

Author : Chris Drawater
Date : April 2008
Version : 2.0

Going about Physical Database Schema Design

Abstract

Physical database design for J2EE/Web applications doesn't need to be an overwhelming formal process – it can be kept simple. Success is governed by understanding the design context.

Document Status

This document is Copyright © 2008 by Chris Drawater.

This document is freely distributable under the license terms of the [GNU Free Documentation License](http://www.gnu.org/copyleft/fdl.html) (<http://www.gnu.org/copyleft/fdl.html>). It is provided for educational purposes only and is NOT supported – use at your own risk !

Introduction

The perceived importance of physical database design has changed in recent years. In many J2EE or Web application developments, the Oracle or PostgreSQL component is now viewed as a side issue to, or a by-product of, the Java coding. Rarely is much consideration given to physical database design.

Considering that the time to execute SQL is probably the biggest single component of the response time of most web based database applications (ie. accessing data within tables), this may not be so wise.

This paper hopes to briefly cover the essence of what is required to conduct a successful physical database design.

For the purposes of this paper, we assume an OLTP system using a relational or object-relational database (eg Oracle or PostgreSQL) is under design. However this design process can equally be applied to other database system types such as DW, MIS or Database Appliance hosted databases.

Abbreviation & Definitions

DBMS – Database management System

DDL → (SQL) Data Definition Language

DFD → Data Flow Diagram (from SSADM)

DML → (SQL) Data Manipulation Language

DSS → Decision Support System

DW → Data Warehouse

ER → Entity Relationship

MIS → Management Information System

OLTP → Online Transaction Processing (ie. no data trawling or MIS etc)

Schema → a collection of DB objects (tables, indexes etc) created to support 1 or more applications. Exact definitions vary across databases and are often related to database user accounts.

SSADM → Structured System Analysis and Design Method

UML → Unified Modelling Language – a diagramtic language for object based design

Use Case → UML diagram capturing a functionality requirement

Sequence Diagram → UML diagram describing the sequential flow of inteactions between objects

Design Goal

The basic aim is to generate a balanced physical database schema design that will →

- avoid unnecessary I/O – data must be written or read from the database as efficiently as possible
- allow for table growth
- avoid excessive overhead on data insert/update
- avoid unnecessary database/table maintenance problems
- achieve predefined performance targets
- avoid data/index block hot spots
- be manageable/maintainable within specific backup or maintenance windows
- work well within the deployment environment constraints

Out of the design process will fall the SQL DDL , complete with DBMS specific features such as →

- indexing
- partitions
- storage parameters
- constraints
- domains or types
- block fill factors (data or index blocks)
- possibly de-normalization
- horizontal or vertical data fragmentation
- OOSQL, XML or LOB support
- Referential integrity
- DBMS specific datatypes

plus also, hopefully →

- a spreadsheet based sizing model

Design Overview

Consider, that on the path to generating the SQL DDL, effectively 3 distinct, but closely related, models will be created →

- Logical - Entity or Object model (*the result of some analysis phase*)
- Data - tables/objects implementation (*effectively DBMS agnostic*)
- Physical (Data) - with indexing/partitions etc (*DBMS specific*)

Each model or design stage follows, or is derived from, the previous model.

Occasionally the data and physical model stages can be combined – at any rate, it is these 2 stages that are commonly referred to as the “DB Design”.

In simple terms, the inputs into the design activity are →

- Logical model (*output from any analysis work*)
 - Usage Model (*how the data will be accessed or used*)
 - Deployment considerations (*uptime, resilience etc*)
- and
- Volumetrics (*how much data*)
 - Throughput (*how many interactions per time period & variation through the day etc*)
 - Performance requirements

and the main outputs are →

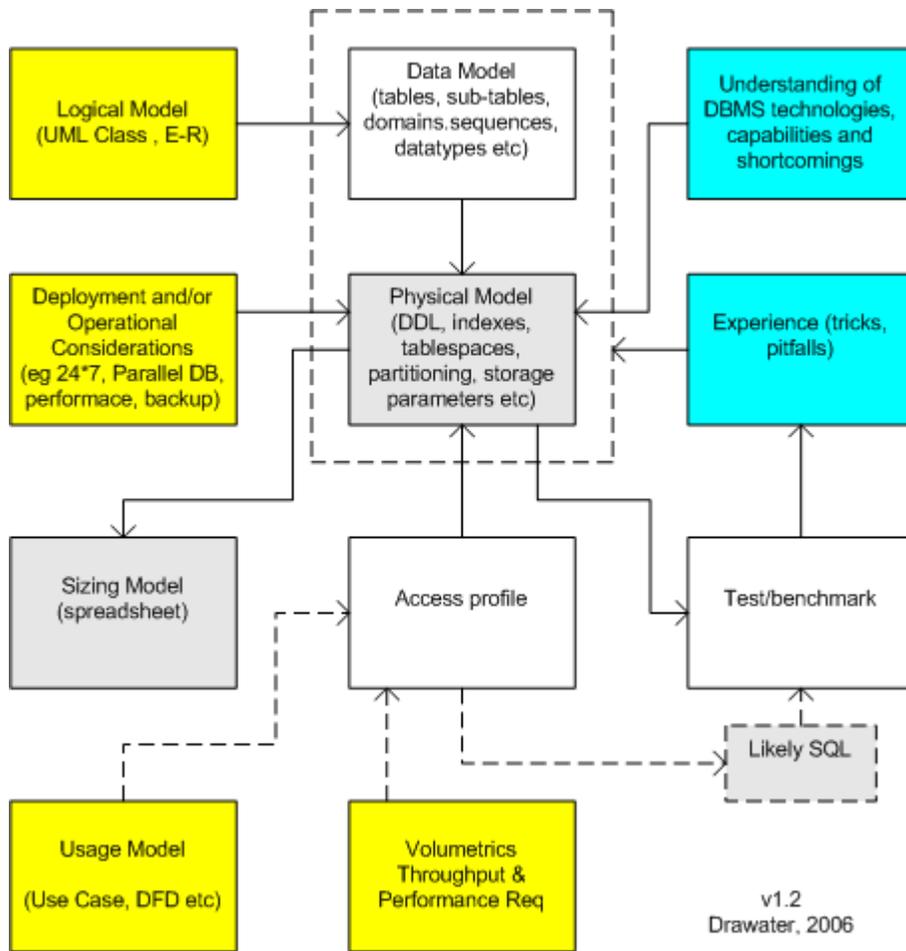
- DBMS specific SQL DDL
- A simple UML Class or similar model (*for documentation and application maintenance purposes*)

and optional (but highly recommended) outputs →

- Sizing spreadsheet
- the likely SQL required for a (pre application code) test or benchmark

Note the minimal documentation – the DDL and the simple UML model (see later section) – all that is required to understand the database schema. Nice and simple.

The relationship between models, inputs and outputs is illustrated below in Figure 1.



In the next few sections, we'll examine more the individual components.

Logical Model

The logical model can take the form of →

- UML
- Spreadsheet with 1 entity/class per sheet
- Good old fashioned ER diagram.

It doesn't matter which diagrammatic techniques or methodologies are used. The important thing is that the data is pre-grouped into entities or objects – think objects, think 'nouns' – if a data body can be described by a 1 word noun (eg employee) then it probably should be a entity/object or subclass of one.

Thinking object wise implicitly installs a fair degree of normalization into logical models....

In addition to an understanding of the basic objects/entities, their field types and sizes, it is also useful to have information such as →

- relationship and cardinality between logical entities
- data volumetrics (for each logical entity)
- level of activity, or whether static
- logical datatypes
- potential key uniqueness/duplicity
- pre-existing data or generated/loaded by the application system
- mandatory or null values
- requirements for system generated keys or OIDs

The (relational but DBMS agnostic) Data Model

The data model is a DBMS independent database design based upon the logical model.

It is at this stage that the designer considers generic indexes, datatypes, de-normalization, referential integrity, primary keys, foreign keys, surrogate keys, constraints etc – all of which should be database agnostic. If possible, it is a good idea to stick with ANSI datatypes and syntax – these can be modified to be DBMS specific in the subsequent physical model, if so desired.

Often as not with OLTP databases , there is a good mapping between the entities/objects and the basic table definitions

It is in the transformation from logical model into data model that different database genre will require different types of tables and levels of normalization →

- OLTP → fairly well normalized (although de-normalized may occur in this stage or during the physical design)
- DW → star or snowflake type schema with fact tables & dimensions, aggregates
- Appliance → normalized or star

Generally, for an OLTP system , we can characterize the tables as →

- Kernel or Core or Primary or Main
- Link or Association or Relationship # eg $A \dashrightarrow B \dashrightarrow C$
- Detail or Collections
- Transient or temp

- Reference or lookup
- Remarks or text
- Queues
- Configuration
- Housekeeping *# eg Max Value*
- Audit trail
- Media Content *# LOB or XML or text*
- Application Index *# like indexing words for text handling*

Most OLTP tables fall into one of these categories – if not, then evaluate carefully the need for them !

Usage Model

Use as much information as available to gain an insight into how the data is likely to be used →

- UML Use Cases and, if available, Sequence Diagrams
- List of business functions or activities(functional decomposition)
- SSADM DFD's
- Anything that describes the interactions against the logical model

Understand the deployment requirements/constraints

Deployment conditions can heavily impact the physical database design. For example, a 24*7 shop will have few downtime windows for database maintenance such as table/index restructuring.

Understand exactly what is required. This might include, amongst others things →

- A parallel implementation : Oracle RAC or grid ? (*hot spot shared data/index blocks and locking may be an issue*)
- 24 * 7 (*online index rebuilds and other object restructuring*)
- MIS/DSS Data Warehouse (*Star or snowflake schema*) or OLTP (*normalized data model*)
- Level of DBA support
- Performance requirements (*max response times, priorities*)
- Backup/recovery (*static tables, read only components*)
- Database maintenance windows (*time to re-organize a large table or index*)
- replication specific requirements (*often require a Primary key or OID or supplemental logging, possibly don't support all datatypes : LOBs etc*)

Understand the DBMS technology

If you do not understand what technologies are provided by the DBMS – their capabilities and limitation , how can use them to achieve your goal ?

Amongst other things, check out →

- IO clustering technologies (*IOT, hash or index clusters, hash tables, partitions*)
- Types of indexes (*Btree, hash, bitmap*)
- Parallel DML or DDL support
- Sequence Numbers

- ANSI compatibility
- OOSQL (*inheritance, nested tables, types*)
- LOB or XML support
- domains,
- datatypes (*type & max sizes*)
- storage (*tablespaces, block fill factors, extent sizes, transportable tablespaces etc*)
- partitions or placement across filesystems
- character sets (*how many Thai characters can a varchar column hold ?*)
- DBMS buffer block sizes (*may impact max column sizes*)
- Online maintenance support
- DBMS data caching technologies
- Referential integrity support
- Constraint support
- Replication (*requirements on tables etc*)

Lastly but most importantly, understand how the DBMS locking works (*does it use shared read locks ?, levels of granularity, lock escalation etc*).

Access Profiles

Data Access profiles (Ref 1) are extremely useful in database design (pretty much obligatory for any serious complete benchmarking) but are often ignored .

Each logical business transaction or UML Sequence Diagram (even Use Case) in our usage model can be broken down into several references to tables within our data model. These approximate to potential SQL statements or groups of SQL. Each transaction or activity will manipulate rows of data and an overview of the different types of transactions against tables can be generated - an access profile.

In conjunction with our volumetric information, access profiles can be generated for either →

- all business/service interactions
- Crucial or key interactions

Use a spreadsheet to create an access profile with the following columns →

- | | |
|------------------------------------|---|
| • Interaction/function | <i>eg. get customer details</i> |
| • Relative Priority | <i>(critical, high, med, low)</i> |
| • Table | <i>eg. cust</i> |
| • SQL access type | <i>eg. unique key , key range or scan</i> |
| • Likely SQL (even if multi table) | <i>eg. select.....</i> |
| • Table rows/ SQL restricted on : | <i>eg. cust_id, *, like 'XYZ%'</i> |
| • rows manipulated | <i>eg. 30</i> |
| • frequency (per hour or day etc) | <i>eg 20/sec</i> |

then sort by table/frequency and carefully examine how each table will be accessed. Is the workload mix what was expected ?

(If a function hits 4 tables then have 4 rows, or if the interaction is a 2 table select then again use 2 rows)

These profiles should clarify or indicate →

- the crucial SQL (useful for benchmarking)
- potential index keys **
- join keys
- potentially updateable keys
- potential IO clustering **
- relative workload against tables
- table storage parameters (for tables with a lot of updates)
- data/index hotspots
- data or index ordering importance or issues
- surrogate keys
- static tables

** cross-check against cardinality shown in the logical models/UML or ER diagrams

Access profiles can also provide a good estimate as to the number of (crucial) SQL transactions per day the system must cope with, and can reveal potential application functionality problems.

Physical Model

The physical design is the deployment design specific to a DBMS and is closely derived from the data model and heavily influenced by the choice of DBMS, the data access profile and environmental requirements. Using the DBMS specific features (see earlier), the model is tuned for both optimal performance AND functionality – for example, in a 24*7 shop it's no use having an initially fast index which degrades quickly but cannot be re-built online!

The DBMS specific physical design DDL is often a superset of the data model design DDL.

For a software vendor supporting many different databases, there should only be 1 DBMS agnostic design but would be more many (derived) physical DBMS specific models.

Testing the Model

If the physical database can be validated before coding, then potentially poor table structures and SQL (the root of most application performance issues) can be identified and tuned.

So test the databases before coding – create the DB from the DDL (ie, the physical model) and run the key (likely) SQL statements identified from the Access Profiles.

To test in a realistic manner →

- Use some quickly hacked out PL/SQL or Java/JDBC to artificially generate data
- Use representative ranges of data
- Run concurrent processes or streams of SQL via the DBMS command line too;
- Test a fairly large volume of data (not 10 rows, use 10s – 100s of GB of data)

Sizing Models

The sizing of the database (or more precisely, the schema objects in the database) is DBMS specific and its calculation requires the exact physical model to be known.

Once the tables, indexes , fill factors and data block size and DBMS block overheads are understood then a data volume sizing spreadsheet can be created.

For each table and each associated index →

- calculate the average row size (see the DBMS specific manual) , then
- the rows per database block (*don't forget the fillfactors etc*), and finally
- the number of blocks required to host all the rows.

The sum total of all the blocks required for all the tables/indexes is the approximate schema size.

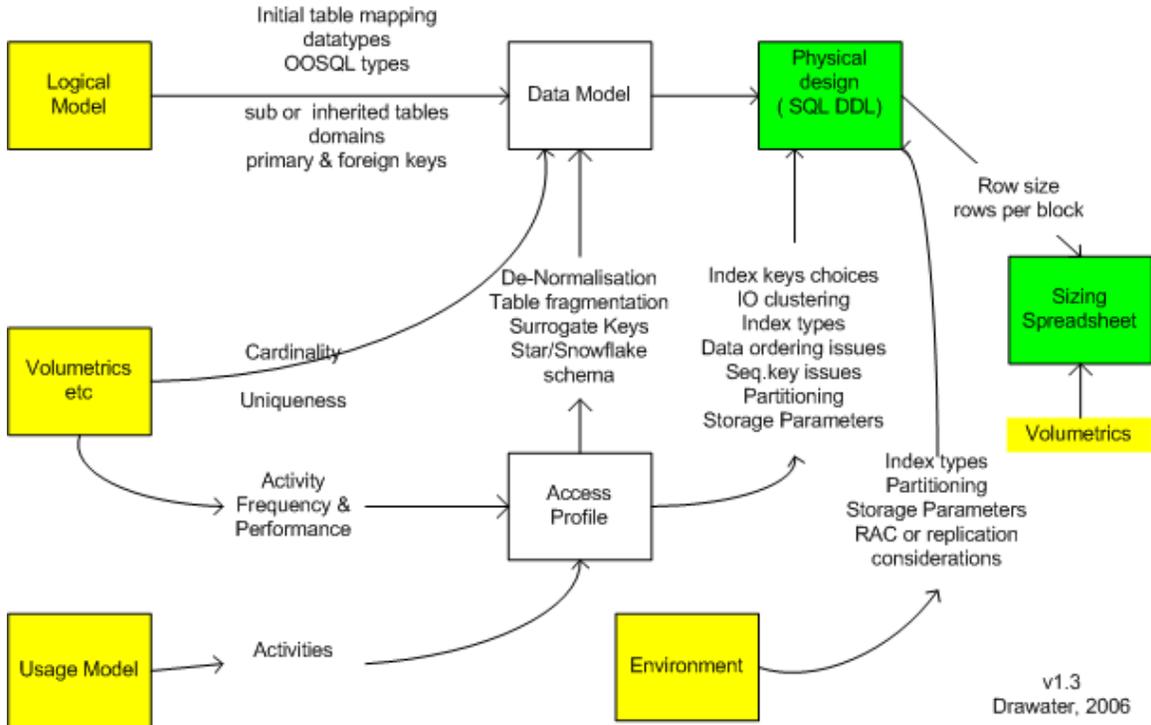
Note on Data and Physical Model diagrams

The author's preference is to use UML Class Diagrams (with <<table>> and other stereotypes) to visually illustrate the nature and cardinality of the relationships/associations between tables (= classes).

There is no need to specify the class attributes (ie effectively the columns) as the actual table column names and datatypes, are documented in the DDL. However, if so desired, any OOSQL related methods associated with the tables can be listed as class operations/methods.

Concluding Remarks

The basic design process is summarized below in Figure 2 – use whichever bits work for your database design work.



Remember - at its optimal, physical database design should be part of a holistic system design, stretching from OS tuning and backup all the way to the JDBC SQL. Never forget the design context.

So how long will the physical DB design take ? There's no real answer. Some people rigorously and formally follow a set process pattern. Others seem to be able to create designs more intuitively and take very little time – experience can count for a lot in this respect.

Chris Drawater has been working with RDBMSs since 1987 and can be contacted at drawater@btinternet.com.

References

1. Drawater (1991), Databases - Physical Database Design, *Computing* 18/04/91