

Index

■ A

- The Agile Manifesto, 390
 - interface, definition, 392
 - process and standardization, 391
 - schemas and users, 394
 - simplicity and minimizing work, 392
 - TDD, 393
 - traditional values, 391
- Amdahl's Law, 46
- Application server
 - applicability limits, 318
 - Avaloq Banking System, 313–315
 - business logic benefits (*see* Three-tier architecture, business logic benefits)
 - soft factors, 318
- Automated code analysis, 171, 173
 - analyzing your code, 174–175
 - dynamic analysis, 174 (*see also* Dynamic analysis)
 - vs.* instrumentation, 175
 - static analysis, 173–174 (*see also* Static analysis)

- Avaloq Banking System, 313–315. *See also* Three-tier architecture, business logic benefits

■ B

- Bulk binding, 121, 143
 - DML, 144–145
 - error handling, 151–155
 - LOG ERRORS clause, 156–157
 - robust, 157–162
 - SAVE EXCEPTIONS with batches, 155–156
 - improvements in 11g, 150–151
 - memory usage monitoring, 148–150
 - performance measurement, 145–148
- Bulk fetching, 121
 - associative array, 127, 128
 - explicit cursor bulk mode, 126
 - explicit fetch calls, 124–125
 - explicit fetch calls bulk mode, 126
 - implicit cursor, 124
 - implicit cursor bulk mode, 125–126
 - implicit fetch calls, 125–127

- implicit fetch calls bulk mode, 126–127
- nested table, 127
- varray, 127

■ C

Code generators, 42

Code modularization

- composability *vs.* decomposability, 327
- continuity, 326
- decomposability, 326
- few interfaces and layering, 327
- information hiding (abstraction), 327
- multiple packages/submodules, 332–335
- packages and associated tables, 327–328
 - access detection by shared pool probing, 329
 - access detection by static analysis, 328
 - access detection with fine-grained auditing, 330–332
 - access detection with triggers, 329–330
- reusability, 326
- schemas as modules, 336
 - migration path, 339
 - object privilege grants, 336–338
 - optimal usage, 338–339
 - other schema object access, 338

- with schemas *vs.* within schemas, 342–343

- within schemas, 339–342

- single task, 327

- small interfaces, 327

- understandability, 326

Code recompilation, 90

Code refactoring

- cursor loop, 137–140

- exception, 135–137

- over bulking, 140–143

Coding conventions

- dynamic code, 431–431

- error handling, 426, 437

- check data and display interface, 441

- error recovery, 440–441

- error reporting, 438–440

- error trapping, 438

- formatting, 427

- case, 427

- comments, 427

- indentation, 428

- functions, 437

- maintainability, 426

- packages, 432–433

- reliability, 426

- security, 426

- speed in coding, 426
- stored procedures, 433
 - calls, 435
 - constants, 435
 - global variables, 436
 - local procedures and functions, 436
 - local variables, 435
 - naming prefixes, 433–434
 - parameters, 434
 - procedure metadata, 436–437
 - types, 436
- trainability, 426
- Contract-Oriented Programming, 213
 - bug-free code principles
 - adopt function-based interfaces, 231
 - assert preconditions rigorously, 229–230
 - avoid correctness-performance trade-offs, 232–233
 - crash on ASSERTFAIL, 231
 - modularize ruthlessly, 230–231
 - Oracle 11g optimized compilation, 233
 - regression test, 231–232
 - contract patterns
 - FUNCTION RETURN BOOLEAN Not-NULL, 228
 - FUNCTION RETURN Not-NULL, 227
 - Not-NUL IN/Not-NULL OUT, 227
 - RETURN TRUE/ASSERTFAIL, 228–229
 - design by contract, 213
 - assertions, 215
 - book references, 215
 - invariants, 215
 - postconditions, 214–215
 - preconditions, 214
 - software contracts, 213–214
 - implementing PL/SQL contracts
 - basic ASSERT procedure, 216–217
 - enforcing contracts, 220–222
 - function prototype, 223–224
 - improvements, 222–223
 - local/global assert, 219–220
 - standard package-local assertion, 218–219
 - testing odd and even integers
 - complementary function for evenness, 225–226
 - function prototype and coding conventions, 224–225
 - minimal implementation, 225
- Cost-Based Optimizer (CBO), 242
 - default statistics, 279–281
 - extended statistics, 275–279

- Extensible Optimizer
 - dynamic PL/SQL function statistics, 284–285
 - PL/SQL function procedure, 281–282
 - PRODUCTS table, data profile, 282
 - statistics type, 282–284
 - function-based indexes, 273–275
 - Query Transformation and Function Executions, 242
 - displaying function calls, 243
 - PL/SQL function, projection of, 243
 - Cursors, 291
 - code indentation, 430
 - dynamic REF cursors (*see* Dynamic REF cursors)
 - explicit cursors, 292
 - anatomy, 293–294
 - bulk processing, 294–295
 - business requirement, 292
 - open, fetch and close, 292, 293
 - REF cursors, 295–296
 - two-fold problem, 293
 - implicit cursors, 296
 - anatomy, 297–299
 - extra fetch theory, 299–300
 - for ... in cursor, 297, 299
 - select ... into cursor, 298–299
 - TKPROF output, extra fetch test, 299
 - static REF cursors (*see* Static REF cursors)
- D**
- Data abstraction. *See* Information hiding
 - Database agnostic*, 370
 - Database Vault, 339, 340, 341, 342
 - Dependency chains, 445, 451
 - adding components to packages, 462–466
 - compilation, 446
 - datatype reference, 457–458
 - DBA_DEPENDENCIES, 446
 - definitions, 445
 - execute immediate, 448
 - parsing, 448
 - procedure recompilation, 447
 - REFERENCED_NAME, 447
 - resource locking, 466–467
 - RESUSE SETTINGS, 446
 - shortening, 452–456
 - statement execution, 448
 - synonyms, 466
 - SYS_STUB_FOR_PURITY_ANALYSIS, 451–452
 - table alterations, view, 458–462

- triggers, 467
 - create/replace, 470–471
 - follows tr_pay_risk_rating, 470
 - high risk payments, 469
 - initially disabled, 471–472
 - multiple triggers, 468
 - RISK_RATING, 468
 - tr_pay_follow_up, 471
 - update trigger, 472
- Dynamic analysis, 174
 - code coverage testing, 202–204
- DBMS_HPROF, 206
 - ANALYZE function, 207
 - configuring, 207
 - example session, 207–208
 - output, 208–209
 - plshprof, 207
- DBMS_PROFILER and DBMS_TRACE, 197
 - configuring, 197–198
 - example profiling, 198–199
 - output, 199–202
- performance metrics, 202, 204
- performance profile, 197
- unexecuted code, 204–206
- unintentional side benefit, 197

- Dynamic REF cursors
 - DBMS_SQL, 311–312
 - procedure, 308
 - SQL injection, threat, 309–311
- Dynamic SQL handling, 19
 - DBMS_SQL package, 27–28
 - dynamic cursors, 24–27
 - dynamic thinking, 28–33
 - native dynamic SQL, 21–23
 - object dependencies, 41
 - negative effects, 41–42
 - positive effects, 42–43
 - performance and resource utilization, 37–38
 - anti-patterns, 38–39
 - implementation types, 39–41
 - security issues, 33–37

E

- Eiffel, 229
- Entity-relationship diagram, 334, 335
- Error handling, 151, 437
 - check data and display interface, 441
 - error recovery, 440–441
 - error reporting, 438–440
 - humans, 438
 - machines, 438
 - output parameters, 438

error trapping
 exceptions, 438
 skeleton procedure, 438

LOG ERRORS clause, 156–157

robust bulk bind, 157
 contiguous array, 158
 earlier versions of Oracle, 161–162
 non-contiguous array, 158–161

SAVE EXCEPTIONS with batches, 155–156

Evolutionary data modeling, 367
 database and agile development, 369–370

database designs, 371

database refactoring
 Abbreviated Employee Data for Department, 375
 Employee Data Request, 374–375
 Human Resources schema, unchanged, 372, 373

PL/SQL (*see* PL/SQL, evolutionary data modelling)

system development, 368–369

Exception handling. *See* Error handling

Explicit cursors, 292
 anatomy, 293–294
 bulk processing, 294–295

business requirement, 292

open, fetch and close, 292, 293

REF cursors, 295–296

two-fold problem, 293

■ F

F-bounded polymorphism, 344

Formatting code, 427
 case, 427
 comments
 advantage, 427
 comparisons, 427

indentation, 428
 case statements, 430
 cursors, 430
 for loop statements, 430–431
 if statements, 429–430
 insert statements, 429
 select statements, 428–429
 string concatenation, 431
 update and delete statements, 429

French Horn, 396

■ G

Generic types, 344

Gustafson's Law, 46

H

Human Resources (HR) database schema

- target design, 380
- unchanged, 372

I, J, K

Identifiers

- data type, 185, 186
- scope of, 185
- types, 184
- unused identifiers, 192–194
- USER_IDENTIFIERS, columns, 186

Implicit cursors, 296

- anatomy, 297–299
- extra fetch theory, 299–300
- for ... in cursor, 297, 299
- select ... into cursor, 298–299
- TKPROF output, extra fetch test, 299

Information hiding, 327

Instrumentation, 175

- built-in profiles, 409–410
- conditional compilation, 409
- definition, 402
- extended SQL trace data, 410
- identification
 - DBMS_APPLICATION_INFO.SET_MODULE, 407–409

DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS, 405–406

DBMS_SESSION.SET_IDENTIFIER, 404–405

ILO (*see* Instrumentation Library for Oracle (ILO))

- measurement intrusion, 404
- program slow, 402–404

Instrumentation Library for Oracle (ILO), 410

- instrumenting code benefits, 411
- performance problem diagnosis, 414–418

SQLCODE and SQLERRM, 412

triggers, insert/update, 413

working procedure, 411

Invalidations, 445

dependency chains, 455 (*see also* Dependency chains)

packages, 455

view

ADJUST function, 459–460

table TRANS, 458–459

Iterative design, 369

L

Laws of computing, 46

LISTAGG function, 241

Local Problem Fatal (L_PB_FATAL), 439

Logical loops, 43

■ **M, N**

- Manual code analysis, 173
- MapReduce programming model, 45, 51.
 See also Parallel execution servers,
 MapReduce
- Memory allocation, 361
- Memory deallocation, 361
- Metamodel, 333–335
- Moore’s Law, 46

■ **O**

- Object-oriented programming
 - additional attributes and methods in
 subtypes, 354–355
 - assessment, 356
 - dynamic dispatch with dynamic SQL,
 352–353
 - dynamic dispatch with static
 dispatcher, 353–354
 - multiple instances and in-memory
 references, 348–351
 - persistence of objects as records, 355
 - reuse of data and associated
 functionality, 343
 - reuse with adaptations, 343
 - subtypes, 351–352
 - user-defined types, 344–345
 - inheritance and dynamic dispatch,
 345–346
 - limitations, 347–348
 - persistence, 348

- OLTP. *See* Online transaction processing
 system (OLTP)
- Online transaction processing system
 (OLTP), 50
- Oracle SQL Developer, 100
- Oracle’s data dictionary
 - INHERITED and OVERRIDING, 183
 - PARAMETERS and RESULTS, 182
 - USER_ARGUMENTS, 179–181
 - USER_DEPENDENCIES, 177
 - USER_PROCEDURES, 177–179
 - USER_SOURCE, 176–177
 - USER_TRIGGERS, 183
 - USER_TYPE_METHODS, 182–183
 - USER_TYPES, 181–182

■ **P**

- Packages
 - kernel packages, 432
 - naming, 433
 - user interface, 433
- Parallel execution servers, MapReduce
 - guidance, 69
 - parallel pipelined table functions, 69
 - pipelined table function, 52
 - ordering *vs.* clustering, 62
 - partition by any, 63
 - partition by hash, 63

- partition by range, 63
- partitioning, 62
- Parallel hardware architectures
 - clustered systems of SMP computers, 48
 - distributed/networked systems of SMP computers, 48
 - multi-core single processor single computer, 48
 - single core single processor isolated computer, 48
 - symmetric multiprocessing systems (SMP), 48
- Parallel processing, 45
 - Amdahl's Law, 46
 - benefits, 45
 - candidate workloads, 50
 - non-OLTP workloads, 50–51
 - OLTP systems, 50
 - data growth, 47
 - vs.* distributed processing, 47
 - goals, 48
 - degree of parallelism, 49
 - scaling up, 49
 - speedup, 49
 - Gustafson's Law, 46
 - hardware architectures, 47
 - clustered systems of SMP computers, 48
 - distributed/networked systems of SMP computers, 48
 - multi-core single processor single computer, 48
 - single core single processor isolated computer, 48
 - symmetric multiprocessing systems (SMP), 48
- MapReduce programming model, 45, 51
 - guidance, 69
 - parallel pipelined table functions, 69
 - pipelined table functions, 52–69
- Moore's Law, 46
- PL/SQL, 51
 - job queue processes, 52
 - parallel DDL, 51
 - parallel DML, 51
 - parallel execution servers, 52
 - parallel query, 51
 - user session processes, 52
- Parallel programming technique. *See* MapReduce programming model
- Performance, 395
 - definition, 395, 396
 - feature aspects, 398–399
 - functional requirements, 396
 - instrumentation (*see* Instrumentation)

- Method R
 - identify the task, 414
 - measure the task, 415
 - optimize, 415
 - repeat, 416
- musical note, 396
- profiling (*see* Profiling)
- resource utilization
 - Matt’s payroll report, 398
 - Nathan’s sales report, 398
- response time, 397
- throughput, 397
- Pipelined table functions, 52
- PL/SCOPE
 - configuration, 187–188
 - data type checks, 196
 - identifiers (*see* Identifiers)
 - impact analysis
 - database application changes, 188
 - PLSCOPE_SUPPORT_PKG, 189
 - PRINT_IMPACTS, 189
 - schema changes, 188
 - naming standards compliance
 - validation, 191–192
 - scope issues, 194–195
- PL/SQL, 1
 - access layer creation
 - Alter GetEmployeeData Procedure, 383–385
 - Create
 - GetEmployeeData_MultiPhoneNo Procedure, 386–387
 - Execution and Results Set Procedure, 378–379
 - Execution Script Procedure, 378
 - Extended Phone Number Functionality Testing, 388–390
 - Human Resource schema, target design, 380
 - Refactoring Exercise, API, 379–380
 - Return Employee Data by Department Procedure, 376–378
 - Verifying and Recompiling related objects, 381–382
- bulk SQL operations, 121
 - array processing, 123–124
 - bulk binding, 121, 143–162
 - bulk collect overhead monitoring, 131–135
 - bulk fetching, 121, 124–127, 143
 - client bulk processing, 164–168
 - code refactoring, 135–143
 - collection-style datatypes, 127–131
 - DIY-style store, 121–122
 - examples, 122
 - memory consumption, 162–164
- conditional compilation, 71, 83
 - benefits, 87–89

- code generation, 86–87
- controlling, 91–92
- directives, 83–85
- inquiry variables, 93–96
- invalidations, 89–91
- context switch, 3
- DUAL table access, 8
 - arithmetic operations/DATE manipulations, 8–9
 - master table, 10
 - sequences, 9–10
- evolutionary data modeling
 - Agile Manifesto, 390–392
 - architect, application design, 393
 - interface, definition, 392
 - schemas and users, 394
 - TDD, 393–394
- excessive commits, 17
- excessive database link calls, 16
- excessive function calls, 10
 - costly function calls, 13–15
 - unnecessary function execution, 11–13
- excessive parsing, 17–18
- excessive use of triggers, 17
- interpreted *vs.* native compilation, 13
- lookup queries, 5–8
 - array-caching technique, 6, 8
 - associative array, 7–8
- memory management, 356
 - associative array index, 362, 363, 364, 365
 - memory usage measurement, 356, 358, 359, 360, 361
- nested row-by-row processing, 4–5
- object-oriented programming, 343
 - additional attributes and methods in subtypes, 354–355
 - assessment, 356
 - dynamic dispatch with dynamic SQL, 352–353
 - dynamic dispatch with static dispatcher, 353–354
 - multiple instances and in-memory references, 348–351
 - persistence of objects as records, 355
 - reuse of data and associated functionality, 343
 - reuse with adaptations, 343
 - subtypes, 351–352
 - user-defined types, 344–345 (*see also* User-defined types (UDT))
- parallel processing
 - job queue processes, 52
 - parallel DDL, 51
 - parallel DML, 51
 - parallel execution servers, 52
 - parallel query, 51

- user session processes, 52
- programming, 313
 - abbreviations, 320–323
 - business logic, 313
 - code and data modularization (*see* Code modularization)
 - database as application server (*see* Application server)
 - prefixes and suffixes for identifiers, 323–326
 - requirements, 319
 - uniformity, 319, 320
- row-by-row processing, 1–4
 - rewritten code, 3–4
 - MERGE statement, 5
 - nested cursors, 4–5
- vs.* SQL, 1, 18
- warnings, 71
 - ALTER SESSION privilege, 72
 - compilation, 80–82
 - elimination, 78
 - vs.* errors, 76–78
 - ignore, 78
 - PLW-05005, 79
 - PLW-06002, 79
 - PLW-06010, 79
 - PLW-07204, 79
 - informational-type, 71
 - level, 72
 - other types, 82
 - performance-type, 71
 - severe-type, 71
 - usage, 73–76
- PL/SQL functions, 235
 - CBO (*see also* Cost-Based Optimizer)
 - context-switching, 236–241
 - encapsulating SQL lookups
 - autotrace output, SQL with PL/SQL, 246
 - autotrace output, SQL-only report, 247
 - create PL/SQL function, 245
 - SQL with PL/SQL, 245
 - SQL-only solution, 246
 - tkprof output, 247
 - executions, 242
 - CBO, 242–244
 - internal optimizations, 244
 - query rewrite, 242–244
 - LISTAGG functions, 241
 - NO_DATA_FOUND exception, 255
 - optimizer difficulties, 249–251
 - parallel query, 255
 - performance, 236
 - predicate ordering, 251–253
 - predictability, 236

- read-consistency trap, 253–255
 - reduce the impact of, 256
 - sense of perspective, 256–257
 - SQL, 258
 - SQL alternatives, 257–258
 - views, 258–262
 - virtual column, 262–264
 - reducing executions, 264
 - caching techniques, 266–271
 - deterministic functions, 271–273
 - pre/post computing, SQL hints, 264–266
 - side effects, 236
 - STRAGG functions, 241
 - suboptimal data access, 245–248
 - tuning technique, 285–289
 - PLSHPROF, 207
 - elapsed time report, 210
 - function drill down, 210
 - index report, 209
 - Profiling
 - benefits, 401
 - definition, 400
 - performance problem diagnosis, 419–422
 - response time, decomposition of, 400, 401
 - sequence diagram, 400
 - Program profiling, 174
- Q**
- Quest Code Tester for Oracle, 99
- R**
- Read No Package State (RNPS), 451
 - Read-consistency trap
 - create PL/SQL function, 253
 - Session One’s long running query, 254
 - Session One’s Resultset, 254
 - Session Two’s customer update, 254
 - REF cursors
 - dynamic REF cursors, 296, 307
 - static REF cursors, 296, 301
 - Remote objects, 42
 - Reusability, 343
- S**
- Slow-by-slow processing. *See* PL/SQL, row-by-row processing
 - SQL
 - bulk operations, 121
 - array processing, 123–124
 - bulk binding, 121, 143–162

- bulk collect overhead monitoring, 131–135
- bulk fetching, 121, 124–127, 143
- client bulk processing, 164–168
- code refactoring, 135–143
- collection-style datatypes, 127–131
- DIY-style store, 121–122
- examples, 122
- memory consumption, 162–164
- code generators, 42
- dynamic handling
 - DBMS_SQL package, 27–28
 - dynamic cursors, 24–27
 - dynamic thinking, 28–33
 - native dynamic SQL, 21–23
 - object dependencies, 41–43
 - performance and resource utilization, 37–41
 - security issues, 33–37
- logical loops, 43
- memory management, 356
 - associative array index, 362–365
 - memory usage measurement, 356–361
- object-oriented programming, 343
 - additional attributes and methods in subtypes, 354–355
 - assessment, 356
 - dynamic dispatch with dynamic SQL, 352–353
 - dynamic dispatch with static dispatcher, 353–354
 - multiple instances and in-memory references, 348–351
 - persistence of objects as records, 355
 - reuse of data and associated functionality, 343
 - reuse with adaptations, 343
 - subtypes, 351–352
 - user-defined types, 344–345 (*see also* User-defined types (UDT))
- programming, 313
 - abbreviations, 320–323
 - business logic, 313
 - code and data modularization (*see* Code modularization)
 - database as application server (*see* Application server)
 - prefixes and suffixes for identifiers, 323–326
 - requirements, 319
 - uniformity, 319–320
- remote objects, 42
- Static analysis, 173
 - data dictionary, 176
 - USER_ARGUMENTS, 179–181
 - USER_DEPENDENCIES, 177
 - USER_PROCEDURES, 177–179

- USER_SOURCE, 176–177
- USER_TRIGGERS, 183
- USER_TYPE_METHODS, 182–183
- USER_TYPES, 181–182
- PL/SCOPE (*see* PL/SCOPE)
- source text searching, 328
- user dependencies, 328

Static REF cursors

- and client, 304–305
- cursor variable restrictions, 303–304
- goals, 303
- package parameter specifications, 301
- parsing, 305–307
- statement caching, 303
- strong typed, 302
- weak-typed, 302

STF\$ function, 437

STRAGG function, 241

SYS_STUB_FOR_PURITY_ANALYSIS, 451

■ T

Test driven development (TDD), 393–394

Three-tier architecture, business logic benefits

- availability from any environment, 315
- centralized single-tier deployment, 316
- consistency, 315
- distributed transactions, 316

- Java stored procedures, 317
- .NET and C-based external procedures, 317
- performance, 315
- PL/SQL stored procedures, 316
- scalability, 316
- security, 315
- simplicity, 315
- stability and reliability, 316

Tuning PL/SQL

- session cache function, 287–288
- user defined, 286–287

■ U, V

Unit test repository

- create/select the repository, 101–102
- importing tests, 103
- managing the tests, 102–103
- saving tests, 108

Unit testing, 97

- broadening
 - adding static lookup values, 109–110
 - dynamic query creation, 112–113
 - seeding test implementations, 111–112
 - static drop-down list, 110, 111
- build tests, 98–99
- code coverage statistics, 106

- in command line, 119–120
- create/select the repository, 101–102
- create/update database connection, 100, 101
- debugging tests, 98, 108, 109
- definition, 98
- features
 - building suites, 119
 - exporting and importing tests, 117–118
 - library components, 114–117
 - running reports, 113
 - synchronizing tests, 118–119
- first implementation creation, 104–105
- importing tests, 103
- maintaining the repository, 102–103
- parameters, 106, 107
- procedure, 103–104
- process validation, 107, 108
- running tests, 108
- saving tests, 108

- startup and teardown processes, 105–106
- tools, 99
 - Oracle SQL Developer, 100
 - Quest Code Tester for Oracle, 99
 - utPLSQL, 99
 - Unit Test Wizard, 104
- Upcall/extension mechanism, 334
- User-defined types (UDT)
 - inheritance and dynamic dispatch, 345–346
 - limitations, 347–348
 - persistence, 348
- utPLSQL, 99
- ututil, 119

■ W, X

- Workload balancing, 50–51
- Write No Database State (WNDS), 451

■ Y, Z

- You Ain't Gonna Need It (YGANI), 371