

# A Practical Introduction to the Oracle Relational Database

Copyright 2002, John B. Matthews  
Distribution per GNU Free Documentation License  
<http://www.gnu.org/copyleft/fdl.html>

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b><i>Introduction to Oracle</i></b> ..... | <b>3</b> |
| 1.1      | Prerequisites .....                        | 3        |
| 1.2      | Objectives.....                            | 3        |
| 1.3      | Methods.....                               | 3        |
| 1.4      | Resources .....                            | 3        |
| <b>2</b> | <b><i>Databases</i></b> .....              | <b>4</b> |
| 2.1      | Abstraction .....                          | 4        |
| 2.2      | Tabular model.....                         | 4        |
| 2.3      | Hierarchical model .....                   | 4        |
| 2.4      | Network model.....                         | 4        |
| 2.5      | Relational model .....                     | 4        |
| 2.6      | Object-relational .....                    | 5        |
| 2.7      | Physical structure .....                   | 5        |
| 2.8      | Logical structure.....                     | 6        |
| 2.9      | Oracle processes.....                      | 7        |
| 2.10     | Memory structure.....                      | 7        |
| 2.11     | Distributed Database.....                  | 8        |
| 2.12     | Database Objects .....                     | 8        |
| <b>3</b> | <b><i>SQL*Plus</i></b> .....               | <b>9</b> |
| 3.1      | Overview .....                             | 9        |
| 3.2      | Executive commands .....                   | 10       |
| 3.3      | Editing commands .....                     | 10       |
| 3.4      | Buffer commands.....                       | 11       |
| 3.5      | Formatting commands.....                   | 12       |
| 3.6      | Interactive commands.....                  | 13       |
| 3.7      | Variables .....                            | 15       |
| 3.8      | Example .....                              | 15       |

|          |  |           |
|----------|--|-----------|
| <b>4</b> | <b><i>SQL</i></b> .....                | <b>17</b> |
| 4.1      | Introduction.....                      | 17        |
| 4.2      | Statements.....                        | 17        |
| 4.3      | Basic elements .....                   | 19        |
| 4.4      | Operators.....                         | 20        |
| 4.5      | Functions.....                         | 22        |
| 4.6      | Expressions, conditions & queries..... | 22        |
| 4.7      | Constraints .....                      | 23        |
| 4.8      | Examples.....                          | 24        |
| 4.9      | Views .....                            | 25        |
| <b>5</b> | <b><i>PL/SQL</i></b> .....             | <b>25</b> |
| 5.1      | Introduction.....                      | 25        |
| 5.2      | Data types .....                       | 26        |
| 5.3      | Program structure .....                | 27        |
| 5.4      | Cursors.....                           | 29        |
| 5.5      | Collections.....                       | 32        |
| 5.6      | Subprograms .....                      | 33        |
| 5.7      | Database Triggers.....                 | 36        |
| 5.8      | Errors .....                           | 37        |
| <b>6</b> | <b><i>HTML</i></b> .....               | <b>37</b> |
| 6.1      | Oracle Application Server (OAS).....   | 37        |
| 6.2      | WebDB/Portal.....                      | 41        |
| <b>7</b> | <b><i>Java</i></b> .....               | <b>41</b> |
| 7.1      | Java stored procedures .....           | 41        |
| 7.2      | JDBC .....                             | 42        |
| <b>8</b> | <b><i>Forms</i></b> .....              | <b>43</b> |
| 8.1      | Modules.....                           | 43        |
| 8.2      | Blocks .....                           | 43        |
| 8.3      | Object navigator .....                 | 44        |
| 8.4      | Property window.....                   | 44        |
| 8.5      | Layout editor .....                    | 44        |
| 8.6      | PL/SQL editor.....                     | 44        |
| 8.7      | Event triggers.....                    | 45        |
| 8.8      | Items .....                            | 46        |

|      |                               |    |
|------|-------------------------------|----|
| 8.9  | List of values (LOV) .....    | 47 |
| 8.10 | Menus .....                   | 47 |
| 8.11 | Libraries.....                | 48 |
| 8.12 | Multi-form applications ..... | 49 |
| 9    | <i>Reports</i> .....          | 50 |
| 9.1  | Modules.....                  | 50 |
| 9.2  | Data model.....               | 50 |
| 9.3  | Layout editor .....           | 50 |
| 9.4  | Parameter form.....           | 50 |
| 10   | <i>References</i> .....       | 50 |

# 1 Introduction to Oracle

## 1.1 Prerequisites

- Basic knowledge of Relational Database Management Systems.
- Familiarity with at least one programming language.

## 1.2 Objectives

- Database architecture
- SQL
- PL/SQL
- Oracle Forms & Reports
- Problem solving with Oracle Tools

## 1.3 Methods

- Text: *Oracle 9i A Beginner's Guide*
- Lecture
- Laboratory exercises
- Problem solving

## 1.4 Resources

- Course outline
- Oracle documentation (html & pdf)
- Oracle Technology Network: <http://technet.oracle.com>

## 2 Databases

### 2.1 Abstraction

A database model allows data to be organized in a way that is independent of the content or how the data is stored.

### 2.2 Tabular model

- Oldest, simplest information storage scheme.
- $n$  columns per record; one record per row.
- Examples: spreadsheet, phone book, shopping list.

### 2.3 Hierarchical model

- Managed via Indexed Sequential Access Method (ISAM).
- May contain nested data structures.
- Structure tied to problem domain.
- Examples: file system, this document.

### 2.4 Network model

- Allows data to point to other data.
- Quick insertion and fast retrieval.
- Supports chain of inference.
- Difficult to modify structure.

### 2.5 Relational model

Relational database includes:

- Entities (tables)
- Attributes (columns)
- Relationships (E-R diagram)

Metadata

- Information about the data (metadata) is stored in the database.
- Data dictionary holds views of metadata.
- DBA\_, ALL and USER\_ hierarchy of views for objects.
- DBA\_CATALOG: all tables in system.
- ALL\_CATALOG: all tables accessible to user.
- USER\_CATALOG: all tables owned by user.

Structured Query Language (SQL)

- Data accessible via ANSI standard language.
- Easy to modify both content and structure.

### Normalized

- Allows data to be stored in only one place.
- Allows consistent queries & updates

### Multiple views

- Allows data to be viewed in multiple ways.
- Simplifies access.
- Improves security.

### Client-server model

- Allows multiple users to share data.
- Permits centralized maintenance.
- Improves scalability.

## 2.6 Object-relational

- Upward compatible with existing relational model
- Objects are real-world entities
- Objects may be manipulated with via SQL

### Encapsulation

- Puts user defined data structures together with the methods (procedures and functions) that manipulate them.
- Nested data structures may include nested tables and variable sized arrays.
- Type system parallels PL/SQL package structure, which enforces a strong interface between programming constructs.
- Example: PL/SQL Guide, Ch 9.

### Inheritance

- New objects derived from old ones.
- New objects inherit the methods of the parent object.
- Must be simulated via type composition.
- Example: Application Developer's Guide, Ch 18.

### Polymorphism

- Capability of one object to respond differently to the same request.
- Example: SDO geometric objects.

## 2.7 Physical structure

### Parameter file

- Named init<sid>.ora.
- Defines control file locations.
- Defines memory allocation.

### Control files

- At least one control file required.
- Identifies the location of the database files.
- Stores the state of the database.
- Required for startup.

### Database files

- Contain the actual data and database objects.
- Size is determined at creation.
- May be enlarged, but difficult to shrink.
- Internal structure managed by Oracle.

### Redo log files

- At least two redo log files required.
- Stores all the transactions against the database.
- Allows recovery from system failure.
- ARCHIVE LOG mode: logs are retained.
- NOARCHIVE LOG mode: logs are recycled.

## 2.8 Logical structure

### Database

- Comprised of two or more tablespaces.
- May be open or closed.
- A server may have more than one database.

### Tablespaces

- Comprised of one or more schemas.
- System tablespace holds the data dictionary.
- Temporary tablespace used for sorting, etc.
- Tool tablespace used for commonly accessible programming interfaces.
- User tablespace holds user objects: data, indexes, views, etc.

### Schemas

- Owned by a single user.
- May include objects in multiple tablespaces.
- Segments
  - Data segment: specified by the storage clause when the table is created.
  - Index segment: stores index information, often in a tablespace that is separate from data segments.
  - Rollback segment: contains information that allows rollback of uncommitted transactions.
  - Temporary segment: used to hold intermediate results during sorting and query processing.

## Packages

- Collections of related data, procedures and functions.
- Used to hold server-side procedures and functions.

## 2.9 Oracle processes

### Client processes

- Initiates a connection to the server.
- Make database requests to the server.
- Examples: SQL\*Plus, Forms, java application.

### Server processes

- Accepts requests from the client.
- Returns results.
- Example: Oracle listener, WebDB listener.

### Database Support processes

- Database Writer (DBWR)
- Checkpoint (CKPT)
- Log Writer (LGWR)
- System Monitor (SMON)
- Process Monitor (PMON)
- Etc.

## 2.10 Memory structure

### System Global area (SGA)

- Shared by all Oracle processes.
- Size specified in parameter file.
- Four major components:
  1. Data buffer cache
    - Stores recently used data blocks.
    - Least Recently Used (LRU) algorithm.
    - Repeated requests for the same data can be fulfilled from memory (fast) rather than disk (slow).
  2. Dictionary cache
    - Stores data dictionary results.
    - Speeds query processing.

### 3. Redo log buffer

- Stores all transactions until written to disk.

### 4. Shared SQL pool

- Stores parsed SQL statements for reuse.
- Stores shared PL/SQL packages.
- Swapping vs. thrashing

### Program Global Area (PGA)

- Connection specific data allows multiple, distinct connections by the same user.
- Package global cache allows each connection to have its own instance of a shared package.

## 2.11 Distributed Database

Net8 (SQL\*Net) supports many transport layers:

- TCP/IP & SSL
- SPX/IPX
- Named pipes
- LU 6.2

Remote SQL via named database links

Parallel server architecture

Multi-threaded server

## 2.12 Database Objects

Tables

- Entities and their attributes (columns).
- Conceptually similar to a spreadsheet.
- Keyed by one or more attributes.
- Normal form.

Views

- Virtual- or named-queries.
- Used for convenience and security.
- Difficult to update.

Indexes

- Allow data to be searched and sorted quickly.
- Specify one or more attributes.

## Synonyms

- Alias for another object.
- May be global (PUBLIC) or local.

## Privilege

- Object privileges on tables, views, sequences etc.
- System privileges to create, alter and drop objects.

## Role

- Named collection of privileges.
- May be assigned to users or other roles.

## Snapshot

- A remote copy of a table that is periodically synchronized.
- Usually read-only.
- May be updated using symmetric replication.

## Package

- A collection of program objects and code.
- Includes related subprograms (procedures and functions).

## Database link

- A name used to access a table on a remote database.
- May be global or local.

## Sequences

- An object that returns the next number in a sequence.
- A safe way to create unique numbers in a multi-user environment.

## Clusters

- A means to group frequently accessed objects on disk.
- Used to improve access for often-used objects.

# **3 SQL\*Plus**

## **3.1 Overview**

- Used interactively for SQL & PL/SQL.
- Stores the most recent command in a buffer.
- Displays & formats query results for reports.
- Database administration tool.
- Command-line interpreter & shell.

## 3.2 Executive commands

/

- Executes the SQL command or PL/SQL block in the command buffer.
- Entered at the command prompt (>).

@

- @ file\_name[.ext] [arg...]
- Invokes the script in the named file.
- Default extension is “.sql”.

@@

- @@ file\_name[.ext]
- Similar to @, but uses path of caller.
- Used string command files together.

Start

- STA[RT] file\_name[.ext] [arg ...]
- Loads command file into buffer.
- Executes the buffered commands.

Run

- List the buffer contents
- Executes the buffered commands.
- Similar to List followed by /

Execute

- EXEC[UTE] statement
- Executes a single PL/SQL statement.

## 3.3 Editing commands

Edit

- ED[IT] [file\_name[.ext]]
- Invokes the host operating system’s default editor.
- Loads the buffer contents into the editor when no name is specified.

Get

- GET file\_name[.ext] [LIS[T]|NOL[IST]]
- Loads a file into the buffer.

Save

SAV[E] file\_name[.ext] [CRE[ATE]|REP[LACE]|APP[END]]  
Saves the buffer in a file.

## 3.4 Buffer commands

### List

- L[IST] [n|n m|n \*|n LAST|\*|\* n|\* LAST|LAST]
- List all lines.
- List line *n*.
- List lines *n* through *m*.
- List lines *n* through current.
- List lines *n* through last line.
- List current line through line *n*.
- List current line through last line.
- List last line.

### Input

- I[NPUT] [text]
- Adds one or more new lines of text after the current line in the buffer.
- Without text, repeatedly adds lines.

### Delete

- DEL [n|n m|n \*|n LAST|\*|\* n|\* LAST|LAST]
- Options similar to List command.

### Append

- A[PPEND] text
- Adds specified text to the end of the current line in the SQL buffer.
- Use two spaces between command and text.

### Change

- C[HANGE] sepchar old [sepchar [new [sepchar]]]
- Changes the first occurrence of text on the current line in the buffer.
- Sepchar is any non-alphanumeric character such as “/” or “.”.
- Old is the string to search for.
- New replaces old.

### Clear

- CL[EAR] option
- Options include
- BRE[AKS]
- BUFF[ER]
- COL[UMNS]
- COMP[UTES]
- SCR[EEN]
- SQL
- TIMI[NG]

## 3.5 Formatting commands

### Column

- COL[UMN] [{column|expr} [option ...]]
- Specifies display attributes for a given column.
- Options include:
- ALI[AS] alias
- FOR[MAT] format
- HEA[DING] text
- JUS[TIFY] {L[EFT] | C[ENTER] | C[ENTRE] | R[IGHT]}
- LIKE {expr | alias}
- WRA[PPED] | WOR[D\_WRAPPED] | TRU[NCATED]

### Title & Btitle

- TTI[TLE] [printspec [text|variable] ...] | [OFF|ON]
- Formats a title at the top or bottom of each page.
- Printspec includes:
- COL n
- S[KIP] [n]
- TAB n
- LE[FT]
- CE[NTER]
- R[IGHT]
- BOLD
- FORMAT text

### Break

- BRE[AK] [ON report\_element [action [action]]]
- Specifies where and how formatting will change in a report, such as
- Suppressing display of duplicate values for a given column.
- Skipping a line each time a given column value changes.
- Printing COMPUTED figures each time a given column value changes or at the end of the report.
- Break on {column | expr | ROW | REPORT}

### Compute

- COMP[UTE] [function [LAB[EL] text] OF {column} ON {column}]
- Calculates summary functions including:
- Average
- Count
- Maximum
- Minimum
- Standard deviation
- Sum
- Variance

## 3.6 Interactive commands

### Connect

- CONNECT [username/password@tnsname](#)
- Prompts if password omitted.

### Disconnect

- Drops the connection to the database.
- Used for security.

### Prompt

- PROMPT [text]
- Sends the specified text to the user's screen.
- PROMPT 'Preparing report...'

### Accept

- ACC[EPT] variable [PROMPT text] [HIDE]
- Reads a line of input and stores it in a given user variable.
- ACCEPT pswd CHAR PROMPT 'Password: ' HIDE

### Set

- SET system\_variable value
- Sets a system variable to alter the SQL\*Plus environment for your current session.
- Variables include:
- ARRAY[SIZE] {15 | n}
- AUTO[COMMIT] {OFF | ON | IMM[EDIATE] | n}
- AUTOT[RACE] {OFF | ON | TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]
- ECHO {OFF | ON}
- EDITF[ILE] file\_name[.ext]
- FEED[BACK] {6 | n | OFF | ON}
- HEA[DING] {OFF | ON}
- LIN[ESIZE] {80 | n}
- LONG {80 | n}
- NEWP[AGE] {1 | n | NONE}
- NUMF[ORMAT] format
- NUM[WIDTH] {10 | n}
- PAGES[IZE] {24 | n}
- PAU[SE] {OFF | ON | text}
- SERVEROUT[PUT] {OFF | ON} [SIZE n] [FOR[MAT] {WRA[PPED] | WOR[D\_WRAPPED] | TRU[NCATED]}]
- SQLP[ROMPT] {SQL> | text}
- TERM[OUT] {OFF | ON}
- TI[ME] {OFF | ON}
- TIMI[NG] {OFF | ON}

- UND[ERLINE] {- |c| ON | OFF}
- WRA[P] {OFF | ON}

### Show

- SHO[W] option
- Shows environment settings
- Additional options include
- ALL
- BTI[TLE]
- ERR[ORS] [{FUNCTION | PROCEDURE | PACKAGE | PACKAGE BODY | TRIGGER | VIEW | TYPE | TYPE BODY} [schema.]name]
- PARAMETERS [parameter\_name]
- REL[EASE]
- REPF[OOTER]
- REPH[EADER]
- SGA
- SPOO[L]
- SQLCODE
- TTI[TLE]
- USER

### Spool

- SPO[OL] [file\_name.ext] | OFF | OUT]
- Use spool out to send output to printer.
- To create a flat file, use the following

```
SET NEWPAGE 0
SET SPACE 0
SET LINESIZE 80
SET PAGESIZE 0
SET ECHO OFF
SET FEEDBACK OFF
SET HEADING OFF
```

### Describe

- DESC[RIBE] {[schema.]object[@net\_service\_name]}
- Lists the column definitions for the specified table, view, or synonym or the specifications for the specified function or procedure.

### Timing

- TIMI[NG] [START text | SHOW | STOP]
- Records timing data for an elapsed period of time, lists the current timer's name and timing data, or lists the number of active timers.

## 3.7 Variables

### Substitution variables

- DEFINE name = value.
- Use &name to refer to value.
- Use \& to escape a literal ampersand.
- Prompt for undefined variables.
- Use &1, &2 etc. for parameters from the OS command line: sqlplus scott/tiger @script variable.

### Bind variables

- VARIABLE name type
- Use :name to refer to variable in PL/SQL.
- Use PRINT name to examine value.

## 3.8 Example

```
-- SQL*Plus User's Guide, Example 4-25
SPOOL TEMP
CLEAR COLUMNS
CLEAR BREAKS
CLEAR COMPUTES

COLUMN DEPTNO HEADING DEPARTMENT
COLUMN ENAME HEADING EMPLOYEE
COLUMN SAL HEADING SALARY FORMAT $99,999

BREAK ON DEPTNO SKIP 1 ON REPORT
COMPUTE SUM OF SAL ON DEPTNO
COMPUTE SUM OF SAL ON REPORT

SET PAGESIZE 21
SET NEWPAGE 0
SET LINESIZE 30

TTITLE CENTER 'A C M E W I D G E T' SKIP 2 -
LEFT 'EMPLOYEE REPORT' RIGHT 'PAGE:' -
FORMAT 999 SQL.PNO SKIP 2

BTITLE CENTER 'COMPANY CONFIDENTIAL'

SELECT DEPTNO, ENAME, SAL
FROM EMP
ORDER BY DEPTNO;

SPOOL OFF
```

A C M E W I D G E T

EMPLOYEE REPORT PAGE: 1

| DEPARTMENT | EMPLOYEE | SALARY   |
|------------|----------|----------|
| 10         | CLARK    | \$2,450  |
|            | KING     | \$5,000  |
|            | MILLER   | \$1,300  |
| *****      |          | -----    |
| sum        |          | \$8,750  |
| 20         | SMITH    | \$800    |
|            | ADAMS    | \$1,100  |
|            | FORD     | \$3,000  |
|            | SCOTT    | \$3,000  |
|            | JONES    | \$2,975  |
| *****      |          | -----    |
| sum        |          | \$10,875 |

COMPANY CONFIDENTIAL

A C M E W I D G E T

EMPLOYEE REPORT PAGE: 2

| DEPARTMENT | EMPLOYEE | SALARY   |
|------------|----------|----------|
| 30         | ALLEN    | \$1,600  |
|            | BLAKE    | \$2,850  |
|            | MARTIN   | \$1,250  |
|            | JAMES    | \$900    |
|            | TURNER   | \$1,500  |
|            | WARD     | \$1,250  |
| *****      |          | -----    |
| sum        |          | \$9,400  |
| *****      |          | -----    |
| sum        |          | \$29,025 |

COMPANY CONFIDENTIAL

## 4 SQL

### 4.1 Introduction

Based on the work of Dr. E.F. Codd

Originally implemented at IBM

ANSI X3.135-1992, "Database Language SQL"

Declarative vs. procedural language

- Declarative language specifies what result is desired, e.g. ordering a meal from a menu.
- Procedural language specifies how the result is to be obtained, e.g. following a recipe.

Not case sensitive, except quoted text

### 4.2 Statements

DML

- Select

```
SELECT deptno, ename FROM emp;
```

```
SELECT distinct deptno FROM emp;
```

```
SELECT deptno, ename FROM emp  
ORDER BY ename;
```

```
SELECT deptno, ename FROM emp  
WHERE deptno = 10  
ORDER BY ename;
```

- Insert

```
INSERT INTO emp (empno, ename, job, sal, comm, deptno)  
VALUES (7890, 'YOUNG', 'CLERK', 1200, NULL, 40);
```

- Update

```
UPDATE emp SET comm = 200  
WHERE ename = 'TURNER';
```

- Delete

```
DELETE FROM emp WHERE ename = 'YOUNG';
```

- Truncate

```
TRUNCATE TABLE bonus;
```

## DDL

- Alter

```
ALTER TABLE dept MODIFY loc VARCHAR2(14);
```

- Create

```
CREATE TABLE dept
  (deptno NUMBER(2)    primary key using index,
   dname   VARCHAR2(12),
   loc     VARCHAR2(12));
```

```
CREATE TABLE temp as SELECT * FROM emp;
```

```
CREATE USER temp IDENTIFIED BY temp
  DEFAULT TABLESPACE users
  TEMPORARY TABLESPACE temp
  QUOTA UNLIMITED on USERS;
```

- Drop

```
DROP TABLE temp;
```

- Grant

```
GRANT SELECT ON emp TO scott;
GRANT connect TO temp;
```

- Revoke

```
REVOKE SELECT ON emp FROM scott;
```

- Rename

```
RENAME temp TO fred;
```

## Transaction control

- Commit: Make changes permanent.
- Rollback: Undo uncommitted work.

- Savepoint: Identify a point in a transaction to which you can later roll back.
- Set transaction: Identify a transaction as read/write.

### 4.3 Basic elements

#### Literals

- Quoted text: 'This is text.'
- Integer: up to 38 digits.
- Number: scientific notation with 38 digit mantissa and exponent -130 to +125.

#### Built-in data types

- CHAR(n): Fixed length character data.
- VARCHAR2(n): Variable length string up to *n* characters.
- NUMBER(p, s): Number with precision *p* and scale *s*.
- DATE: Calendar date and time with one second precision.
- LOB: Large objects; hold up to four GB.
- BLOB: Binary large object.
- BFILE: A BLOB stored in the local file system.
- CLOB: Character large object.
- ROWID: Internal storage reference.

#### User data types

- Object: includes name attributes and methods.
- Ref: a pointer to an object.
- Varray: variable sized array of objects.
- Nested table: a table within a table.

#### Format Models

- Number formats

```
SQL> SELECT TO_CHAR(12345.67, '$999,999.99')
       2 "Amount" FROM dual;
```

```
Amount
-----
$12,345.67
```

- Date formats

```
SELECT
  TO_CHAR(sysdate, 'DD-Mon-YYYY HH24:MI:SS')
  "Today" FROM dual;
```

```
Today
```

-----  
18-Jan-2001 22:38:04

- Explicit conversion vs. implicit conversion.

#### Nulls

- The value of a column in which nothing is stored.
- Not the same as an empty string or 0 for numbers.
- Use IS NULL or IS NOT NULL for comparisons.

#### Pseudo-columns

- Nextval: Next value in a sequence.
- Currval: Current value in a sequence.
- RowID: A row's address in the database.
- Rownum: The number of the current row.

#### Comments

- /\* comment \*/
- -- comment
- hints in DML
- Comment command

#### Database Objects

- Schema: a logical collection of database objects.
- Owned by a single user.

#### Schema Object Names and Qualifiers

- Name syntax: schema.object[@database\_link]
- One to 30 characters in length.
- Start with a letter.
- May contain underscore.
- \$ and # permitted but discouraged.

## 4.4 Operators

#### Unary and Binary Operators

- Unary operator operates on only one operand.
- Binary operator operates on two operands.

#### Precedence

| <b>Operator</b>               | <b>Operation</b>                     |
|-------------------------------|--------------------------------------|
| +, -                          | identity, negation                   |
| *, /                          | multiplication, division             |
| +, -,                         | addition, subtraction, concatenation |
| =, !=, <, >, <=, >=, IS NULL, | comparison                           |

|                   |                  |
|-------------------|------------------|
| LIKE, BETWEEN, IN |                  |
| NOT               | logical negation |
| AND               | conjunction      |
| OR                | Disjunction      |

### Arithmetic Operators

- +, -, \*, /
- Results are numeric.
- + and - also work on dates.
- `SELECT 3 * (2 + 1) FROM dual;`
- `SELECT TO_CHAR(to_date('1-Jun-2000') - 1/(60*60*24), 'DD-Mon-YYYY HH24:MI') FROM dual;`

### Concatenation Operator

- ||
- Concatenates character strings.
- Result is character.
- `SELECT 'Name: ' || ename FROM emp;`

### Comparison Operators

- =, !=, <, >, <=, >=,
- IS NULL, IS NOT NULL  
`SELECT ename FROM emp WHERE comm IS NULL`
- IN, NOT IN: a list.  
`SELECT ename FROM emp WHERE job IN ('CLERK', 'MANAGER')`
- LIKE: pattern match with '%'.  
`SELECT ename FROM emp WHERE ename LIKE ('A%');`
- BETWEEN: range test.  
`SELECT ename FROM emp WHERE sal BETWEEN 1000 AND 3000;`
- Compares two values and returns true, false or null.

### Logical Operators

- NOT: Logical opposite.
- AND: True if both are true.
- OR: True if either is true.
- If any values are NULL, consult table 3-6, 3-7 or 3-8.
- `SELECT ename FROM emp WHERE job = 'CLERK' AND sal > 1000;`

### Set Operators

- UNION: All rows selected by either query.
- UNION ALL: All rows selected by either query, including all duplicates.
- INTERSECT: All distinct rows selected by both queries.
- MINUS: All distinct rows selected by the first query but not the second.

## 4.5 Functions

### Number functions

- Arithmetic functions like ABS, ROUND, TRUNC.
- Algebraic functions like SQRT, MOD, POWER and SIGN.
- Transcendental functions like SIN, COS and TAN.

### Character functions

- Capitalization functions like UPPER, LOWER and INITCAP.
- String function like LENGTH, INSTR, SUBSTR and CONCAT.
- Fill functions like TRIM, LTRIM, RTRIM, LPAD and RPAD.
- Phonetic search using SOUNDIX.

### Date functions

- The current date and time, SYSDATE.
- Approximate dates using ROUND and TRUNC.
- Month functions ADD\_MONTHS, MONTHS\_BETWEEN, LAST\_DAY.
- Weekday calculations with NEXT\_DAY.
- Time zone conversion via NEW\_TIME.

### Conversion functions

- TO\_CHAR, TO\_DATE, TO\_NUMBER

### Miscellaneous

- GREATEST, LEAST from a list.
- USER: returns the user ID.
- NVL(exp1, exp2): if exp1 is null, return exp2.

### Aggregate functions

- Return a single row based on groups of rows.
- All except COUNT ignore nulls.
- AVG: SELECT AVG (sal) FROM emp;
- COUNT: SELECT COUNT (\*) FROM emp;
- MAX: SELECT MAX (sal) FROM emp;
- MIN: SELECT MIN (hiredate) FROM emp;
- SUM: SELECT SUM (sal) FROM emp;
- VARIANCE, STDEV

## 4.6 Expressions, conditions & queries

### Expressions

- An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value.
- Simple expression, e.g.  $3 * (2 + 1)$

- Compound expression, e.g. TRUNC(SYSDATE) + 1/24
- DECODE(exp, search, result, [search, result,] default)

```
DECODE (TRUNC(grade/10), 9, 'A',
                        8, 'B',
                        7, 'C',
                        6, 'D',
                        'F')
```

### Conditions

- Part of the WHERE clause in SELECT, UPDATE and DELETE.
- Use comparison operators.

### Queries

- Simple queries use column names.
- If two columns from different tables have the same name, use an alias.
- Hierarchical queries use START WITH and CONNECT BY.
- Use ORDER BY [DESC] clause to sort results.
- Sub-query.

## 4.7 Constraints

### Not null

- Requires that a value be supplied on insert.
- Applies to single columns only.

```
ALTER TABLE dept MODIFY deptno
    CONSTRAINT ck_deptno NOT NULL;
```

### Unique

- Requires that no two rows have the same value.
- May comprise multiple columns.

### Primary key

- Only one primary key permitted.
- Must also be unique and not null.
- May comprise multiple columns.

```
ALTER TABLE dept ADD CONSTRAINT pk_dept
    PRIMARY KEY (deptno) USING INDEX;
```

### Foreign key

- Requires values to come from references table.
- May refer to multiple columns.
- REFERENCES clause.

- ON DELETE CASCADE clause.

```
ALTER TABLE emp ADD CONSTRAINT fk_emp
  FOREIGN KEY (deptno) REFERENCES dept(deptno)
  ON DELETE CASCADE;
```

#### Check

- Enforces a condition during insert and update operations.
- CHECK (ename = UPPER(ename))

## 4.8 Examples

### Equi-join

```
SELECT dept.dname, emp.ename FROM dept, emp
WHERE dept.deptno = emp.deptno
ORDER BY dname, ename
```

### Outer join

```
SELECT dept.dname, emp.ename FROM dept, emp
WHERE dept.deptno = emp.deptno(+)
ORDER BY dname, ename
```

### Self-join

```
SELECT e1.ename||' works for '||e2.ename
  "Employees and their Managers"
FROM emp e1, emp e2
WHERE e1.mgr = e2.empno;
```

### Correlated query (sub-select)

```
SELECT deptno, ename, sal
FROM emp x
WHERE sal > (SELECT AVG(sal)
  FROM emp
  WHERE x.deptno = deptno)
ORDER BY deptno;
```

### Group by

```
SELECT deptno, MIN(sal), MAX (sal)
FROM emp
GROUP BY deptno
HAVING MIN(sal) < 1000;
```

## 4.9 Views

### Concept

- A view is a virtual table derived from another table or from two or more joined tables.
- Only the query that creates the view is stored.

### Advantages

- Hides complex joins or frequently used conditions.
- Simplifies some joins.
- Improves performance by allowing the view to be optimized.
- Provides security by hiding sensitive columns

### Disadvantages

- View of joined tables not updateable.
- View of aggregate columns not updateable.
- View with expressions not updateable.
- After adding columns, views may need to be recompiled.

### Examples

```
CREATE VIEW manager_view AS
  SELECT a.ename, b.ename manager
  FROM emp a, emp b
  WHERE a.mgr = b.empno;

SELECT * FROM manager_view ORDER BY ename;

ALTER VIEW manager_view COMPILE;

SELECT text FROM user_views
  WHERE view_name = 'MANAGER_VIEW';

DROP VIEW manager_view;
```

## 5 PL/SQL

### 5.1 Introduction

- Oracle's procedural language for SQL.
- Derived from the Algol-PL/1-Pascal-Ada family of languages.

- Syntax and structure based on Ada.
- Supports principles of software engineering.
- Facilitates large program development.
- Suitable for either server- or client-side applications.
- Recursive block structure:

```

DECLARE --anonymous block, procedure or function
-- declaration section for
-- constant, variable, cursor, exception
-- type, subtype, procedure or function
BEGIN
-- program statements
-- optional nested anonymous block
EXCEPTION
-- optional exception handling
END;
```

## 5.2 Data types

### Character strings

- `VARCHAR2(n)`: Variable length up to  $n$  characters.
- Length up to 32767

### Numeric

- `NUMBER(p, s)`: Number with precision  $p$  and scale  $s$ .
- Subtypes include `DECIMAL`, `FLOAT` and `REAL`.
- Use `INTEGER` to declare integers up to 38 digits
- `BINARY_INTEGER`:  $-2^{32}$  to  $+2^{32} - 1$
- Subtypes include `NATURAL` and `POSITIVE`.
- Uses less storage than `NUMBER`.

### Date

- `DATE`: Calendar date and time with one second precision.

### Boolean

- `BOOLEAN`: stores true false or null.
- Result type of logical and comparison expressions.

### Attributes

- Use `%TYPE` attribute to refer to database column types.
- Use `%ROWTYPE` to refer to database row types.

### Datatype conversion

- Explicit conversions use conversion functions.
- Implicit conversions happen on selection or assignment.
- See PL/SQL User's Guide, Table 2-1.

## 5.3 Program structure

### Declarations

- Constants
- Variables
- Cursors
- Exceptions
- User-defined types

```
pi CONSTANT REAL := 3.14159;
employee_name VARCHAR2(15);
employee_name emp.ename%TYPE;
employee_record emp%ROWTYPE;
birthday DATE;
emp_count NUMBER := 0;
found BOOLEAN := FALSE;
range_error EXCEPTION;
CURSOR c is SELECT * FROM emp;
TYPE Vendors IS TABLE OF VARCHAR2(10);
SUBTYPE Counter IS NATURAL;
```

### Operators

- Assignment uses := operator.
- Arithmetic and comparison operators as in SQL with the addition of exponentiation.
- Comparisons involving nulls always yield NULL.
- Applying the logical operator NOT to a null yields NULL.
- In conditional control statements, if the condition yields NULL, its associated sequence of statements is not executed.

### Logical control structures

```
IF condition THEN
    sequence_of_statements
END IF;
```

```
IF condition THEN
    sequence_of_statements1
ELSE
    sequence_of_statements2
END IF;
```

```
IF condition1 THEN
    sequence_of_statements1
ELSIF condition2 THEN
    sequence_of_statements2
ELSE
```

```

    sequence_of_statements3
END IF;

DECLARE -- correct
    overdrawn BOOLEAN;
BEGIN
    IF new_balance < minimum_balance THEN
        overdrawn := TRUE;
    ELSE
        overdrawn := FALSE;
    END IF;
    ...
    IF overdrawn = TRUE THEN
        RAISE insufficient_funds;
    END IF;
END;
/

```

```

DECLARE -- better
    overdrawn BOOLEAN;
BEGIN
    overdrawn := new_balance < minimum_balance
    IF overdrawn THEN
        RAISE insufficient_funds;
    END IF;
END;
/

```

### Iterative control structures

```

LOOP
    sequence_of_statements
END LOOP;

```

```

LOOP
    ...
    EXIT WHEN condition
    ...
END LOOP;

```

```

WHILE condition LOOP
    sequence_of_statements
END LOOP;

```

```

FOR counter IN [REVERSE] lower_bound..higher_bound LOOP
    sequence_of_statements
END LOOP;

```

## Null statement

- NULL indicates no operation.

```
IF count > 10 THEN
    NULL;
ELSE
    sequence_of_statements
END IF;
```

## Exceptions

- An exception is an abnormal condition that stops execution.
- Handling exceptions allows execution to continue.
- See predefined exceptions in User's Guide.
- User defined exception may be declared.
- Exceptions propagate from inner blocks to outer blocks.
- Use a nested block to retry exceptions.
- Handle the OTHERS exception to preclude propagation.
- When you can't recover, at least display the error.

```
EXCEPTION
    WHEN exception_name1 THEN -- handler
        sequence_of_statements1
    WHEN exception_name2 THEN -- another handler
        sequence_of_statements2
    ...
    WHEN OTHERS THEN          -- optional handler
        sequence_of_statements3
END;
```

```
DECLARE
    s VARCHAR2(5);
BEGIN
    s := 'Testing';
EXCEPTION
    WHEN OTHERS THEN
        DBMS_Output.Put_Line(SQLERRM);
END;
/
```

## 5.4 Cursors

### Introduction

- A cursor is mechanism for examining query results one row at a time.
- Implicit cursor created for all DML statements.
- Explicit cursor required for all queries returning more than one row.

### Cursor attributes

- See PL/SQL User's Guide, Table 5-1
- %FOUND
- %NOTFOUND
- %ISOPEN
- %ROWCOUNT

### Implicit cursors

- Used for SELECT statements that return a single row.
- Used for INSERT, UPDATE and DELETE.
- Automatically closed after use.
- Use SQL% attributes to determine results.

```
DECLARE
  e emp%ROWTYPE;
BEGIN
  -- find the president
  SELECT * INTO e FROM emp
    WHERE emp.job = 'PRESIDENT';

  -- earmark for a bonus
  INSERT INTO bonus VALUES(
    e.ename, e.job, e.sal, e.comm);

  IF SQL%FOUND THEN
    DBMS_Output.Put_Line(
      'Insert into bonus table succeeded.');
```

```
END IF;

EXCEPTION
  WHEN OTHERS THEN
    DBMS_Output.Put_Line(SQLERRM);
END;
/
```

### Explicit cursors

- Used for SELECT statements that may return multiple rows.
- Declared explicitly in declarative part of PL/SQL block.
- Use OPEN, FETCH and CLOSE to manipulate cursor.

```
DECLARE
  CURSOR c IS
    SELECT * FROM dept ORDER BY deptno;
  d dept%ROWTYPE;
  s VARCHAR2(80);
BEGIN
  OPEN c; -- open the cursor
```

```

LOOP
    FETCH c INTO d; -- fetch each row
    EXIT WHEN c%NOTFOUND;
    s := d.deptno || ' ' || d.dname || ', ' || d.loc;
    DBMS_Output.Put_Line(s);
END LOOP;
CLOSE c;

EXCEPTION
    WHEN OTHERS THEN
        IF c%ISOPEN THEN CLOSE c; END IF;
        DBMS_Output.Put_Line(SQLERRM);
END;
/

```

Cursors may have parameters.

```

DECLARE
    CURSOR c (department NUMBER) IS
        SELECT * FROM emp
        WHERE deptno = department
        ORDER BY ename;
    e emp%ROWTYPE;
    s VARCHAR2(80);
BEGIN
    OPEN c(30); -- open the cursor for department 30
    LOOP
        FETCH c INTO e; -- fetch each row
        EXIT WHEN c%NOTFOUND;
        s := e.empno || ' ' || e.ename || ', ' || e.job;
        DBMS_Output.Put_Line(s);
    END LOOP;
    CLOSE c;

EXCEPTION
    WHEN OTHERS THEN
        IF c%ISOPEN THEN CLOSE c; END IF;
        DBMS_Output.Put_Line(SQLERRM);
END;
/

```

Cursor for-loops

- Implicitly declares a variable of %ROWTYPE
- Automatically opens the cursor.
- Fetches one row after another.
- Automatically closes the cursor.

```

DECLARE

```

```

CURSOR c (department NUMBER) IS
  SELECT * FROM emp
  WHERE deptno = department
  ORDER BY ename;
s VARCHAR2(80);
BEGIN
  FOR e IN c(30) LOOP
    s := e.empno || ' ' || e.ename || ', ' || e.job;
    DBMS_Output.Put_Line(s);
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    IF c%ISOPEN THEN CLOSE c; END IF;
    DBMS_Output.Put_Line(SQLERRM);
END;
/

```

## 5.5 Collections

- A collection is an ordered group of elements, all of the same type.
- Each element has a unique subscript (index).
- Similar to one-dimensional arrays.
- DML and assignment (:=) permitted.
- May be passed as parameters.
- Methods: count, delete, exists, extend/trim, first/last, next/prior.

### Varrays

- Have maximum sizes (limit).
- Always have consecutive indexes (dense).
- Data stored in-line (in the same tablespace).
- Retain their ordering and subscripts when stored.
- Suitable for small collections.

### Nested tables

- Can grow and shrink arbitrarily (extend/trim).
- May have gaps between indexes (next/prior).
- Data stored in separate tables.
- Better for large or sparse collections.

### Index-by tables

- Similar to nested tables, but not stored in database.
- Somewhat more flexible inside PL/SQL.
- Useful for storing data for later reference, without having to query the database again to get it.

```

DECLARE
  TYPE DepartmentType IS TABLE OF dept%ROWTYPE

```

```

        INDEX BY BINARY_INTEGER;
    depTable DepartmentType;
    i POSITIVE;
    CURSOR c1 is
        SELECT * FROM dept ORDER BY deptno;
BEGIN
    --load the local table for future reference
    FOR drec IN c1 LOOP
        depTable(drec.deptno) := drec;
    END LOOP;

    --loop through the table
    FOR i IN depTable.FIRST .. depTable.LAST LOOP
        IF depTable.exists(i) THEN
            DBMS_Output.Put_Line(depTable(i).deptno ||
                ' ' || depTable(i).dname);
        END IF;
    END LOOP;

    --another way to loop through the table
    i := depTable.FIRST;
    WHILE i <= depTable.LAST LOOP
        DBMS_Output.Put_Line(depTable(i).deptno ||
            ' ' || depTable(i).dname);
        i := depTable.NEXT(i);
    END LOOP;
END;
/

```

## 5.6 Subprograms

- Named PL/SQL blocks that accept parameters.
- Provide extensibility.
- Encourage modularity.
- Promote reusability.
- Actual parameters arranged by position or by name.
- Names may be overloaded if signatures differ.

### Procedures

- Generally used to perform an action.
- Parameter types: IN, OUT, IN OUT.
- Optional NOCOPY for OUT and IN OUT parameters.

```

CREATE [OR REPLACE] PROCEDURE name [(parameter_list)] IS
    [local declarations]
BEGIN
    executable statements
[EXCEPTION

```

```
    exception handlers]
END [name];
```

### Function

- Generally used to calculate a value.
- Function exits after RETURN statement in body.

```
CREATE [OR REPLACE] FUNCTION name [(parameter_list)] RETURN
datatype IS
    [local declarations]
BEGIN
    executable statements, with at least one RETURN
[EXCEPTION
    exception handlers]
END [name];
```

### Packages

- Named collections of declarations, procedures and functions.
- Package specification includes all variable, cursor, exception, procedure and function declarations visible to callers.
- Package body implements the objects specified in the package specification.
- Package body may have other object not visible to the outside, such as static variables.
- Package body may change without requiring the specification to be re-compiled.
- More efficient on server.

```
-----
-- Module:      Conv
--
-- Copyright: 2000, GCS
--
-- Author:      Dr. John B. Matthews
--
-- Purpose:     Conversion for base 2 - 16 numbers
--
-- Revision history:
--    09/25/2000 JBM Created
-----
```

```
CREATE OR REPLACE PACKAGE Conv IS
-- Convert decimal numbers and strings in base 2 through 16.
-- Warning: Only bases 2 through 16 supported.

    FUNCTION To_Base(dec IN NUMBER, base IN NUMBER)
        RETURN VARCHAR2;
    FUNCTION To_Dec (str IN VARCHAR2, base IN NUMBER := 16)
```

```

    RETURN NUMBER;
FUNCTION To_Hex (dec IN NUMBER) RETURN VARCHAR2;
FUNCTION To_Oct (dec IN NUMBER) RETURN VARCHAR2;
FUNCTION To_Bin (dec IN NUMBER) RETURN VARCHAR2;
PROCEDURE Set_Case(upper IN BOOLEAN := TRUE);

END Conv;
/

CREATE OR REPLACE PACKAGE BODY Conv IS

hexu VARCHAR2(16) := '0123456789ABCDEF';
hexl VARCHAR2(16) := '0123456789abcdef';
hex  VARCHAR2(16) := hexu; -- default to upper case

FUNCTION To_Base (dec IN NUMBER, base IN NUMBER) RETURN
VARCHAR2 IS
    str VARCHAR2(255) := '';
    num NUMBER := ABS(TRUNC(dec));
BEGIN
    LOOP
        str := SUBSTR(hex, MOD(num, base) + 1, 1 ) || str;
        num := TRUNC( num/base );
        EXIT WHEN num = 0;
    END LOOP;
    RETURN str;
END To_Base;

FUNCTION To_Dec (str IN VARCHAR2, base IN NUMBER := 16) RETURN
NUMBER IS
    num NUMBER := 0;
BEGIN
    FOR i IN 1 .. LENGTH(str) LOOP
        num := num * base +
            INSTR(hexu, UPPER(SUBSTR(str, i, 1))) - 1;
    END LOOP;
    RETURN num;
END To_Dec;

FUNCTION To_Hex (dec IN NUMBER) RETURN VARCHAR2 IS
BEGIN
    RETURN To_Base(dec, 16);
END To_Hex;

FUNCTION To_Oct (dec IN NUMBER) RETURN VARCHAR2 IS
BEGIN
    RETURN To_Base(dec, 8);
END To_Oct;

```

```

FUNCTION To_Bin (dec IN NUMBER) RETURN VARCHAR2 IS
BEGIN
    RETURN To_Base(dec, 2);
END To_Bin;

PROCEDURE Set_Case(upper IN BOOLEAN := TRUE) IS
BEGIN
    IF upper THEN
        hex := hexu;
    ELSE
        hex := hexl;
    END IF;
END;

END Conv;
/
show errors;

execute DBMS_Output.Put_Line(Conv.To_Dec('FFFF'));
execute DBMS_Output.Put_Line(Conv.To_Dec(str => 'FFFF'));
execute DBMS_Output.Put_Line(Conv.To_Hex(65535));
execute DBMS_Output.Put_Line(Conv.To_Oct(8 * (8 * (8 + 1) + 1)
+ 1));
execute DBMS_Output.Put_Line(Conv.To_Bin(255));
execute DBMS_Output.Put_Line(Conv.To_Bin(Conv.To_Dec('755',
8)));

```

## 5.7 Database Triggers

### DML triggers

- PL/SQL procedures that execute in response to some operation.
- Trigger fires on INSERT, UPDATE and/or DELETE.
- On INSERT, new value is available, old is NULL.
- On UPDATE, both old and new values are available.
- On DELETE, old value is meaningful, new is NULL.
- BEFORE trigger allows new value to be updated.
- AFTER trigger sees changes from any BEFORE trigger.
- FOR EACH ROW option fires once for each row affected.
- INSERTING, UPDATING and DELETING predicates specify the operation that invoked the trigger.

```

CREATE OR REPLACE TRIGGER Print_Salary_Changes
BEFORE DELETE OR INSERT OR UPDATE OF sal ON emp
FOR EACH ROW
DECLARE
    old_sal NUMBER := NVL(:old.sal, 0);

```

```

    new_sal NUMBER := NVL(:new.sal, 0);
    change  NUMBER := new_sal - old_sal;
BEGIN
    DBMS_Output.Put ('Old salary: ' || old_sal || ', ');
    DBMS_Output.Put ('New salary: ' || new_sal || ', ');
    DBMS_Output.Put_Line('Change: ' || change);
END;
/
UPDATE emp SET sal = sal + 100.00 WHERE deptno = 10;

```

### System triggers

- DDL statements: CREATE, ALTER or DROP statements.
- System events: STARUP, SHUTDOWN or SEVERERROR.
- Client events: LOGON, LOGOFF.
- ON SCHEMA fires only for creating user's events.
- ON DATABASE fires for any user's events.

### Instead-of triggers

- Used on views to replace a pending DML operation.
- Allows updating the underlying tables.
- FOR EACH ROW required.

### Trigger applications

- Provide sophisticated auditing.
- Prevent invalid transactions.
- Enforce referential integrity on remote database.
- Enforce complex business rules.
- Enforce complex security authorizations.
- Provide transparent event logging.
- Automatically generate derived column values.
- Enable building complex views that can be updated.
- Track system events and set contexts.

## 5.8 Errors

- Use show errors command to examine user\_errors table.
- Use list command to see referenced line numbers.
- Reference errors in Oracle Error Messages manual.

## 6 HTML

### 6.1 Oracle Application Server (OAS)

- Oracle's HTTP server is based on Apache.
- Able to dispatch dynamic web pages from many sources:
  - PL/SQL
  - Oracle Forms and Reports

- Java, JSP, EJB
- Perl
- PL/SQL gateway interprets URL's as PL/SQL procedure calls:
- <protocol>://<host>:<port>/<dad>/<package>.<procedure>
- HTTP GET parameters passed as PL/SQL parameters.
- Data access descriptor specifies authentication.

```
-----
-
--
-- Module:      cremp.sql
--
-- Copyright:  2001, GCS
--
-- Author:      Dr. John B. Matthews
--
-- Purpose:     Create PL/SQL Package for employee maintenance
--
-- Revision history:
--   01/30/2001 JBM Created
--   02/02/2001 JBM Revise Banner
-----
-
```

```
CREATE OR REPLACE PACKAGE Employee IS
  PROCEDURE Home;
  PROCEDURE Enter (id in NUMBER);
  PROCEDURE Updater (id NUMBER, jobd VARCHAR2,
    hired VARCHAR2, salary NUMBER);
  PROCEDURE Select_Emp;
END Employee;
/
```

```
CREATE OR REPLACE PACKAGE BODY Employee IS
```

```
PageColor VARCHAR2(6) := 'CCFFFF'; -- Cyan
TabColor   VARCHAR2(6) := 'FFFFCC'; -- Yellow
```

```
PROCEDURE Banner(title VARCHAR2) IS
BEGIN
  http.htmlOpen;
  http.headOpen;
  http.title(title);
  http.headClose;
  http.bodyopen(cattributes => 'BGOLOR=#' || PageColor);
  http.hr;
  http.tableopen(cattributes => 'WIDTH="100%"');
  http.tablerowopen;
```

```

    http.tabledata('<H1>' || title || '</H1>', 'LEFT');
    http.tabledata(
        cvalue =>'<img SRC="/images/cap.gif" ALT="capitol">',
        cattributes => 'ALIGN="RIGHT" VALIGN="TOP"');
    http.tablerowclose;
    http.tableclose;
    http.hr;
END Banner;

PROCEDURE Enter(id NUMBER) IS
    CURSOR ce IS
        SELECT ename, job, hiredate, sal
        FROM emp
        WHERE emp.empno = id;
    name emp.ename%TYPE;
    jobd emp.job%TYPE;
    hired emp.hiredate%TYPE;
    salary emp.sal%TYPE;
BEGIN
    Banner('Enter Employee data');
    OPEN ce;
    FETCH ce INTO name, jobd, hired, salary;
    CLOSE ce;
    http.header(3, 'Updating employee data for ' || name || '.');
    http.formOpen('Employee.Updater');
    http.formHidden('id', id);
    http.p('Job description:');
    http.formText('jobd', 20, 20, jobd);
    http.p(chr(38) || 'nbsp');
    http.p('Hire date:');
    http.formText('hired', 20, 20,
        TO_CHAR(hired, 'DD-Mon-YYYY HH24:MI'));
    http.p(chr(38) || 'nbsp');
    http.p('Monthly salary:');
    http.formText('salary', 10, 10, salary);
    http.paragraph;
    http.formReset;
    http.formSubmit(NULL, 'Update');
    http.formClose;
    http.bodyClose;
    http.htmlClose;
END Enter;

PROCEDURE Updater (id NUMBER, jobd VARCHAR2, hired VARCHAR2,
salary NUMBER) IS
BEGIN
    Banner('Update complete');
    UPDATE emp SET

```

```

        job = jobd,
        hiredate = TO_DATE(hired, 'DD-Mon-YYYY HH24:MI'),
        sal = salary
WHERE empno = id;
http.header(3, 'The data has been updated. ' ||
    htf.anchor(owa_util.get_owa_service_path ||
        'Employee.Home', htf.img('/images/home32.gif')));
http.p(id || ';' || jobd || ';' || hired || ';' ||
salary);
http.bodyClose;
http.htmlClose;
END Updater;

PROCEDURE Select_Emp IS
    CURSOR ce IS
        SELECT empno, ename, job
        FROM emp
        ORDER BY ename;
    id    emp.empno%TYPE;
    name  emp.ename%TYPE;
    job   emp.job%TYPE;
BEGIN
    Banner('Select an Employee');
    http.header(3, 'Click on a link to update Employee data. ' ||
        htf.anchor(owa_util.get_owa_service_path ||
            'Employee.Home', htf.img('/images/home32.gif')));

    http.tableopen(cattributes => 'BORDER=1 BGCOLOR=#'
        || TabColor);
    http.tablerowopen;
    http.tableheader('ID', 'LEFT');
    http.tableheader('Name', 'LEFT');
    http.tableheader('Job', 'LEFT');
    http.tableheader('Command', 'LEFT');
    http.tablerowclose;
    OPEN ce;
    LOOP
        FETCH ce INTO id, name, job;
        EXIT WHEN ce%NOTFOUND;
        http.tablerowopen;
        http.tabledata(id, 'LEFT');
        http.tabledata(name, 'LEFT');
        http.tabledata(job, 'LEFT');
        http.tabledata(
            htf.anchor(owa_util.get_owa_service_path ||
                'Employee.Enter?id=' || id, 'Update'), 'LEFT');
        http.tablerowclose;
    END LOOP;

```

```

CLOSE ce;
htp.tableclose;

htp.bodyclose;
htp.htmlclose;
END Select_Emp;

PROCEDURE Home IS
BEGIN
  Banner('Employee Maintenance');
  htp.p('<H3>');
  htp.ulistopen;
  htp.listItem;
  htp.anchor(owa_util.get_owa_service_path ||
    'Employee.Select_Emp',
    'Select an Employee for update.');
```

```

  htp.listItem;
  htp.anchor(owa_util.get_owa_service_path ||
    'owa_util.showsource(''Employee'')',
    'View source code.');
```

```

  htp.ulistclose;
  htp.p('</H3>');
  htp.bodyclose;
  htp.htmlclose;
END Home;

END Employee;
/
```

## 6.2 WebDB/Portal

- Web-based form, report, graph and site development tool.
- Web-based site and developer administration tool.
- Pure HTML 3.x and JavaScript.
- Source generated for all packages.

## 7 Java

### 7.1 Java stored procedures

- Java Virtual Machine (JVM) runs inside Oracle.
- Java source may be compiled inside Oracle.
- Externally compiled Java classes may be loaded into Oracle via the loadjava command line utility.
- To be called from SQL or PL/SQL, define an interface to the Java.

create or replace java source named "MyTimestamp" as

```
import java.lang.String;
```

```

import java.sql.Timestamp;

public class MyTimestamp
{
    public static String getTimestamp()
    {
        return (new
            Timestamp(System.currentTimeMillis())).toString();
    }
};
/

create or replace function CallMTS return varchar2
as language java
name 'MyTimestamp.getTimestamp() return java.lang.String';
/

set serveroutput on
execute DBMS_Output.Put_Line(CallMTS);

COLUMN CallMTS FORMAT A25
select CallMTS, to_char(sysdate, 'YYYY-MM-DD HH24:MI:SS') from
dual;

```

## 7.2 JDBC

- Standard Java interface for connecting to RDBMS from Java.
- Defined in java.sql Java documentation.
- Oracle JDBC server driver is always available in the JVM.
- Oracle JDBC OCI driver requires client side software.
- Oracle JDBC thin driver is pure Java.
- Works from Java application, servlet or bean.

```

// This sample lists all the names from the Employee table.

// set CLASSPATH=.;f:\oracle\jdbc\lib\classes111.zip
// javac Employee.java
// java Employee

import java.sql.*;

class Employee
{
    public static void main (String args []) throws SQLException
    {
        // Load the Oracle JDBC driver
        DriverManager.registerDriver(new
            oracle.jdbc.driver.OracleDriver());
    }
}

```

```

// Connect to the database, using the parameter syntax
// "jdbc:oracle:thin:@<host>:<port>:<sid>",
// "<user>", "<password>".
Connection conn = DriverManager.getConnection
    ("jdbc:oracle:thin:@MyServer:1521:ORCL",
     "scott", "tiger");

// Create a Statement
Statement stmt = conn.createStatement ();

// Execute it
ResultSet rset = stmt.executeQuery
    ("select ENAME, HIREDATE, SAL from EMP order by ENAME");

// Iterate through the result set and print      String
name; Date hired; int salary;
while (rset.next ())
{
    name = rset.getString(1);
    hired = rset.getDate(2);
    salary = rset.getInt(3);
    System.out.println
        (name + " " + hired + " " + salary);
}
}
}

```

## 8 Forms

- Has built-in transaction functionality.
- Has pre-defined master-detail control.
- Easy Oracle connectivity.
- Highly portable to multiple platforms.
- Expensive.
- Thick client vs. three-tier architecture.

### 8.1 Modules

- Form module binary file (<module>.fmb).
- Form module executable file (<module>.fmx).
- Form module text file (<module>.fmt).

### 8.2 Blocks

#### Base table block

- Created with wizard or manually.
- Items are associated with a database table.
- Holds WHERE and ORDER BY expressions for querying.

- Arranged in single- or multi-record format.
- Optional scroll bar for multi-record block.
- Specifies default navigation sequence.

#### Control blocks

- Items are *not* associated with a database table.
- Used for buttons, summary items and lookup results.

#### Master-detail

- A base table block with two related tables.
- Coordinates querying between the two blocks.
- Relation specifies a join condition.

### 8.3 Object navigator

- Displays hierarchy of form objects.
- Expand and collapse outline view.
- Create, delete and copy objects.
- Quickly select objects for editing.
- Ownership view vs. Visual view

### 8.4 Property window

- Displays the properties of the currently selected object.
- Open additional property windows to compare properties.
- Double-click on item in object navigator to see properties.
- Shift-double-click to open a fresh property window.

### 8.5 Layout editor

- Used to graphically create and arrange windows, canvass-views, blocks and items.
- Use layout tools to select and manipulate objects.
- Use drawing tools to create background boilerplate.
- Use item tools to create database items.
- Use shift-click or lasso to select multiple items.
- Move items with the mouse or nudge them with the arrow keys.
- Resize items by dragging their selection points.
- Constrain an object's shape by shift dragging.
- Align objects using the alignment tool.
- Group objects to manipulate as a whole.

### 8.6 PL/SQL editor

- Used to add program control to forms.
- Create and edit event triggers, subprograms and packages.
- Use compile button to check syntax before running form.

## 8.7 Event triggers

### Event-driven model

- Standard database transaction processing.
- Standard GUI event handling.
- Triggers supplement or replace the standard handling.

### Trigger prefix

- Pre: Fires before an action.
- Post: Fires after an action.
- When: Fires in addition to an action.
- On: Fires instead of the standard action.
- Key: Replaces standard function-key processing.

### Query triggers

- Pre-Query: set up query conditions, alter where clause.
- Post-Query: decode look-up items items.

```
POST-QUERY ON :emp.department
BEGIN
  SELECT dname INTO :emp.department
  FROM dept
  WHERE dept.deptno = :emp.deptno;
END;
```

### Validation triggers

- When-Validate-Item.
- When-Validate-Record.

### Transaction triggers

- Pre-, On-, Post-Insert, Update and Delete.
- Pre-Commit, Post-Forms-Commit, Post-Database-Commit.
- Use Pre- and Post- triggers to supplement DML.
- Use On- triggers to access a non-Oracle database.

### Navigational triggers

- Fires when focus changes from one item or level to another.
- Pre-Form, Pre-Block and Pre-Text-Item.
- Post-Text-Item, Post-Block and Post-Form.

### Interface triggers.

- When-button-pressed.
- When-checkbox-changed.
- When-radio-changed.
- When-window-, When-image- and Key- triggers.

### Master-detail triggers

- On-Check-Delete-Master.
- On-Clear-Details.
- On-Populate-Details.

### Trigger scope

- Form level triggers apply to the whole form.
- Block level triggers apply to a single block.
- Item level triggers apply to a single item.

### Trigger code

## 8.8 Items

### Buttons

- A button is used to execute commands or initiate action.
- Use When-Button-Pressed trigger for program control.

### Check boxes

- A check box is a two-state item indicating yes/no, on/off etc.
- Can hold char, number or date data.
- Properties define values for checked and unchecked state.
- Other values may be either rejected or accepted (as on or off).
- Used in combination for multiple selection, e.g. day-available.
- State set from database, by program or using space bar.
- Use When-Checkbox-Changed trigger for program control.
- Use `Get_Item_Property` and `Set_Item_Property` to change state.
- Use `Convert_Other_Value` to validate state.

### Radio groups

- A radio group is a multi-state item indicating one of two or more mutually exclusive choices.
- Properties define which state to associate with each value.
- Other values may be rejected or assigned to a specific state.
- Maintained as a group by Forms.
- State set from database, by program or using space bar.
- Use When-Radio-Changed trigger for program control.
- Use `Get_Item_Property` and `Set_Item_Property` to change state.
- Use `Convert_Other_Value` to validate state.

### List items

- Text list: a fixed-size, scrolling list box.
- Poplist: a list that pops up when an indicator is pressed.
- Combo box: combines text list with poplist.

### Text items

- A text item is used to display and edit text.
- Formatting properties for alignment, bevel and font attributes.
- Function properties for maximum length, security, auto-skip and keep position.
- Validation properties for length, case, range or mandate.
- May be single- or multi-line.
- Single-line text may have a format mask.
- Multi-line text may be wrapped and may have an optional vertical scroll bar.

### Display items

- A display item is a text item that cannot be changed by the user.
- Used often to display results of Post-Query trigger results.
- Uses less memory than a Text item.

## 8.9 List of values (LOV)

### LOV

- Filled with data derived from database query.
- May be attached to a text item for validation.
- May be invoked programmatically, e.g. from a button.
- May be positioned on the screen or auto-centered.
- Auto-reduction feature as value is typed.
- Return items used to populate form.

### Record group

- Static record group contains constant data.
- Non-query record group is built programmatically.
- Query record group gets data from a query.
- Use `Populate_Group_With_Query` to replace a record group's query at runtime..

### Program control

- Use `List_values` in a `When-New-Item-Instance` or `Key-Listval` trigger.
- Use `Show_LOV` to display a LOV from a button.
- Use `Set_LOV_PropertyY` to change properties including position and the underlying record group.

## 8.10 Menus

### Modules

- Menu module binary (<module>.mmb).
- Menu executable (<module>.mmx).

### Default vs. custom menu

- Default menu specified as DEFAULT in form properties.
- Found in file menundef.mmb.
- Menu commands implemented as Do\_Key calls.

### Menu items

- Main menu lists pull-down menu names.
- Each menu lists menu items.
- Each menu items can have a submenu.
- For operator ease, limit submenus to one level.

### Menu commands

- A menu item's Command Type determines its function.
- Null means a menu separator.
- Menu means a submenu.
- PL/SQL means a PL/SQL command or subprogram call.

### PL/SQL\

- Invoke a form with Open\_Form, New\_Form or Call\_Form.
- Execute operating system commands with Host.
- Invoke another Oracle tool with Run\_Product.
- Call Do\_Key to execute built-in functions.
- Call Execute\_Trigger to invoke a form trigger.

### Menu program units

- Can't reference the value of a form object directly.
- Use the function Name\_In to get the value.
- Use the procedure Copy to assign values to objects.

## 8.11 Libraries

### Modules

- Library module binary (<module >.pll).
- Library module executable (<module >.plx).
- Library text file (<module>.pld).

### Library program units

- Can't reference the value of a form object directly.
- Use the function Name\_In to get the value.
- Use the procedure Copy to assign values to objects.

## 8.12 Multi-form applications

### Open\_Form

- Open a new form, while leaving the current form open.
- Use the activate option to make the new form get focus.
- Second form can use the same session or open a new one.
- Avoid using with forms that contain root windows.

### New\_Form

- Closes the current form and opens the new one.
- Use rollback option if data not saved.

### Call\_Form

- Suspend the current form and open a new form.
- Control returns to previous form after `Exit_Form`.

## **9 Reports**

### **9.1 Modules**

### **9.2 Data model**

Queries

Database columns

Formula columns

Summary columns

Data links

Placeholders

### **9.3 Layout editor**

Header

Trailer

Body

Margin

### **9.4 Parameter form**

## **10 References**

Portfolio, T., Russell, J., *PL/SQL User's Guide and Reference*, 2001, Oracle Corp.

Russell, J., *Oracle9i Application Developer's Guide—Fundamentals*, 2001, Oracle Corp.

Watt, S., *SQL\*Plus User's Guide and Reference*, 2001, Oracle Corp.