *writers*, but they do *not* block other *readers*.

If a statement retrieves a data block and finds that it has been modified since the statement began, it searches the "rollback segments" for the prior version of the block. If the prior version has aged out of the rollback segments, the statement fails with the well-known ORA-1555 error: "Snapshot too old".

This isolation level can cause inconsistent results if used in inappropriate circumstances. For example, it does not prevent the "Lost Update" problem described in Reference [3] as follows.

> *"Transaction A retrieves some tuple p at time t1; transaction B retrieves that same tuple p at time t2; transaction A updates [and commits] the tuple (on the basis of the values seen at time t1) at time t3; and transaction B updates [and commits] the same tuple (on the basis of the values seen at time t2, which are the same as those seen at time t1) at time t4.*
>
> *Transaction A's update is lost at time t4, because transaction B overwrites it without even looking at it."*

The Case of the Phantom Fortune

Here is a PL/SQL procedure that you can use to simulate the "Lost Update" problem that can arise when using statement-level consistency. It withdraws the indicated amount from one account and deposits it into a second account. After reading the current balances in each account, it purposely sleeps for 60 seconds to allow you to start another transaction from another database session, this time to transfer money from a third account to the second account (to which you are simultaneously attempting to transfer money in the first session).

You will find that money is successfully subtracted from the first account *and* the third account, but the second account does *not* receive both amounts!

Create a table and populate it as described below. At the start of the test, each account contains exactly ten 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);
```

Here is the PL/SQL program needed for the test. Note that it uses the "sleep" procedure, which is part of the "user_lock" package. To create this package and give execute permissions to public, you will need to log in as SYS and run the "userlock.sql" script in the $ORACLE_HOME/rdbms/admin directory. Also note that the parameter to the sleep procedure is expressed in hundredths of seconds. "userlock.sleep(6000)" suspends program execution for 60 seconds.

```
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(6000);
  --
  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;
```

Execute the following command to transfer five dollars from the first account to the second account.

```
execute debit_credit(1,2,5);
```

Before the first command has been completed, switch to another database session and execute the following command to transfer five dollars from the third account to the second account.

```
execute debit_credit(3,2,5);
```

You will find that both statements complete successfully, however the balance in the second account is only *fifteen* dollars (instead of twenty dollars) even though the balance in the other two accounts has dropped from ten dollars to five dollars. Five dollars has done a vanishing trick!

Transaction-level Consistency

This *non-default* isolation level avoids most errors that can occur at the default isolation level and is activated using the transaction setting "isolation_level=serializable". It is referred to as "snapshot isolation with the first-updater-wins rule" in the academic literature. Every SQL statement operates on a snapshot of the database containing only data values that were committed before the *transaction* began. *Every* statement within the *same* transaction operates on the *same* snapshot and, therefore, this isolation level provides *transaction-level* consistency.

The other significant difference between this non-default isolation level and the default isolation level is that Oracle will *abort* a transaction that attempts to modify a data item that was modified *after* the transaction began[4]. This is called the "first-updater-wins" rule. If you use this isolation level to run the test described in the previous section, the second transaction will abort with the error message "can't serialize access for this transaction" (ORA-8177).

[4] Oracle will also abort a transaction if it cannot verify that the data item was *not* modified after the transaction began. The details can be found in the Oracle 10*g* Concepts Manual (Reference [10]).

Write Skew

While transaction-level consistency does a good job at avoiding a plethora of errors including "Lost Updates" (Reference [3]) as well as "Dirty Reads", "Non-repeatable Reads" and "Phantoms" (Reference [11]), it is subject to a class of error referred to as "Write Skew" (Reference [5]).

Here are three examples of "Write Skew". The first example is taken verbatim from the chapter on Data Consistency and Concurrency in the Oracle 10*g* Concepts Manual (Reference [10]).

> *"One transaction checks that a row with a specific primary key value exists in the parent table before inserting corresponding child rows. The other transaction checks to see that no corresponding detail rows exist before deleting a parent row.*
>
> *In this case, both transactions assume (but do not ensure) that data they read will not change before the transaction completes. The read issued by transaction A does not prevent transaction B from deleting the parent row, and transaction B's query for child rows does not prevent transaction A from inserting child rows.*
>
> *This scenario leaves a child row in the database with no corresponding parent row."*

The second example is from Reference [5].

> *"Suppose X and Y are data items representing bank balances for a married couple, with the constraint that X+Y > 0 (the bank permits either account to overdraw as long as the sum of the account balances remains positive). Assume that initially X = 70 and Y = 80.*
>
> *Transaction T1 reads X and Y, then subtracts 100 from X, assuming it is safe because the two data items added up to 150. Transaction T2 concurrently reads X and Y, then subtracts 100 from Y, assuming it is safe for the same reason."*

The final example is paraphrased from Chapter 3 (Locking and Concurrency) in Tom Kyte's best-selling book, "Expert One-On-One Oracle" (Reference [7]).

Two tables, A and B, initially contain no rows. Session 1 uses transaction-level consistency and executes the command "insert into A select count(*) from B". Session 2, also using transaction-level consistency, executes the command "insert into B select count(*) from A". Both sessions then commit successfully.

Both table A and table B now contain a single data item with value 0. It is easy to see that this cannot happen if the sessions were executed serially in some order.

Ensuring Serializability of Transaction-level Consistency

While transaction-level consistency does not always guarantee consistent results, it *is* possible for a set of transactions using transaction-level consistency to operate "with serializable effect". For example, Reference [5] rigorously proves that the transactions comprising the TPC-C benchmark (Reference [11]) *always* operate with serializable effect when using transaction-level consistency.

Reference [5] also explains how to determine if the transactions comprising an arbitrary application always operate with "serializable effect" when using transaction-level consistency. However, automated tools are not yet available for the purpose and, therefore, this sort of analysis may not be feasible in a system containing thousands of different transaction types.

There are two methods of achieving serializable results when using transaction-level consistency.

The first method is to force Oracle to acquire *table-level* read-locks on *every* table that is read or modified during a transaction. This is achieved using the database initialization parameter "serializable=true" (Oracle 9*i* and prior versions) or the "hidden" parameter "_serializable=true" (Oracle 10*g*).

The second method leverages the "first-updater-wins" rule and requires the enforcement of *one* of the following "sufficient conditions" for *every* pair of write-transactions comprising an application.

1. The transactions must operate on separate tables or on different areas of tables. For example, an application might be coded in such a way that a write-transaction reads and modifies the data of only one department of the organization at a time.

2. Both transactions must update at least one common record. If a suitable record does not exist, then an artificial record can be created. Reference [5] refers to this strategy as "materializing the conflict". If the transactions attempt to execute concurrently, then the "first-updater-wins" rule will cause one of them to fail with an ORA-8177 error (which is the needed behavior).

The above technique is a modified version of a technique listed in an academic paper published a few months ago (Reference [4]). The "SELECT FOR UPDATE" solution (a.k.a. "Do-It-Yourself" Referential Integrity) suggested in the chapter on Data Consistency and Concurrency in the Oracle Concepts Manual (Reference [10]) is a variation of the technique. The same effect can also be achieved by defining the appropriate "foreign-key" constraint[5].

Summary

It is important to understand each isolation level and choose one that maximizes concurrency but avoids inconsistent results. In some cases, program modifications are necessary to avoid inconsistent results.

Acknowledgements

I am grateful to Venkat Devraj, CEO of ExtraQuest Corporation and the author of Oracle 24x7 Tips and Techniques, and to Ravi Kulkarni, Senior Database Administrator at Corio, for carefully reading this essay and providing helpful comments.

References

[1] **ANSI**. X3.135-1992, Database Language SQL, 1993. Available at *http://www.cs.pdx.edu/~len/587/sql-92.pdf*.

[5] Oracle uses "SELECT FOR UPDATE" while checking foreign-key constraints.

[2] R. **Bamford** and K. Jacobs. Method and Apparatus for Providing Isolation Levels in a Database System. United States Patent No. 5,870,758, 1996. Available at *http://www.uspto.gov/*, on payment of a $3 download fee.

[3] C. **Date**. An Introduction to Database Systems, Sixth Edition. Addison Wesley, 1994, Chapter 14.

[4] 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*.

[5] 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*.

[6] K. **Jacobs**, R. Bamford, G. Doherty, K. Haas, M. Holt, F. Putzolu, and B. Quigley. Concurrency Control: Transaction Isolation and Serializability in SQL92 and Oracle7. Oracle White Paper, Part No. A33745, 1995. Available on request from Oracle Support.

[7] T. **Kyte**. Expert One-On-One Oracle. Wrox Press, 2001, Chapter 3.

[8] T. **Kyte**. Inside Multiversioning. Slide presentation at the Northern California User Group Fall 2004 conference, 2004. Available at *http://www.nocoug.org/download/2004-08/RWCons.ppt*.

[9] S. **Lu**, Member, A. Bernstein, and P. Lewis. Correct Execution of Transactions at Different Isolation Levels. 2004. Available at *http://www.cs.wayne.edu/~shiyong/papers/tkde04.pdf*.

[10] **Oracle**. Concepts. 2004, Chapter 13. Available at *http://www.oracle.com/technology/software/index.html*.

[11] **TPC**. TPC-C Benchmark Specification. Available at *http://www.tpc.org/tpcc/*.

Iggy Fernandez is the Lead DBA for a Silicon Valley startup and is Oracle 10*g* certified. Previously, he was the Manager of Database Administration for Corio, an Application Services Provider (ASP) and was responsible for a mixed portfolio of nearly one thousand Oracle and SQL Server databases. He is interested in best practices for Oracle database administration and is writing a book called "A Structured Approach to Oracle Database Administration using Oracle 10*g*" which seeks to apply I.T. Service Management (ITSM) techniques to Oracle database administration. You can contact him at *iggy_fernandez@hotmail.com*.