

**PONTIFICIA UNIVERSIDAD CATOLICA DEL
ECUADOR SEDE AMBATO (PUCESA)**

Escuela de Ingeniería de Sistemas

**DISERTACIÓN DE GRADO PREVIA LA
OBTENCIÓN DEL TÍTULO DE INGENIERA DE
SISTEMAS**

**“ESTUDIO DE LAS HERRAMIENTAS RAD (Rapid
Application Development) DE ORACLE PARA EL
DESARROLLO DE APLICACIONES”**

Myriam Elizabeth Alvarez Miranda

DIRECTOR: Ing. Galo López

Ambato, 2005



**SECRETARIA
ESCUELA DE INGENIERIA
DE SISTEMAS**

[Handwritten signature]
28-X-2005

**PONTIFICIA UNIVERSIDAD CATOLICA DEL
ECUADOR SEDE AMBATO (PUCESA)**

Escuela de Ingeniería de Sistemas

**DISERTACIÓN DE GRADO PREVIA LA
OBTENCIÓN DEL TÍTULO DE INGENIERA DE
SISTEMAS**

**“ESTUDIO DE LAS HERRAMIENTAS RAD (Rapid
Application Development) DE ORACLE PARA EL
DESARROLLO DE APLICACIONES”**

Director:



Ing. Galo López

Revisores:



Ing. _____

Ing. _____

Myriam Elizabeth Alvarez Miranda

Ambato, 2005

DEDICATORIA

A Dios Todopoderoso, por haberme ayudado y guiado en el transcurso de mi existencia.

A mis padres y hermanos, por su apoyo constante en todas las actividades que he emprendido.

A mi esposo y mis hijos, por su paciencia y sacrificio, principalmente en esta última etapa de mis estudios.

AGRADECIMIENTO

A la Pontificia Universidad Católica del Ecuador sede Ambato, su personal docente, administrativo y de servicios, quienes durante mi etapa estudiantil, supieron apoyarme en forma desinteresada, para culminar mi carrera profesional.

A mi director de tesis, Ing. Galo López, que demostró su profesionalismo en la dirección del presente trabajo.

INDICE

DEDICATORIA.....	I
AGRADECIMIENTO	II
INDICE	III
INDICE DE TABLAS.....	VII
INDICE DE GRÁFICOS	VIII
RESUMEN.....	1
INTRODUCCIÓN	2
CAPITULO I.....	3
PLANTEAMIENTO DEL PROBLEMA.....	3
1.1.- JUSTIFICACIÓN	3
1.2.- DELIMITACIÓN	3
1.3.- IMPORTANCIA.....	4
1.4.- OBJETIVOS.....	4
1.4.1.- General.....	4
1.4.2.- Específicos	4
1.5.- MARCO REFERENCIAL.....	5
1.5.1.- Marco Teórico	5
1.6.- HIPÓTESIS.....	6
1.7.- METODOLOGÍA DE LA INVESTIGACIÓN.....	6
1.7.1.- Tipo de Investigación.....	6
1.7.2.- Técnicas de Recolección de Datos.....	6
1.8.- METODOLOGÍA INFORMÁTICA	7
CAPITULO II.....	10
FUNDAMENTOS TEÓRICOS.....	10
2.1.- BASES DE DATOS RELACIONALES.....	10
2.2.- SQL (LENGUAJE DE CONSULTA ESTRUCTURADO)	13
2.3.- MODELO ENTIDAD - RELACIÓN.....	14
Llaves Primarias:	22
Llaves Candidatas:	22
Llaves Foráneas:.....	22
2.4.- MANIPULACIÓN DE DATOS.....	25
2.4.1.- Consultas	25
2.4.2.- Insertar Registros.....	30
2.4.2.1.- Insertar registros únicos:.....	30
2.4.2.2.- Insertar Registros de otra Tabla:	31
2.4.3.- Actualizar Registros.....	32
2.4.4.- Eliminar o Borrar Registros.....	33
2.4.5.- Subconsultas	33
2.5.- PL/SQL (LENGUAJE PROCEDURAL)	37
2.5.1.- Reglas para Bloques	39
2.5.2.- Alcance de Objetos de Bloque.....	40
2.5.3.- Declarar Variables y Constantes.....	40
2.5.4.- Registros PL/SQL.....	41
2.5.5.- Asignaciones PL/SQL.....	42

2.5.6.-	Cursores.....	43
2.5.6.1.-	Cursores Implícitos.....	43
2.5.6.2.-	Cursores Explícitos.....	43
2.5.7.-	Manejo de Excepciones (Errores).....	46
2.5.8.-	Estructuras de Control.....	50
2.5.8.1.-	IF.....	50
2.5.8.2.-	Ciclos.....	50
2.5.9.-	Procedimientos y Funciones.....	53
2.5.10.-	Paquetes.....	56
2.5.11.-	Triggers (Disparadores).....	58
CAPITULO III.....		60
ADMINISTRACIÓN ORACLE.....		60
3.1.-	ADMINISTRACIÓN.....	60
3.1.1.-	Usuarios.....	60
3.1.2.-	Seguridades.....	62
3.1.3.-	Objetos.....	63
3.2.-	HERRAMIENTAS DE DESARROLLO (ORACLE DEVELOPER).....	66
3.3.-	HERRAMIENTAS DEVELOPER PARA WINDOWS 95/98/NT/2000.....	69
3.3.1.-	Pantalla Principal de Forms Builder.....	69
3.3.1.1.-	Navegador de Objetos Oracle Forms.....	70
3.3.1.2.-	Conectar a la Base de Datos.....	73
3.3.1.3.-	Diseño de Formas.....	73
3.3.1.4.-	Crear y Ejecutar una Forma con un Bloque Simple.....	76
3.3.1.5.-	Editor Layout.....	84
3.3.1.6.-	Grabar, Compilar y Ejecutar Formas.....	87
3.3.1.7.-	Grabar una Forma.....	87
3.3.1.8.-	Compilar/Construir una Forma.....	88
3.3.1.9.-	Ejecutar una Forma.....	89
3.3.1.10.-	Ejecutar una Forma como Aplicación Java o Applet.....	90
3.3.1.11.-	Consultas.....	91
3.3.1.12.-	Formas.....	94
3.3.1.13.-	Crear una Forma Master - Detalle.....	95
3.3.1.14.-	Crear el bloque Master.....	95
3.3.1.15.-	Crear el Bloque Detalle.....	96
3.3.1.16.-	Propiedades de Relación.....	99
3.3.1.17.-	Unidades de Programa.....	102
3.3.1.18.-	Extender Formas Master-Detalle.....	105
3.3.1.19.-	Listas de Valores (LOV) y Campos de Tablas No-Base... ..	105
3.3.1.20.-	Campos Tabla No-Base.....	112
3.3.1.21.-	Crear un Trigger.....	114
3.3.2.-	Unidades de Programa Oracle Forms.....	117
3.3.2.1.-	Crear Unidades de Programa.....	118
3.3.2.2.-	Adicionar un Nuevo ítem al Bloque Empleado.....	118
3.3.2.3.-	Crear un Procedimiento para Contar los Miembros del Departamento.....	119
3.3.2.4.-	Crear un Trigger para Llamar al Procedimiento.....	121
3.3.2.5.-	Grabar, Compilar/Construir y Ejecutar la Forma.....	121

3.3.2.6.- Crear Procedimientos Almacenados en SQL*Plus	122
3.3.2.7.- Procedimientos Almacenados y Triggers adicionales	125
3.3.2.8.- Generar identificadores únicos o consecutivos usando la tabla	125
3.3.2.9.- Generar identificadores únicos o consecutivos (Secuencia Oracle)	126
3.3.2.10.- Verificar restricciones en el cliente	127
3.3.2.11.- Adicionar Alertas (Mensajes Pop-up).....	128
3.3.2.12.- Llenar items con valores por omisión	129
3.3.2.13.- Funcionalidad de los botones de comando	130
3.3.2.14.- Pasar parámetros entre formas	130
3.3.3.- Oracle Reports.....	132
3.3.3.1.- Navegador de Objetos de Oracle Reports.....	133
3.3.3.2.- Crear un Reporte de Tabla Simple	135
3.3.3.3.- Grabar, Generar y Ejecutar el Reporte	140
3.3.3.4.- Crear un Reporte Master-Detalle	140
3.3.3.5.- Forma de Parámetro en Oracle Reports.....	142
3.3.3.6.- Construir una Forma con Parámetros Personalizados	144
3.3.4.- Opciones en Oracle Forms.....	146
3.3.4.1.- Jerarquía de Opciones.....	147
3.3.5.- Graphics y Chart.....	153
3.3.5.1.- Adicionar un Chart a una Forma	154
3.3.6.- Oracle Schema Builder	158
CAPITULO IV.....	165
DIAGNÓSTICO DEL SISTEMA ACTUAL	165
4.1.- RESULTADOS DE LAS TÉCNICAS DE INFORMACIÓN.....	165
4.1.1.- Ubicación y Propietarios	165
4.1.2.- Sistema Actual	165
4.2.- DEFICIENCIAS DEL SISTEMA	165
4.3.- ANÁLISIS DE REQUERIMIENTOS	166
4.4.- NUEVO SISTEMA.....	167
CAPITULO V	168
ANÁLISIS DEL SISTEMA PROPUESTO.....	168
5.1.- ESTUDIO DE FACTIBILIDAD	168
5.1.1.- Planificación de las actividades	169
5.1.2.- Recursos del proyecto	169
5.1.3.- Costo del sistema actual.....	170
5.1.4.- Costo del sistema propuesto	170
5.2.- ANÁLISIS DE RIESGOS.....	172
CAPITULO VI.....	175
DISEÑO DEL SISTEMA	175
6.1.- ANÁLISIS ESTRUCTURADO	175
6.1.1.- Componentes.....	175
6.1.2.- Análisis de Flujo	176
6.1.2.1.- Descripción del Diccionario de Datos	176
6.1.2.2.- Diagrama de Flujo de Datos (DFD)	177
6.1.2.3.- Jerarquía de Procesos	184

6.1.2.4.- Referencia de procesos	184
6.1.2.5.- Entidades externas	185
6.1.2.6.- Almacenamientos	185
6.1.3.- <i>Modelo Relacional</i>	185
6.1.4.- <i>Modelo Entidad Relación</i>	186
CAPITULO VII	187
DESARROLLO DEL SISTEMA	187
7.1.- IMPLEMENTACIÓN	187
7.1.1.- <i>Creación de Tablas</i>	187
7.1.2.- <i>Insertar datos</i>	204
7.2.- PRUEBAS	204
CONCLUSIONES.....	205
RECOMENDACIONES.....	206
BIBLIOGRAFÍA	207
GLOSARIO.....	209
ANEXOS	211
ANEXO 1.- MANUAL DE USUARIO	211
ANEXO 2.- MANUAL DEL ADMINISTRADOR	211

INDICE DE TABLAS

Tabla 2.1. Componentes del Modelo Entidad-Relación	16
Tabla 2.2. Convenciones para Identificar Relaciones	19
Tabla 2.3. Ejemplo de Relaciones	20
Tabla 2.4. Tipos de Relaciones	20
Tabla 2.5. Regla de Normalización	21
Tabla 2.6. Tipos de Restricciones	21
Tabla 2.7. Símbolos para la Carta de Instancia de Tabla	25
Tabla 3.1. Funciones Básicas dentro de una Forma	94
Tabla 5.1. Planificación de Actividades	169
Tabla 5.2. Coeficientes a, b, c y d para las Ecuaciones	171
Tabla 5.3. Ciclo de Vida del Proyecto	171
Tabla 6.1. Símbolos Gráficos para Componentes del Sistema	175

INDICE DE GRÁFICOS

Fig. 1.1. Ciclo de Desarrollo de Sistemas.....	7
Fig. 2.1. Operaciones DML.....	11
Fig. 2.2. Modelos de Datos	15
Fig. 2.3. Modelo Entidad-Relación.....	16
Fig. 2.4. Convenciones del Modelo Entidad Relación.....	17
Fig. 2.5. Ejemplo de Modelo Entidad-Relación	18
Fig. 2.6. Subgrupo del Modelo Entidad-Relación	19
Fig. 2.7. Relaciones Recursivas.....	20
Fig. 2.8. Pasos del Diseño de Bases de Datos.....	23
Fig. 3.1. Módulos de una Aplicación	63
Fig. 3.2. Componentes de un Proyecto, Project Builder	64
Fig. 3.3. Archivos de un Proyecto.....	64
Fig. 3.4. Registros de un Proyecto.....	65
Fig. 3.5. Registros Globales y Locales.....	66
Fig. 3.7. Herramientas Procedure Builder y Query Builder.....	68
Fig. 3.8. Herramientas Schema Builder y Translation Builder	68
Fig. 3.9. Opciones Developer.....	69
Fig. 3.10. Pantalla de Ingreso a Forms Builder.....	69
Fig. 3.11. Pantalla Principal de Oracle Forms	70
Fig. 3.12. Conexión a la Base de Datos.....	73
Fig. 3.13. Ventana Típica de Forms.....	74
Fig. 3.14. Bloque Simple y Bloque Simple con Campo de Observación.....	75
Fig. 3.15. Master/Detalle y Master/Detalle con Campo de Observación.....	75
Fig. 3.16. Asistente de Bloque de Datos.....	77
Fig. 3.17. Tipo de Bloque de Datos.....	77
Fig. 3.18. Tabla y Columnas del Bloque de Datos.....	78
Fig. 3.19. Navegador del Bloque de Datos.....	78
Fig. 3.20. Seleccionar Columnas para el Bloque de Datos	79
Fig. 3.21. Ventana final del Asistente de Bloque de Datos.....	79
Fig. 3.22. Ventana Asistente Layout.....	80
Fig. 3.23. Asistente Layout, Seleccionar el Lienzo	80
Fig. 3.24. Asistente Layout, Seleccionar Bloque de Datos	81
Fig. 3.25. Asistente Layout, Modificar Estilo de Campos.....	81
Fig. 3.26. Asistente Layout, Estilo del Layout.....	82
Fig. 3.27. Asistente Layout, Título y Número de Registros	83
Fig. 3.28. Asistente Layout, Ventana Final	83
Fig. 3.29. Editor de Layout.....	84
Fig. 3.30. Modificar ancho de Campo	85
Fig. 3.31. Ventana de Propiedades	85
Fig. 3.32. Ventana de Propiedades Bloque EMPLEADO.....	86
Fig. 3.33. Grabar una Forma.....	87
Fig. 3.34. Ejecutar una Forma.....	89
Fig. 3.35. Forma como Aplicación Java o Applet	90
Fig. 3.36. Oracle Forms Server	91

Fig. 3.37. Modelo Entidad Relación Uno a Muchos	95
Fig. 3.38. Bloque de Datos Master.....	96
Fig. 3.39. Asistente Bloque de Datos.....	97
Fig. 3.40. Tipo de relación.....	97
Fig. 3.41. Relación Master - Detalle.....	98
Fig. 3.42. Forma Master – Detalle	99
Fig. 3.43. Propiedades de la relación Departamento-Emppleado.....	100
Fig. 3.44. Principales Propiedades de Master/Detalle.....	100
Fig. 3.45. Navegador de Objetos en Master/Detalle.....	103
Fig. 3.46. Editor PL/SQL para una Unidad de Programa.....	103
Fig. 3.47. Crear un nuevo LOV	106
Fig. 3.48. Asistente LOV	106
Fig. 3.49. Condición LOV	107
Fig. 3.50. Columnas LOV	107
Fig. 3.51. Propiedades de Columnas LOV	108
Fig. 3.52. Items y Parámetros LOV.....	108
Fig. 3.53. Título y Posiciones de LOV	109
Fig. 3.54. Número de Registros LOV.....	109
Fig. 3.55. Asociar LOV a Campos	110
Fig. 3.56. Paso Final del Asistente LOV	110
Fig. 3.57. LOV en Navegador de Objetos.....	111
Fig. 3.58. Forma con LOV	111
Fig. 3.59. Crear un Campo tabla no-base	113
Fig. 3.60. Propiedades del Item de Texto	113
Fig. 3.61. Crear un Trigger.....	115
Fig. 3.62. Editor PL/SQL para el Trigger	116
Fig. 3.63. Forma con LOV	117
Fig. 3.64. Nuevos Items	119
Fig. 3.65. Forma incluida un nuevo Item	122
Fig. 3.66. Alertas	129
Fig. 3.67. Oracle Reports	132
Fig. 3.68. Opciones Oracle Reports	133
Fig. 3.69. Ventana Inicial de Oracle Reports	133
Fig. 3.70. Navegador de Objetos de Reports.....	134
Fig. 3.71. Asistente Reports	136
Fig. 3.72. Sentencia SQL para el Reporte	137
Fig. 3.73. Seleccionar Columnas para el Reporte.....	137
Fig. 3.74. Funciones Aplicadas a los Campos del Reporte.....	138
Fig. 3.75. Cambiar Etiquetas y Anchos en los Campos del Reporte	138
Fig. 3.76. Modelos de Reporte	139
Fig. 3.77. Pre visualización del Reporte.....	139
Fig. 3.78. Actividad de Generación del Reporte	140
Fig. 3.79. Reporte Master/Detalle.....	142
Fig. 3.80. Alerta para Forma de Parámetros.....	143
Fig. 3.81. Forma de Parámetros.....	144
Fig. 3.82. Constructor de Forma de Parámetros.....	144
Fig. 3.83. Forma de Parámetros Personalizada.....	145

Fig. 3.84. Ejecución de la Forma de Parámetros..... 145

Fig. 3.85. Opciones por Omisión..... 146

Fig. 3.86. Propiedad Menu Module..... 147

Fig. 3.87. Ventana Menu Editor..... 148

Fig. 3.88. Items de Opciones 149

Fig. 3.89. Propiedades Forma en Blanco 152

Fig. 3.90. Forma con Opciones Personalizadas 152

Fig. 3.91. Graphics Builder..... 153

Fig. 3.92. Forma Master/Detalle Departamento-Empleados 154

Fig. 3.93. Asistente Chart..... 155

Fig. 3.94. Esquema y Subtipo de Chart..... 155

Fig. 3.95. Seleccionar Bloque de Datos para Chart 156

Fig. 3.96. Campos para el eje X del Chart 156

Fig. 3.97. Campos para el eje Y del Chart 157

Fig. 3.98. Nombre del Archivo Chart 157

Fig. 3.99. Forma con Chart..... 158

Fig. 3.100. Pantalla de Oracle Scheme 159

Fig. 3.101. Crear una Tabla en Schema Builder..... 159

Fig. 3.102. Cambiar Características de Columnas..... 160

Fig. 3.103. Tabla Cliente en Schema Builder 161

Fig. 3.104. Sentencia SQL Generada por Schema Builder..... 161

Fig. 3.105. Hacer Commit a la Base de Datos 162

Fig. 3.106. Objetos del Esquema..... 162

Fig. 3.107. Incluir Tablas Existentes en Schema Builder..... 163

Fig. 3.108. Relacionar Tablas en Schema Builder..... 163

Fig. 3.109. Tablas Relacionadas en Schema Builder..... 164

Fig. 3.110. SQL para Mostrar las Relaciones..... 164

Fig. 6.1. DFD Nivel 0, Diagrama de Contexto..... 178

Fig. 6.2. DFD Nivel 1, Sistema Producción de Leche..... 179

Fig. 6.3. DFD Nivel 2, Subproceso Producción de Leche..... 179

Fig. 6.4. DFD Nivel 3, Subproceso Registro de Producción de Leche 180

Fig. 6.5. DFD Nivel 2, Subproceso Producción 181

Fig. 6.6. DFD Nivel 3, Subproceso Registrar Productos 182

Fig. 6.7. DFD Nivel 2, Subproceso Pedidos 183

Fig. 6.8. DFD Nivel 2, Subproceso Clientes y Proveedores..... 183

Fig. 6.9. Modelo Relacional..... 185

Fig. 6.10. Modelo Entidad-Relación 186

RESUMEN

El avance vertiginoso que se presenta día a día, principalmente durante las últimas décadas, en lo que se refiere al campo tecnológico, obliga a que las empresas públicas y privadas inviertan tanto en la actualización permanente de hardware y software, y en la capacitación de su personal, para garantizar el cumplimiento de sus objetivos.

En lo referente al campo informático, el desarrollo de hardware ocurre con mayor rapidez que el desarrollo del software, este último trata de aprovechar las ventajas en cuanto a capacidad y velocidad que ofrece el primero.

Las herramientas de desarrollo actuales, en una gran mayoría, presentan interfaces gráficas, intuitivas, que con la ayuda de asistentes, ayudan a los desarrolladores a realizar actividades que normalmente se tenían que realizar en forma manual, invirtiendo gran cantidad de tiempo y recursos.

En el caso de desarrollo de sistemas de información y gestión, uno de sus principales componentes es, sin duda, la Base de Datos, que permite almacenar y manipular la información, contando adicionalmente con sistemas de seguridad para garantizar su integridad.

La mayoría de Bases de Datos, utilizan lenguajes y sublenguajes normalizados, que toman como base el SQL, y manejan más o menos los mismos elementos básicos: tablas, vistas, procedimientos, triggers, etc.

Con el apoyo de dichas herramientas, se implementa un sistema para el control de producción de leche en la hacienda “El Nilo y El Rosario” del cantón Pillaro en la Provincia del Tungurahua.

INTRODUCCIÓN

El avance científico y tecnológico de los momentos actuales, tanto en Hardware como en Software, requiere de profesionales capaces de aprovechar las herramientas disponibles, para así obtener productos informáticos robustos y confiables.

El desarrollo acelerado de herramientas de Software, cada vez más poderosas, requiere esfuerzos adicionales para una constante actualización de quienes se encuentran involucrados en la creación de aplicaciones.

La generalización de la programación orientada a objetos, ha permitido cierta estandarización tanto para el diseño como para la implementación de aplicaciones, realizadas anteriormente con lenguajes procedurales como C, Pascal, C++, etc.

Es así como los ambientes de desarrollo de aplicaciones RAD, tales como Visual Basic, Visual C, Delphi, Visual Foxpro, y Oracle han alcanzado gran acogida, para el desarrollo de aplicaciones a gran escala.

Los ambientes de desarrollo RAD, presentan facilidades para la realización, desde el diseño de bases de datos (back end), hasta el diseño de las interfaces de usuario (front end).

CAPITULO I

PLANTEAMIENTO DEL PROBLEMA

1.1.- JUSTIFICACIÓN

Actualmente se dispone de muchas herramientas para el desarrollo de aplicaciones, como Access, Visual Basic, Visual FoxPro, Delphi, etc., todas tienen integrados ambientes de desarrollo RAD, y utilizan los principios básicos de la Programación Orientada a Objetos.

El acceso a Bases de Datos a través de ODBC's, en algunos casos, presenta ciertas dificultades para los diseñadores y programadores, por tal motivo, empresas de Bases de Datos a gran escala como ORACLE, han implementado herramientas para realizar sus operaciones a través de conexiones nativas.

Por tanto, se justifica el estudio de las herramientas de desarrollo y de la Base de Datos ORACLE, para aplicaciones de mediana y gran complejidad.

1.2.- DELIMITACIÓN

El presente estudio tiene como finalidad aplicar los conceptos básicos de la Programación Orientada a Objetos, de las Bases de Datos, para determinar su aplicabilidad con las herramientas RAD de ORACLE, analizar, diseñar e implementar un sistema para el CONTROL DE PRODUCCIÓN DE LECHE en la Hacienda "El Nilo y El Rosario", ubicada en el Cantón Pillaro de la Provincia del Tungurahua.

1.3.- IMPORTANCIA

El estudio de las herramientas RAD de ORACLE, es importante por cuanto permitirá a los desarrolladores de sistemas, trabajar con un mismo tipo de herramienta, que evite el exceso de tráfico que puede ocurrir al trabajar con conexiones ODBC's, principalmente en aplicaciones que manejan gran cantidad de transacciones, y disponer no solo de unas cuantas herramientas para realizar su trabajo.

Además, se debe tener presente, que en la ciudad, provincia, y país, muchas empresas medianas y grandes, se encuentran realizando migraciones de otras herramientas a ORACLE.

1.4.- OBJETIVOS

1.4.1.- General

- Realizar un estudio de las herramientas de desarrollo RAD de ORACLE, para la generación de aplicaciones.

1-4-2- Específicos

- a) Estudiar los conceptos básicos para el manejo de la Base de Datos ORACLE, y de las herramientas RAD de ORACLE para el diseño de Aplicaciones.
- b) Analizar las principales características de las herramientas RAD de ORACLE.
- c) Utilizar las herramientas RAD, para el desarrollo de una aplicación para el Control de Producción de Leche en la Hacienda "El Nilo y El Rosario".

1.5.- MARCO REFERENCIAL

1.5.1.- Marco Teórico

El objetivo primordial de la Escuela de Ingeniería en Sistemas de la PUCESA, es sin duda alguna alcanzar la calidad educativa, para hacerlo cuenta con un Plan de Estudios que día a día tiene que ir actualizándose, de acuerdo a los avances tecnológicos que alcanza la ciencia.

El conocimiento y utilización de herramientas actuales, permite que los estudiantes alcancen una profesionalización acorde a los requerimientos del medio y una inserción rápida en el proceso productivo.

La época actual, ha puesto en manos de la sociedad nuevas tecnologías que principalmente incrementan la productividad y bienestar del hombre.

La esfera educativa, no puede permanecer al margen del acelerado auge que ha tomado la tecnología de punta, y sus constantes cambios, para asumir la nueva relación hombre – técnica, por lo que se propone facilitar un estudio sobre herramientas RAD, con acceso a bases de datos a través de conexiones nativas y su utilización en Internet.

Forms Developer es el Medio Ambiente de Desarrollo Rápido de Aplicaciones de ORACLE (RAD), que permite la construcción de aplicaciones a gran escala, que utiliza características declarativas, de forma que los desarrolladores de aplicaciones, las pueden crear rápidamente a partir de definiciones simples, y permite:

- Presentar aplicaciones en Internet.
- Producir aplicaciones eficientes y escalables
- Construir aplicaciones grandes, consistentes, y manejables en forma rápida
- Usar herramientas de cuarto nivel y servidores de Bases de Datos.

1.6.- HIPÓTESIS

El uso de herramientas RAD de ORACLE, permite el desarrollo de aplicaciones robustas, eficientes y escalables, ahorrando tiempo y esfuerzo de los desarrolladores.

1.7.- METODOLOGÍA DE LA INVESTIGACIÓN

1.7.1.- Tipo de Investigación

ANALÍTICO.- Identificar mediante los resultados obtenidos, las ventajas que presenta el uso de herramientas RAD de ORACLE.

INDUCTIVO.- Del estudio de obtendrá conclusiones generales que serán expuestas y analizadas.

TEÓRICO SISTEMÁTICO.- Para plantear el desarrollo de una aplicación de una manera técnica.

1.7.2.- Técnicas de Recolección de Datos

- Internet
- Documentación de Oracle (libros, manuales, y revistas)
- Entrevistas
- Observación
- Lectura Científica
- Técnicas de Análisis y Síntesis

1.8.- METODOLOGÍA INFORMÁTICA

Desde el concepto hasta su producción, se utilizará la metodología del Ciclo de Vida Clásico, conocido como Modelo en Cascada, por cuanto se requiere un trabajo sistemático y secuencial para el desarrollo de un sistema informático, el mismo que esta formado por múltiples estados, que permiten transformar los requerimientos de información del negocio en una aplicación operacional.

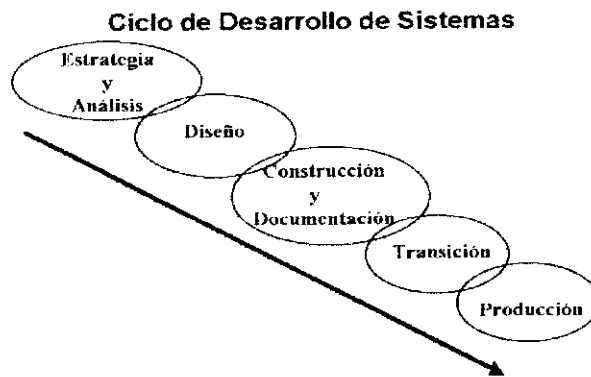


Fig. 1.1. Ciclo de Desarrollo de Sistemas

Estrategia Y Análisis

- ☑ Estudiar y analizar los requerimientos del negocio. Con la intervención de los usuarios y jefes, identificar los requerimientos de información. Incorporar la misión de la empresa y aplicación, tanto como cualquier especificación del futuro sistema.
- ☑ Construir modelos del sistema. Transferir el desarrollo narrativo del negocio, de la fase de estrategia y análisis, en una representación gráfica de las necesidades de información y reglas del negocio. Confirmar y redefinir el modelo con los analistas y expertos.

Diseño

- ☑ Diseñar la base de Datos. Con los mapas del modelo Entidad – Relación, transformar entidades a tablas, atributos a columnas, relaciones a llaves foráneas y reglas del negocio a restricciones.

Construcción y Documentación

- ☒ Construir el prototipo del sistema. Escribir y ejecutar los comandos para crear las tablas y objetos soportados por la base de datos.
- ☒ Desarrollar la documentación de usuario, texto de ayuda en pantalla, y manuales de operación para soporte del usuario y operación del sistema.

Transición

- ☒ Refinar el prototipo. Mover una aplicación en producción con pruebas de aceptación del usuario. Conversión de datos existentes, y operaciones paralelas. Realizar cualquier modificación requerida.

Producción

- ☒ Entregar el sistema a los usuarios. Operar el sistema de producción. Monitorear su desempeño y, reforzar y refinar el sistema.

El diseño de un sistema de Base de datos relacional, incluye convertir un modelo en una representación de software que trabaje. Las entidades u objetos percibidos por el usuario son transformados en tablas en la base de datos. Todas las formas de diseño incluyen una mezcla de reglas, juicios, y sentido común.

Durante el diseño, el objetivo es diseñar sistemas fiables, de alto desempeño, usando las estrategias del análisis. Los siguientes factores clave deben ser considerados:

- 1.- **Desempeño.**- El diseño inicial de un sistema tiene un enorme impacto en su desempeño final. Generalmente el impacto es mucho más grande que cualquier afinación remedial.
- 2.- **Aplicación Integrada.**- Los sistemas de aplicación son típicamente realizados por grupos de desarrolladores, sin alguna especificación de diseño desde la cuál partir, los desarrolladores utilizan su propio estilo. No solo buenos diseños promueven un buen aspecto y percepción, pero también

ayuda a asegurar que todos los componentes de la aplicación resultante se encuentren integrados entre sí.

- 3.- Integración con Otros Sistemas.-** A menudo, hay requerimientos que un nuevo sistema se integre con sistemas existentes o con sistemas a ser contruidos en otras plataformas. Un buen diseño extiende los beneficios de integración en sistemas corporativos.
- 4.- Documentación y Comunicación.-** La mayor parte del trabajo de un diseñador es comunicar las decisiones del diseño a otros. Estas decisiones, por lo menos deben estar bien documentadas.
- 5.- Escalabilidad.-** Acometer el desempeño durante el diseño en lugar de durante la producción. Por ejemplo, desarrollando una aplicación en un medio ambiente pequeño, controlado no verifica las situaciones del mundo real o grandes grupos de datos, factores que pueden revelar diseños imperfectos.
- 6.- Evitar Reinventar la Rueda.-** Muchos de los problemas que se deben enfrentar durante el desarrollo, ya han sido encontrados por otros antes. Se debe usar soluciones de diseños existentes, comprobados, cuando se pueda.

CAPITULO II

FUNDAMENTOS TEÓRICOS

2.1.- BASES DE DATOS RELACIONALES

Una base de datos está generalmente definida como un conjunto integrado de datos que modelan un universo. Este universo está compuesto por objetos interrelacionados, los objetos de un mismo tipo constituyen una **entidad** y la relación entre entidades se denomina **asociación**.

El proceso de descripción de **asociaciones** y **entidades** se llama **modelamiento**, y se lo hace con la ayuda de un **modelo de datos**; los modelos de datos más generales son:

- ☒ **Modelo jerárquico.**
- ☒ **Modelo en red**
- ☒ **Modelo objeto**
- ☒ **Modelo relacional:** El principio básico de este modelo es representar tanto las **entidades** como las **asociaciones** con la ayuda de **relaciones** denominadas también **tablas**. Una **tabla** está compuesta por **filas** y **columnas**, cada **fila** representa un objeto (proveedor - artículo) las **columnas** representan los **atributos** de dicho objeto. Una **tabla** es una **estructura**.

Las funciones de los Sistemas Manejadores de Bases de Datos DBMS (Data Base Management Systems) son:

- ☒ Permitir la definición de todos los datos
- ☒ Permitir manipular datos
- ☒ Establecer controles para la seguridad de estos datos
- ☒ Permitir los accesos concurrentes.

Los objetivos de un DBMS son:

- ☑ **Definición de datos:** sirve para modificar la estructura o añadir campos, la misión es describir y definir todos los esquemas que participen en la base de datos. Consiste en la descripción de los objetos que se van a representar. La descripción de todas las estructuras que forman una base de datos.
- ☑ **Definición de vistas:** Es una visión parcial de la tabla, “cuando en una tabla, alguna parte de ésta no se quiere que alguien la manipule”
- ☑ **Manipulación de datos:** recoge todas las operaciones de intercambio de datos entre las tablas, estas operaciones pueden ser de consulta o de puesta al día (inserción, modificación, supresión).

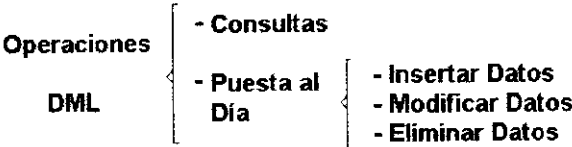


Fig. 2.1. Operaciones DML.

- ☑ **Seguridad de los datos:** consiste en garantizar que sólo los usuarios autorizados puedan efectuar operaciones correctas sobre la Base de Datos, para ello se dispone de 2 tipos.
 - Control sobre la base de datos
 - Control sobre las tablas

El principal objetivo de la implantación de una base de datos es poner a disposición de un gran numero de usuarios un conjunto integrado de datos, estos datos podrán ser manipulados por los diferentes usuarios, y es ahora cuando se debe garantizar la coherencia de los datos después de las diversas manipulaciones. Esto se garantiza con la ayuda del concepto de **transacción**, que se define como una unidad lógica de tratamiento, que aplicada a un estado coherente de una base de datos, restituye un nuevo estado coherente de la base de datos, pero con sus datos modificados (únicamente puede ser completamente modificado o anulado).

El ciclo de vida de una base de datos se puede descomponer en 3 etapas:

- ☑ **Concepción:** Consiste en reproducir el mundo real con la ayuda de uno de los modelos de datos conocidos (relacional). El resultado de esta fase es un esquema escrito según un formalismo cualquiera, no interpretable por el DBMS.
- ☑ **Creación de la Base de Datos vacía:** traducir el esquema en ordenes comprensibles para el DBMS, como resultado se obtiene la estructura de la base de datos, desprovista de cualquier tipo de información.
- ☑ **Explotación:** Los registros serán manipulados con la ayuda de los lenguajes de programación. Es ahora, cuando los usuarios pueden consultar los datos y ponerlos a punto durante el resto de la vida de la base de datos.

Para evitar problemas que se presentan por una mala concepción de la Base de Datos, se debe garantizar que no exista duplicación de datos en las tablas:

- ☑ **Redundancia de datos:** Si un cliente ha realizado más de un pedido, todos los datos de este cliente estarán repetidos tantas veces como pedidos haya, lo mismo sucede para los artículos esto es opuesto al principal objetivo de una base de datos que consiste en evitar la repetición de los mismos.
- ☑ **Puestas al día múltiple:** Para poder asegurar la coherencia de los datos, es necesario efectuar múltiples puestas a día, por ejemplo, cuando un cliente cambia de dirección
- ☑ **Incoherencia de los datos:** Si una operación de puesta al día múltiple, no se ha realizado completamente, el estado de la base de datos queda incoherente y puede producir errores importantes.
- ☑ **Pérdida de datos:** La supresión de una línea, en la tabla de pedidos, entraña la pérdida de todos los datos relativos a un cliente, si no ha efectuado ningún otro pedido. Esto es cierto también para un artículo que no ha sido pedido por ningún otro cliente. Estas anomalías constituyen lo que se ha convenido en llamar **comportamiento anormal de las tablas**, para evitar esto, existe un proceso llamado **normalización**,

que entre otras cosas intenta esclarecer los conceptos de **dependencia funcional** y **estado de las tablas**.

- ☑ **Dependencia funcional:** Concepto que se aplica a las columnas, y consiste en hacer corresponder un único valor a aquella columna o columnas que se consideren más significativas.
- ☑ **Estado de la tabla:** Se dice que una tabla está en primera forma normal (1FN), si toda columna de la tabla sólo tiene valores atómicos, un valor es atómico si él no es divisible.

Un paso crítico en la implementación del modelo de datos relacional, conocido como **normalización**, fue propuesta por Tedd Codd en los años 70, en es un proceso que consiste de la descomposición de tablas para un procesamiento eficiente, como en el caso de la primera forma normal que dice: **la información debe ser atómica**. Desde la perspectiva de sistemas orientados a objetos, este proceso crea algunos problemas. Primero la relación como resultado de la normalización no corresponda con el objeto del mundo real. Segundo, no hay un camino para volver a reutilizar parte de la relación, y el proceso es irreversible una vez que la relación es mapeada fuera, es imposible desmontarlo en componentes significativos.

2.2.- SQL (Lenguaje de Consulta Estructurado)

SQL (Structured Query Language), es una herramienta para organizar, gestionar y recuperar datos almacenados en una Base de Datos, es decir permite interactuar con la Base de Datos, y específicamente con un tipo específico llamado **Base de Datos Relacional.g**

Para ello dispone de los siguientes sublenguajes:

- ☑ **DDL** (Data Definition Language) Lenguaje de descripción de datos.
- ☑ **DML** (Data Manipulation Language) Lenguaje de manipulación de datos
- ☑ **DCL** (Data Control Language) Lenguaje de control de datos.

Y trabaja con las siguientes estructuras básicas:

- ☑ **Base de datos:** Compuesta de un conjunto de tablas del sistema, creadas implícitamente por el mismo sistema, y por un conjunto de tablas y vistas creadas por el usuario.
- ☑ **Tablas:** El universo está descrito con la ayuda de tablas, cada una representa a una entidad o a una asociación entre entidades. Las tablas están compuestas de columnas o de líneas o filas llamadas convencionalmente **campos** y **registros**. Una columna representa un atributo de la entidad, y para describirla es necesario especificar un nombre y un tipo de dato, una particularidad de las columnas es que pueden permitir o no valores nulos.
- ☑ **Fila:** Combinación de los diferentes atributos del objeto (**registro**).
- ☑ **Vistas:** Tabla virtual, definida sobre las tablas base descritas por el usuario, su objeto es permitir a los usuarios manipular un subconjunto de datos.
- ☑ **Usuarios:** Uno de los objetivos de un DBMS, es garantizar la seguridad de los datos, para hacerlo, el lenguaje de programación genera todos los usuarios y sus permisos de acceso para acceder a una base de datos, cada usuario debe disponer de su autorización correspondiente, y es el propietario de la base de datos DBA (DataBase Administrator, Administrador de la Base de Datos) quien debe otorgar y retirar permisos de acceso.
- ☑ **Índices:** Se utilizan para aumentar el rendimiento del sistema y asegurar la integridad de los datos. Para cada tabla que se crea, se asocia automáticamente una tabla índice, que contiene la posición del registro según la columna especificada como índice.

2.3.- Modelo Entidad - Relación

Los modelos son la piedra angular del diseño. Los ingenieros construyen un modelo de un carro o de un edificio (y sus detalles) antes de ponerlo en producción,

de la misma manera, los diseñadores del sistema desarrollan modelos para explorar ideas y mejorar la comprensión de la base de datos.

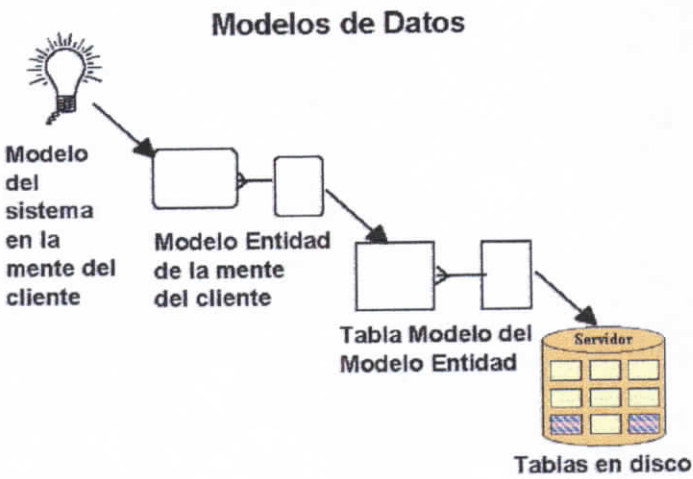


Fig. 2.2. Modelos de Datos

Los modelos ayudan a comunicar los conceptos en la mente de la gente, y pueden ser usados para los siguientes propósitos: Comunicación, Categorización, Descripción, Especificación, Investigación, Desarrollo, Análisis, Imitación.

El objetivo es producir un modelo, que se ajuste a varios de estos usos, puede ser comprendido por cualquier usuario final, pero contiene suficiente detalle para construir un sistema de base de datos.

Los modelos entidad-relación son derivados de las especificaciones o narrativas del negocio, conocidas como **Reglas del Negocio**. Este modelo representa gráficamente las necesidades y reglas de información del negocio.

Los modelos entidad-relación separan la información requerida por un negocio de las actividades realizadas dentro del mismo. Sin embargo los negocios pueden cambiar sus actividades, el tipo de información tiende a permanecer constante, por tanto las estructuras de datos también tienden a ser constantes.

Modelo Entidad-Relación

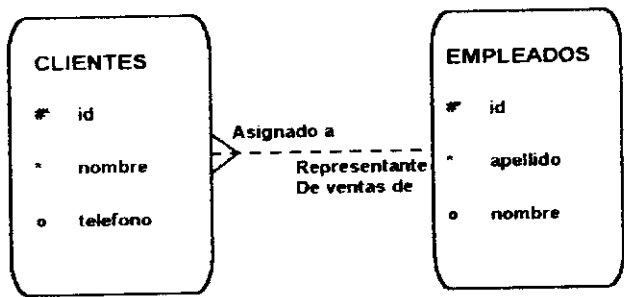


Fig. 2.3. Modelo Entidad-Relación

Los beneficios que se obtienen del Modelo Entidad-Relación son:

- ✓ Documentar los requerimientos de información para la organización, en un formato claro y preciso.
- ✓ Proveer un mapa gráfico, fácil de entender, para el diseño de la base de datos.
- ✓ Desarrollar y refinar el modelo fácilmente
- ✓ Proveer un cuadro claro del alcance de los requerimientos de información.
- ✓ Ofrecer un cuadro de trabajo efectivo, para integrar múltiples aplicaciones, proyectos de desarrollo y software de aplicación comprado.

Los componentes del Modelo Entidad-Relación son:

Componente	Descripción
Entidad	Una cosa que tiene significado, acerca de la cuál se necesita conocer información
Atributo	Alguna cosa que describe o cualifica una entidad
Relación	Una asociación con nombre entre entidades, que muestran opcionabilidad o grado

Tabla 2.1. Componentes del Modelo Entidad-Relación

El Modelo Entidad-Relación está compuesto de entidades, atributos y relaciones.

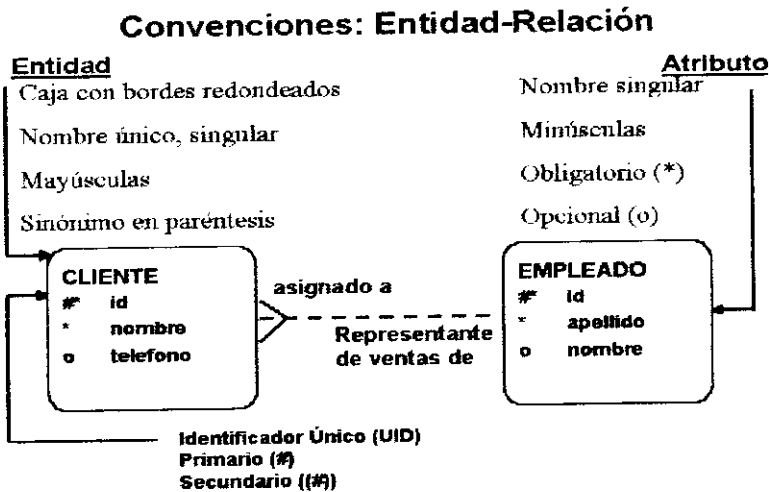


Fig. 2.4. Convenciones del Modelo Entidad Relación

Entidades.- Una entidad representa una cosa que tiene significado, acerca del sistema del negocio, o una categoría discreta o colección de datos relacionados. Ejemplos: cliente, orden y empleado.

Para representar una entidad en un modelo, las convenciones son:

- ☒ Cajas redondeadas de cualquier dimensión
- ☒ Nombre de la entidad único, singular y en mayúsculas
- ☒ Nombre sinónimo opcional, en mayúsculas entre paréntesis

Atributos.- Los atributos describen las entidades y mantienen la información específica que debe ser conocida. Por ejemplo, para la entidad **cliente** los atributos pueden ser: **cédula, nombre, teléfono y dirección.**

Si una entidad no tiene atributos que necesiten ser conocidos, desde el punto de vista del negocio, entonces no está dentro del alcance de los requerimientos del sistema, y no deberían aparecer en el modelo.

Cada uno de los atributos puede ser requerido u opcional. Para representar una entidad en el modelo, las convenciones son:

- ☒ Usar nombres en singular y en minúsculas

- ☑ Marcar atributos obligatorios, o valores que deben ser conocidos con un asterisco (*)
- ☑ Marcar atributos opcionales, o valores que pueden ser conocidos con “o”.

Identificadores únicos.- Un identificador único (UID), es cualquier combinación de atributos o relaciones, o ambas, que sirven para distinguir ocurrencias de una entidad. Cada ocurrencia de entidad debe ser identificable en forma única.

- ☑ Marcar cada atributo que es parte de UID con un símbolo de número #
- ☑ Marcar UID secundarios con un signo de números entre paréntesis (#)

Relaciones.- Cada entidad debe tener una relación, que representa los requerimientos de información y reglas del negocio. La relación es una asociación en dos direcciones entre dos entidades, o entre una entidad y ella misma. Cuando una entidad tiene una relación con sí misma, se dice que es una **relación recursiva**.

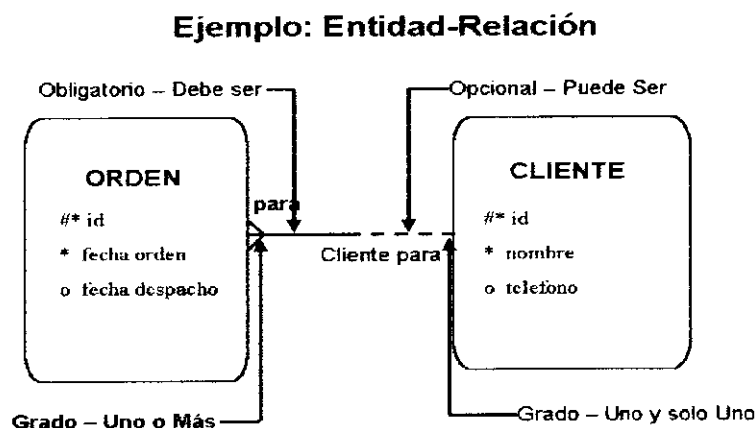


Fig. 2.5. Ejemplo de Modelo Entidad-Relación

Cada dirección de una relación contiene:

- ☑ Un nombre, por ejemplo: **enseña a**, o **asignado a**
- ☑ Opcionalmente, **debe ser** o **puede ser**
- ☑ Un grado, **uno y solo uno** o **uno o más**.

Cardinalidad es sinónimo de grado.

Entidad fuente [**puede ser** | **debe ser**] nombre de relación [**Uno o solo Uno** | **Uno o Más**] entidad destino.

Las convenciones utilizadas son:

Símbolo	Descripción
Línea entrecortada	Elemento opcional, indica “puede ser”
Línea continua	Elemento obligatorio, indica “debe ser”
Línea Angular	Elemento de grado, indica “Uno o Más”
Línea Simple	Elemento de grado, indica “Uno y solo Uno”

Tabla 2.2.Convenciones para Identificar Relaciones

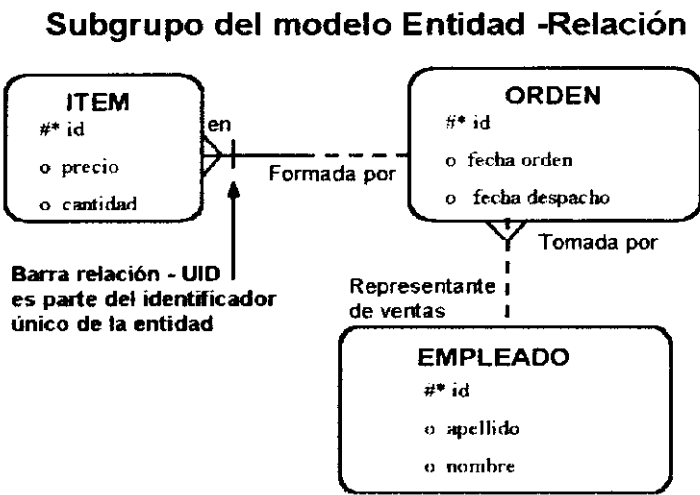


Fig. 2.6. Subgrupo del Modelo Entidad-Relación

Una entidad puede ser identificada en forma única a través de una relación. Utilizar una barra UID para indicar que una relación es parte del identificador único de la entidad. La relación contenida en un UID debe ser obligatoria y, uno y solo uno en la dirección que participa en el UID, por ejemplo, cuando se ordenan ítems, se tiene un número de orden e ítem con un número de ítem de línea única. Pero cuando otra orden es colocada, ese número de ítem ya no es único. Por tanto, el ítem es identificado en forma única por su atributo número y el número de orden específico al cuál el ítem está relacionado.

Número de orden	Número ítem	Número de producto
100	1	209
100	2	399
100	3	876

101	1	630
101	2	297

Tabla 2.3. Ejemplo de Relaciones

Una relación entre una entidad y ella misma se llama **relación recursiva**, y es representada por una línea circular.



Fig. 2.7. Relaciones Recursivas

Los tipos de relaciones son:

Tipo	Descripción
Uno-a-Uno	Grado de uno y solo uno en ambas direcciones, estos tipos son raros, y pueden realmente ser la misma entidad, o un atributo de la entidad.
Muchos-a-Uno	Grado de uno o más en una dirección y un grado de uno y solo uno en la otra dirección, es muy común.
Muchos-a-Muchos	Grado de uno o más en ambas direcciones, es muy común, y se puede resolver con una entidad de intersección.

Tabla 2.4. Tipos de Relaciones

Antes de realizar el diseño de la base de datos, se deben eliminar los problemas de la redundancia de datos, normalizando el modelo de datos. Modificar el modelo de datos para soportar diferentes requerimientos funcionales y alternar diseños de bases de datos por normalización del almacenamiento de datos antes de crear la base de datos. Los beneficios de la normalización son:

- ☑ Minimizar la redundancia de datos
- ☑ Reducir los problemas de integridad
- ☑ Identificar entidades perdidas, relaciones y tablas

Para desarrollar el proceso de normalización, se tienen las siguientes reglas:

Regla	Descripción
Primera forma normal (1FN)	Todos los atributos deben ser valores simples y no repetidos (atómicos).
Segunda forma normal (2FN)	Un atributo debe depender en el identificador único de la entidad completa
Tercera forma normal (3FN)	Un atributo no UID no puede depender de otro atributo no UID.

Tabla 2.5. Regla de Normalización

Las restricciones de Integridad y Llaves, aseguran que los usuarios desarrollen solo operaciones que dejan a la base de datos en un estado correcto y consistente, para dar fuerza a las restricciones de integridad de datos. Todas las restricciones de integridad de datos deben ser forzadas por el servidor de base de datos o la aplicación. Las llaves corresponden a las restricciones de integridad. Los tres tipos de llaves son llaves primarias, llaves únicas y llaves foráneas.

Los tipos de restricciones son:

Tipo de Restricción de Integridad	Descripción
Entidad	Parte de una llave primaria no puede ser NULL, y el valor debe ser único.
Referencial	Los valores de llave foránea deben hacer juego con una llave primaria o ser NULL.
Columna	Los valores en la columna, deben hacer juego con los tipos de datos definidos.
Definida de Usuario	Los valores deben cumplir con las reglas del negocio.

Tabla 2.6. Tipos de Restricciones

Ejemplos de restricción de integridad de datos definidos por el usuario:

- ☑ Un empleado en el departamento de finanzas no puede tener un título de programador.
- ☑ La comisión de una persona de ventas no puede exceder el 50% del salario base.

PASOS DEL DISEÑO DE BASES DE DATOS

A partir del Modelo Entidad-Relación crear una Carta de Instancia de Tabla

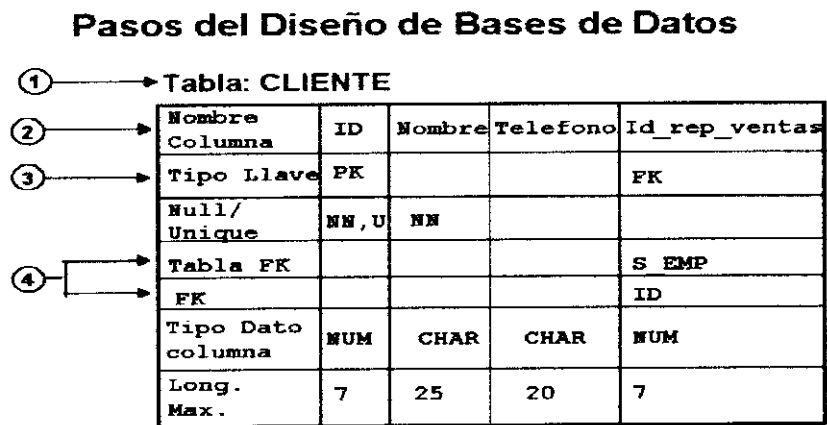


Fig. 2.8. Pasos del Diseño de Bases de Datos

1.- Entidad simple a tabla

La Carta de Instancia de Tabla, debe contener las columnas para el nombre de la tabla, los nombres de las columnas, los tipos de llaves, referencias nulas y únicas, información de llave foránea, tipo de dato de columna y longitud máxima, y datos de ejemplo. Registrar el nombre de la tabla.

Atributos a columnas y documento de datos, ejemplo. Etiquetar claramente los nombres de columna y sus tipos de datos genéricos asociados, por ejemplo, caracter, número o fecha.

Cada atributo en el modelo entidad relación, a un nombre de columna en la tabla, adicionar ejemplos de datos en la carta, para mostrar el contenido de la tabla en una forma visual. Marcar los atributos obligatorios con un asterisco, a columnas definidas como NOT NULL (NN).

Evitar nombrar las columnas con palabras reservadas, como **NUMBER**, los nombres de columna con abreviaciones consistentes como **NO** o **NUM** (pero no ambas) para evitar confusiones del programador y el usuario.

Identificadores únicos a llaves primarias, asegurando incluir cualquier componente de llave foránea de la llave primaria.

Los UID identificados en el modelo entidad relación, por un símbolo de número (#) para las columnas de llave primaria y etiquetarlo como **PK**. Los UID que contienen un atributo simple a una columna simple **PK**. Etiquetar las referencias no nulo y única como **NN** y **U**. Los UID que incluyen atributos múltiples a una **PK** compuesta, etiquetarlas como **NN** y **U1**. Si se tiene una llave primaria alterna, etiquetarla como **NN** y **U**, pero escoger solo una **PK**.

Si el UID de la entidad incluye la relación (indicada por la barra **UID**), adicionar una columna **FK** para cada relación y etiquetarla como tipo **PK** y **FK**. Adicionar columnas **FK** al final de la carta de instancia de tabla o a la derecha de todas las columnas, aún si **FK** es parte de **PK**. Escoger un nombre único para cada columna **FK** y adicionar un dato de ejemplo.

2.- Relaciones a llaves foráneas:

El último paso es cambiar las relaciones a llaves foráneas, considerando dos tipos de relaciones: **Muchos-a-Uno**, si la relación es parte del identificador único, ya se la tiene marcada. Asegurar que se etiqueten las relaciones obligatorias con la referencia **NN**. Escoger un nombre único para la columna de llave foránea.

- ☒ **Relaciones Muchos-a-Uno.-** Toma la llave primaria del extremo **Uno** y se la coloca en la tabla en el extremo **Muchos** como una llave foránea, técnica aplicable a las relaciones recursivas.
- ☒ **Relación opcional Uno-a-Uno.-** Se puede colocar la llave foránea en la tabla a cualquier lado de la relación, asegurando colocar la referencia **U**.
- ☒ **Relación Uno-a-Uno Obligatoria.-** Colocar la llave foránea única en la tabla en el fin obligatorio y etiquetarla como **NN** para reforzar el requerimiento obligatorio y **U** para reforzar la relación **Uno-a-Uno**.

Se deben considerar los siguientes requerimientos adicionales:

- ☒ Diseñar los índices, que son objetos de la base de datos que proveen acceso directo y rápido a las filas. Se puede crear índices para llaves alternas, llaves foráneas y columnas frecuentemente usadas en las condiciones de búsqueda.

- ☑ Establecer definiciones de vistas, que son tablas lógicas basadas en una o más tablas o vistas. Las vistas pueden restringir el acceso, proveer una mejor presentación de la información, y pueden contener una consulta compleja.
- ☑ Planificar el espacio de almacenamiento físico, que es la cantidad de espacio requerido para almacenar los datos de una tabla en la base de datos.
- ☑ Redefinir las restricciones de integridad.

Los símbolos utilizados para documentar la Carta de Instancia de Tabla son:

Símbolo	Definición
PK	Columna Llave Primaria
FK	Columna Llave Foránea
FK1, FK2	Dos llaves foráneas dentro de la misma tabla
FK1, FK1	Dos columnas dentro de la misma llave foránea compuesta
NN	Columna NO NULA
U	Columna única
U1, U1	Dos columnas que combinadas son únicas

Tabla 2.7. Símbolos para la Carta de Instancia de Tabla

2.4.- Manipulación de Datos

2.4.1.- Consultas

La consulta de una Base de Datos, se realiza a través de la instrucción **SELECT**, y existen dos tipos de consultas diferenciadas.

Consulta no Calificada

- ☑ Cuando se quiere visualizar algunas columnas en concreto

SELECT lista de columnas

FROM lista de tablas;

Ejemplo:

SELECT idcliente, nombre, ape1, ape2

FROM clientes;

- ☒ Cuando se quiere visualizar todas las columnas de una tabla única:

```
SELECT *  
FROM clientes;
```

- ☒ Cuando se quiere visualizar datos de dos o más tablas, que no tienen columnas que tengan el mismo nombre:

```
SELECT nombre, ape1, ape2, idarticulo, descripción  
FROM clientes, artículos;
```

- ☒ Cuando se quiere visualizar datos de dos o más tablas, que tienen columnas que tengan el mismo nombre:

```
SELECT clientes.nombre, clientes.ape1, clientes.ape2,  
proveedores.nombre, proveedores.ape1, proveedores.ape2  
FROM clientes, proveedores;
```

Consulta Calificada

Se hace a partir de una consulta simple y una cláusula **WHERE**, ésta puede estar seguida por una o varias condiciones, que a su vez están relacionadas entre sí por los operadores lógicos **AND**, **OR** y **NOT**.

Existen 3 tipos de condiciones:

- ☒ Condiciones de comparación.
- ☒ Condiciones de combinación
- ☒ Condiciones de subsistencia.

Las condiciones permiten comparar una columna o una expresión con otra columna expresión o lista de columnas, con el siguiente formato:

```
SELECT nombre-columna(s)  
FROM lista de tablas  
WHERE condición(es);
```

Los formatos generales de las condiciones son:

- ☒ Expresión Operador relacional Expresión
- ☒ Expresión [NOT] BETWEEN Expresión AND Expresión
- ☒ Expresión [NOT] IN (lista de valores)
- ☒ Nombre-columna [NOT] LIKE "cadena de caracteres"
- ☒ Nombre-columna [NOT] MATCHES "cadena de caracteres"
- ☒ Nombre-columna IS [NOT] NULL

Donde:

Expresión: es un nombre de columna, una constante o una combinación de las dos, relacionadas por los operadores aritméticos (multiplicación, división, suma, resta):

* / + -

Una expresión puede incluir funciones predefinidas:

SYSDATE: Fecha del sistema.

USER: nombre del usuario.

TODAY: fecha actual

Ejemplo:

```
SELECT *  
FROM pedidos  
WHERE fecha_pedido = TODAY;
```

Además, se puede utilizar los operadores de relación (igual, diferente, mayor que, menor que, mayor o igual que, menor o igual que):

= <> > < >= <=

Ejemplo:

```
SELECT *  
FROM pedido  
WHERE cantidad > 100 AND fecha_pedido = TODAY;
```

Con el predicado **BETWEEN**, se puede comparar el valor de la expresión situada a la izquierda de la palabra **BETWEEN**, con los valores comprendidos en el intervalo definido por las expresiones a la izquierda y derecha de la palabra clave **AND**.

Ejemplo: se desea consultar todas las columnas de la tabla artículos, donde el precio unitario sea mayor o igual que 100 y menor o igual que 300.

```
SELECT *  
FROM artículos  
WHERE precio_unitario >=100 AND precio_unitario <=300;
```

Con el predicado **IN**, se puede comparar el valor de la expresión, situado a la izquierda de la palabra **IN**, con la lista de valores encerrados entre paréntesis situados a la derecha de la misma. Ejemplos:

```
SELECT *  
  FROM clientes  
 WHERE ciudad = "Ambato"  
    OR ciudad = "Quito"  
    OR ciudad = "Guayaquil";
```

Se puede escribir como:

```
SELECT *  
  FROM clientes  
 WHERE ciudad IN ("Ambato","Quito", "Guayaquil");
```

El predicado **LIKE** permite utilizar una comparación de semejanza, entre el valor de una columna y el de una cadena de caracteres, utilizando los caracteres genéricos de sustitución (%, _). Ejemplo:

```
SELECT *  
  FROM clientes  
 WHERE ape1 LIKE "A%"
```

%: sustituye a varios caracteres.

_ : sustituye a un solo caracter.

```
SELECT *  
  FROM clientes  
 WHERE ape1 LIKE "_T%";
```

El predicado **MATCHES**, es un complemento al predicado **LIKE**, y ofrece los siguientes caracteres de sustitución (*, ?, %, _). Ejemplos:

```
SELECT *  
  FROM clientes  
 WHERE ape1 LIKE "A%" OR ape1 LIKE "B%" OR ape1 LIKE "C%";
```

```
SELECT *  
  FROM clientes  
 WHERE ape1 MATCHES "[A-C]*";
```

El predicado **IS [NOT] NULL**, permite una comparación sobre valores nulos.

Ejemplo:

```
SELECT nombre, ciudad  
  FROM clientes  
 WHERE ciudad IS NOT NULL;
```

Ordenación de Resultados

Los resultados obtenidos de una consulta SQL, se los puede ordenar hasta por 8 columnas diferentes en una misma instrucción, con el formato:

```
ORDER BY [ASC/DESC] lista de columnas
```

Ejemplo:

```
SELECT *
  FROM clientes
 WHERE ciudad IS NOT NULL.
 ORDER BY DESC apel;
```

En forma adicional, SQL ofrece cinco funciones básicas para realizar cálculos estadísticos, con las que se puede determinar el número de líneas por tabla o por grupo que cumple una condición, calcular la suma y la media de columnas numéricas o determinar el valor máximo o mínimo de una columna.

☒ **COUNT (* | DISTINCT | nombre columna)**

Para contar el número de líneas que satisface la sentencia:

```
SELECT COUNT(*)
  FROM articulo;
```

Para obtener las distintas ciudades que se tiene, si se tiene 2 con el mismo nombre, sólo se cuenta 1 al ser repetidas.

```
SELECT DISTINCT ciudad
  FROM clientes;
```

☒ **SUM ([DISTINCT] columna)**

Suma los valores de la columna que satisface la sentencia.

La opción DISTINCT suma los valores únicos de la columna.

La columna debe de ser de tipo numérico y puede ser una expresión. Por ejemplo, si la tabla artículos tiene la información:

<u>Idarticulos</u>	<u>Existencias</u>	<u>precio</u>
1	10	15
2	5	10
3	5	10

Para sumar la columna artículos:

```
SELECT SUM(existencias*precio)
  FROM articulos;
```

Para sumar los valores únicos de todos los registros en todas las existencias.

```
SELECT SUM(DISTINCT existencias)
  FROM articulos;
```

☒ **AVG ([DISTINCT] columna)**

Calcula la media de los valores de la columna que satisfacen la sentencia.

La opción DISTINCT calcula la media de valores únicos.

Para calcular el precio de los artículos con precio único.

```
SELECT AVG(DISTINCT precio)
  FROM articulos;
```

☒ **MAX (columna)**

Visualiza el máximo de los valores de la columna, que satisfacen la sentencia.

La columna debe de ser de tipo numérico y puede ser una expresión.

Para calcular el precio máximo que se cobra de los artículos.

```
SELECT MAX(precio)
FROM articulos;
```

☒ **MIN (columna)**

Visualiza el mínimo de los valores de la columna que satisfacen la sentencia.

La columna debe de ser una expresión y tiene que ser numérica.

Para calcular el precio mínimo que cobra por los artículos

```
SELECT MIN(precio)
FROM articulos;
```

Cuando existen valores nulos, la función **COUNT** toma en cuenta estas líneas, las demás funciones NO.

2.4.2.- Insertar Registros

Para agregar un registro en una tabla, se utiliza la sentencia INSERT, con el formato:

```
INSERT INTO nombre_tabla [(campo1, campo2, ..., campoN)]
VALUES (valor1, valor2, ..., valorN);
```

Se pueden insertar registros de dos formas: Insertar registros únicos ó Insertar registros contenidos en otra tabla.

2.4.2.1.- Insertar registros únicos:

Se graba en el **campo1** el **valor1**, en el **campo2** el **valor2**, y así sucesivamente. Si no se especifican los campos, se deben ubicar los valores para todos los campos de la tabla, en el orden en que se encuentran.

2.4.2.2.- Insertar Registros de otra Tabla:

Para insertar registros obtenidos desde otra tabla, la sintaxis es:

```
INSERT INTO nombre_tabla (campo1, campo2, ..., campoN)
SELECT TablaOrigen.campo1, TablaOrigen.campo2, ..., TablaOrigen.campoN
FROM TablaOrigen;
```

Se seleccionan los campos **1, 2, ..., n** de la tabla origen y se graban en los campos **1, 2, ..., n** de nombre_tabla. La sentencia **SELECT** puede incluir la cláusula **WHERE** para filtrar los registros a copiar. Si nombre_tabla y TablaOrigen poseen la misma estructura la sintaxis se puede simplificar como:

```
INSERT INTO nombre_tabla
SELECT TablaOrigen.* FROM TablaOrigen;
```

Con lo que, todos los campos de **TablaOrigen** se graban en **nombre_tabla**, para esto es necesario que todos los campos de **TablaOrigen** estén contenidos con igual nombre en **nombre_tabla**, dicho de otra forma, **nombre_tabla** debe tener todos los campos de **TablaOrigen** (con igual nombre e igual tipo).

En este tipo de consulta hay que tener especial atención con los campos contadores o auto numéricos, puesto que, al insertar un valor en un campo de este tipo se escribe el valor que contenga su campo homólogo en la tabla origen, no incrementándose como corresponde.

Se puede utilizar la sentencia **INSERT INTO** para agregar un registro único a una tabla, utilizando la sintaxis de la consulta de adición de registro único, tal y como se mostró anteriormente. En este caso, el código especifica: el nombre y el valor de cada campo del registro. Se debe especificar cada uno de los campos del registro al que se va a asignar un valor así como el valor para dicho campo. Cuando no se especifica dicho campo, se inserta el valor predeterminado o **Null**. Los registros se agregan al final de la tabla.

También se puede utilizar **INSERT INTO** para agregar un conjunto de registros pertenecientes a otra tabla o consulta utilizando la cláusula **WHERE** en la

Col1, col2.- nombres de los campos que se desea cambiar, en una misma sentencia **UPDATE** se pueden actualizar varios campos de cada registro en la tabla.

Expr1, expr2.- valores que se asignan a las columnas, pueden ser constantes o expresiones.

La cláusula **WHERE** tiene el mismo formato que la sentencia **SELECT** y determina los registros que serán modificados, en el caso de no encontrarse una cláusula **WHERE**, toda la tabla será actualizada. Ejemplos:

- ☒ Incrementar un 10% al precio del producto cuyo código es '254' en la tabla artículo:

```
UPDATE articulo
SET precio = precio * 1.1
WHERE codigo = '254';
```

- ☒ Fijar la fecha de matrícula a los alumnos que no la tienen, con la fecha actual del sistema:

```
UPDATE matricula
SET fecha_matricula = TODAY()
WHERE fecha_matricula IS NULL;
```

2.4.4.- Eliminar o Borrar Registros

Para borrar registros de una tabla de la base de datos, se utiliza la sentencia **DELETE**, con el formato:

```
DELETE FROM nombre_tabla
[WHERE condición];
```

La cláusula **WHERE** tiene el mismo formato que en la sentencia **UPDATE** y determina cuáles registros serán borrados, sin no se especifica una cláusula **WHERE**, se borran todos los registros. Ejemplos:

- ☒ Borrar el registro del artículo cuyo código es '369':

```
DELETE FROM articulo
WHERE codigo = '369';
```

2.4.5.- Subconsultas

Las subconsultas:

- ✓ Sirven para realizar una consulta sobre la base de un criterio desconocido.
- ✓ Se utilizan en sentencias de manipulación de datos.
- ✓ Sirven para ordenar datos.

Una subconsulta es una sentencia **SELECT** embebida en una cláusula de otra sentencia **SQL**.

El formato general es:

```
SELECT lista_de_selección
FROM tabla
WHERE expresión operador
(SELECT lista_de_selección
FROM tabla);
```

La subconsulta se ejecuta antes de la consulta principal, y el resultado es usado por la consulta más externa.

Recomendaciones:

- ✓ Una subconsulta debe estar encerrada entre paréntesis.
- ✓ Dos clases de operadores de comparación son usados en las subconsultas, los de simple fila y los de múltiple fila.
- ✓ Una subconsulta debe aparecer a la derecha del operador.
- ✓ Las subconsultas pueden ser usadas en muchos comandos **SQL**.
- ✓ Las subconsultas no pueden contener una cláusula **ORDER BY**

Ejemplos:

- ✓ Visualizar los empleados de tabla emp:

```
SELECT *
FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30

7876	ADAMS	CLERK	7788	23-MAY-87	1100	20
7900	JAMES	CLERK	7698	03-DEC-81	950	30
7902	FORD	ANALYST	7566	03-DEC-81	3000	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	10

- ☒
 Listar los empleados que tengan el mismo cargo que SMITH

```

SELECT ename "NOMBRE", JOB "CARGO"
      FROM emp
     WHERE job =
           (SELECT job
            FROM emp
            WHERE ename = 'SMITH');
    
```

NOMBRE	CARGO
SMITH	CLERK
ADAMS	CLERK
JAMES	CLERK
MILLER	CLERK

- ☒
 Listar los empleados que trabajan en el mismo departamento que ADAMS

```

SELECT ename "EMPLEADO", job "CARGO"
      FROM emp
     WHERE deptno =
           (SELECT deptno
            FROM emp
            WHERE ename = 'ADAMS');
    
```

EMPLEADO	CARGO
SMITH	CLERK
JONES	MANAGER
SCOTT	ANALYST
ADAMS	CLERK
FORD	ANALYST

Para usar las funciones de grupo dentro de una subconsulta:

- ☒
 Visualizar el nombre, cargo y salario de todos los empleados que ganan menos que el promedio general

```

SELECT ename "EMPLEADO", job "CARGO", sal "SUELDO"
      FROM emp
     WHERE sal <
           (SELECT AVG(sal)
            FROM emp);
    
```

EMPLEADO	CARGO	SUELDO
SMITH	CLERK	800
ALLEN	SALESMAN	1600
WARD	SALESMAN	1250
MARTIN	SALESMAN	1250
TURNER	SALESMAN	1500

ADAMS	CLERK	1100
JAMES	CLERK	950
MILLER	CLERK	1300

Se generan errores cuando una subconsulta retorna mas de una fila, y se usa un operador de comparación de simple fila:

```
SELECT ename, job, sal
FROM emp
WHERE deptno =
      (SELECT deptno
FROM dept
WHERE deptno = 10
OR dname = 'SALES');
ERROR:
ORA-01427: single-row subquery returns more than one row
```

Para corregir el error, se debe cambiar el operador de fila simple por uno de fila múltiple como **IN**:

```
SELECT ename, job, sal
FROM emp
WHERE deptno IN
      (SELECT deptno
FROM dept
WHERE deptno = 10
OR dname = 'SALES');
```

ENAME	JOB	SAL
-----	-----	-----
ALLEN	SALESMAN	1600
WARD	SALESMAN	1250
MARTIN	SALESMAN	1250
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
KING	PRESIDENT	5000
TURNER	SALESMAN	1500
JAMES	CLERK	950
MILLER	CLERK	1300

Se puede usar subconsultas en la cláusula HAVING:

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
HAVING AVG(sal) >
      (SELECT AVG(sal)
FROM emp
WHERE deptno = 20);
```

DEPTNO	AVG(SAL)
-----	-----
10	2916.6667

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
```

Sentencias que forman el bloque.

EXCEPTION

Todos los manejadores de excepciones.

END;

Marca de fin de bloque.

Ejemplo:

DECLARE

COSTO_TEMP NUMBER(10,2);

BEGIN

SELECT costo FROM articulo INTO costo_temp WHERE codigo = '234';

IF costo_temp > 0 THEN

UPDATE articulo SET costo = (costo_temp*1.1) WHERE codigo = '234';

ELSE

UPDATE articulo SET costo = 21.32 WHERE codigo = '234';

END IF;

COMMIT;

EXCEPTION

WHEN NO_DATA_FOUND THEN

INSERT INTO errores (codigo, mensaje) VALUES(99, 'codigo 234 no existe');

END;

Se pueden observar varios elementos nuevos, no cubiertos en **SQL**, que son las extensiones de **PL/SQL**:

- ☑ **Variables y Constantes.**- objetos que son usados para almacenar y manipular datos en el ámbito de bloque. Pueden ser de los tipos de datos: **CHAR, VARCHAR2, NUMBER, DATE o BOOLEAN.**
- ☑ **Soporte SQL.**- todas las sentencias **SQL** son soportadas dentro de bloques **PL/SQL**, incluyendo sentencias de control de transacciones.
- ☑ **Tipos de Datos Compuestos.**- los registros permiten que grupos de campos sean definidos y manipulados en bloques **PL/SQL**.
- ☑ **Control de Flujo.**- condiciones, ciclos, saltos y etiquetas proveen acciones condicionales, pruebas, saltos y control del programa interactivo.
- ☑ **Funciones pre construidas.**- La mayoría de funciones de datos **SQL** son soportadas dentro de bloques **PL/SQL**.
- ☑ **Manejo de Cursor.**- cursores (un área de memoria que mantiene un grupo resultante), pueden ser explícitamente definidos y manipulados permitiendo el procesamiento de múltiples filas. Un grupo de atributos del

sistema **PL/SQL** proveen la habilidad para verificar el estado interno del cursor.

- ☑ **Manejo de Excepciones.**- los bloques tienen la habilidad para atrapar y manejar condiciones de error locales (excepciones implícitas). Se puede también generar excepciones explícitas, que tratan con errores lógicos y de datos.
- ☑ **Almacenamiento de Código.**- los bloques pueden ser almacenados dentro de una base de datos Oracle como: procedimientos, funciones, paquetes (un grupo de bloques) y disparadores.

2.5.1.- Reglas para Bloques

Cada unidad de **PL/SQL** puede constituir un bloque, y como mínimo, las sentencias ejecutables, deben estar limitadas por las palabras **BEGIN** y **END**.

Las sentencias **SELECT** dentro de bloques **PL/SQL** son **SQL** incrustadas (una categoría **ANSI**), y como tal, deben retornar una sola fila. Las sentencias **SELECT** que no retornan filas o retornan más de una fila, generan un error. Si se quiere tratar con grupos de filas, se deben colocar los datos retornados en un cursor. La cláusula **INTO** es obligatoria para las sentencias **SELECT** dentro de bloques **PL/SQL** (las cuales no están dentro de la definición de un cursor), se podría almacenar los valores retornados desde un **SELECT**.

Si variables u objetos **PL/SQL** son definidos para uso dentro de un bloque, entonces se debe también tener una sección **DECLARE**.

Si se incluye una sección **EXCEPTION** las sentencias dentro de ésta, sólo son procesadas si ocurre la condición a la cuál se refieren. La ejecución del bloque es terminada luego que una rutina de manejo de excepciones se ejecuta.

Los bloques **PL/SQL** pueden ser anidados, y ocurren dondequiera que una sentencia ejecutable pueda ser colocada (incluyendo la sección **EXCEPTION**).

2.5.2.- Alcance de Objetos de Bloque

El alcance de un objeto define donde el objeto es visible, esto es, el área de programa lógico que puede legalmente referirse a un objeto dado.

El alcance de una variable u objeto (cursor, constante, etc.) es determinado por el bloque donde está declarado. Es solo disponible hasta que el bloque donde está definido finaliza su ejecución. Se debe recordar que las variables de bloque son definidas bajo la palabra clave **DECLARE**.

Para bloques anidados, un objeto definido en un bloque padre está disponible dentro de todos sus hijos (bloques anidados). Lo contrario no es válido, los objetos definidos en un bloque hijo no son visibles para el padre.

Si un bloque anidado define un objeto con el mismo nombre que un objeto en su bloque padre, entonces solo el objeto local es visible.

2.5.3.- Declarar Variables y Constantes

PL/SQL soporta una variedad de tipos de datos que se pueden usar para declarar variables y constantes. Se puede asignar valores a las variables al declararlas, y cambiar el valor de una variable a través de asignaciones posteriores.

Se puede asignar el valor de una constante mientras se declara, este valor es fijo y no puede ser cambiado durante la ejecución, ejemplos:

Declaración de variables:

```
DECLARE
  V_NUM1 NUMBER NOT NULL := 10109;
  NUM8 NUMBER(3,1);
  XYZ NUMBER(2,2) := 31.8;
  ABC12 NUMBER(9,2) := XYZ * 131;
  V_CHR1 CHAR(89);
  V_CHR2 VARCHAR2(12) := "ENERO";
  Fecha_actual := SYSDATE;
```

Verd_Falso BOOLEAN;

Declaración de constantes:

```
DECLARE
    PI CONSTANT NUMBER(9,3) := 3.142;
    VAT CONSTANT NUMBER(4,2) := 17.5;
```

Cuando se declara variables **PL/SQL** para mantener valores de columnas, se debe asegurar que el tipo de variable es el mismo que el tipo de la columna, caso contrario se obtiene un error en tiempo de ejecución. Se puede usar el atributo **%TYPE** para definir una variable con la misma definición de la columna, como está definida en el diccionario de datos Oracle. El atributo es prefijado con el nombre de esquema, tabla y columna, y es usado donde se requiere el tipo de dato en la sección **DECLARE**. **PL/SQL** determina el tipo de dato y tamaño de la variable cuando se accede al bloque, y es siempre compatible con la columna usada. Ejemplo:

```
DECLARE
    V_NUM1 articulo.costo%TYPE;
```

2.5.4.- Registros PL/SQL

Un registro **PL/SQL** es una variable que contiene un grupo de campos separados. Cada campo es individualmente direccionable. Se puede referenciar los nombres de campo tanto en asignaciones como en expresiones. Los campos dentro de un registro puede tener diferentes tipos de datos y tamaños, similar a las columnas de una tabla de la base de datos. Los registros son una forma conveniente para almacenar una fila completa de una tabla de la base de datos.

Utilizar el atributo **%ROWTYPE** para declarar un registro que contenga un grupo de columnas de una tabla o vista de la base de datos. Los campos dentro del registro toman sus nombre y tipos de dato, desde las columnas de la tabla o vista.

Declarar el registro en la sección **DECLARE** junto con cualquier otra variable o constante requerida. Ejemplo:

```
DECLARE
    REC1 articulo%ROWTYPE;
    REC4 articulo%ROWTYPE;
```

Coloca **REC1** como un objeto registro, que mantiene los campos que hacen juego con las columnas de la tabla **ARTICULO**. No contiene valor alguno hasta que sea llenado.

Asignar valores en un registro **PL/SQL** nombrando el registro luego de la palabra clave **INTO** de una sentencia **SELECT**. La palabra clave **INTO** define la especificación del nombre para el área(s) de almacenamiento del valor(es) requeridos.

```
BEGIN
    SELECT * FROM articulo INTO REC1 WHERE codigo = '234';
END;
```

Se puede asignar todos los valores del registro de un registro a otro, que tenga las mismas definiciones. Ejemplo:

```
BEGIN
    REC4 := REC1;
END;
```

Para referenciar los valores de campo dentro de un registro **PL/SQL**:

```
BEGIN
    REC4 := REC1;
    IF REC4.costo > 0 THEN
        REC4.cantidad := 10;
    ELSE
        REC4.cantidad := 7;
    END IF;
END;
```

2.5.5.- Asignaciones PL/SQL

Se puede asignar valores a las variables en cualquier parte de un bloque **PL/SQL**, incluyendo en la creación de una variable. Ejemplos:

```
NUM1 := NUM1 + NUM2 + (NUM3 * 3);
ANUM3 := costo_prom * 7 + NVL(prom_costo_compra, 0);
SENT := 1;
STR1 := 'No restar hasta uno = ' || TO_CHAR(enviado);
CHRI := 'ABCDEFG';
FLAG1 := TRUE;
FLAG1 := FALSE;
NOMBRE := UPPER(titulo);
```


2.5.6.- Cursores

2.5.6.1.- Cursores Implícitos

Siempre que se emite una sentencia **SQL**, el servidor de la Base de Datos abre un área de memoria que se conoce como **cursor**, a donde el comando es pasado y ejecutado.

Cuando la parte ejecutable de un bloque **PL/SQL** emite un comando **SQL**, **PL/SQL** crea un cursor implícito que tiene un identificador **SQL**. **PL/SQL** maneja este cursor.

PL/SQL provee algunos atributos que permiten evaluar lo que ocurre cuando el cursor implícito fue usado la última vez. Se puede usar estos atributos en sentencias **PL/SQL** tales como funciones, pero no se los puede usar dentro de sentencias **SQL**.

Los atributos del cursor son:

- ☑ **%ROWCOUNT**.- número de filas procesadas por una sentencia **SQL**.
- ☑ **%FOUND**.- **TRUE** si al menos una fila fue procesada.
- ☑ **%NOTFOUND**.- **TRUE** si no hay filas procesadas.
- ☑ **%ISOPEN**.- **TRUE** si el cursor está abierto o **FALSE** si el cursor no ha sido abierto o ha sido cerrado. Sólo se usa con cursores explícitos. Ejemplo:

```
DECLARE
    ROW_DEL NO NUMBER;
BEGIN
    DELETE * FROM articulo;
    ROW_DEL NO := SQL%ROWCOUNT;
END;
```

2.5.6.2.- Cursores Explícitos

Las sentencias **SELECT** que ocurren dentro de bloques **PL/SQL** son conocidas como incrustadas, deben retornar solo una fila. Se puede definir una

sentencia **SELECT** con un cursor (un área de memoria), ejecutar la consulta y luego manipular las filas retornadas dentro del cursor. Los cursores son controlados a través de cuatro sentencias:

- ☑ **DECLARE.-** define el nombre y estructura del cursor, junto con la sentencia **SELECT** que llenará el cursor con datos. La consulta es validada pero no ejecutada.
- ☑ **OPEN.-** ejecuta la consulta que rellena el cursor con filas.
- ☑ **FETCH.-** carga la fila direccionada por el apuntador del cursor en las variables, y mueve el apuntador del cursor a la siguiente fila, listo para el siguiente **fetch**.
- ☑ **CLOSE.-** libera los datos dentro del cursor y cierra el cursor. El cursor puede ser reabierto para refrescar sus datos.

Los cursores son definidos dentro de la sección **DECLARE** de un bloque **PL/SQL**. Ejemplo:

```
DECLARE
    CURSOR mi_cursor IS SELECT codigo, costo FROM articulo;
```

El cursor es definido por la palabra clave **CURSOR**, seguida por el identificador del cursor (**mi_cursor**) y luego la sentencia **SELECT** usada para rellenarlo, que puede ser cualquier consulta válida.

Los cursores son abiertos con la sentencia **OPEN**, que rellena el cursor con datos.

```
BEGIN
    OPEN mi_cursor;
END;
```

Para acceder a las filas de datos dentro del cursor, se usa la sentencia **FETCH**.

```
DECLARE
    CURSOR mi_cursor IS SELECT codigo, costo FROM articulo;
    V_codigo NUMBER(10);
    V_costo NUMBER(10,2);
BEGIN
    OPEN mi_cursor;
    FETCH mi_cursor INTO v_codigo, v_costo;
END;
```

La sentencia **FETCH** lee los valores de columna, para la fila de cursor actual, y los coloca en las variables especificadas. El apuntador del cursor es actualizada para apuntar a la siguiente fila. Si el cursor no tiene más filas, las variables serán colocadas a **null** en el primer intento de **FETCH**, los siguientes intentos de **FETCH** alcanzan una excepción.

La sentencia **CLOSE** libera el cursor y cualquier fila dentro de él, se puede abrir el cursor de nuevo para refrescar los datos en él.

```
DECLARE
    CURSOR mi_cursor IS SELECT codigo, costo FROM articulo;
    V_codigo NUMBER(10);
    V_costo NUMBER(10,2);
BEGIN
    OPEN mi_cursor;
    FETCH mi_cursor INTO v_codigo, v_costo;
    CLOSE mi_cursor;
END;
```

Para procesar todas las filas dentro de un cursor, se necesita colocar la sentencia **FETCH** en un ciclo, y verificar el atributo del cursor **NOTFOUND** para ver si se ha alcanzado una fila o no.

```
DECLARE
    CURSOR mi_cursor IS SELECT codigo, costo FROM articulo;
    V_codigo NUMBER(10);
    V_costo NUMBER(10,2);
BEGIN
    OPEN mi_cursor;
    LOOP
        FETCH mi_cursor INTO v_codigo, v_costo;
        EXIT WHEN mi_cursor%NOTFOUND;
    END LOOP;
    CLOSE mi_cursor;
END;
```

Los registros **PL/SQL** pueden estar basados en un cursor, lo que es conveniente para procesar filas desde un grupo activo de datos. Ejemplo:

```
DECLARE
    CURSOR mi_cursor IS SELECT codigo, costo FROM articulo;
    Registro mi_cursor%ROWTYPE;
BEGIN
    OPEN mi_cursor;
    LOOP
        FETCH mi_cursor INTO registro;
        EXIT WHEN mi_cursor%NOTFOUND;
        IF registro.codigo = '234' THEN
            registro.costo = 19.10;
```

```

        END IF;
    END LOOP;
    CLOSE mi_cursor;
END;
```

Se puede usar la cláusula **WHERE CURRENT OF** para ejecutar comandos **DML** en la fila actual del cursor, lo que facilita la actualización de filas. Ejemplo:

```

DECLARE
    CURSOR mi_cursor IS SELECT codigo, costo FROM articulo;
    Registro mi_cursor%ROWTYPE;
BEGIN
    OPEN mi_cursor;
    LOOP
        FETCH mi_cursor INTO registro;
        EXIT WHEN mi_cursor%NOTFOUND;
        IF registro.codigo = '234' THEN
            DELETE FROM articulo WHERE CURRENT OF mi_cursor;
        END IF;
    END LOOP;
    CLOSE mi_cursor;
END;
```

Notar que no se requiere especificar explícitamente la fila que se quiere borrar, **PL/SQL** suministra el identificador de fila requerida desde el registro actual en el cursor, asegurando que solo la fila correcta sea borrada.

Es posible cambiar el grupo resultante retornado usando parámetros, los cuales permiten especificar el criterio de selección al abrir el cursor.

```

DECLARE
    CURSOR mi_cursor (param1 NUMBER) IS
        SELECT codigo, costo
            FROM articulo WHERE codigo = param1;
    Registro mi_cursor%ROWTYPE;
BEGIN
    OPEN mi_cursor(21);
    FETCH mi_cursor INTO registro;
    CLOSE mi_cursor;
    OPEN mi_cursor(101);
    FETCH mi_cursor INTO registro;
    CLOSE mi_cursor;
END;
```

2.5.7.- Manejo de Excepciones (Errores)

Las excepciones son identificadores **PL/SQL** que son llamadas durante la ejecución de un bloque para terminar su acción. Un bloque es siempre terminado

cuando **PL/SQL** alcanza una excepción, que puede definir manejadores de errores para capturar las excepciones y desarrollar algunas acciones finales antes de salir del bloque. Si **PL/SQL** llama a una excepción dentro del bloque, entonces la excepción no será propagada fuera del bloque o del medio ambiente.

Hay dos clases de excepciones:

- ☑ **Predefinidas**, errores predefinidos de Oracle que están asociados con códigos de error preestablecidos.
- ☑ **Definidas por el Usuario**, declaradas y llamadas por el usuario cuando se requieren específicamente dentro de un bloque. Se puede asociar una excepción definida por el usuario con un código de error si se desea.

Si un error ocurre dentro de un bloque **PL/SQL**, se pasa el control a la sección **EXCEPTION** del bloque. Si no existe la sección **EXCEPTION** dentro del bloque o la sección **EXCEPTION** no maneja el error ocurrido, entonces el bloque es terminado con una excepción no manejada. Las excepciones se propagan a través de los bloques anidados hasta que un manejador de excepciones, que pueda manejar el error, sea encontrado. Si no se encuentra un manejador de excepciones en ningún bloque, el error es pasado hasta el medio ambiente principal. Las excepciones ocurren cuando se genera un error Oracle (automáticamente llama una excepción) o se alcance un error explícitamente con la sentencia **RAISE**.

Los dos errores más comúnmente originados en una sentencia **SELECT** ocurren cuando no retorna filas o retornan más de una fila (recordar que esto no es permitido). El siguiente ejemplo trata con estas dos condiciones.

```
DECLARE
    V_costo NUMBER(10,2);
    V_codigo NUMBER(10);
BEGIN
    SELECT codigo, costo FROM articulo INTO v_codigo, v_costo
        WHERE codigo > 21;
    IF v_costo > 0 THEN
        UPDATE articulo SET costo = (v_costo * 1.1) WHERE codigo > 21;
    ELSE
        UPDATE articulo SET costo = 21.32 WHERE codigo > 21;
    END IF;
    COMMIT;
EXCEPTION
```

```

        WHEN NO_DATA_FOUND THEN
            INSERT INTO Errores (codigo, mensaje)
                VALUES (99, 'No encontrado');
        WHEN TOO_MANY_ROWS THEN
            INSERT INTO Errores (codigo, mensaje) VALUES (99, 'Demasiados');
    END;

```

Se genera un error porque hay más de un registro con código mayor que 21. La excepción alcanzada pasará a la sección **EXCEPTION**, donde cada acción manejada será verificada. Las sentencias dentro de la acción **TOO_MANY_ROWS** será ejecutada y luego el bloque se termina.

Si algunos otros errores ocurren, esta sección **EXCEPTION** no podría manejarlos, porque no tiene definidos como acciones verificables. Para cubrir todos los errores posibles, se puede especificar todas las acciones con **OTHERS**.

```

DECLARE
    V_costo NUMBER(10,2);
    V_codigo NUMBER(10);
BEGIN
    SELECT codigo, costo FROM articulo INTO v_codigo, v_costo
        WHERE codigo > 21;
    IF v_costo > 0 THEN
        UPDATE articulo SET costo = (v_costo * 1.1) WHERE codigo > 21;
    ELSE
        UPDATE articulo SET costo = 21.32 WHERE codigo > 21;
    END IF;
    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO Errores (codigo, mensaje)
            VALUES(99, 'No encontrado');
    WHEN TOO_MANY_ROWS THEN
        INSERT INTO Errores (codigo, mensaje) VALUES(99, 'Demasiados');
    WHEN OTHERS THEN
        INSERT INTO Errores (codigo, mensaje)
            VALUES(99, 'Algunos errores ocurren');
END;

```

Este bloque atrapará todos los errores. Si la excepción no es “no hay filas retornadas” o “demasiadas filas retornadas”, entonces la acción **OTHERS** se desarrolla.

PL/SQL provee dos funciones especiales, que pueden ser usadas dentro de la sección **EXCEPTION**: **SQLCODE** y **SQLERRM**. **SQLCODE** es el código de error Oracle para la excepción, **SQLERRM** es el mensaje de error Oracle para la

excepción. Se puede usar estas funciones para detectar que error ha ocurrido (es muy útil en la acción **OTHERS**). **SQLCODE** y **SQLERRM** deben ser asignados a variables antes de usarlas.

```

DECLARE
    V_costo NUMBER(10,2);
    V_codigo NUMBER(10);
    Error_mensaje VARCHAR2(100);
    Error_codigo NUMBER;

BEGIN
    SELECT codigo, costo FROM articulo INTO v_codigo, v_costo
        WHERE codigo > 21;
    IF v_costo > 0 THEN
        UPDATE articulo SET costo = (v_costo * 1.1) WHERE codigo > 21;
    ELSE
        UPDATE articulo SET costo = 21.32 WHERE codigo > 21;
    END IF;
    COMMIT;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO Errores (codigo, mensaje)
            VALUES(99, 'No encontrado');
    WHEN TOO_MANY_ROWS THEN
        INSERT INTO Errores (codigo, mensaje) VALUES(99, 'Demasiados');
    WHEN OTHERS THEN
        Error_mensaje:= SUBSTR(SQLERRM,1,100);
        Error_codigo := SQLCODE;
        INSERT INTO Errores (codigo, mensaje)
            VALUES(error_codigo, error_mensaje');

END;

```

Si se necesita explícitamente alcanzar un error, se puede usar la sentencia **RAISE**. Por ejemplo, para alcanzar un error si una sentencia **SELECT** no encuentra una fila:

```

DECLARE
    V_costo NUMBER(10,2);
    V_codigo NUMBER(10);
    Error_mensaje VARCHAR2(100);
    Error_codigo NUMBER;

BEGIN
    SELECT codigo, costo FROM articulo INTO v_codigo, v_costo
        WHERE codigo < 0;
    RAISE esto_esta_mal

EXCEPTION
    WHEN esto_esta_mal THEN
        INSERT INTO Errores (codigo, mensaje) VALUES(99, 'mal');

END;

```

No debería encontrar ninguna fila con un código menor que 0, por lo que, fuerza a **PL/SQL** pasar el control a una acción propia definida en la sección **EXCEPTION**.

2.5.8.- Estructuras de Control

PL/SQL provee varias estructuras de control posibles, para controlar el flujo lógico dentro de un bloque, estas no son diferentes a las estructuras encontradas en la mayoría de lenguajes de programación procedurales, y son:

2.5.8.1.- IF

Provee el control selectivo de acciones sobre la base del resultado de una condición.

```
IF A = 1 THEN
    D := 1 + E;
    E1 := H * 1.176;
ELSIF A = 2 THEN
    E := 1 + D;
    E1 := H * 1.1876;
    E2 := 3;
ELSIF A = 3 THEN
    B := 1.123;
ELSE
    B := 2.01;
END IF;
```

Se puede tener muchas sentencias **ELSIF**. La sección **ELSE** captura cualquier condición no atrapada. No es obligatorio tener las secciones **ELSIF** o **ELSE**.

2.5.8.2.- Ciclos

Provee la repetición de grupos de sentencias sin imponer una condición.

EXIT.- Provee una función de corte para un ciclo.

```
LOOP
    I := I + 1;
    IF I = 10 THEN
        EXIT;
    END IF;
END LOOP;
```



```
LOOP
  I := I + 1;
  EXIT WHEN I = 10;
END LOOP;
```

Notar que, si no se ha obtenido una condición de salida (y una sentencia **EXIT**) se tendrá un ciclo indefinido. El ciclo básico consiste de un grupo de sentencias entre **LOOP** y **END LOOP**. La sentencia **EXIT** termina el ciclo más cercano (a menos que explícitamente se nombre al ciclo que se quiere terminar). Notar que, los dos ciclos son funcionalmente idénticos, y demuestran dos formas alternas para usar la sentencia **EXIT**.

Ciclo FOR.- Provee la ejecución interactiva de sentencias sobre la base de un contador.

```
FOR Cont IN 12 .. 150 LOOP
  A := A + (Cont * 10);
  EXIT WHEN A > 12900;
END LOOP;
FOR Cont IN REVERSE 18 .. 121 LOOP
  A := A + (Cont * 10);
END LOOP;
```

Los ciclos **FOR** tienen la misma estructura que los ciclos básicos, pero tienen una sentencia de control adicional al frente. Esta sentencia adicional controla el número de interacciones ejecutadas por el ciclo (provista para no usar la sentencia **EXIT** para salir del ciclo). Por omisión, el ciclo incrementa la variable de control (la cuál debe ser declarada de la forma usual) comenzando en el valor más bajo y avanzando hasta el valor más alto, usar la cláusula **REVERSE** para hacer que el ciclo decremente la variable de control desde el valor más alto hasta el valor más bajo. Si se quiere un salto en el incremento, diferente de uno, se debe cambiar la variable de control dentro del ciclo.

Ciclo WHILE.- Provee una sentencia de ejecución interactiva, basada en una condición.

```
WHILE B > 0 AND Artículo <> 'Horno' OR Artículo <> 'Cocina' LOOP
  FOR Cont IN REVERSE 18 .. 121 LOOP
    A := A + (Cont * 10);
    IF A > 12900 THEN
      Artículo = 'Cocina';
    END IF;
```

```

END LOOP;
END LOOP;

```

La condición es evaluada al inicio del ciclo, si es falsa al inicio entonces el ciclo no se realiza. Notar que si la condición(es) de **WHILE** no cambian dentro del ciclo, entonces el ciclo nunca finaliza. Se puede incluir una o más sentencias **EXIT** si se requiere una salida del ciclo.

Es posible usar etiquetas con las estructuras de control, las cuales son encerradas entre los símbolos << y >>. Se puede pasar a ellas con las sentencias **GOTO** (pero es una mala práctica de programación), es útil para salir de ciclos o bloques anidados.

```

<<BloqueExterno>>
DECLARE
    V_costo NUMBER(10,2);
    V_Codigo NUMBER(10);
BEGIN
    SELECT codigo, costo FROM articulo INTO v_codigo, v_costo
        WHERE codigo > 21;
    IF v_costo > 0 THEN
        UPDATE articulo SET costo = (v_costo * 1.1) WHERE codigo > 21;
    ELSE
        UPDATE articulo SET costo = 21.32 WHERE codigo > 21;
    END IF;
<<BloqueInterno1>>
DECLARE
    I NUMBER(4);
    AB CHAR(10);
    Cont NUMBER(3);
    A NUMBER(6);
    <<CicloExterno>> LOOP
        I := I + 1;
        IF I = 5 THEN
            GOTO DOCOMMITTAG;
        END IF;
        IF I = 10 THEN
            EXIT CicloExterno;
        END IF;
        <<CicloInterno>> FOR Cont IN 12 .. 150 LOOP
            A := A + (CNT * 10);
            EXIT CicloExterno WHEN A > 12900;
        END LOOP CicloInterno;
    END LOOP CicloExterno;
    <<HacerCommit>>
    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO Errores (codigo, mensaje)
            VALUES(99, 'No Encontrado');

```

```
WHEN TOO_MANY_ROWS THEN
    INSERT INTO Errores (codigo, mensaje) VALUES(99, 'Demasiados');
END;
```

2.5.9.- Procedimientos y Funciones

Un procedimiento o función, es un bloque **PL/SQL** con nombre, almacenados normalmente en la base de datos dentro de las especificaciones del paquete (que es una envoltura para un grupo de bloques con nombre), pero pueden ser almacenados en la base de datos individualmente. La ventaja de colocar un bloque en la base de datos es pasada a la vez que es almacenada. Cuando es ejecutada, Oracle ya tiene el bloque compilado y es mucho más rápido. Es una buena forma de agrupar la funcionalidad de la aplicación, y exponer solo llamado a funciones (no el código en sí). Es posible invocar un procedimiento almacenado o función desde la mayor parte de herramientas Oracle incluyendo **SQL *Plus**. También es posible enlazar un bloque a una base de datos o trigger.

Los beneficios de usar procedimientos y funciones son:

1. Mejorar la Seguridad e Integridad de Datos
 - ☒ Controlar el acceso indirecto a objetos desde usuarios no privilegiados con privilegios de seguridad.
 - ☒ Asegurar que las acciones relacionadas son desarrolladas juntas, o no del todo, por acciones para tablas relacionadas a través de una ruta simple.
2. Mejorar el Desempeño
 - ☒ Evitar detenciones para múltiples usuarios, explotando **SQL** compartido.
 - ☒ Evitar detenciones **PL/SQL** en tiempo de ejecución, por saltos de tiempo de compilación.
 - ☒ Reducir el número de llamadas a la base de datos, y decrementar el tráfico de red, por empaquetar comandos.
3. Mejorar el Mantenimiento

- ☑ Modificar rutinas en línea sin interferir con otros usuarios.
- ☑ Modificar una rutina par afectar múltiples aplicaciones.
- ☑ Modificar una rutina para eliminar pruebas duplicadas.

Los procedimientos son simplemente bloques **PL/SQL** con nombre, son creados por un esquema particular, y similar a otros objetos de la base de datos son propios de ese esquema. Los privilegios para el procedimiento son otorgados / revocados igual que cualquier otro objeto, excepto que se otorga / revoca el derecho **EXECUTE** y no **SELECT**, **UPDATE** etc. Ejemplo de bloque sin nombre:

```

DECLARE
    V_costo NUMBER(10,2);
BEGIN
    SELECT costo FROM articulo INTO v_costo WHERE codigo = 21;
    IF v_costo > 0 THEN
        UPDATE articulo SET costo = (v_costo * 1.1) WHERE codigo = 21;
    ELSE
        UPDATE articulo SET costo = 21.32 WHERE codigo = 21;
    END IF;
    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO Errores (codigo, mensaje)
            VALUES(99, 'Codigo 21 No Encontrado');
END;

```

Este bloque podría ser grabado a un archivo de comando y ejecutado desde allí, pero debería ser pasado cada vez que sea llamado. Para grabar este bloque a la base de datos se necesita ponerle un nombre:

```

CREATE OR REPLACE PROCEDURE Mi_procl
    (v_costo NUMBER(10,2))
IS BEGIN
    SELECT costo FROM articulo INTO v_costo WHERE codigo = 21;
    IF v_costo > 0 THEN
        UPDATE articulo SET costo = (v_costo * 1.1) WHERE codigo = 21;
    ELSE
        UPDATE articulo SET costo = 21.32 WHERE codigo = 21;
    END IF;
    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO Errores (codigo, mensaje)
            VALUES(99, 'Codigo 21 No Encontrado');
END Mi_procl;

```

El nombre del procedimiento **Mi_procl**, es definido en la primera línea. **CREATE OR REPLACE** indica crear un nuevo procedimiento, o si un

procedimiento con el mismo nombre ya existe en el esquema, reemplazarlo (se puede omitir **OR REPLACE**, si se quiere que esto no ocurra).

La sección **DECLARE** es implícita, la sección entre la definición del procedimiento y la sentencia **IS BEGIN**, no es explícitamente nombrada, pero es encerrada entre paréntesis. La sentencia **END** finaliza un bloque con nombre. No se requiere nada para ejecutar un procedimiento, simplemente llamarlo por su nombre.

Si se invoca el procedimiento en **SQL *Plus**, se debería prefijar el nombre del procedimiento con el comando **EXECUTE**.

Para obtener valores hacia y fuera del procedimiento, se definen las variables en la sección de declaración implícita como **IN**, **OUT** o **IN OUT**.

```
CREATE OR REPLACE PROCEDURE Mi_proc1
(v_codigo IN NUMBER,
v_aux IN OUT CHAR,
T_costo OUT NUMBER)
V_costo NUMBER(10,2))
IS BEGIN
  SELECT costo FROM articulo INTO v_costo WHERE codigo = v_codigo;
  IF v_costo > 0 THEN
    UPDATE articulo SET costo = (v_costo *1.1)
      WHERE codigo = v_codigo;
  ELSE
    UPDATE articulo SET costo = 21.32 WHERE codigo = v_codigo;
  END IF;
  T_costo:= v_costo;
  COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    INSERT INTO Errores (codigo, mensaje)
      VALUES(99, 'codigo No encontrado');
END Mi_proc1;
```

Para ejecutar el procedimiento en **SQL *PLUS**:

```
EXECUTE Mi_proc1 (21, :PCOSTO)
```

PCOST es el nombre de una variable **SQL *Plus**. **21** será pasado al procedimiento, el valor de **T_COSTO** será pasado fuera del procedimiento y almacenado en **:PCOSTO**.

Una variable **IN OUT** puede pasar valores hacia y fuera del procedimiento.

Se debe recordar que, un procedimiento puede leer una lista de valores, pero no podrá retornar directamente ninguno (pero si puede retornar indirectamente valores). Una función puede también leer una lista de valores, pero retorna explícitamente un resultado simple, que es normalmente asignado a una variable o usado en la condición de una estructura de control del programa (sentencias **IF**, etc.).

La siguiente función muestra la sintaxis de una función **PL/SQL**, notar que, al igual que en el procedimiento, la cláusula **OR REPLACE** puede ser omitida si no se quiere el reemplazo de una función existente con el mismo nombre. Las definiciones de funciones varían de las definiciones de un procedimiento en que, se debe nombrar explícitamente una variable para retornar un valor a través de la sentencia **RETURN**.

```
CREATE OR REPLACE FUNCTION Mi_func1
RETURN NUMBER
IS Rcosto NUMBER(10,2);
BEGIN
SELECT costo FROM articulo INTO Rcosto WHERE codigo = 21;
RETURN (Rcosto);
END Mi_func1;
```

Para ejecutar el procedimiento en SQL *PLUS:

```
EXECUTE :Pcosto := Mi_func1
```

Pcosto es el nombre de una variable **SQL *Plus**, el valor de compra del artículo **21** será pasado desde la función y almacenado en **:PCOSTO**.

2.5.10.- Paquetes

Un paquete es un grupo de funciones y/o rutinas relacionadas. Se usan para agrupar bloques **PL/SQL** que forman una aplicación común, o están unidos a una función simple del negocio. Los paquetes consisten de una **especificación** y un **cuerpo**. La especificación del paquete lista las interfaces públicas para los bloques dentro del cuerpo del paquete. El **cuerpo** del paquete contiene los bloques **PL/SQL** públicos y privados que forman la aplicación, los bloques privados no son definidos en la especificación del paquete y no pueden ser llamados por cualquier rutina que no esté definida dentro del cuerpo del paquete. El beneficio de los paquetes es la mejora

en la organización de los bloques de procedimientos y funciones, permitiendo actualizar los bloques que forman el cuerpo del paquete sin afectar la especificación (por los usuarios que tienen derechos sobre el paquete) y permiten otorgar derechos de ejecución una sola vez, en lugar de hacerlo a cada bloque.

Para crear la especificación del paquete se utiliza una variación del comando **CREATE**, colocando cada cabecera de bloque **PL/SQL** que será público dentro del paquete. Ejemplo:

```
CREATE OR REPLACE PACKAGE Mi_paq1 AS
    PROCEDURE Mi_proc1 (v_codigo IN NUMBER, var1 IN OUT CHAR, Tcosto
OUT NUMBER);
    FUNCTION Mi_func1;
END Mi_paq1;
```

Para crear el cuerpo del paquete se especifica cada bloque **PL/SQL** que forma el paquete, notar que no se están creando estos bloques en forma separada (no se requiere **CREATE OR REPLACE** para las definiciones procedimiento y funciones). Ejemplo:

```
CREATE OR REPLACE PACKAGE BODY Mi_paq1 AS
    PROCEDURE Mi_proc1
        (v_codigo IN NUMBER,
        Var1 IN OUT CHAR,
        Tcosto OUT NUMBER)
        V_costo NUMBER(10,2))
    IS BEGIN
        SELECT costo FROM articulo INTO v_costo
            WHERE codigo = v_codigo;
        IF v_costo > 0 THEN
            UPDATE articulo SET costo = (v_costo * 1.1)
                WHERE codigo = v_codigo;
        ELSE
            UPDATE articulo SET costo = 21.32
                WHERE codigo = v_codigo;
        END IF;
        Tcosto := v_costo;
        COMMIT;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            INSERT INTO Errores (codigo, mensaje)
                VALUES(99, 'codigo no encontrado');
    END Mi_proc1;
    FUNCTION Mi_func1
        RETURN NUMBER
    IS Rcosto NUMBER(10,2);
    BEGIN
        SELECT costo FROM articulo INTO Rcosto WHERE codigo = 21;
```

```

        RETURN (Rcosto);
    END Mi_func1;
END Mi_paq1;

```

Se puede ejecutar un bloque de paquete público así:

```
EXECUTE :Pcosto := Mi_paq1.Mi_func1
```

Se utiliza **DROP PACKAGE** y **DROP PACKAGE BODY** para remover los objetos tipo paquete de la base de datos.

2.5.11.- Triggers (Disparadores)

Un trigger es un bloque de código **PL/SQL** enlazado y ejecutado por cualquier evento que ocurra a una tabla de la base de datos. Los disparadores son implícitamente invocados por los comandos **DML**. Los disparadores son almacenados como texto y compilados al momento de la ejecución. No se puede usar las sentencias **COMMIT**, **ROLLBACK** y **SAVEPOINT** dentro de bloques trigger. Recordar que, los disparadores pueden ser ejecutados cientos de veces para una actualización grande (que pueden afectar seriamente el desempeño de ejecución de **SQL**).

Los disparadores pueden ser llamados antes (**BEFORE**) o después (**AFTER**) de los eventos: **INSERT**, **UPDATE** y **DELETE**.

Los disparadores pueden ser de tipo **SENTENCIA** o **ROW**. Los disparadores **SENTENCIA** se disparan antes o después de la ejecución de la sentencia que causa que el trigger se dispare. Los disparadores **ROW** se disparan antes o después que cualquier fila afectada es procesada. Ejemplo de trigger **SENTENCIA**:

```

CREATE OR REPLACE TRIGGER Mi_trig1
    BEFORE DELETE OR INSERT OR UPDATE ON articulo
BEGIN
    IF (TO_CHAR(SYSDATE,'dy') IN ('sab','dom'))
        OR (TO_CHAR(SYSDATE,'hh24:mi') NOT BETWEEN '08:30'
            AND '18:30') THEN
        RAISE_APPLICATION_ERROR(-20500,'La tabla esta
            segura');
    END IF;
END;

```


Luego de la sentencia **CREATE OR REPLACE** esta el identificador de objeto (**TRIGGER**) y el nombre del objeto (**Mi_trig1**). Este trigger especifica que antes que cualquier evento cambie los datos en la tabla **ARTICULO**, este bloque **PL/SQL** será compilado y ejecutado. El usuario no podrá actualizar la tabla fuera de las horas normales de trabajo. Ejemplo de trigger **ROW**:

```
CREATE OR REPLACE TRIGGER Mi_trig2
AFTER DELETE OR INSERT OR UPDATE ON articulo
FOR EACH ROW
BEGIN
    IF DELETING THEN
        INSERT INTO Historico (codigo, articulo, fecha_borrado)
        VALUES (:OLD.codigo, :OLD.articulo, SYSDATE);
    ELSIF INSERTING THEN
        INSERT INTO Nuevos (codigo, articulo, fecha_ingreso)
        VALUES (:NEW.codigo, :NEW.articulo, SYSDATE);
    ELSIF UPDATING ('codigo') THEN
        INSERT INTO Cambios (cod_ant, cod_nuevo, articulo, fecha_actualiz)
        VALUES (:OLD.codigo, :NEW.codigo, :NEW.articulo,
        SYSDATE);
    ELSE /* Actualizar cualquier campo diferente de CODIGO */
        INSERT INTO Otros (codigo, articulo, fecha_actualiz)
        VALUES (:OLD.codigo, :NEW.articulo, SYSDATE);
    END IF
END;
```

En este caso, se ha especificado que el trigger sea ejecutado luego que cualquier evento cambie los datos sobre cualquier fila afectada. Dentro del cuerpo del bloque **PL/SQL**, se puede verificar cuales acciones de actualización están siendo desarrolladas para la fila actualmente afectada, y tomar la acción que sea apropiada. Notar que, se puede especificar los valores viejos y nuevos de las filas actualizadas, prefijando los nombres de columna con los calificadores **:OLD** y **:NEW**.

CAPITULO III

ADMINISTRACIÓN ORACLE

3.1.- Administración

En un medio ambiente multiusuario, es deseable mantener la seguridad del acceso a la base de datos y su uso. La seguridad de la base de datos Oracle permite:

- ☑ Controlar el acceso a la base de datos
- ☑ Dar acceso a objetos específicos en la base de datos
- ☑ Confirmar los privilegios dados y recibidos, en el Diccionario de Datos
- ☑ Crear sinónimos para objetos de la base de datos

Hay más de ochenta privilegios del sistema disponibles para usuarios y roles. Los privilegios del sistema son típicamente proporcionados por el administrador de la base de datos.

Los privilegios típicos del DBA son:

<u>Privilegio del Sistema</u>	<u>Operaciones Autorizadas</u>
Create User	permite crear otros usuarios
Drop User	elimina otro usuario
Drop Any Table	elimina una tabla en cualquier esquema
Backup Any Table	respalda cualquier tabla en cualquier esquema

3.1.1.- Usuarios

El Administrador de la Base de Datos, crea un usuario y le asigna un grupo de privilegios del sistema, que determinan lo que el usuario puede hacer en el ámbito de la base de datos, para lo cuál se utiliza el siguiente formato:

CREATE USER nombre_usuario **IDENTIFIED BY** clave;

Donde:

Nombre_usuario es el nombre del usuario a ser creado


```
CREATE ROL contabilidad;  
GRANT create table, create view TO contabilidad;  
GRANT contabilidad TO carlosr, manuelp;
```

Al momento de crear un usuario se asigna su clave de acceso, si dispone de los privilegios necesarios, puede en cualquier momento cambiar su clave, con el comando **ALTER USER**, con el formato:

```
ALTER USER usuario IDENTIFIED BY clave;
```

Donde:

Usuario Nombre del usuario.

Clave Especifica la nueva clave

3.1.2.- Seguridades

La seguridad de la base de datos puede ser clasificada en dos categorías: **seguridad del sistema** y **seguridad de datos**.

La **seguridad del sistema** cubre el acceso y uso de la base de datos a nivel del sistema, tal como nombre y clave de usuario, espacio de disco ubicado para los usuarios y operaciones del sistema permitidas al usuario.

La **seguridad de datos** cubre el acceso y uso de los objetos de la base de datos, y las acciones que los usuarios pueden tener sobre los objetos.

Privilegios: son los derechos que tiene un usuario para ejecutar sentencias **SQL** particulares. El DBA, es el usuario de más alto nivel, con la habilidad de otorgar acceso a los usuarios tanto a la base de datos como a sus objetos. Los usuarios requieren privilegios para tener acceso a la base de datos y **privilegios de objeto** para manipular el contenido de los objetos en la base de datos. Los usuarios también pueden tener privilegios para otorgar privilegios adicionales a otros usuarios o a roles, los cuales son llamados grupos de privilegios relacionados.

Esquemas: son colecciones de objetos, tales como tablas, vistas y secuencias. El esquema pertenece a un usuario de la base de datos y tiene el mismo nombre que el usuario.

3.1.3.- Objetos

Las aplicaciones hoy en día, pueden ser muy complejas, y consisten de cientos de diferentes módulos, y virtualmente, de millones de líneas de código.

Mientras la aplicación crece, se necesita manejar todos sus componentes.

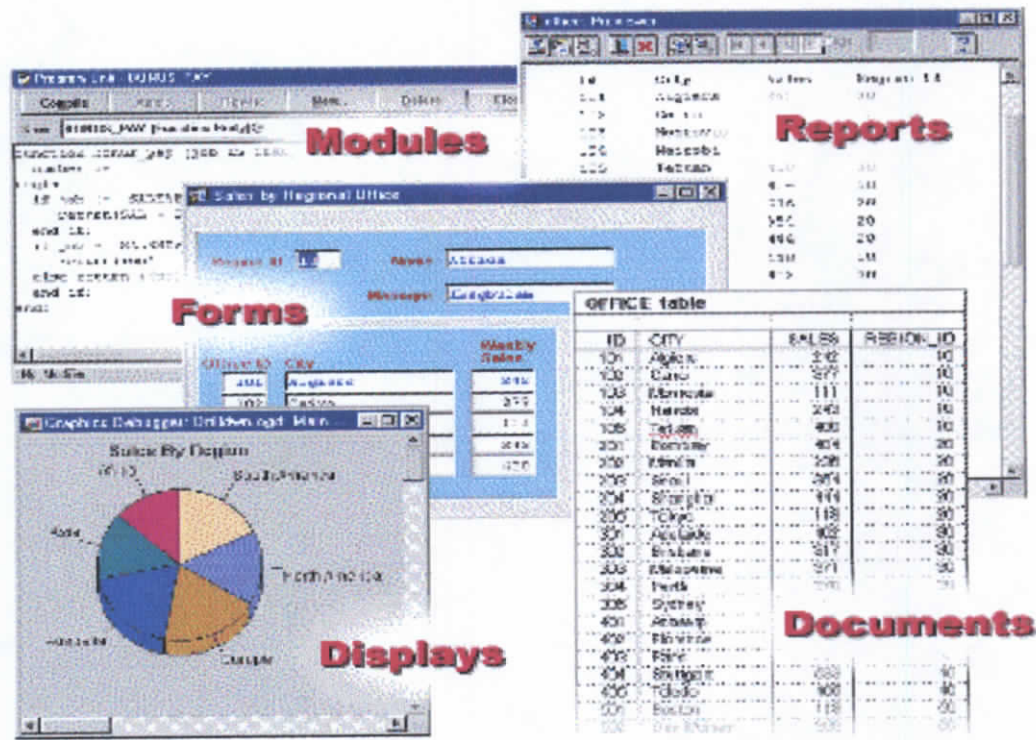


Fig. 3.1. Módulos de una Aplicación

Los proyectos, son listas de archivos que asocian las aplicaciones con las herramientas utilizadas, y permiten definir las tareas más comunes.

En resumen, con un simple comando **Build**, **Project Builder** compila todos los archivos de acuerdo a su tipo.

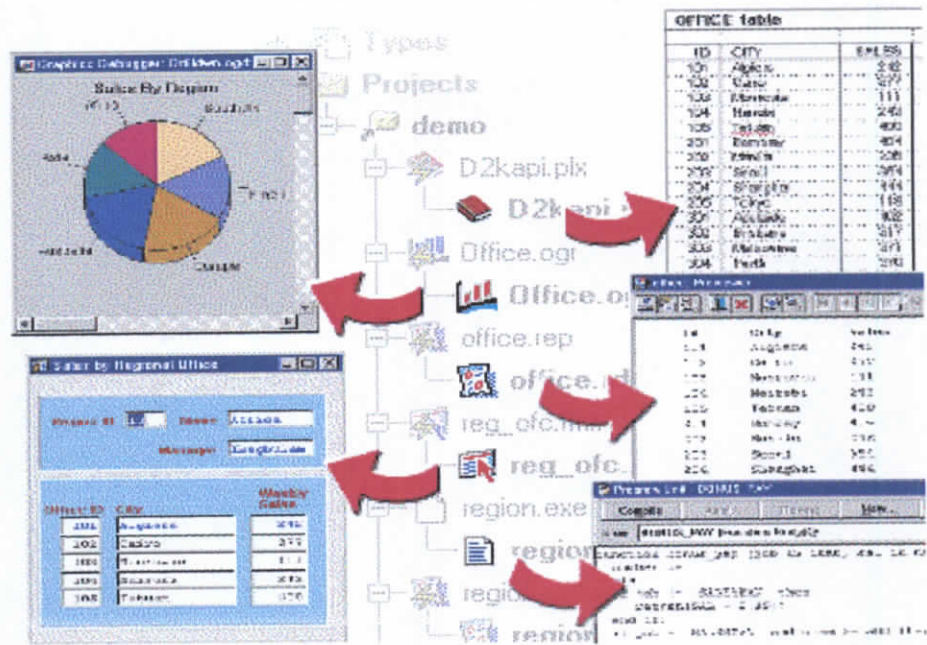


Fig. 3.2. Componentes de un Proyecto, Project Builder

Por tanto, las partes que conforman un proyecto son:

Archivos del Proyecto: Los archivos del proyecto son todos aquellos asociados con la aplicación, pueden ser de cualquier tipo, incluyendo formas, reportes, visualizaciones, módulos 3GL, documentación, y scripts de prueba.

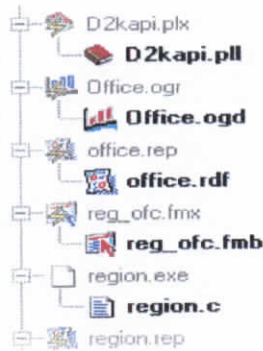


Fig. 3.3. Archivos de un Proyecto

Registros: Los registros son lugares de almacenamiento de información del proyecto. La información de tipo es usada para categorizar los archivos del proyecto y determinar las acciones que se aplican a ellos. También existe un registro especial

denominado **Connections**, que almacena cadenas de conexión a la base de datos predefinidas, que se pueden asociar con los archivos.

Un proyecto se refiere a dos tipos de registros: **Registro Global**, y uno o más **Registros de Usuario**.

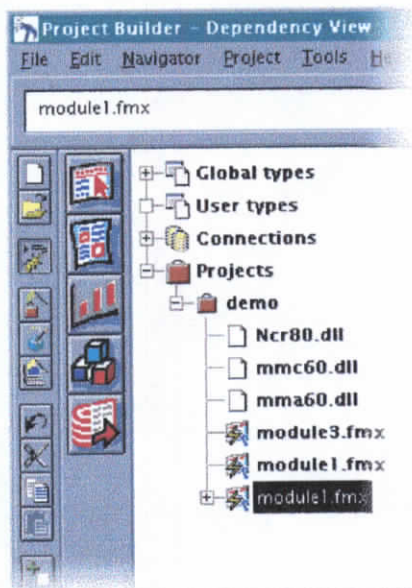


Fig. 3.4. Registros de un Proyecto

- **Registros Globales:** Contienen información común al medio ambiente de desarrollo completo; por ejemplo, los tipos que se aplican a los archivos del proyecto. **Global Registry** es creado y mantenido por un Administrador del Proyecto.
- **Registros de Usuario:** Contienen la información que se ha definido para los proyectos, tal como: tipos que son específicos al proyecto y conexiones a las fuentes de datos que contienen información del proyecto.

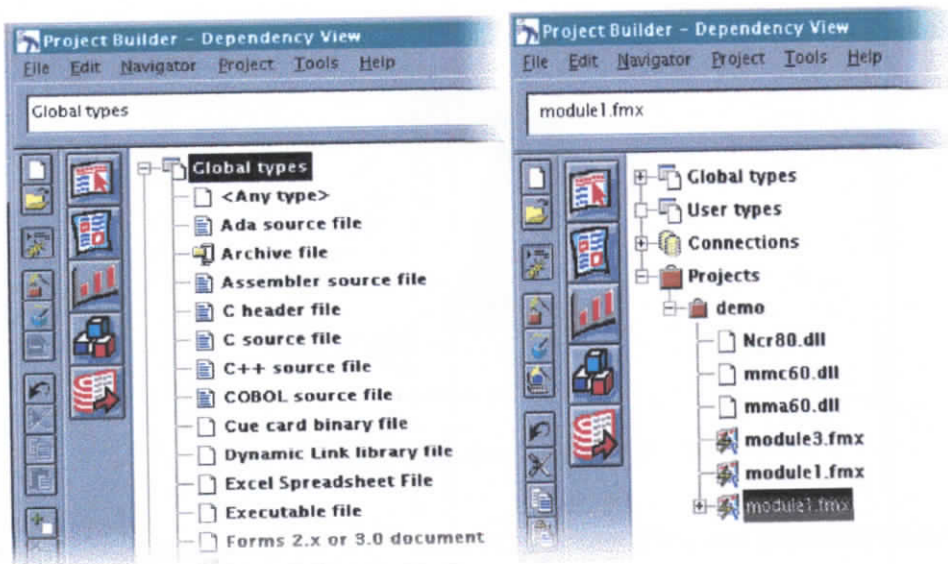


Fig. 3.5. Registros Globales y Locales

3.2.- Herramientas de Desarrollo (Oracle Developer)

Developer es un medio Ambiente de Desarrollo Rápido de Aplicaciones Oracle (RAD), que permite la construcción de aplicaciones de base de datos a gran escala, así como para internet. Utiliza características declarativas, de forma que los desarrolladores de aplicaciones comerciales, las pueden crear rápidamente a partir de definiciones de bases de datos.

Developer permite:

- ☑ Presentar aplicaciones en Internet.
- ☑ Producir aplicaciones eficientes y escalables
- ☑ Construir aplicaciones grandes, consistentes, y manejables en forma rápida
- ☑ Usar herramientas de tercer nivel y servidores de bases de datos.

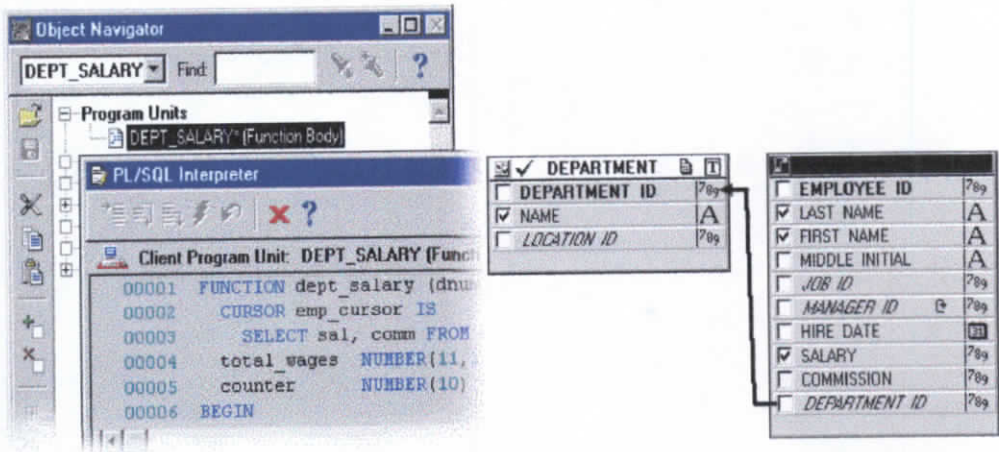


Fig. 3.7. Herramientas Procedure Builder y Query Builder

- 3. **Schema Builder:** Herramienta GUI para crear, copiar, modificar y remover objetos de la base de datos y sus relaciones.
- 4. **Translation Builder:** Herramienta para traducir las aplicaciones Developer a más de cuarenta lenguajes.

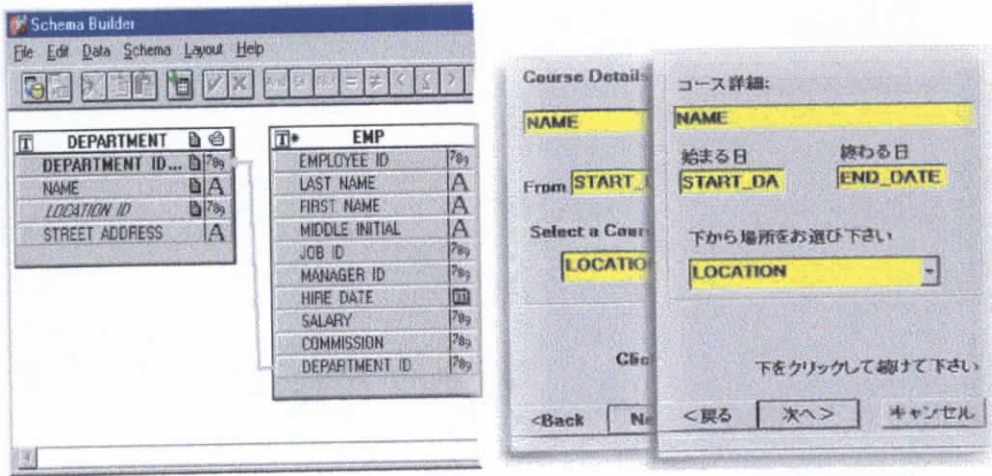


Fig. 3.8. Herramientas Schema Builder y Translation Builder

- 5. **Project Builder:** Un sistema para manejo y desarrollo de proyectos que manipula el código fuente para todos los componentes **Developer**. Se utiliza para manejar visualmente los componentes que mejoran las aplicaciones (Figura 3.2.).
- 6. **PECS (Performance Evaluation/Collection System):** Para perfilar la ejecución y desempeño de aplicaciones desarrolladas con herramientas Oracle.

7. **Discoverer:** Herramienta para navegar a través de los datos, que puede ser personalizada para usuarios finales.

3.3.- Herramientas Developer para Windows 95/98/NT/2000

Una instalación típica de **Developer** contiene varias opciones:

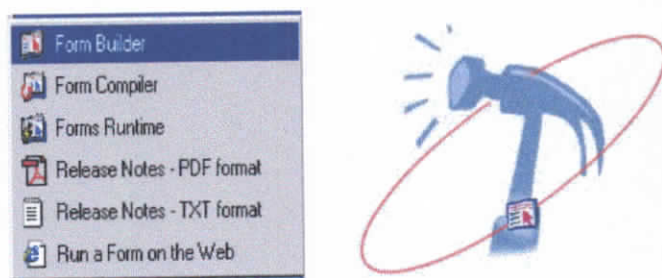


Fig. 3.9. Opciones Developer

3.3.1.- Pantalla Principal de Forms Builder

Una vez que se ejecuta **Forms Builder**, aparece la ventana:

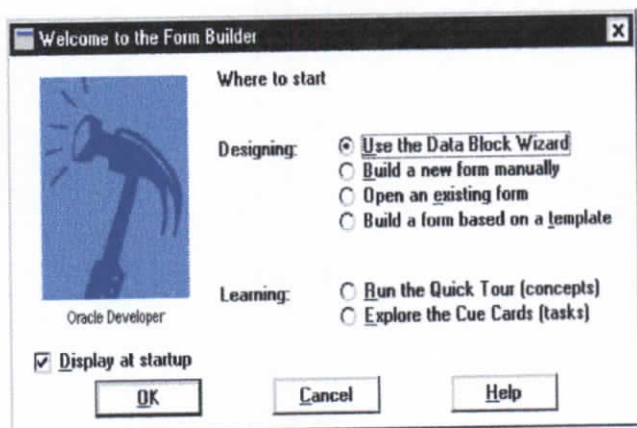


Fig. 3.10. Pantalla de Ingreso a Forms Builder

Forms visualiza esta ventana de bienvenida, como una forma rápida para iniciar el trabajo y diseñar formas. La forma más general para trabajar con formas es a través del **Navegador de Objetos**.

En la ventana de bienvenida hacer clic en el botón **Cancel** o seleccionar la opción **Build a Form Manually** y hacer clic en **Ok**.

3.3.1.1.- Navegador de Objetos Oracle Forms

Luego de iniciar Oracle Forms, se presenta la siguiente pantalla principal.

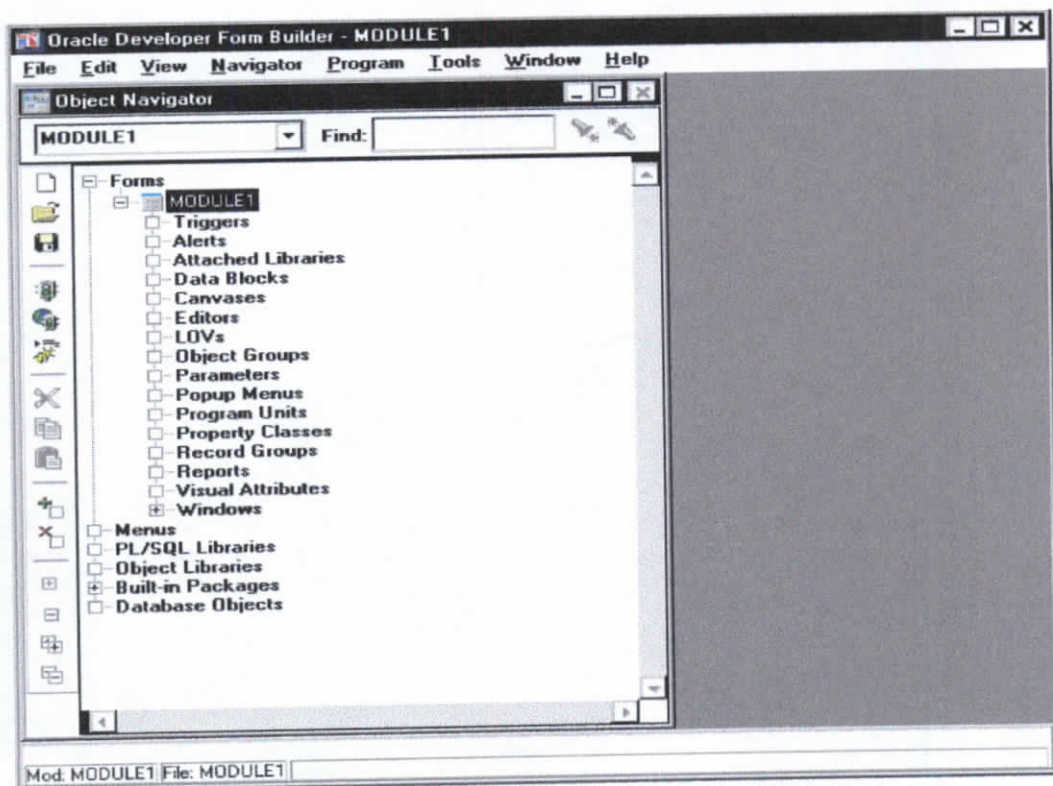


Fig. 3.11. Pantalla Principal de Oracle Forms

A esta pantalla se la conoce como **Navegador de Objetos**, y visualiza todos los elementos de una forma, bloques de datos, ítems de opciones, librerías de código definidas por el usuario, procedimientos y funciones preconstruidos, y objetos de la base de datos (tablas, vistas). La visualización se presenta en forma de árbol, al hacer clic en el signo +, se expande el ítem actual para mostrar los detalles, y al hacer clic en el signo –, se ocultan los detalles de la lista actual.

La **barra de opciones**, en la parte superior, provee acceso a la mayoría de características de Oracle Forms.

- ☑ **File** tiene opciones para crear, abrir, grabar e imprimir formas, también incluye una opción para conectar a la base de datos y la opción **Administration** usada para compilar formas y el código de opciones. La última opción es **Exit**, para salir de Oracle Forms.
- ☑ **Edit** tiene opciones para cortar, copiar y pegar objetos, para invocar un editor y deshacer una operación.
- ☑ **View** intercambia la visualización del navegador de objetos entre elementos visuales y elementos del usuario.
- ☑ **Navigator** tiene ítems que controlan la visualización del navegador de objetos incluyendo expandir y ocultar ramas de árboles. Puede ser usada para crear y borrar objetos con las opciones **Create** y **Delete**.
- ☑ **Program** tiene ítems que controlan la forma de compilación, generación y ejecución (**Run**). Disparadores, Procedimientos y otro código **PL/SQL**, también pueden ser creados desde esta opción. La opción **Smart Triggers** visualiza los disparadores más usados de acuerdo al tipo de objeto, actualmente seleccionado en el navegador de objetos.
- ☑ **Tools** tiene opciones para intercambiar entre varias pantallas incluyendo: Editor Layout, Editor Menu y Navegador de Objetos. También hay varios asistentes incluyendo: Data Block, Layout, Chart y LOV (Lista de Valores).
- ☑ **Windows** visualiza ítems para cada una de las ventanas abiertas. Por omisión, se visualiza sólo un ítem.
- ☑ Finalmente la opción **Help** se usa para visualizar tópicos de ayuda.

Por omisión, se presenta una barra de botones al lado izquierdo de la ventana **Navegador de Objetos**, cada uno de los cuáles realiza algunas funciones de las opciones. Por ejemplo, al hacer clic en la carpeta abierta es igual que seleccionar la opción **File | Open**. Para ver que hace un botón, colocar el cursor del ratón sobre el botón y aparece un mensaje corto que describe su función.

En **Developer**, una forma, reporte o gráfico es generalmente referido como **Módulo**. Cuando se inicia por primera vez **Forms Builder**, se crea un módulo por omisión llamado: **MODULE1**

El contenido del **Navegador de Objetos** está dispuesto en 6 secciones:

1. **Forms:** Contiene objetos forma y todos sus componentes, tales como bloques de datos, disparadores, alertas, lienzos, unidades de programa (procedimientos y funciones), parámetros y ventanas. Más de una forma puede ser abierta a la vez.
2. **Menus:** Contiene objetos menú y todos sus componentes, como items de opciones, parámetros y unidades de programa. Más de un menú puede ser abierto a la vez.
3. **PL/SQL Libraries:** Son librerías de código **PL/SQL** (almacenadas ya sea en la Base de Datos o en archivos ***.plb**). Las librerías pueden ser compartidas entre diferentes formas y entre diferentes desarrolladores.
4. **Object Libraries:** Son librerías que contienen una colección de objetos (cualquier cosa que puede aparecer en una forma).
5. **Built-in Packages:** Lista todos los paquetes (procedimientos **PL/SQL**) disponible por omisión para todas las formas, reportes, etc.
6. **Database Objects:** Lista todos los objetos de la Base de Datos (Tablas, Vistas, Librerías **PL/SQL**, y Unidades de Procedimiento Almacenadas) en el esquema de la Base de Datos del usuario Oracle.

Mientras los objetos son creados, son visualizados bajo la sección apropiada. Opciones de contexto sensitivas pueden ser visualizadas para cada objeto, al hacer clic con el botón derecho sobre el objeto.

3.3.1.2.- Conectar a la Base de Datos

El primer paso para trabajar con las herramientas **Developer**, es establecer una conexión a una Base de Datos Oracle. Seleccionar la opción **File | Connect**, ingresar el nombre de usuario y la clave.

Para el campo **Database**, ingresar el nombre de servicio para la Base de Datos Oracle. Por ejemplo, si se está usando Personal Oracle Lite podría ser **ODBC:POLITE**; si se tiene una Base de Datos Local dejar este campo en blanco; si se está trabajando en un medio cliente / servidor se debe configurar **Oracle Net8** para establecer una conexión con el servidor, entonces suministrar el nombre de servicio Oracle Net8.

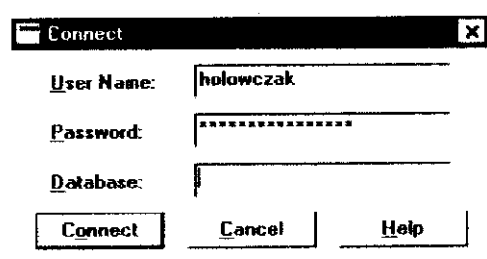


Fig. 3.12. Conexión a la Base de Datos

Hacer clic en el botón **Connect** para completar la conexión. Si el nombre de usuario o la clave, se ingresan en forma incorrecta, la caja de diálogo se vuelve a presentar.

Cada vez que cualquier herramienta de **Developer** es ejecutada, el primer paso es conectar a una Base de Datos Oracle.

3.3.1.3.- Diseño de Formas

Un esquema de Base de Datos típico, contiene docenas de tablas, cada una con varias columnas de varios tipos de datos. Se desarrollan aplicaciones (formas, reportes, opciones, etc.) para hacer que el usuario realice el trabajo de manipular datos en esas tablas de una manera fácil. En **Oracle Forms**, una forma (o forma de

entrada de datos) actúa similar a una ventana en un esquema de Base de Datos, con un enfoque individual a una o pocas tablas a la vez. En resumen, una forma puede dar indicaciones, de forma que el usuario conozca que tipo de dato debe ingresar, y lo manipula.

Por omisión, cada forma tiene la capacidad para consultar datos existentes en la tabla, modificar datos existentes y adicionar nuevos datos (registros) a la tabla. Una forma es construida usando uno o más **bloques de datos** que corresponden a tablas en la Base de Datos. Los campos dentro del bloque de datos corresponden a columnas en la tabla de la Base de Datos. Un bloque de datos es similar a un objeto **DataWindow** de PowerBuilder.

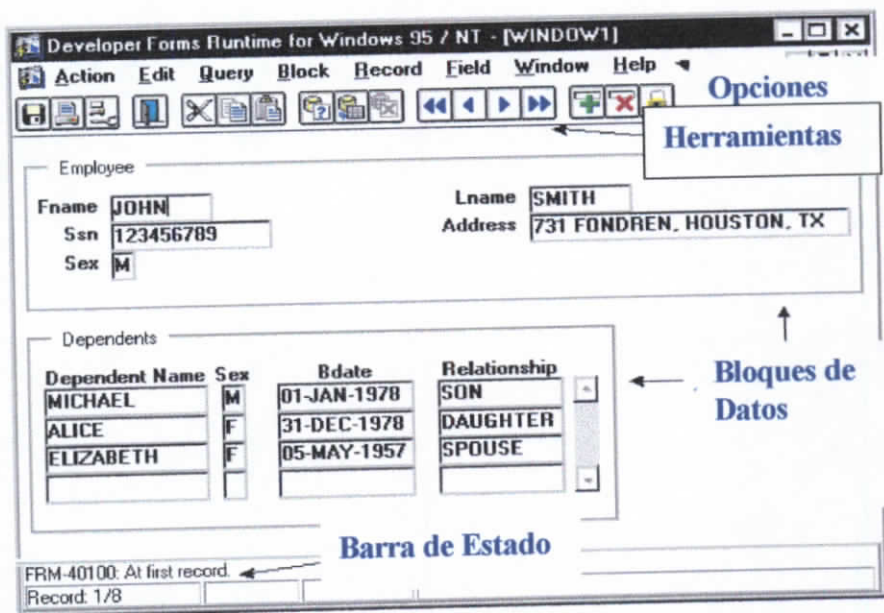


Fig. 3.13. Ventana Típica de Forms

En la Figura 3.13, la forma tiene dos bloques de datos, uno para la tabla **EMPLEADO**, y una para la tabla **DEPENDIENTE**, que son organizados como **Master/Detalle** donde un registro de empleado (master) está asociado con uno o más registros de dependiente (detalle).

Por omisión, las formas también tienen una barra de botones y una de opciones, que pueden ser usadas para avanzar a través de los registros en un bloque, navegar entre bloques, actualizar y desarrollar consultas sobre las tablas, insertar, actualizar y

borrar registros, limpiar la forma y salir de ella. Finalmente, en la parte inferior de cada forma está la barra de estado que visualiza cualquier indicación relevante o mensaje de error y una indicación de los registros en el bloque de datos actual.

Se pueden diseñar cuatro tipos principales de formas.

- ☑ Forma **Bloque Simple**, contiene un bloque de datos simple, que corresponde a una sola tabla de la Base de Datos.
- ☑ Forma **Bloque Simple con Campo de Observación**, contiene un bloque de datos simple, que corresponde a una sola tabla de la Base de Datos, con la adición de uno o más campos que visualizan datos de otras tablas.

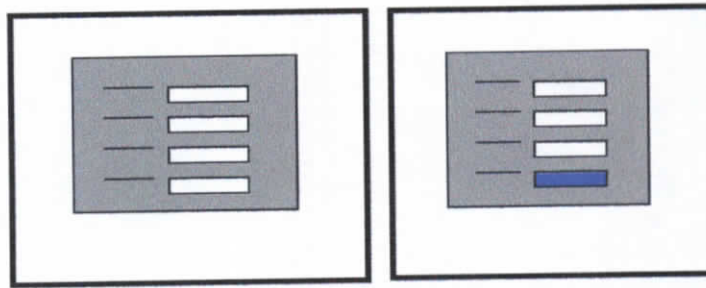


Fig. 3.14. Bloque Simple y Bloque Simple con Campo de Observación

- ☑ Forma **Master/Detalle**, contiene dos bloques de datos, que son organizados en una relación master/detalle (uno a muchos).
- ☑ Forma **Master/Detalle con Campo de Observación**, similar a Master/Detalle pero con campos adicionales de observación, en los bloques master y/o detalle.

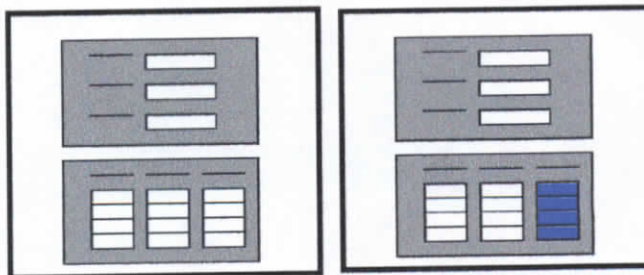


Fig. 3.15. Master/Detalle y Master/Detalle con Campo de Observación

Hay ciertas variaciones adicionales tales como forma Master/Detalle/Detalle con tres bloques y así. Sin embargo, los cuatro tipos indicados son los más comunes.

3.3.1.4.- Crear y Ejecutar una Forma con un Bloque Simple

En general, un **Bloque de Datos** en una forma, se corresponde a una tabla en la Base de Datos, que se conoce como **Tabla Base para el Bloque de Datos**. Para cada tabla visualizada en la forma, se debe crear un nuevo bloque de datos. Por ejemplo, crear un bloque de datos para la tabla **EMPLEADO**.

Un bloque aparece sobre un **Lienzo** que está contenido en una ventana. Hay muchas opciones para controlar múltiples vistas de lienzo, y múltiples ventanas por forma.

Hay varias formas para crear un bloque de datos: una es definir manualmente la tabla base y columnas, y sus posiciones en la forma, pero es muy tedioso.

Oracle Forms provee dos asistentes para crear bloques de datos:

1. El Asistente **Data Block** guía el desarrollo a través de pasos, para seleccionar la tabla base y columnas.
2. El Asistente **Layout** guía el desarrollo para la organización de la tabla base y columnas en la forma.

Crear un Nuevo Bloque: Para crear un nuevo bloque, seleccionar **Tools | Data Block wizard**, y aparece la ventana siguiente:

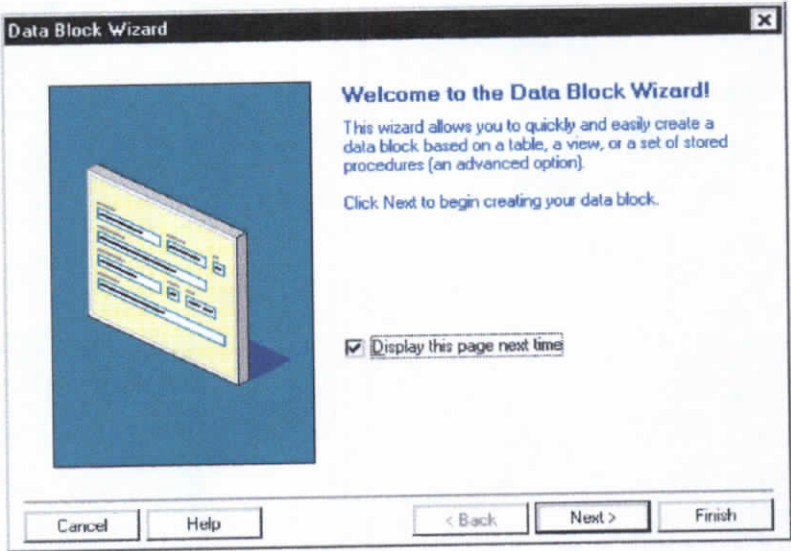


Fig. 3.16. Asistente de Bloque de Datos

Al hacer clic en el botón **Next**, se presenta la ventana:

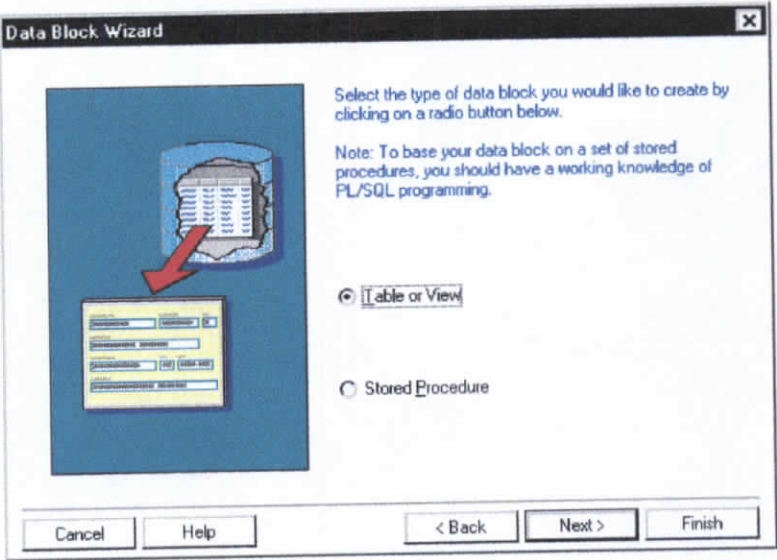


Fig. 3.17. Tipo de Bloque de Datos

Hay dos tipos de bloques de datos que pueden ser creados. **Table/View** y **Stored Procedure**. Seleccionar **Table/View** y hacer clic en el botón **Next**.

En la siguiente ventana, seleccionar una tabla base y las columnas para el bloque de datos.

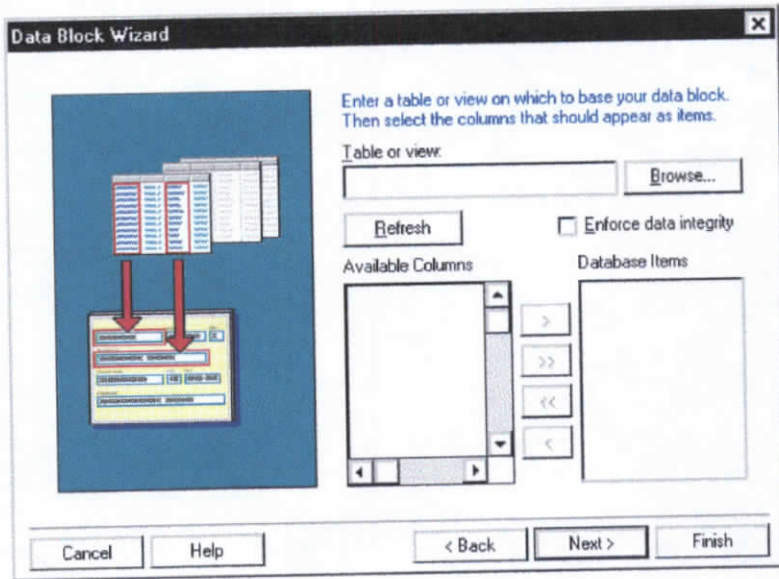


Fig. 3.18. Tabla y Columnas del Bloque de Datos

Hacer clic en el botón **Browse...** a la derecha del campo **Table or View**:

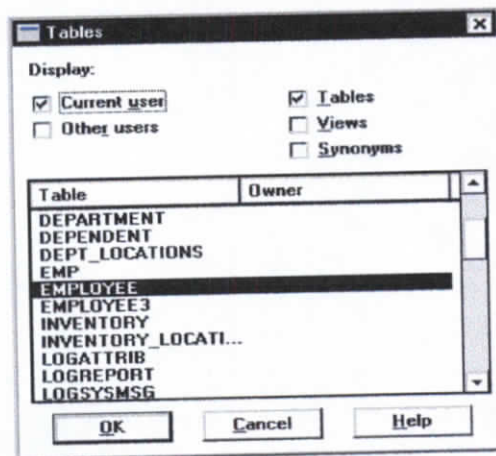


Fig. 3.19. Navegador del Bloque de Datos

Verificar que los botones **Current User** y **Tables** estén seleccionados, en la lista de tablas, seleccionar el nombre de la tabla y hacer clic en el botón **OK**.

El asistente, con el nombre de la tabla y la lista de columnas disponibles se presenta. Para incluir una columna en el bloque de datos, seleccionar el nombre de la columna y hacer clic en la flecha derecha, para moverla a la ventana de la derecha.

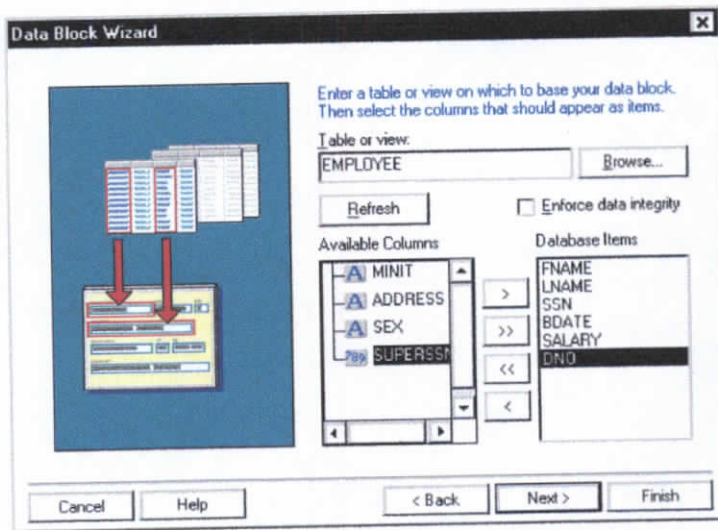


Fig. 3.20. Seleccionar Columnas para el Bloque de Datos

Hacer clic en el botón **Next**, y se presenta el diálogo final del asistente:

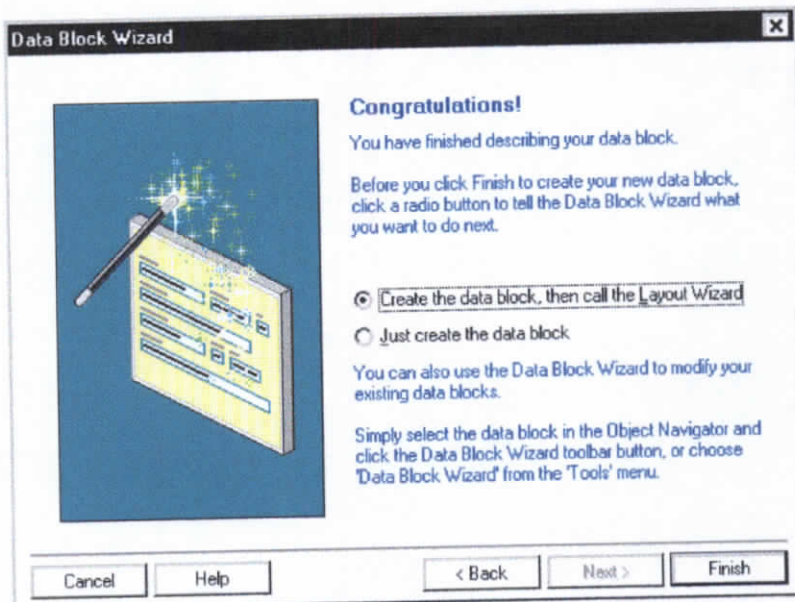


Fig. 3.21. Ventana final del Asistente de Bloque de Datos

Para especificar como quedará la nueva forma, seleccionar la opción **Create the block, then call the Layout wizard** y hacer clic en el botón **Finish**.

El bloque de datos es creado (se lo puede ver en el Navegador de Objetos). Los objetos creados incluyen el bloque de datos **EMPLEADO** que contiene los items para cada una de las columnas seleccionadas en el asistente.

Una vez que el nuevo bloque de datos e items han sido creados, aparece la primera ventana del Asistente Layout:

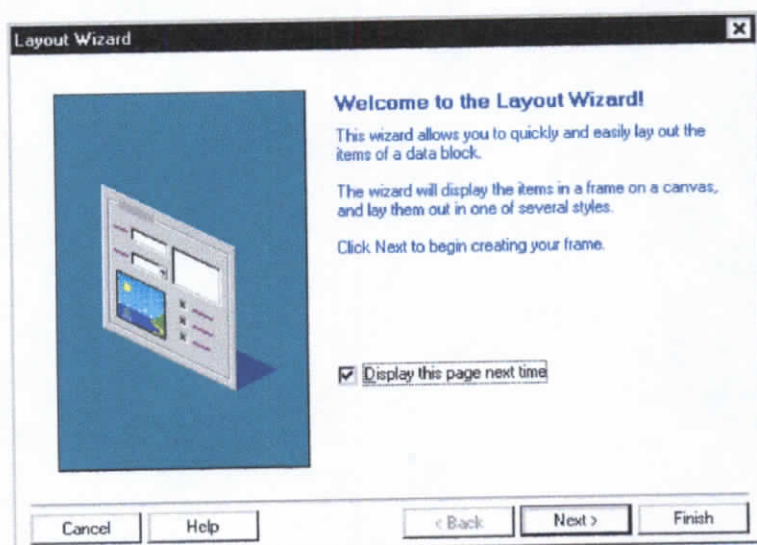


Fig. 3.22. Ventana Asistente Layout

Hacer clic en el botón **Next**:

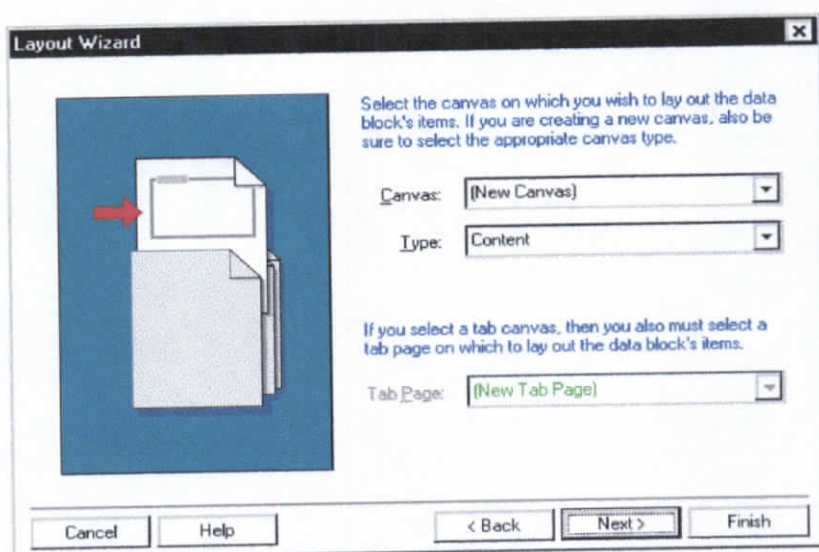


Fig. 3.23. Asistente Layout, Seleccionar el Lienzo

El layout para un bloque de datos puede ser colocado sobre cualquier lienzo existente, si existen, caso contrario solo la opción crear un nuevo lienzo está disponible.

Hacer clic en el botón **Next**:

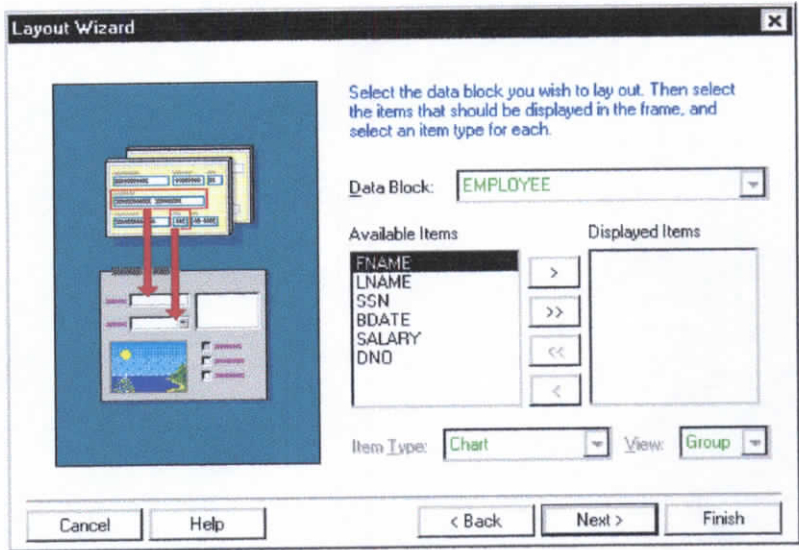


Fig. 3.24. Asistente Layout, Seleccionar Bloque de Datos

Las columnas de una tabla base sobre un bloque de datos, pueden ser adicionadas al layout. El bloque de datos que se está colocando se ilumina automáticamente, en este caso **EMPLEADO**. Mover todas las columnas desde **Available Items** hacia **Displayed Items** haciendo clic en la doble flecha derecha y luego en el botón **Next**.

En la siguiente ventana, los campos **Labels**, **Field Sizes** y **Field Heights** pueden ser alterados, cambiarlos y hacer clic en el botón **Next**:

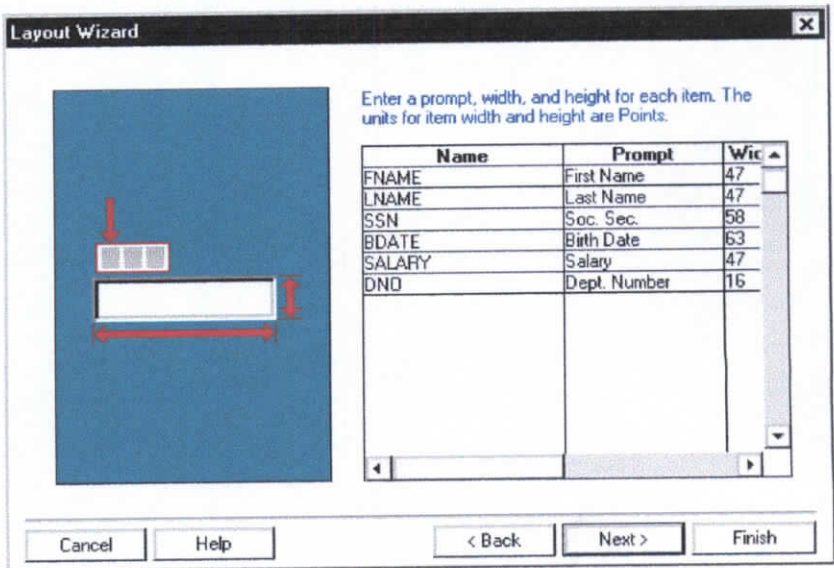


Fig. 3.25. Asistente Layout, Modificar Estilo de Campos

En la siguiente ventana:

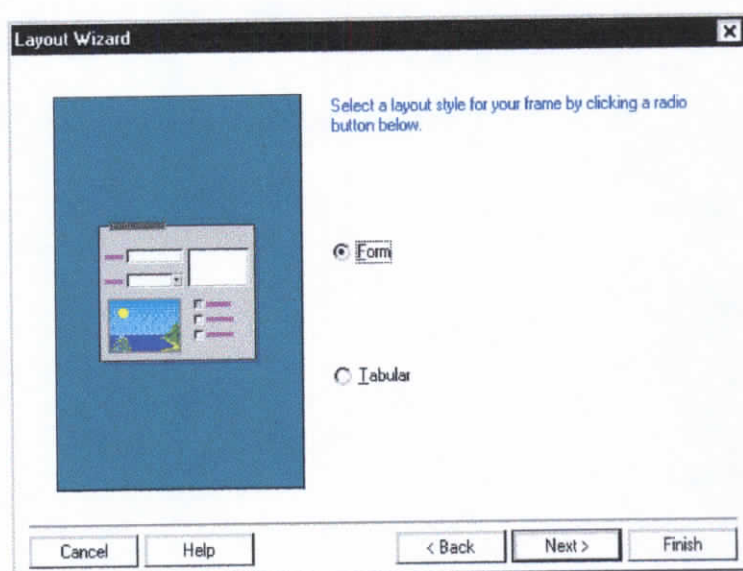


Fig. 3.26. Asistente Layout, Estilo del Layout

Hay dos estilos principales para las formas, **Tabular** arregla las etiquetas de campo en la parte superior de la forma y un número de registros hacia abajo (como una hoja de cálculo); y **Form** coloca las etiquetas de campo a la izquierda de los campos. Generalmente un registro se visualiza a la vez.

Una vez seleccionado el estilo, presionar el botón **Next**.

En el siguiente diálogo, ingresar un título para el cuadro (alrededor del bloque de datos) y verificar la opción para incluir una barra de desplazamiento.

El número de registros visualizados en una **Forma** debería ser 1, mientras que en una **Tabular** puede ser más de 1.

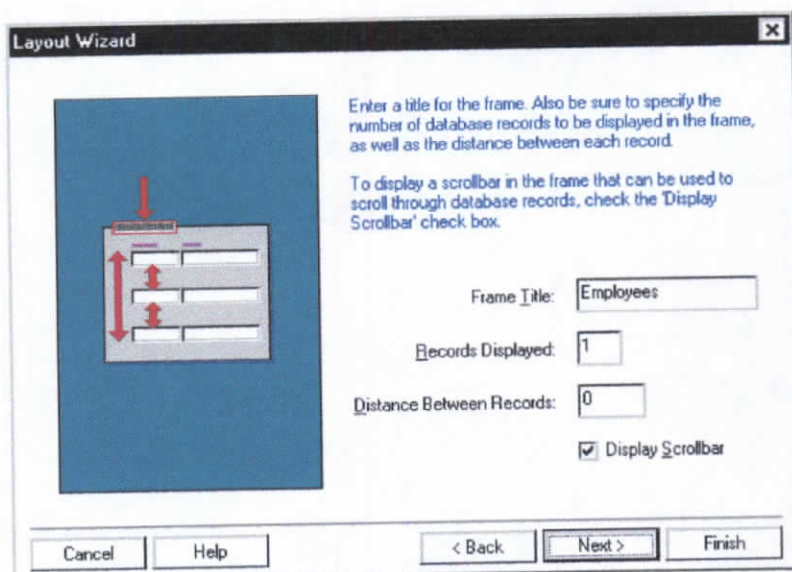


Fig. 3.27. Asistente Layout, Título y Número de Registros

Hacer clic en el botón **Next** y aparece la caja de diálogo final para del Asistente.

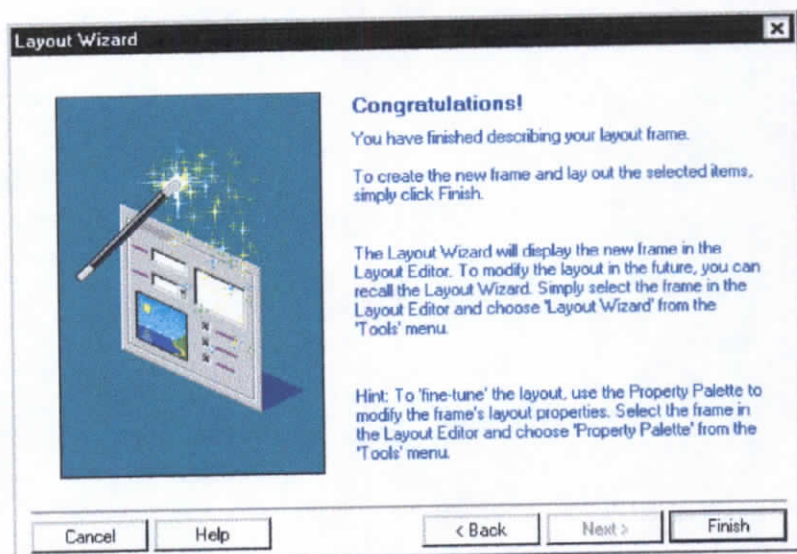


Fig. 3.28. Asistente Layout, Ventana Final

Hacer clic en el botón **Finish** para crear el layout. Un nuevo **Lienzo** será creado, y debe aparecer el **Editor Layout**, y aparece la nueva forma con el bloque de datos.

3.3.1.5.- Editor Layout

Para ver la forma actual, sus bloques y campos, seleccionar **Tools | Layout Editor** (sino se encuentra visualizada).

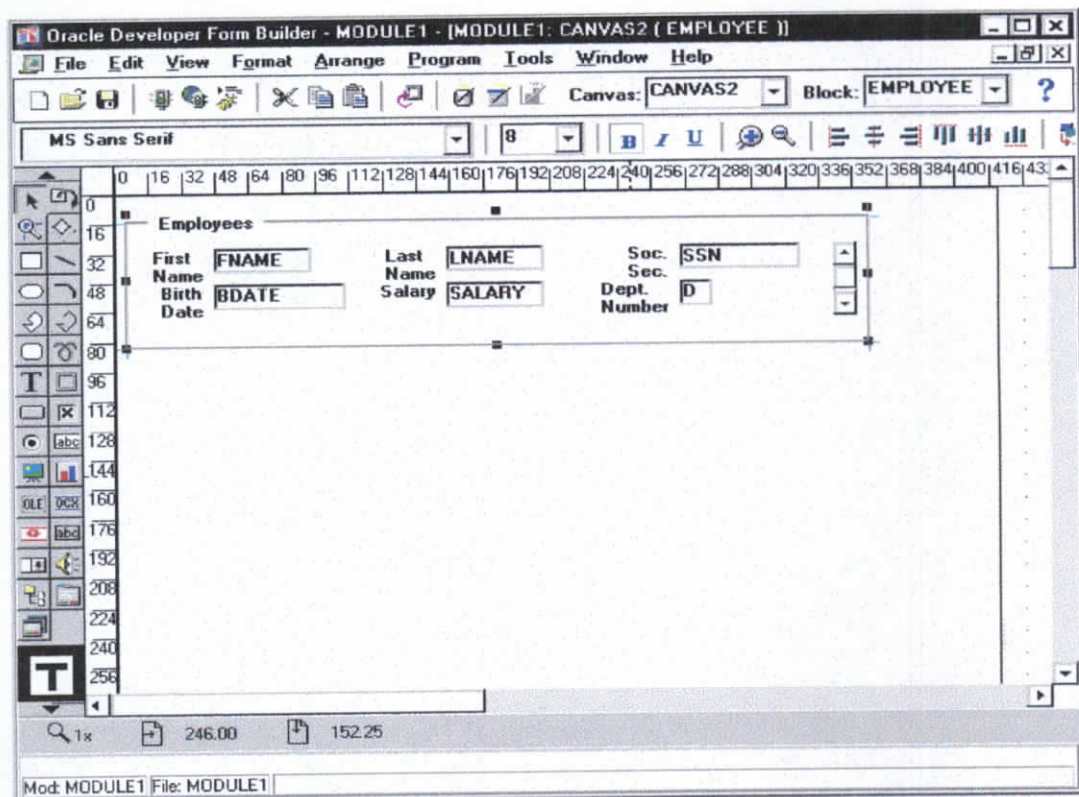


Fig. 3.29. Editor de Layout

En el Editor, hacer clic y arrastrar los campos y etiquetas para moverlos. Otros textos líneas, cajas, etc., pueden ser adicionados con la herramienta **Paleta** del lado izquierdo de la ventana.

Algunas operaciones incluyen:

- ☒ Mover un campo o etiqueta de campo, con clic y arrastrar el ítem con el botón izquierdo del ratón.
- ☒ Cambiar el texto de una etiqueta, seleccionar la herramienta texto **T** y hacer clic sobre la etiqueta. Para finalizar la edición hacer clic fuera del texto.

- ✓ Adicionar texto al bloque, seleccionar la herramienta texto  y hacer clic sobre un área abierta. Ingresar el nuevo texto, para cambiar el tipo de letra, marcar el texto, y seleccionar **Format | Font**. Para cambiar el color del texto de una etiqueta, usar la herramienta apuntador  para marcar la etiqueta, y luego hacer clic en el botón **Text Color** .
- ✓ Cambiar el ancho de un campo haciendo clic en el campo, arrastrar uno de los marcadores.

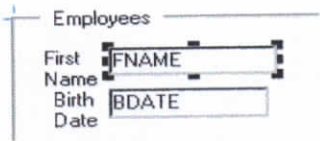


Fig. 3.30. Modificar ancho de Campo

- ✓ Usar la herramienta Zoom In  para acercar la forma, o la herramienta Zoom Out  para alejar la forma.

Visualizar Propiedades

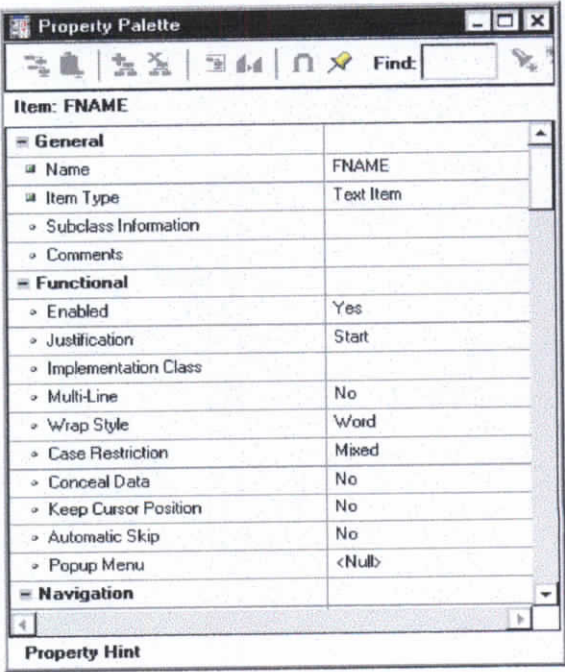


Fig. 3.31. Ventana de Propiedades

En el Editor de Layout, hacer doble clic sobre un objeto para ver las **Propiedades del Objeto**. En forma opcional, hacer clic sobre un objeto con el botón derecho del ratón y seleccionar la opción **Properties**.

Cada tipo de objeto tiene un diferente grupo de propiedades, así: al hacer doble clic sobre el bloque de datos **EMPLEADO** en el Navegador de Objetos, se visualiza las propiedades del bloque de datos.

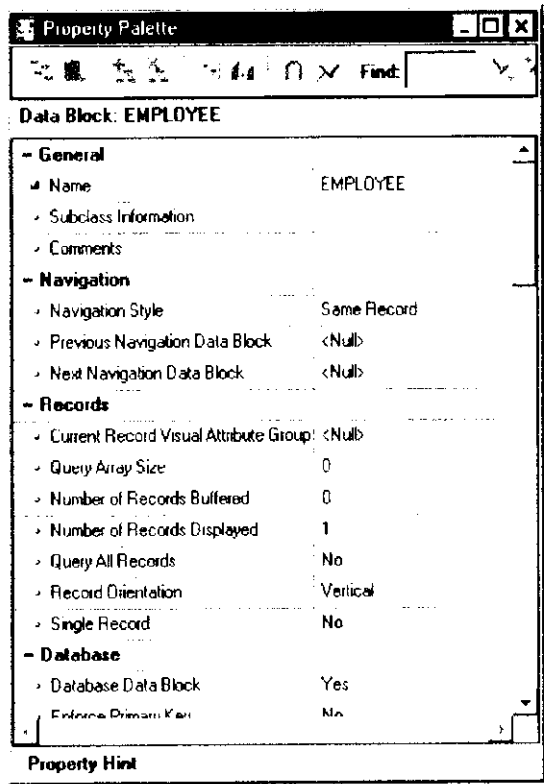


Fig. 3.32. Ventana de Propiedades Bloque EMPLEADO

Algunas propiedades de interés para un bloque de datos son:

- ☑ **Cláusula WHERE:** filtra la selección de filas desde la tabla base.
- ☑ **Cláusula ORDER BY:** altera el orden de los registros visualizados.
- ☑ **Delete Allowed, Insert Allowed, Query Allowed y Update Allowed:** permiten o previenen dichas acciones. Por ejemplo, al colocar **Delete Allowed** a **FALSE** se previene a los usuarios borrar registros de la tabla base correspondiente.

Hay otras propiedades que afectan los tipos de letra, colores, tamaño y posición de los objetos en el bloque.

3.3.1.6.- Grabar, Compilar y Ejecutar Formas

Las formas pueden ser grabadas como archivos en un directorio. El **código fuente** para una forma se almacena con la extensión ***.fmb**, y el **Compilado** con la extensión ***.fmx**.

En Windows 95/98/NT/2000, las formas pueden ser grabadas en una carpeta (subdirectorio) del disco duro local, en un servidor de archivos o en un disquete. Se recomienda mantener directorios separados para formas de diferentes proyectos.

3.3.1.7.- Grabar una Forma

Para grabar una forma, seleccionar **File | Save**, y se presenta la ventana

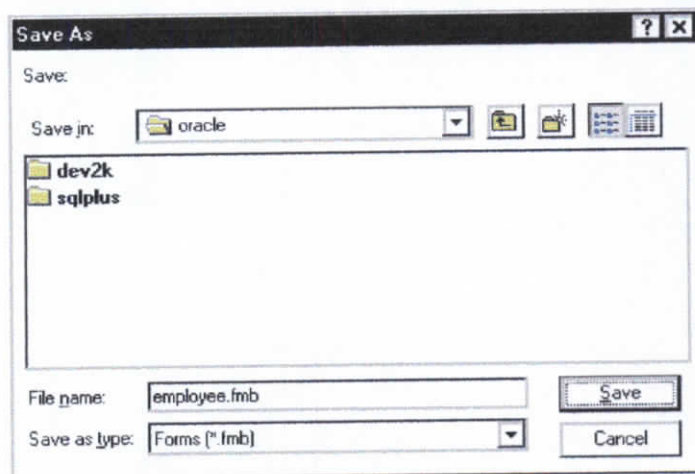


Fig. 3.33. Grabar una Forma

Notar que en las formas y otros archivos de la compañía u organización, podrían ser grabados en disquete o en un directorio especial. Ingresar el nombre de

archivo en el campo **File Name:**, con la extensión **.fmb**, luego hacer clic en el botón **Save**. Por ejemplo **empleado.fmb**.

Para mejorar la portabilidad de las formas entre diferentes sistemas operativos, usar nombres de archivo que comiencen con una letra, no contengan espacios o signos de puntuación (excepto el caracter subrayado), con letras minúsculas y que sean relativamente cortos.

Si se está trabajando en una forma y se desea grabarla con un nombre diferente, seguir los mismos pasos anteriores, seleccionando la opción **Save As** en lugar de **Save**.

3.3.1.8.- Compilar/Construir una Forma

Antes que una forma pueda ser ejecutada, debe ser **compilada** y **construida**. Al compilar se ejecuta un compilador **PL/SQL** en todos los procedimientos y disparadores de la forma, al construir se crea el archivo **.fmx** que sirve para ejecutar la forma.

Para compilar una forma, primero asegurar que está grabada. Seleccionar **Program | Compile**, y luego **All**.

Si una forma tiene errores (en el código **PL/SQL**, o si el esquema de la base de datos cambia) se visualizan en una ventana. Un registro del proceso de compilación, incluyendo los mensajes de error, se mantiene en un archivo con la extensión **.err**, con el mismo nombre de la forma.

Para construir el archivo **.fmx** para la forma, seleccionar **File | Administration**, y escoger **Compile File**. Si la generación de la forma es satisfactoria, la frase **Module built successfully** aparece en la barra de estado.

Alternativamente, Oracle Forms por omisión construye automáticamente la forma cada vez que es ejecutada desde el diseñador de formas. Para ver los detalles seleccionar **Tools | Preferences**.

3.3.1.9.- Ejecutar una Forma

Luego que la forma es grabada y construida, puede ser ejecutada con la opción **Program | Run Form** y seleccionar **Client/Server**. Se ejecuta el programa **Oracle Runform** con el nombre de la forma como parámetro.

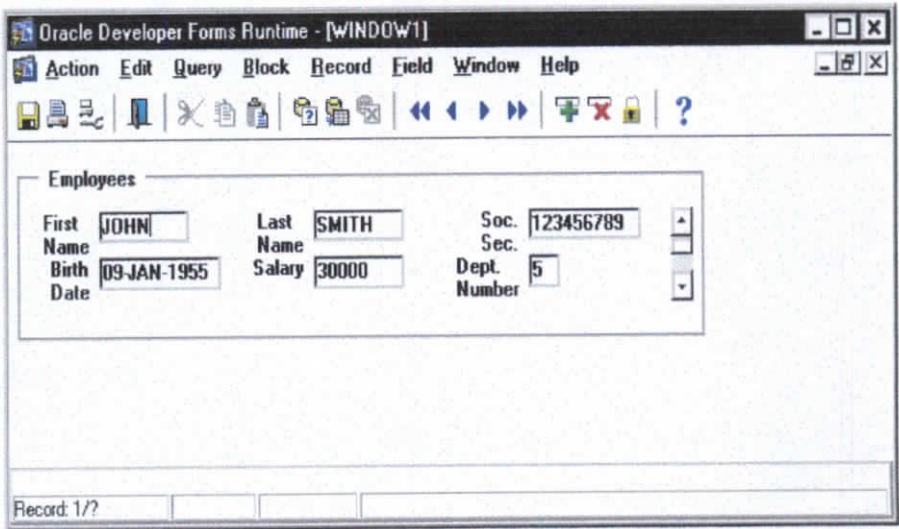


Fig. 3.34. Ejecutar una Forma

Usar las opciones y herramientas para consultar la forma, ingresar nuevos datos y grabar datos nuevos y cambiados.

3.3.1.10.- Ejecutar una Forma como Aplicación Java o Applet

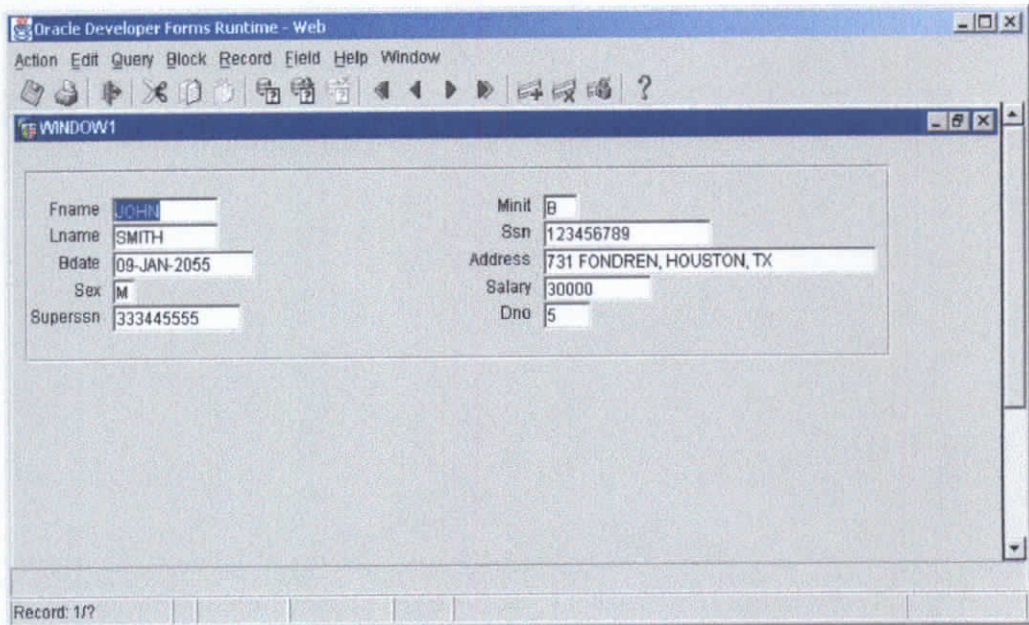


Fig. 3.35. Forma como Aplicación Java o Applet

Si se tiene instalado **Forms Server** (sólo en Windows NT/2000) se puede también ejecutar la forma en modo **Web**, al seleccionar **Program | Web**. En Oracle Forms, el archivo **.fmx** es cargado y ejecutado en una Aplicación Java. En este modo, el servidor de Forms maneja el procesamiento del negocio lógico, e interactúa con los elementos de la interfase de usuario que se ejecutan en la Aplicación Java.

Otra opción para ejecutar **Oracle Forms** es desde un navegador Web como un Applet Java, para lo cuál es necesario contar con un servidor Web (tal como **Oracle Application Server** o **Oracle WebDB**). Oracle provee su propio medio ambiente de tiempo de ejecución Java (JRE Java Runtime Environment) llamado **J-Initiator** que debe ser instalado en cada cliente.

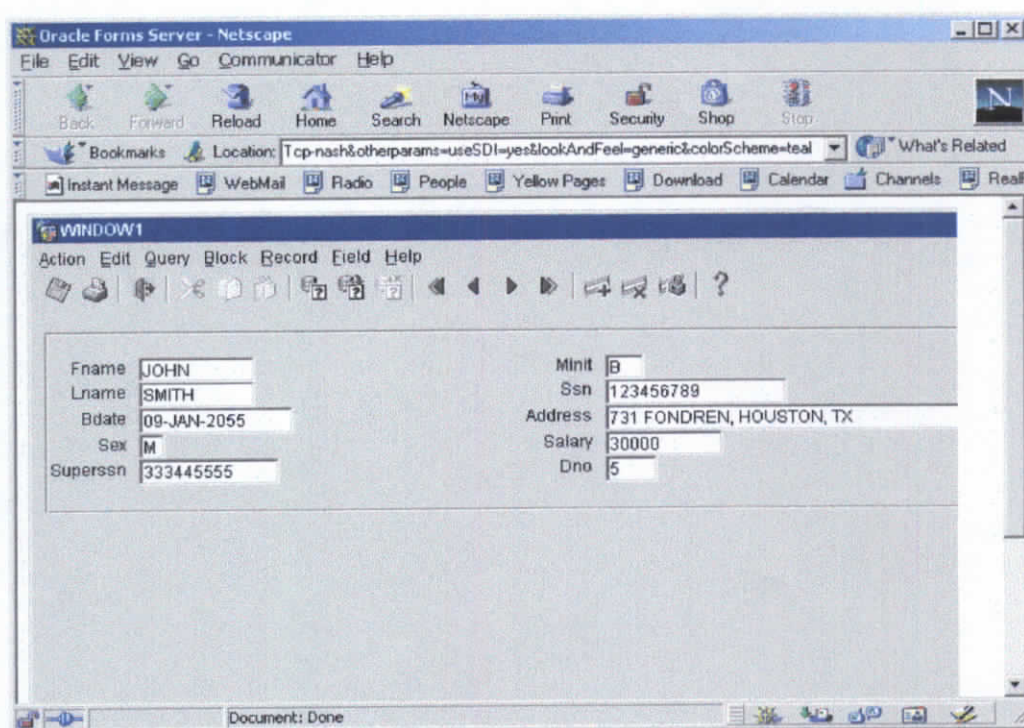


Fig. 3.36. Oracle Forms Server

Forms provee una página Web simple de prueba, ubicada bajo el grupo de programas **Oracle Forms**, llamada **Run a Form on the Web**, que verifica si el servidor Forms, tiene una carga balanceada, y la capa intermedia se está ejecutando correctamente.

3.3.1.11.- Consultas

Cuando la forma se ejecuta por primera vez, no aparecen registros, el usuario puede comenzar a ingresar nuevos registros de datos.

Para realizar una consulta, el usuario debe ingresar un criterio, valores para los campos, que vienen a ser parte de la consulta y son suministrados a la base de datos, hacer clic en el botón **Query** o seleccionar **Query | Enter Query** para ingresar al modo de consulta, la forma es limpiada y el usuario puede navegar en los distintos campos.

Al hacer clic otra vez en el botón **Query**, se ejecuta la consulta actual con cualquier dato suministrado como criterio en una cláusula **WHERE**. Si no existe un criterio, todos los registros en la tabla son visualizados. Por ejemplo, al ingresar el valor **5** en el campo **DNO** mientras se está en modo de consulta, causa que la consulta **SQL** siguiente sea suministrada a la base de datos:

```
SELECT Nombre, Apellido, ssn, bdate, Sueldo, dno
FROM Empleado
WHERE dno = 5;
```

La mayoría de comandos de las opciones y de la barra de botones, pueden ser activados con combinaciones de teclas o teclas de función. Para Windows 95/98/NT/2000:

F7 Coloca la forma en modo de consulta

F8 Ejecuta una consulta

Para ver una lista de las teclas y sus funciones seleccionar: **Help | Keys**.

Una vez consultados los datos (recuperados de la base de datos), son almacenados en un buffer del bloque de datos asociado en la forma, significa que una copia local de los datos se almacena en la memoria del cliente. Se colocan bloqueos apropiados sobre los datos de forma que el cliente mantiene una vista consistente de los datos en las tablas asociadas.

El usuario puede navegar a través de los datos en un bloque, presionando las teclas fecha arriba y fecha abajo para avanzar entre varios registros. Para cambiar los datos en una forma, el usuario puede posicionarse sobre el campo de interés, e ingresar el nuevo dato.

Para ingresar nuevos datos en la forma, el usuario puede avanzar hasta el último registro, moverse hasta un registro en blanco, ingresar los datos en los campos.

Para grabar los cambios y nuevos registros en la forma, seleccionar **Action | Save**, cualquier registro cambiado se envía a la base de datos como una sentencia **UPDATE** de **SQL**, y cualquier nuevo registro es enviado como una sentencia **INSERT**. Estas sentencias son suministradas como una simple unidad lógica de

trabajo. Así, si una falla ocurre durante cualquiera de estas sentencias, la transacción completa será vuelta atrás (**ROLLBACK**).

Si el usuario intenta limpiar el bloque o salir de la forma, y hay algunos cambios que todavía no han sido aplicados a la base de datos, se solicita hacer **COMMIT** los cambios antes de salir o limpiar la forma. El bloque es limpiado cuando se ingresa al modo de consulta. Un bloque también puede ser limpiado con las opciones **Clear Block** y **Clear Form**.

Para salir de una forma en ejecución, seleccionar **Action | Exit**.

Un grupo de funciones básicas que se pueden desarrollar en cualquier forma son:

Funciones	Notas
Consultar una Forma	Para ver todos los registros en la tabla, simplemente Ejecutar la consulta (opción Query o presionar F8 en Windows). Para filtrar el registro, ingresar al modo consulta , especificar los parámetros en los campos y Ejecutar la Consulta .
Insertar Nuevos Registros	Avanzar hasta el último registro hasta obtener un registro en blanco. Ingresar datos para cada campo y luego hacer Commit los cambios con la opción Action Save . Para formas master/detalle, primero consultar un registro master o insertar y grabar un registro master (en el bloque de datos master), luego navegar al bloque detalle e insertar nuevos registros.
Actualizar Registros Existente	Consultar la base de datos y avanzar al registro de interés. Cambiar los valores en los campos y grabar los cambios. Para formas master/detalle, los cambios para la clave en el bloque master pueden ser prevenidos si existen registros detalle (para asegurar la integridad relacional).
Borrar Registros	Consultar la base de datos y avanzar hasta el registro de interés. Seleccionar Records Delete record , luego grabar los cambios. Para formas master/detalle, al intentar borrar un registro master cuando existen registros detalle resulta en un error. En tales casos, todos los registros detalle deberían primero ser borrados y luego el registro master.

Tabla 3.1. Funciones Básicas dentro de una Forma

3.3.1.12.- Formas

Master/Detalle

Una forma Master/Detalle es aquella que contiene dos bloques ordenados en una relación master-detalle.

La relación Master/Detalle es una relación común entre entidades, que en un diagrama entidad relación se conoce como **Uno a Muchos (Padre – Hijo)**. En el diseño físico de la base de datos, un simple registro **Master** de una tabla, referencia a uno o más registros **Detalle** en otra tabla. Un registro en la tabla detalle se relacionará exactamente a un registro master de la tabla master. Ejemplos:

- ☒ Una Orden con muchos Items.
- ☒ Un Departamento con muchos Empleados.
- ☒ Un Empleado con muchos Dependientes.
- ☒ Una Compañía con muchos Oficinas.

Oracle Forms implementa la relación master - detalle usando dos bloques de datos, el primero corresponde a la tabla master y el segundo corresponde a la tabla detalle. Las funciones principales en una forma Master-Detalle son:

- ☒ **Oracle Forms** coordina valores entre los dos bloques a través de una serie de **disparadores** en el ámbito de forma y bloque.
- ☒ **Oracle Forms** garantiza que el bloque detalle visualizará solo registros que estén asociados con el registro actual del bloque master.

Notar que una forma Master - Detalle es simplemente una forma de ver los datos en dos tablas relacionadas. **Forms** no afecta el esquema en términos de crear, eliminar o forzar restricciones de integridad referencial en el ámbito de la base de datos.

3.3.1.13.- Crear una Forma Master - Detalle

Crear una forma que permita al usuario consultar un departamento de la compañía y todos los empleados en dicho departamento.

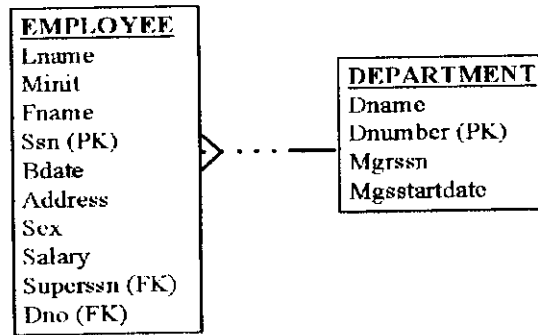


Fig. 3.37. Modelo Entidad Relación Uno a Muchos

3.3.1.14.- Crear el bloque Master

En el Navegador de Objetos, hacer clic en **Forms**, crear una nueva forma con **File | New**, escoger **Form**.

Crear un nuevo bloque llamado **DEPARTAMENTO**, que contenga todas las columnas de la tabla **DEPARTAMENTO**:

1. Seleccionar **Tools | Data Block wizard**.
2. Crear un bloque de datos para una tabla/vista.
3. Especificar la tabla **DEPARTAMENTO** y seleccionar todas las columnas.
4. Crear el bloque de datos y pasar al Asistente de Layout.
5. Aplicar el bloque de datos **Departamento** a un nuevo lienzo.
6. Adicionar todas las columnas como Items Visualizados.
7. Cambiar las etiquetas para los campos
8. Escoger **Form layout**.

- 9. Especificar al cuadro el título **Departamentos** y seleccionar solo 1 registro a ser visualizado.
- 10. Grabar la forma como **departamento1.fmb**, compilar y ejecutar.
- 11. Realizar consultas para verificar su funcionamiento.

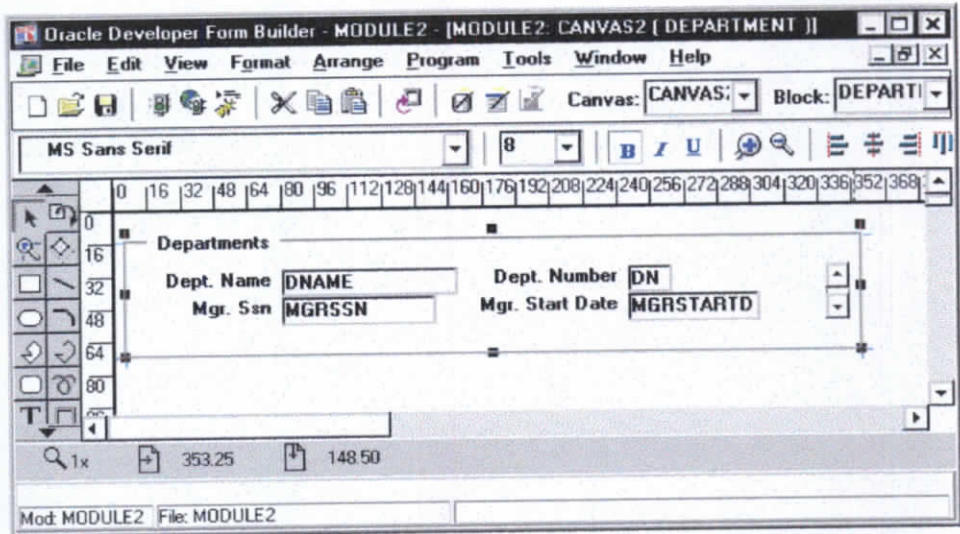


Fig. 3.38. Bloque de Datos Master

3.3.1.15.- Crear el Bloque Detalle

Una vez creado el bloque **Master**, se puede crear el bloque **Detalle EMPLEADO** y asociarlo con el bloque master. Los pasos a seguir son:

- 1. Seleccionar **Tools | Object Navigator**.
- 2. En el Navegador de Objetos, hacer clic sobre **Data Blocks** de la forma **DEPARTAMENTO1**.
- 3. Seleccionar **Tools | Data Block wizard**.
- 4. Seleccionar la tabla **EMPLEADO** e incluir las columnas.
- 5. Como ya existe un bloque de datos en la forma, el siguiente paso en el asistente es crear una relación entre el bloque de datos existente (**DEPARTAMENTO**) y el nuevo bloque que se está creando.

El asistente puede construir la relación, sobre la base de las restricciones en el ámbito de tabla del esquema de la base de datos, pero el desarrollador puede también especificar la relación en forma manual. Quitar la selección de la opción **Auto-join data blocks**. Hacer clic en el botón **Create Relationship** para listar los bloques de datos disponibles.

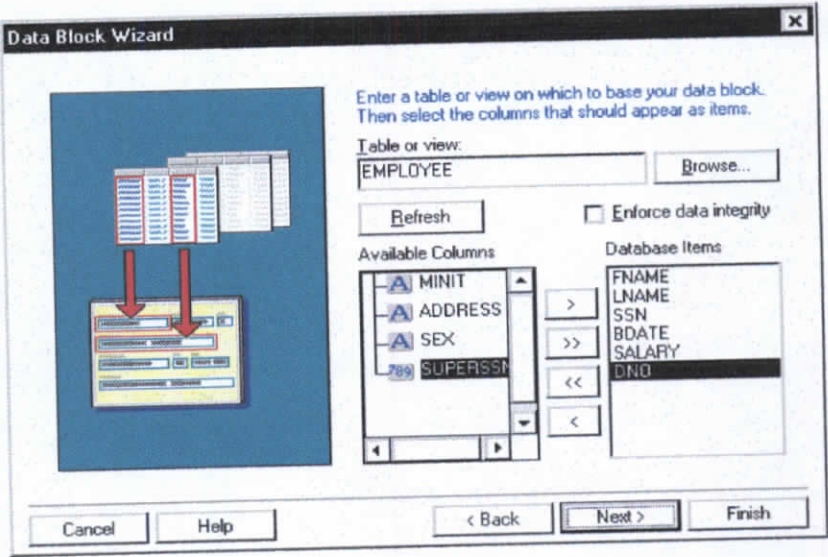


Fig. 3.39. Asistente Bloque de Datos

En la caja de diálogo **Relation Type**, escoger **Based on a Join Condition** y hacer clic en el botón **Ok**.

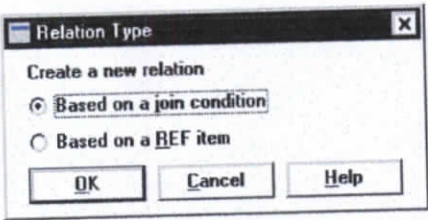


Fig. 3.40. Tipo de relación

Cuando la lista de bloques aparece, escoger el bloque de datos **DEPARTAMENTO**.

Definir el ítem Detalle y el ítem Master, y la condición de unión:

Empleado.Dno = Departamento.Dnumero

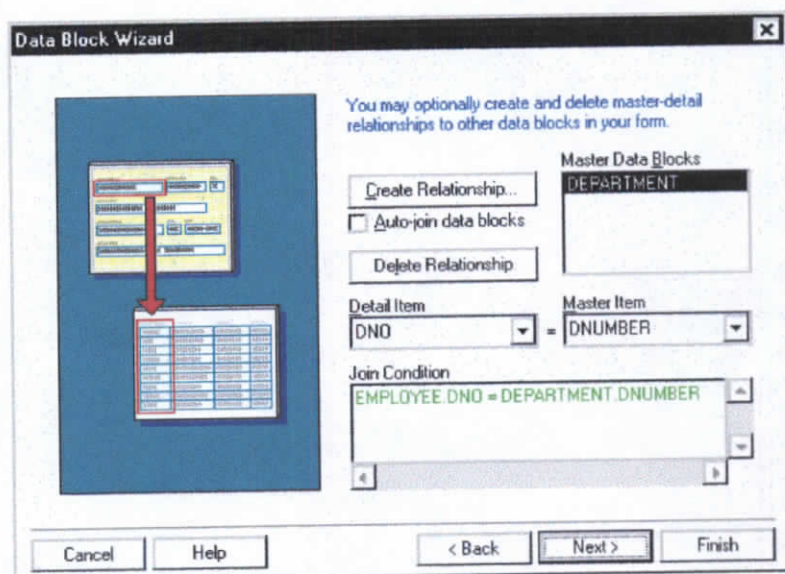


Fig. 3.41. Relación Master - Detalle

6. Crear el bloque de datos y llamar al asistente Layout.
7. Asegurar de escoger el lienzo existente (CANVAS2) e incluir todos los items, (excepto la llave foránea) a ser visualizados.
8. Cambiar las etiquetas para los campos y escoger **Tabular layout**.
9. Ingresar el nombre del cuadro **Empleados** y seleccionar el número de registros a ser visualizados con una distancia 0 entre registros.
10. Grabar la forma, compilar y ejecutar.

Para navegar entre los bloques Master y Detalle, usar:

- ☑ Bloque Siguiente: **CTRL-PageDown** o seleccionar **Block | Next**
- ☑ Bloque Previo: **CTRL-PageUp** o seleccionar **Block | Previous**

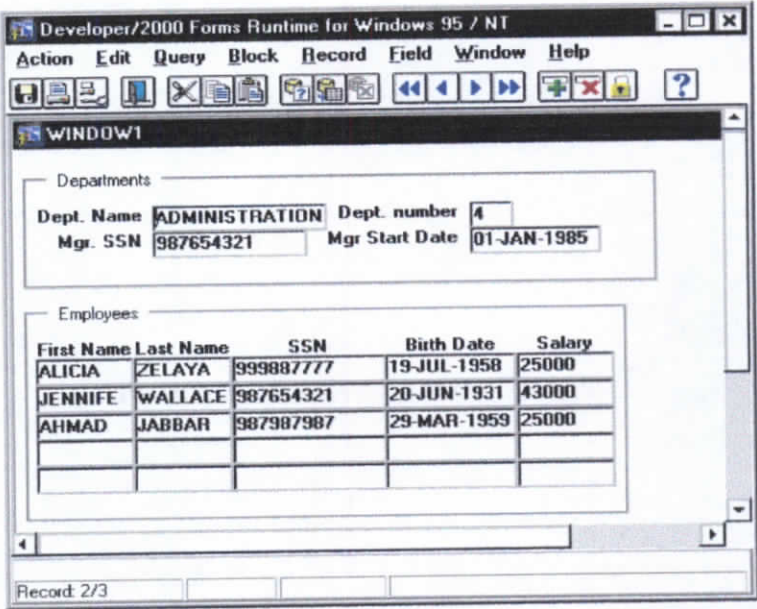


Fig. 3.42. Forma Master – Detalle

3.3.1.16.- Propiedades de Relación

Hay un número de propiedades en una forma Master/Detalle que pueden ser cambiados, el Navegador de Objetos tiene varios nuevos objetos, incluyendo relaciones.

Para ver las propiedades para la relación **Departamento_Empleado**, abrir el bloque **Departamento** y luego abrir el bloque **Relations**, haciendo clic en los símbolos +. Luego hacer clic en la relación **Departamento_Empleado** con el botón derecho del ratón y seleccionar **Properties**.

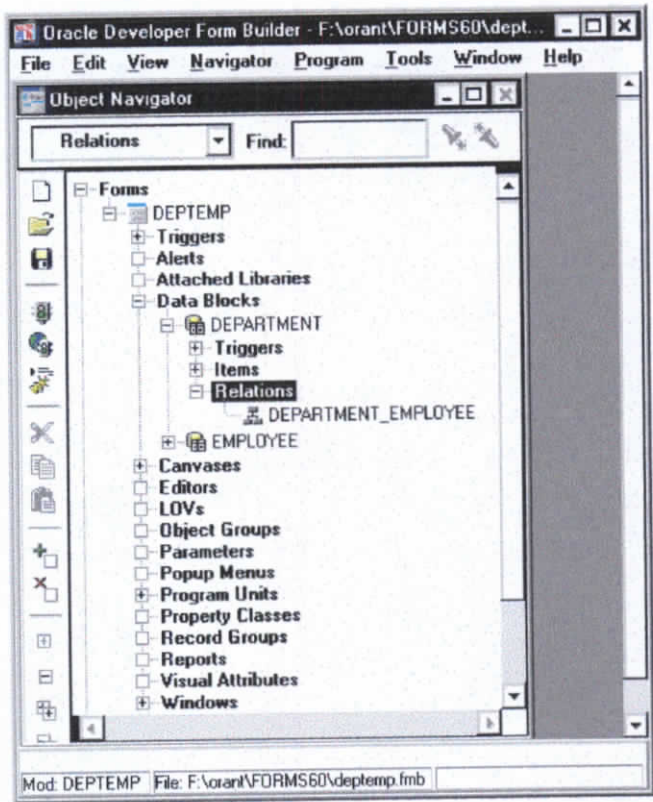


Fig. 3.43. Propiedades de la relación Departamento-Empleado

Las propiedades principales son:

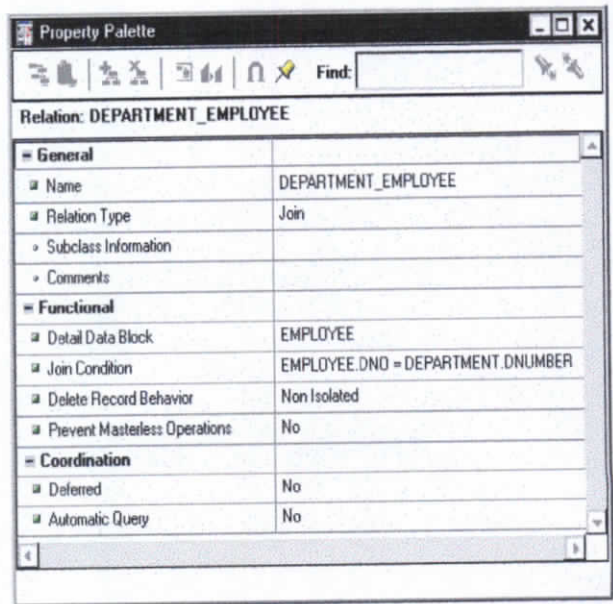


Fig. 3.44. Principales Propiedades de Master/Detalle

- ☒ **Name:** nombre de la relación, generalmente los nombres de los bloques.

- ☑ **Relation Type:** tipo de relación: **Join** o **Ref**. Una relación tipo **Join** usa el típico Join de **SQL** (en la cláusula **Where**) para colocar las dos tablas juntas (bloques de datos). La relación tipo **Ref** es usada para abstraer tipos de datos y referencias de objetos.
- ☑ **Detail Data Block:** nombre del bloque de datos detalle especificado, y cuando fue creado.
- ☑ **Join Condition:** la condición por defecto para las consultas a la base de datos, especificada cuando se creó el bloque de datos detalle.
- ☑ **Delete Record Behavior:** se usa para especificar cómo el borrado de un registro en el bloque master afecta los registros en el bloque detalle. Las posiciones son:
 - **Non-isolated:** previene el borrado de un registro master si existen registros detalle asociados en la base de datos.
 - **Isolated:** borrar el registro master, no afecta a los registros de detalle asociados en la base de datos.
 - **Cascading:** borrando el registro master, automáticamente se borra cualquier registro detalle asociado.
- ☑ **Coordination – Deferred:** Indica cuando los registros detalle deben ser consultados, al consultar un registro master.
 - **Yes:** la forma no consulta los registros detalle hasta que el usuario navegue al bloque detalle.
 - **No:** los registros detalle son alcanzados inmediatamente cuando el usuario consulta el registro master.

Diferido es colocado en **Yes**, en casos donde hay una gran cantidad de registros detalle para cada registro master. En tales casos, una gran cantidad de datos deben ser consultados y entregados al cliente, cada vez que un nuevo registro es visualizado en el bloque master. Cuando diferido es colocado en **Yes**, el usuario puede avanzar al registro master de interés y navegar al bloque detalle (**CTRL+PageDown**) para consultar los registros detalle.

- ☑ **Coordination - Auto-query:** Aplicado a consultas diferidas solamente

- **Yes:** la consulta es automáticamente ejecutada cuando el usuario navega al bloque detalle.
- **No:** la consulta debe ser ejecutada manualmente por el usuario luego que se pasa al bloque detalle.
- ☑ **Prevent Masterless operation:** especifica si los usuarios pueden consultar o insertar registros en un bloque detalle cuando un registro master no está colocado.
 - **Yes:** los usuarios no pueden consultar o insertar cuando no hay un registro master en su lugar.
 - **No:** los usuarios pueden consultar o insertar cuando no hay un registro master en su lugar.

Estas posiciones son usadas para **afinar** el desempeño de una forma master-detalle, en casos donde una gran cantidad de registros detalle están asociados con cada registro master, es una buena idea colocar **coordination-Deferred** a **Yes** para evitar la transferencia innecesaria de datos entre el servidor y el cliente. También ayuda a aumentar la velocidad de visualización de registros master, mientras el usuario avanza libremente a través de ellos, sin una pausa para consultar y entregar registros detalle.

3.3.1.17.- Unidades de Programa

Cuando se crea una forma Master-Detalle, se generan tres **Unidades de Programa** básicas (Procedimientos **PL/SQL**), los cuales son usados para coordinar la consulta y navegación de los registros detalle, cuando el registro master es cambiado (cuando el usuario avanza a diferentes registros master), actualizados o borrados. Estos procedimientos son listados en el Navegador de Objetos:

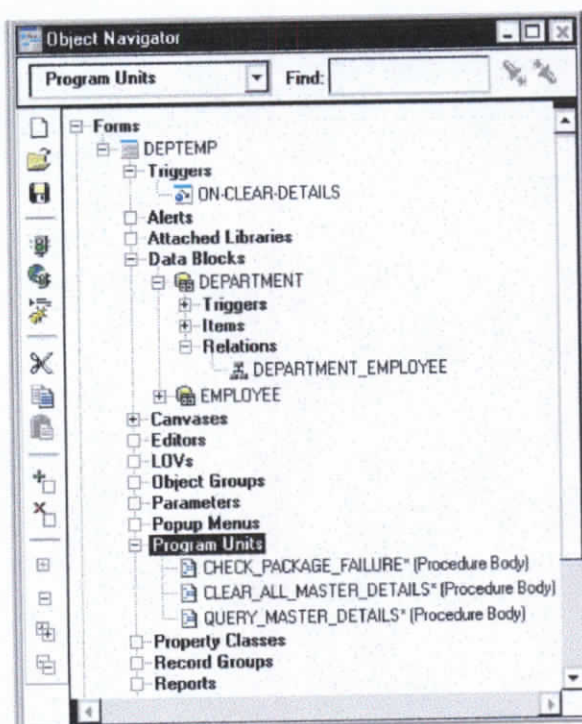


Fig. 3.45. Navegador de Objetos en Master/Detalle

PL/SQL es un lenguaje procedural de Oracle, extensión de **SQL**. Para ver el código **PL/SQL** para una Unidad de Programa, hacer clic derecho sobre el nombre de la unidad de programa y escoger **PL/SQL Editor**. Por ejemplo, el código para el procedimiento **CHECK_PACKAGE_FAILURE** es:

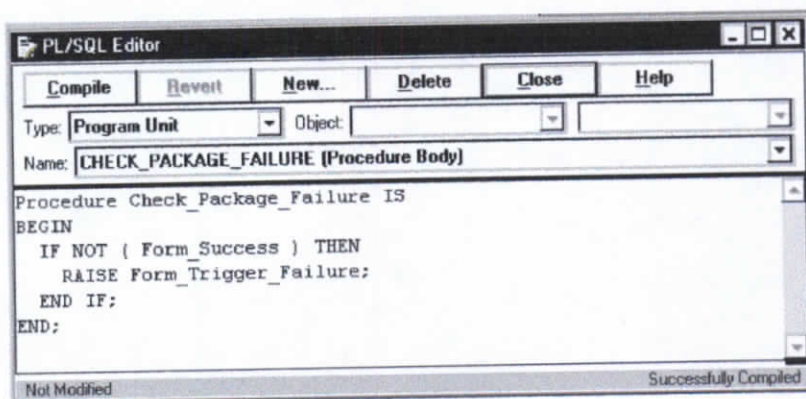


Fig. 3.46. Editor PL/SQL para una Unidad de Programa

El procedimiento **CHECK_PACKAGE_FAILURE** verifica si un comando previo o sentencia **SQL**, fue satisfactorio o falló.

El código **PL/SQL** para el procedimiento **QUERY_MASTER_DETAILS** es.

```

PROCEDURE Query_Master_Details(rel_id Relation,detail CHAR) IS
    oldmsg CHAR(2);    -- Posición del nivel de mensaje anterior
    reldef CHAR(5);      -- Relación de Posición Diferida
BEGIN
    --
    -- Inicializar Variable(s) Locales
    --
    reldef := Get_Relation_Property(rel_id, DEFERRED_COORDINATION);
    oldmsg := :System.Message_Level;
    --
    -- Si NO es diferida, Ir a detalles y ejecutar la consulta.
    --
    IF reldef = 'FALSE' THEN
        Go_Block(detail);
        Check_Package_Failure;
        :System.Message_Level := '10';
        Execute_Query;
        :System.Message_Level := oldmsg;
    ELSE
        --
        -- Relación es diferida, marcar el bloque detalle como no-coordinado
        --
        Set_Block_Property(detail,                                COORDINATION_STATUS,
        NON_COORDINATED);
    END IF;
EXCEPTION
    WHEN Form_Trigger_Failure THEN
        :System.Message_Level := oldmsg;
        RAISE;
END Query_Master_Details;

```

El procedimiento **QUERY_MASTER_DETAILS** es ejecutado siempre que el usuario navega al bloque detalle en una forma master - detalle. Si la propiedad **Deferred Coordination** está colocada en **False**, entonces el bloque detalle es automáticamente consultado.

Finalmente, el procedimiento **CLEAR_MASTER_DETAILS** es llamado cada vez que un nuevo registro master es recuperado (cuando el usuario avanza a un diferente registro master) o borrado. En este caso, si cualquiera de los registros detalle han sido insertados, actualizados o borrados, los cambios deben ser grabados en la base de datos antes que el bloque detalle sea limpiado. Si hay cualquier cambio pendiente de ser grabado, se solicitará al usuario grabar aquellos cambios antes que el bloque detalle sea limpiado.

3.3.1.18.- Extender Formas Master-Detalle

Las formas Master-Detalle pueden ser extendidas para incluir niveles adicionales de detalle. Por ejemplo, si se desea incluir los **Dependientes** como detalle del bloque **Empleados**. Los pasos indicados pueden ser repetidos a través de varios niveles de detalle que sean adicionados, cada uno de los cuales adiciona un nuevo bloque que sea relacionado al anterior.

3.3.1.19.- Listas de Valores (LOV) y Campos de Tablas No-Base

Para el usuario es incómodo tener que recordar varios códigos y números mientras se ingresan datos en una forma. Por ejemplo, en la forma **Empleado**, es difícil recordar todos los números de departamento cuando se inserta un nuevo empleado.

Una solución para este problema, es hacer una lista de valores apropiados cuando el usuario navega hasta el campo **Número de Departamento (DNO)**. En **Oracle Forms**, son llamados **Lista de Valores (LOV)**, que están basadas en un **Grupo de Registros** (consulta que retorna una colección de registros), que pueden ser usados para llenar bloques o **LOV** y pueden ser usados en procedimientos.

Cuando el usuario navega a un ítem con un **LOV** enlazado a él, la tecla **LOV** (**F9** en Windows) llama al **LOV**. Entonces, la consulta asociada con el grupo de registros se ejecuta, y los resultados son visualizados en una ventana pop up. Una vez que el usuario hace una selección de la lista, el valor o valores son retornados a la forma y colocadas en los campos apropiados.

Para crear un **LOV**, se siguen los siguientes pasos:

Hacer clic en el nodo **LOV** del Navegador de Objetos, luego seleccionar **Navigator | Create**. En el diálogo escoger si se desea crear el **LOV** en forma manual o si se desea usar el Asistente **LOV**.

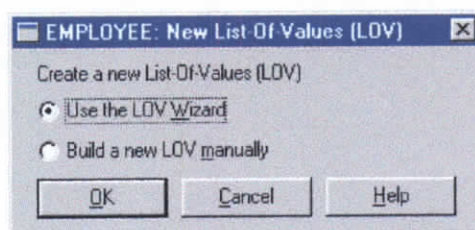


Fig. 3.47. Crear un nuevo LOV

Seleccionar **Use the LOV Wizard** y hacer clic en el botón **Ok**, para ingresar al asistente:

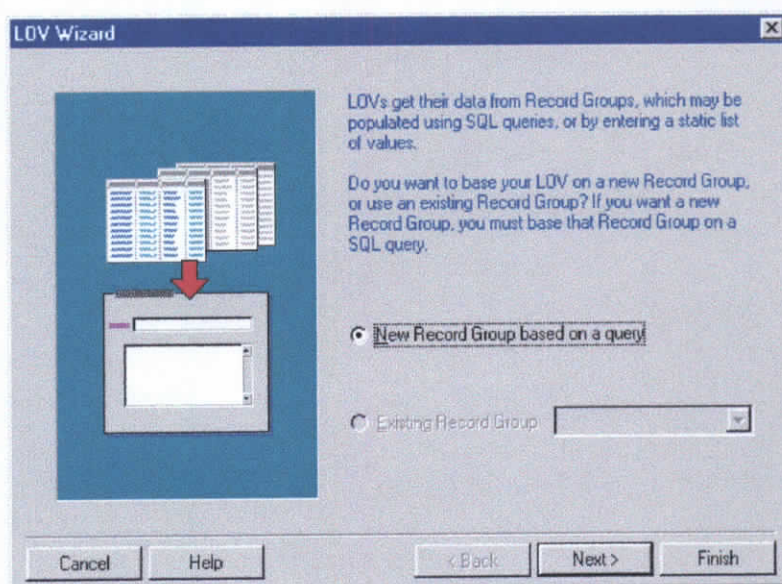


Fig. 3.48. Asistente LOV

Por omisión se crea un nuevo **Grupo de Registro**, presionar el botón **Next**.

Ingresar la consulta, por ejemplo: **SELECT Dnumero FROM Departamento**, y presionar el botón **Next**

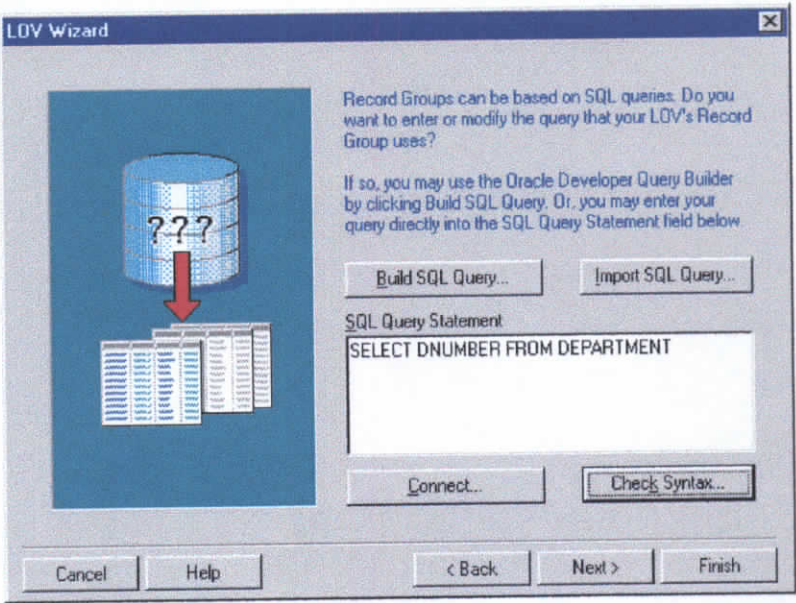


Fig. 3.49. Condición LOV

Luego, especificar cuales columnas en el grupo de registros serán retornadas para el **LOV**. En este caso, el **Número de Departamento**, y presionar **Next**.

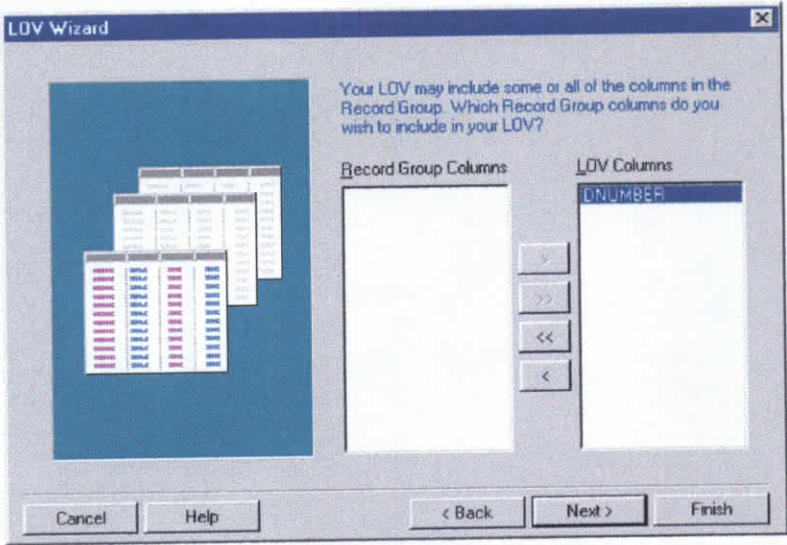


Fig. 3.50. Columnas LOV

Especificar las propiedades de visualización de las columnas **LOV**.

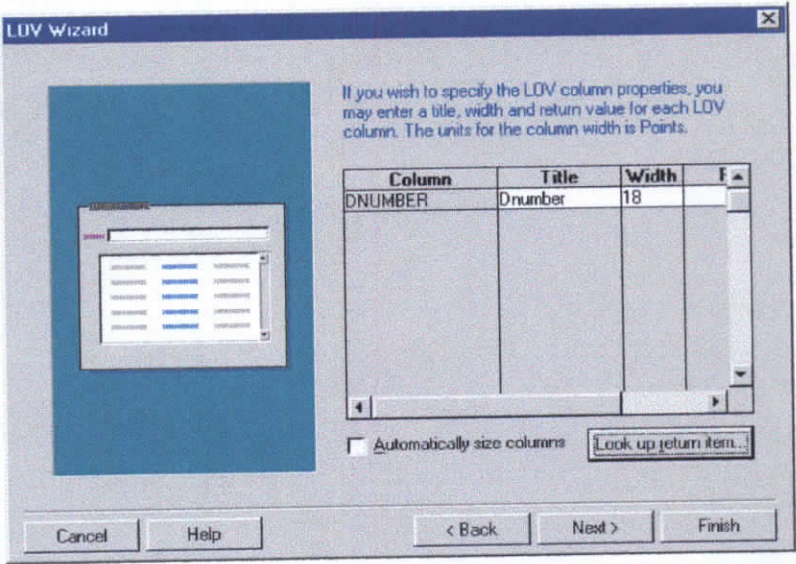


Fig. 3.51. Propiedades de Columnas LOV

Para colocar una columna LOV a un campo, hacer clic en la entrada **Dnumero** y luego hacer clic en el botón **Look up return Ítem....** Seleccionar el campo **Empleado.Dno** y hacer clic en el botón **Ok**.

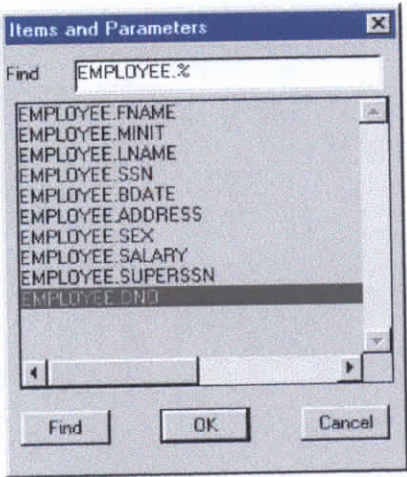


Fig. 3.52. Items y Parámetros LOV

Si el LOV contiene más de un ítem, debe también ser especificado, hacer clic en el botón **Next**.

Especificar un título para la ventana **LOV**, el tamaño y posición del **LOV**, y hacer clic en el botón **Next**.

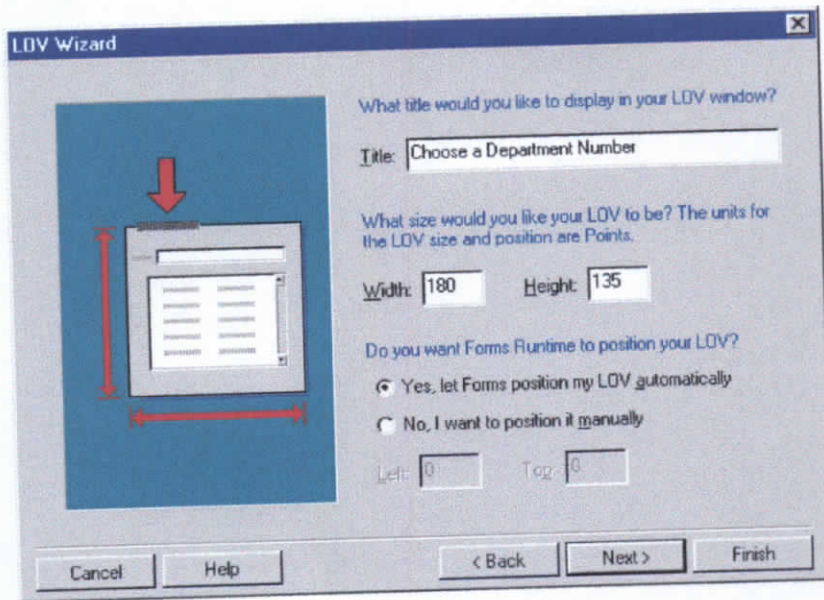


Fig. 3.53. Título y Posiciones de LOV

Las propiedades avanzadas, se refieren a la forma en que los registros deben ser traídos para el LOV. Si hay muchos registros retornados por el **LOV**, se puede considerar cambiar estas opciones, y presionar el botón **Next**.

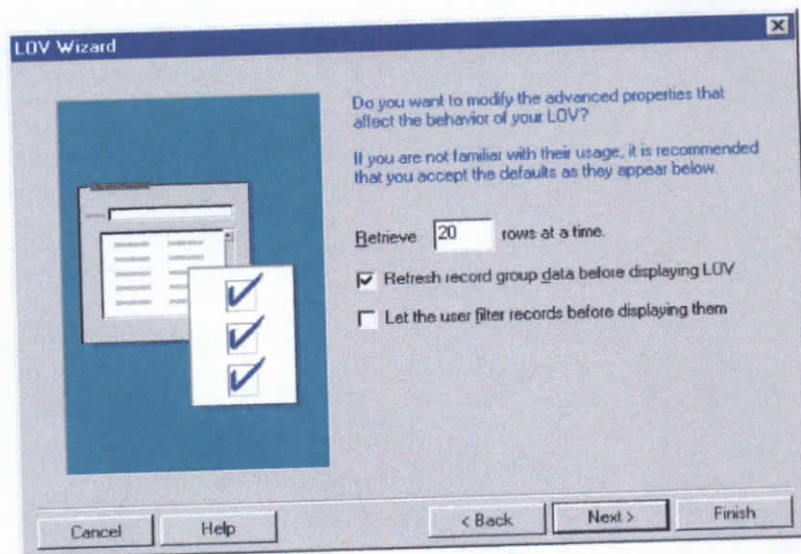


Fig. 3.54. Número de Registros LOV

En este paso final, el LOV retorna algunas columnas, y deben ser asignadas a campos en la forma. Se necesita asociar el **LOV** con un ítem particular de la forma.

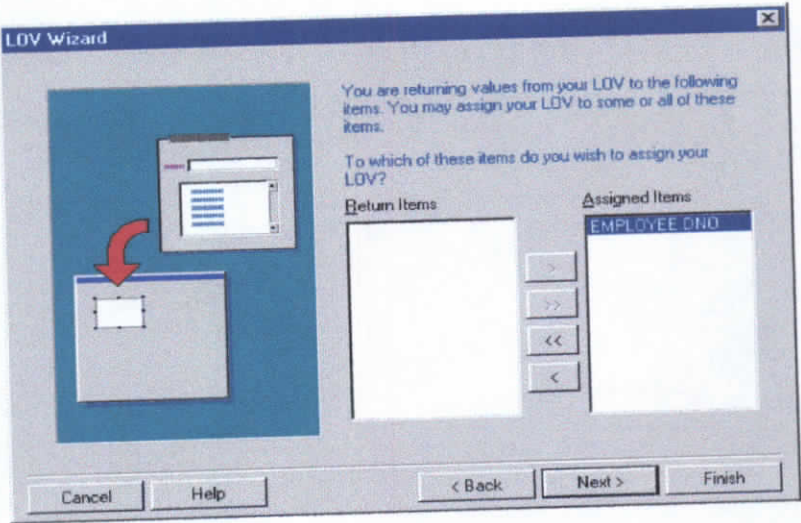


Fig. 3.55. Asociar LOV a Campos

Finalmente, hacer clic en el botón **Finish** para crear el Grupo de Registros y el LOV.

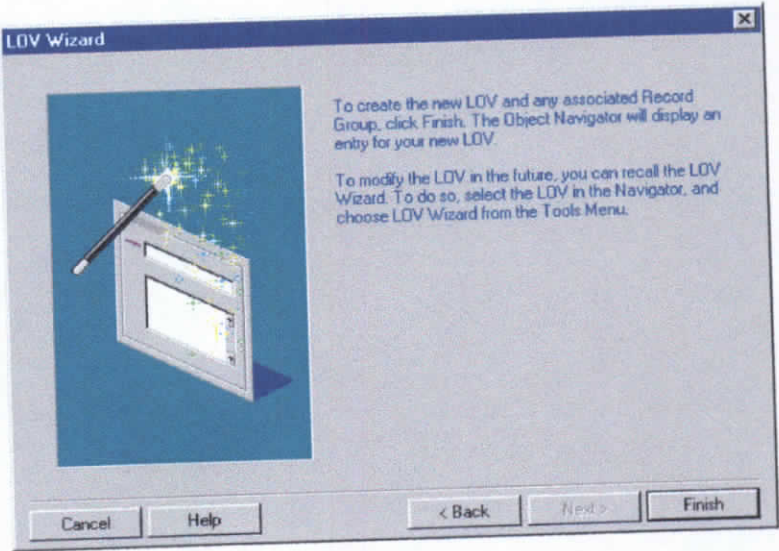


Fig. 3.56. Paso Final del Asistente LOV

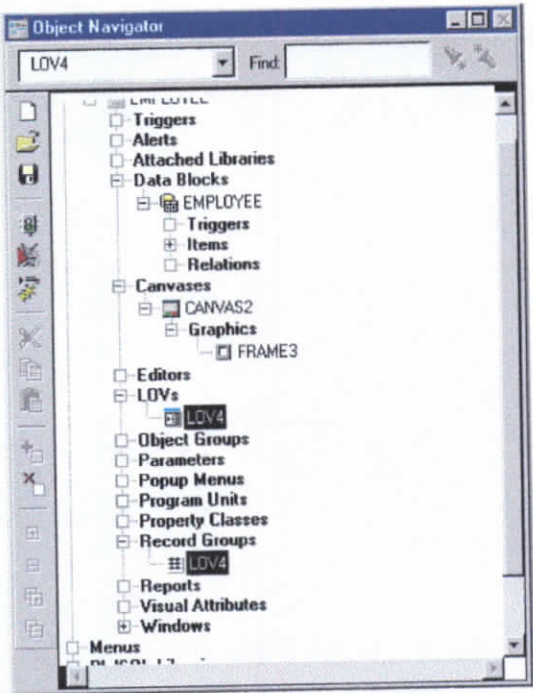


Fig. 3.57. LOV en Navegador de Objetos

Grabar, Compilar y Ejecutar la forma, al navegar hasta el campo **Número de Departamento**, aparece el mensaje: **List of Values** indicando que existe una lista disponible para este campo.

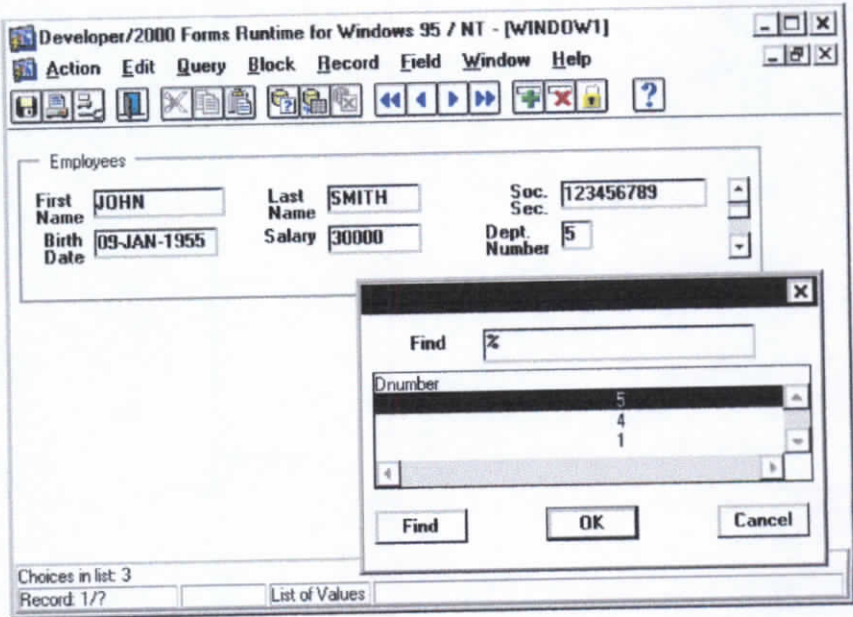


Fig. 3.58. Forma con LOV

3.3.1.20.- Campos Tabla No-Base

Se ha creado una **Lista de Valores**, pero no es clara para muchos usuarios, ya que únicamente se visualizan números.

Podría ser más claro si se incluye el nombre del departamento en la lista de valores pop up, pero éste no es parte de la tabla o bloque de datos **Empleado**. Por tanto, se necesita visualizar esta información para el bloque de datos **Empleado** en un campo **tabla no-base**.

Para adicionar el ítem y hacerlo funcional:

1. Primero, el nuevo ítem tendrá que ser adicionado al bloque de datos **Empleado**, que será llamado por ejemplo **Nombre_Dept**.
2. El campo **Nombre_Dept** debe ser llenado con el nombre de departamento, cada vez que el valor del campo **DNO** cambie. Un **trigger** será creado para manejar esta funcionalidad.
3. Finalmente, la lista de valores será cambiada para incluir el Nombre del Departamento en el **LOV**.

Para adicionar un nuevo ítem a un bloque de datos existente:

- ☒ Cambiar al Editor Layout, seleccionar **Tools | Layout Editor**.
- ☒ Hacer clic en la herramienta **Text Ítem** , y crear un nuevo campo en el bloque **Empleado**, haciendo clic y arrastrando el ratón hasta el ítem **Dno**.

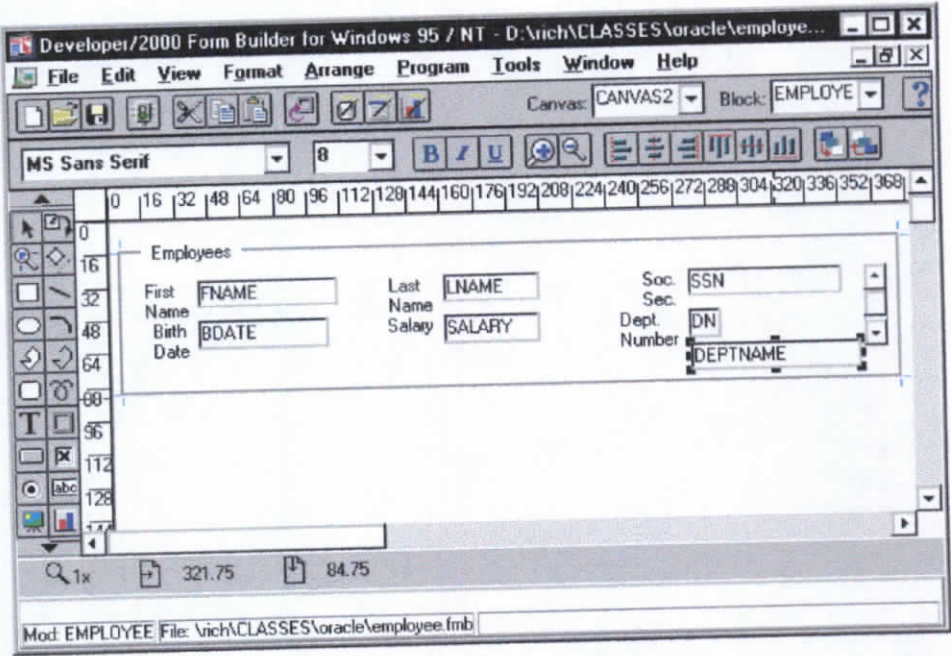


Fig. 3.59. Crear un Campo tabla no-base

- ☑ Visualizar las propiedades para el nuevo Ítem de Texto, haciendo doble clic en él. Cambiar su nombre a **Nombre_Dept** y sus propiedades:

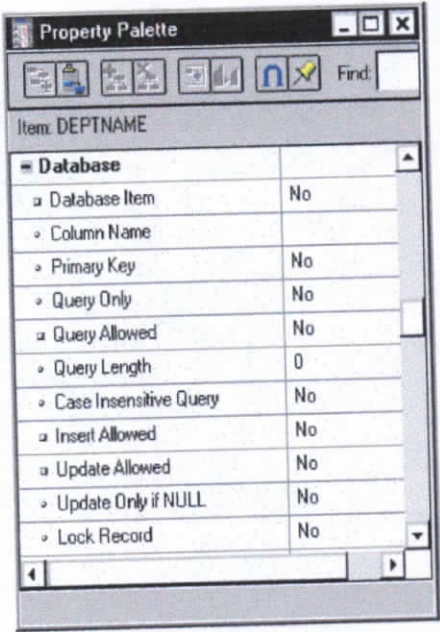


Fig. 3.60. Propiedades del Ítem de Texto

- Enabled:** Yes
- Keyboard Navigable:** No
- Database Item:** No
- Query Only:** No

Primary Key: No

Insert Allowed: No

Query Allowed: No

Update Allowed: No

- ☒ Colocar **Keyboard Navigable** a **No**, previene al usuario posicionarse sobre este campo, puesto que no es usado para entrada de datos o consulta.
- ☒ **Database Item** en **No**, indica que este campo no es una columna actual en la tabla **Empleado**.
- ☒ Las otras propiedades son colocadas a **No** para evitar que el campo participe en cualquiera de las actividades: Borrar, Insertar, Actualizar y Consultar.

El nuevo ítem **Nombre_Dept** es adicionado al bloque de datos **Empleado** como un campo de **tabla no-base**.

3.3.1.21.- Crear un Trigger

Se necesita crear un trigger para llenar el campo **Nombre_Dept**, siempre que el valor de **DNO** cambie. Los **Triggers** en **Oracle Forms** tienen cuatro formas generales:

- ☒ **PRE:** el trigger se dispara antes que un evento sea ejecutado.
- ☒ **POST:** el trigger se dispara luego que un evento es ejecutado.
- ☒ **WHEN:** el trigger se dispara en lugar de la ejecución de un evento.
- ☒ **KEY:** el trigger se dispara cuando una tecla particular es presionada.

Para llenar el campo **Nombre_Dept** con el nombre de departamento apropiado, justo luego que cambie el campo **Dno** en el bloque **Empleado**, entonces usar el trigger **Post-Change** en el campo **Empleado.Dno**.

- ☒ Cambiar a la vista del Navegador de Objetos, seleccionar **Tools | Object Navigator**.

- ☑ Abrir el bloque de datos **Empleado** y el ítem **Dno**.
- ☑ Hacer clic en la propiedad **Triggers** y crear un nuevo trigger con la opción **Navigator | Create**.
- ☑ Aparece una lista de nombres de posibles disparadores, escoger **Post-Change** y hacer clic en el botón **Ok**.

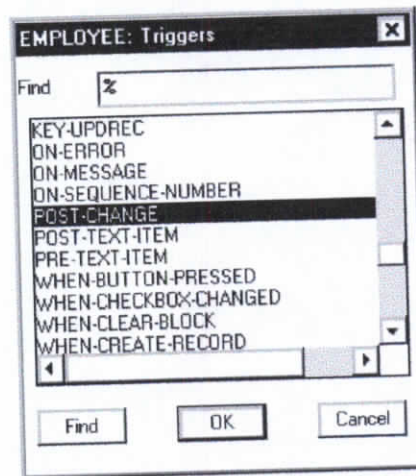


Fig. 3.61. Crear un Trigger

- ☑ Cuando el editor **PL/SQL** aparece, ingresar el código:
 - Llenar el campo de tabla no base :Empleado.Nombre_Dept
 - usando un valor de la tabla Departamento.
 - BEGIN**
 - SELECT** Dnombre
 - INTO** :Empleado.Nombre_Dept
 - FROM** Departamento
 - WHERE** Departamento.Dnumero = :Empleado.Dno;
 - END;**

Los items en un bloque deben ser precedidos por dos puntos, para diferenciarlos de las columnas en una tabla.
- ☑ Compilar el trigger haciendo clic en el botón **Compile**.

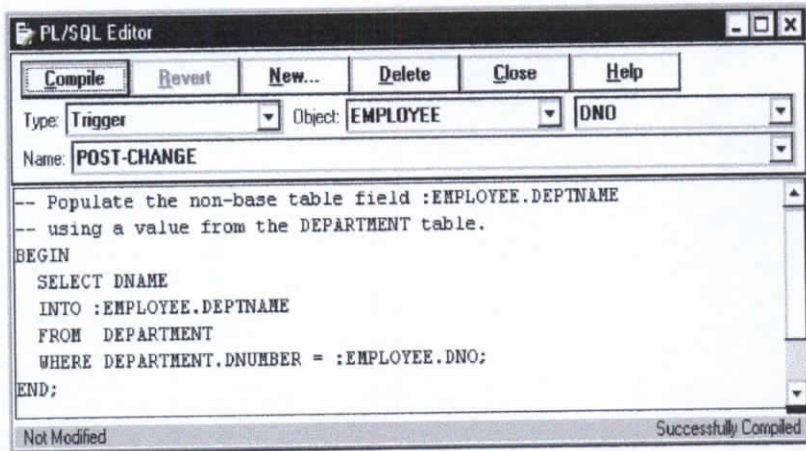


Fig. 3.62. Editor PL/SQL para el Trigger

- ☑ Cerrar el editor **PL/SQL**, hacer clic en el botón **Close**. Grabar, compilar/construir y ejecutar la forma para verificar la funcionalidad.

Para crear una nueva Lista de Valores:

Una nueva lista de valores debe ser creada para visualizar tanto el número de departamento y el nombre.

- ☑ En el Navegador de Objetos, hacer clic en el **LOV** para la forma **Empleado**. Seleccionar **Navigator | Create**. Escoger el Asistente **LOV**.
- ☑ Ingresar la siguiente sentencia SQL en el campo **Query Text**:

```
SELECT Departamento.Dnumero, Departamento.Dnombre
FROM Departamento
```

Hacer clic en el botón **Next**.

Incluir las columnas **Dnumero** y **Dnombre** para el Grupo de Registros en el **LOV**.

- ☑ Ingresar el título para **LOV** y retornar **Dnumero** y **Dnombre** desde el **LOV**.
- ☑ Finalmente, grabar, compilar y ejecutar la forma.

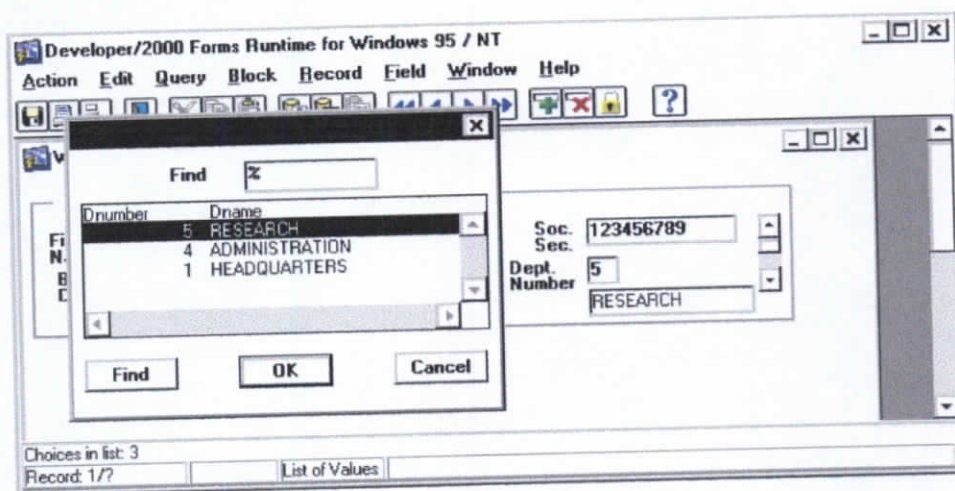


Fig. 3.63. Forma con LOV

3.3.2.- Unidades de Programa Oracle Forms

Oracle Forms provee un mecanismo para almacenar procedimientos (llamados **Unidades de Programa**) escritas en lenguaje **PL/SQL** dentro de una forma. Las unidades de programa pueden ser usadas para adicionar funcionalidad, que no es posible a través de **SQL** o de la interfase de Oracle **Forms**.

Al almacenar una Unidad de Programa dentro de una forma, muchos bloques en la forma pueden tomar ventaja del mismo procedimiento, lo cual reduce la duplicación de código y mejora el mantenimiento del código.

Los procedimientos escritos en **PL/SQL** pueden también ser almacenados dentro de la Base de Datos Oracle en sí, como un objeto en el esquema. Tales Unidades de Programa Almacenado (llamadas Procedimientos Almacenados) son ideales en situaciones donde existen reglas del negocio estandarizadas a gran nivel, o aplicaciones lógicas pueden ser implementadas a través de muchas formas o aplicaciones. En suma, los procedimientos que requieren varias consultas a la base de datos pueden ser mucho mejor implementadas en la base de datos, para reducir el tráfico de red y puede mejorar el desempeño significativamente. Un usuario puede crear procedimientos almacenados con la herramienta Oracle **SQL*Plus**.

Oracle **Forms** puede hacer llamados tanto a Unidades de Programa como a Unidades de Programa Almacenados. Esta flexibilidad permite a los diseñadores de aplicaciones extenso control sobre la ejecución de la aplicación y facilidades para desarrollar afinamientos.

3.3.2.1.- Crear Unidades de Programa

Para aumentar en la forma **Empleado** otros miembros del departamento, se requiere aumentar campos, y escribir procedimientos **PL/SQL**, **Triggers** que llamarán al procedimiento cada vez que se requiera un nuevo registro del empleado.

3.3.2.2.- Adicionar un Nuevo ítem al Bloque Empleado

Adicionar un nuevo ítem al bloque **Empleado**, para almacenar otra información del departamento, en forma similar que en la sección **Adicionar un Nuevo Ítem a un Bloque Existente**.

Por ejemplo, llamar al campo **Otros_Miembros**, asegurar que no es un campo de la base de datos y que un usuario no será capaz de navegar hasta éste, insertar, actualizar o borrar sus datos. Ir a las **Propiedades** y colocar:

Enabled: True
Navigable: No
Database Item: No
Query Only: No
Primary Key: No
Insert Allowed: No
Query Allowed: No
Update Allowed: No

Adicionar una etiqueta para el campo, pasando al Editor Layout, escoger la herramienta **Texto** y hacer clic en el campo **Otros_Miembros**.

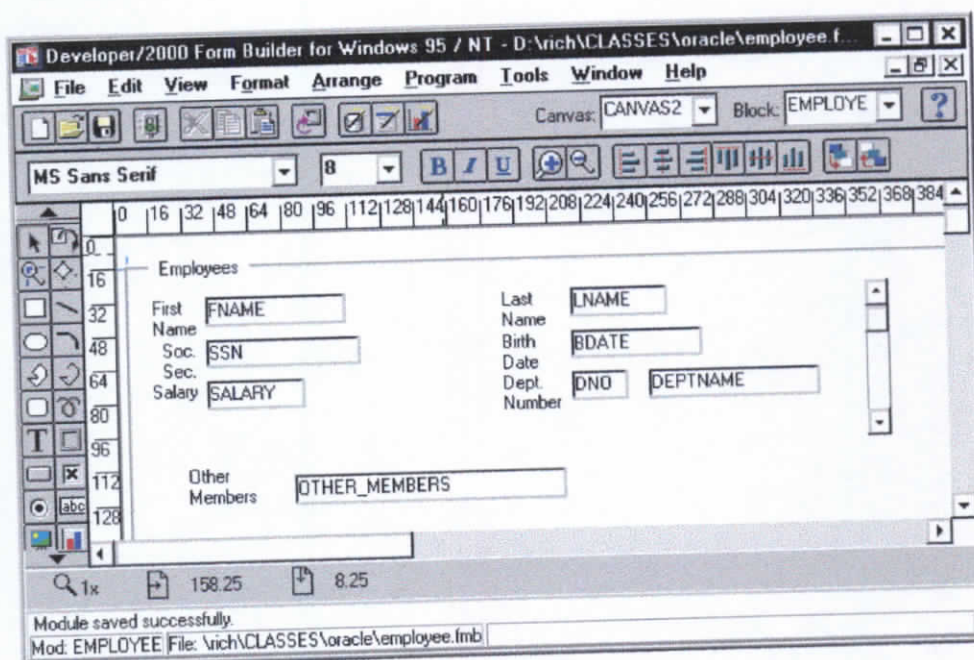


Fig. 3.64. Nuevos Items

3.3.2.3.- Crear un Procedimiento para Contar los Miembros del Departamento

Crear una Unidad de Programa (procedimiento) en Oracle **Forms**, para contar el número de miembros en el mismo departamento, para lo cuál se debe pasar un parámetro llamado **Num_Emp_Ssn**, que contendrá el número de seguridad social del empleado. El procedimiento retornará un valor en el parámetro **Num_Miembros** que contendrá el número de empleados que están en el mismo departamento.

Para crear este procedimiento, pasar al **Navegador de Objetos**, avanzar en **Program Units**, seleccionar **Navigator | Create**, en la caja de diálogo se solicita el nombre del nuevo procedimiento, ingresar **Num_Miembros** y presionar **Ok**.

En el Editor **PL/SQL** ingresar el siguiente cuerpo de procedimiento:

```
-- Dado el SSN del empleado, retornar el número de empleados
-- en el mismo departamento.
PROCEDURE Num_Miembros (Num_Emp_Ssn IN NUMBER,
                        Num_Miembros OUT NUMBER) IS
    Cuenta_Aux NUMBER; -- Contador
BEGIN
    Cuenta_Aux := 0; -- Inicializar a 0
    -- Probar y llenar Cuenta_Aux con la cuenta de otros
```

```

-- empleados en el Departamento
SELECT COUNT(*) INTO Cuenta_Aux
  FROM Empleado
  WHERE Empleado.Dno =
    (SELECT Empleado.Dno
     FROM Empleado
     WHERE Empleado.SSN = Nun_Emp_Ssn);
-- Ver si se obtiene un resultado satisfactorio
IF (NVL(Cuenta_Aux, 0) <> 0) THEN
  Num_Miembros := (Cuenta_Aux - 1);
ELSE
  Num_Miembros := 0;
END IF;
END;
```

Este procedimiento consta de las secciones:

La cabecera del procedimiento indica que un parámetro, **Num_Emp_Ssn** de tipo **NUMBER** debe ser pasado al procedimiento como valor. El otro parámetro **Num_Miembros** también de tipo **NUMBER** donde un valor será devuelto al programa que llama al procedimiento.

Para finalizar la cabecera del procedimiento, la variable **Cuenta_Aux** es declarada para ser usada sólo dentro del procedimiento.

La sentencia **BEGIN** inicia el cuerpo del procedimiento, **Cuenta_Aux** se inicializa a 0, y se realiza una consulta a la base de datos para encontrar el número total de empleados del mismo departamento que el empleado cuyo SSN se suministró.

El resultado en **Cuenta_Aux** es verificado para comprobar si es **NULL**. Si **Cuenta_Aux** no es **NULL** y no es 0, entonces **Num_Miembros** se coloca como **Cuenta_Aux - 1**, de otro modo se coloca en 0.

La última sentencia **END** finaliza el cuerpo del procedimiento.

Asegurar que el procedimiento se compila con el botón **Compile**. Cualquier error en el procedimiento será iluminado. Algunos problemas comunes incluyen olvidar usar **:=** en una sentencia de asignación y dejar una sentencia **IF ... THEN** sin su **END IF**.

Cuando esté verificado el procedimiento, cerrar el editor.

3.3.2.4.- Crear un Trigger para Llamar al Procedimiento

Crear un trigger para llamar al procedimiento **Num_Miembros**, en el **Navegador de Objetos** abrir el bloque **Empleado** y el ítem **SSN**, iluminar la rama **Trigger** para el ítem **SSN**, escoger **Navigator | Create**.

En la lista, escoger **Post-Change** y presionar el botón **Ok** para abrir el editor **PL/SQL** para el nuevo trigger, ingresar el código:

```
DECLARE
    cuenta_retorno NUMBER;
BEGIN
    -- Llamar al procedimiento Forms para obtener la cuenta
    -- de otros en el Departamento.
    Num_Miembros (:Empleado.Ssn, cuenta_retorno);
    -- Asignar la cuenta de retorno al campo en el
    -- bloque Empleado.
    :Empleado.Numero_Miembros := cuenta_retorno;
END;
```

Compilar para verificar el trigger y cerrar el Navegador de Objetos.

3.3.2.5.- Grabar, Compilar/Construir y Ejecutar la Forma

Una vez adicionado un nuevo campo de tabla no-base al bloque **Empleado**, grabar el trabajo con **File | Save As**.

Grabar, Compilar/Construir y Ejecutar la forma, para verificar su funcionalidad, cuando se avance a un nuevo registro de empleado, el campo **Numero_Miembros** se llena automáticamente con valores que indican el número de empleados en el mismo departamento del empleado.

Developer/2000 Forms Runtime for Windows 95 / NT - [WINDOW1]

Action Edit Query Block Record Field Window Help

Employees

First Name JOYCE Last Name ENGLISH

Soc. Sec. 453453453 Birth Date 31-JUL-1962

Salary 25000 Dept. Number 5 RESEARCH

Other Members 3

Record: 6/8

Fig. 3.65. Forma incluida un nuevo Item

3.3.2.6.- Crear Procedimientos Almacenados en SQL*Plus

Un procedimiento (Unidad de Programa) adicionado en Oracle **Forms** para desarrollar una consulta a la base de datos, es almacenado como parte del código fuente de la forma y es compilado cuando la forma es construida, y cada llamado al procedimiento produce:

1. Un llamado local al trigger **Post-Change** al procedimiento que pasa un **Ssn** del empleado.
2. Un acceso remoto desde el procedimiento a la base de datos pasando una consulta **SQL**.
3. Un retorno remoto de la base de datos que contiene un contador.
4. Una pequeña cantidad de procesamiento, en el procedimiento (**IF ... THEN**).
5. Un retorno local desde el procedimiento al trigger.

En este intercambio, el costo más alto está en el tráfico de red requerido entre el cliente y el servidor (pasos 2 y 3), y los costos más bajos en los llamados locales entre el trigger y procedimiento y el procesamiento local (pasos 1, 4 y 5).

La consulta **SQL** en los pasos 2 y 3, no retorna una cantidad significativa de datos, sin embargo, hay situaciones donde, si una gran cantidad de datos son retornadas para cada consulta, tales consultas impactan el desempeño.

Una solución para este problema es mover las consultas y tantos procesamientos como sea posible lo más cerca a los datos en la base de datos, para el ejemplo, implementar el procedimiento **Num_Miembros** en la base de datos Oracle, con la herramienta Oracle **SQL*Plus**, para crear un procedimiento almacenado.

Ingresa a **SQL*Plus**, escribir la sentencia **CREATE PROCEDURE**, se puede grabarla en un archivo de texto y ejecutarla con el comando **START** de **SQL*Plus**.

```
CREATE PROCEDURE Num_Miembros
  (Num_Emp_Ssn IN NUMBER,
   Num_Miembros OUT NUMBER) AS
  -- Dado el SSN del empleado, retornar el número de los otros
  -- empleados en el mismo Departamento.
  Cuenta_Aux NUMBER; -- Contenedor para el contador
BEGIN
  Cuenta_Aux := 0; -- Inicialiar a 0
  -- Probar y llenar Cuenta_Aux con la cuenta de otros
  -- empleados en el Departamento
  SELECT COUNT(*) INTO Cuenta_Aux
    FROM Empleado
   WHERE Empleado.Dno =
         (SELECT Empleado.Dno
          FROM Empleado
         WHERE Empleado.Ssn = Num_Emp_Ssn);
  -- Ver si se obtiene un resultado significativo
  IF (NVL(Cuenta_Aux, 0) > 0) THEN
    Num_Miembros := (Cuenta_Aux - 1);
  ELSE
    Num_Miembros := 0;
  END IF;
END;
```

Ingresa la sentencia, ejecutarla con el comando **RUN** de **SQL*Plus**, o con el slash /

Si cuando se crea un procedimiento almacenado se recibe un mensaje como:

ORA-01031: insufficient privileges

Indica que el usuario no se tiene privilegios para realizar esta actividad.

Para verificar que el procedimiento se ha compilado satisfactoriamente, recurrir a la vista **User_Errors**:

```
SQL> SELECT * FROM User_Errors;
no rows selected
```

Cuando aparece el mensaje **no rows selected**, indica que no hay errores en el trigger.

También se puede ver los errores, con el comando **SQL*Plus Show Errors**, aunque en algunas versiones, no trabaja debido a una falta de buffer de memoria en el computador cliente.

```
SQL> CREATE PROCEDURE test AS
2 BEGIN
3 SELECT SYSDATE INTO :temp FROM dual;
4 END;
Warning: Procedure created with compilation errors.
SQL> show errors
buffer overflow. Use SET command to reduce ARRAYSIZE or increase MAXDATA.
No errors.
```

Si aparece el mensaje **buffer overflow**, colocar la variable **ARRAYSIZE** a un número más bajo que 2 con el comando:

```
SQL> SET ARRAYSIZE 2
```

Luego usar el comando **show errors**:

```
SQL> show errors
Errors for PROCEDURE TEST:
LINE/COL ERROR
-----
3/23          PLS-00049: bad bind variable 'TEMP'
```

Para visualizar los procedimientos almacenados que se han creado, consultar la vista **User_Source** del diccionario de datos.

La diferencia entre el procedimiento almacenado creado en el esquema de la base de datos con SQL*Plus y el creado en Oracle Forms es la forma de la cabecera:

```
CREATE PROCEDURE Num_Miembros
  (Num_Emp_Ssn IN NUMBER,
   Num_Miembros OUT NUMBER) AS
```

En Oracle Forms:

```
PROCEDURE Num_Miembros
  (Num_Emp_Ssn IN NUMBER,
```


Num_Miembros OUT NUMBER) IS

Cualquier Forma, Reporte u otro procedimiento puede hacer llamados al procedimiento **Num_Miembros** como si fuera un procedimiento local.

El trigger **Post-Change** en **:Empleado.Ssn** automáticamente llama al procedimiento almacenado **Num_Miembros** ubicado en el esquema.

Para ver las Unidades de Procedimiento Almacenado disponibles en Oracle **Forms**, pasar al **Navegador de Objetos** y abrir el árbol **Database Objects**, aparece una lista de esquemas (usuarios), avanzar y abrir el árbol **Stored Program Units** del usuario.

3.3.2.7.- Procedimientos Almacenados y Triggers adicionales

Hay varias formas para que los procedimientos almacenados y disparadores puedan ser usados, para asegurar las reglas del negocio y procesar datos en la aplicación.

3.3.2.8.- Generar identificadores únicos o consecutivos usando la tabla

Muchas de las llaves usadas en las tablas están basadas en un número secuencial. Por ejemplo en la tabla **Departamento**, **DNumero** es un entero que podría incrementarse para cada ingreso. Para generar automáticamente un nuevo número de departamento, crear una consulta para capturar el valor actual más grande de **DNumero** y sumar 1, luego asignarlo al ítem **DNumero** en el bloque de datos **Departamento**. Un trigger debe ser creado en el ámbito de bloque y debería ejecutarse antes que cualquier registro sea insertado en la base de datos, el trigger **Pre-Insert** en el bloque de datos **Departamento**:

```
BEGIN
  SELECT MAX( DNumero ) + 1
  INTO :Departamento.DNumero
  FROM Departamento;
```

END;

3.3.2.9.- Generar identificadores únicos o consecutivos (Secuencia Oracle)

Cuando muchos usuarios trabajan en la misma forma para insertar registros, se genera problemas de concurrencia. Para cada nuevo registro a ser insertado, un examen a la tabla completa debe ser realizado para encontrar el identificador más grande.

Una solución es usar una **Secuencia Oracle**, objeto del esquema que mantiene la pista de un entero simple, y automáticamente lo incrementa cuando se accesa.

Por ejemplo en la tabla **Departamento**, **DNumero** es un entero que debería ser incrementado para cada nuevo registro ingresado, para generar automáticamente un nuevo **DNumero**, crear la secuencia usando **SQL*Plus**:

```
CREATE SEQUENCE Departamento_seq  
  INCREMENT BY 1  
  START WITH 1  
  MAXVALUE 99999  
  NOCYCLE;
```

La secuencia **Departamento_seq** comienza contando en 1, incrementa en 1 hasta que alcance 99999, luego de lo cuál genera un error. Esto es razonable porque una organización nunca tendrá más de 99999 departamentos.

Luego, asignar el siguiente valor de secuencia al ítem **DNumero** del bloque de datos **Departamento**. Un trigger debe ser creado en el ámbito de bloque, y se debe ejecutar antes que cualquier nuevo registro sea insertado en la base de datos, el trigger **Pre-Insert**:

```
BEGIN  
  -- Obtener el siguiente valor para DNumero desde la  
  -- secuencia Departamento_seq.  
  SELECT Departamento_seq.NextVal  
    INTO :Departamento.DNumero  
    FROM dual;  
END;
```

La tabla **dual** es una tabla ficticia que contiene exactamente un registro (fila), se puede usarla en una sentencia **SELECT ... INTO** para retornar en forma consistente

un valor simple. **NextVal** accesado en la sentencia, retorna el valor de la secuencia actual y luego lo incrementa automáticamente.

El disparador **Pre-Insert** se ejecuta justo antes que la sentencia **INSERT** para un nuevo registro sea enviado a la base de datos.

Las **Secuencias Oracle** no son aplicables para formas master-detalle, están diseñadas para entregar identificadores únicos para llaves primarias.

3.3.2.10.- Verificar restricciones en el cliente

En cualquier sistema de base de datos, es deseable restringir los valores que pueden tomar ciertas columnas, adicionando restricciones a las tablas de la base de datos, para forzar las restricciones, sin embargo, solo toma lugar cuando una transacción alcanza la base de datos. En suma, muchas restricciones sobre los datos que involucran valores de datos de otras tablas, podrían no ser implementadas fácilmente como restricciones de tabla.

Muchos tipos de restricciones sobre los datos pueden ser verificadas en el cliente usando disparadores. Por ejemplo, en la forma **Empleado**, sería deseable forzar una restricción para que un empleado no pueda ser su propio supervisor, es decir que para cualquier registro **Ssn** no puede ser igual a **SuperSsn**. Para implementar la restricción, crear el trigger **When-Validate-Item** en el ítem **SuperSsn** del bloque de datos **Empleado**:

```
BEGIN
  IF (:SuperSsn = :Ssn) THEN
    MESSAGE('Empleado no puede ser supervisor del mismo!');
    RAISE Form_Trigger_Failure;
  END IF;
END;
```

Form_Trigger_Failure causa que la ejecución de la validación del ítem sea detenida, y el usuario tendrá que cambiar **SuperSsn** antes de moverse a otro ítem.

3.3.2.11.- Adicionar Alertas (Mensajes Pop-up)

El procedimiento **Message** es usado para visualizar el error en la barra de estado de la ventana. Una mejor opción es forzar al usuario que ponga mayor atención al problema, creando una caja de diálogo pop-up, de forma que el usuario lea el mensaje y para continuar deba presionar el botón **Ok** (o cualquier otro), que en Oracle **Forms** se conoce como **Alerta**, que se coloca de la siguiente forma:

- ☒ En el **Navegador de Objetos** visualizar el área **Alerts**, seleccionar **Create**, hacer clic en el nombre por omisión dado para la nueva alerta y cambiarlo: **Alerta_Supervisor**

- ☒ Pasar a la paleta de propiedades y cambiar las propiedades:

Title: Mensaje de Alerta de Supervisor

Message: Empleado no puede ser su propio Supervisor

Alert Style: Stop

Button 1 Label: Ok

Dejar en blanco las etiquetas **Button 2** y **Button 3**

Default Alert button: Button 1

Dejar las otras propiedades con sus valores por omisión y cerrar la paleta de propiedades.

- ☒ Cambiar el trigger **When-Validate-Item** del ítem **SuperSsn** del bloque Empleado a:

DECLARE

 alerta_retorno **NUMBER**;

BEGIN

IF (:SuperSsn = :Ssn) **THEN**

 alerta_retorno := **SHOW_ALERT** ('ALERTA_SUPERVISOR');

RAISE Form_Trigger_Failure;

END IF;

END;

El procedimiento **Show_Alert** llama la alerta especificada, y obtiene el valor de retorno (según el botón presionado por el usuario) para asignarle a una variable local **alerta_retorno**.

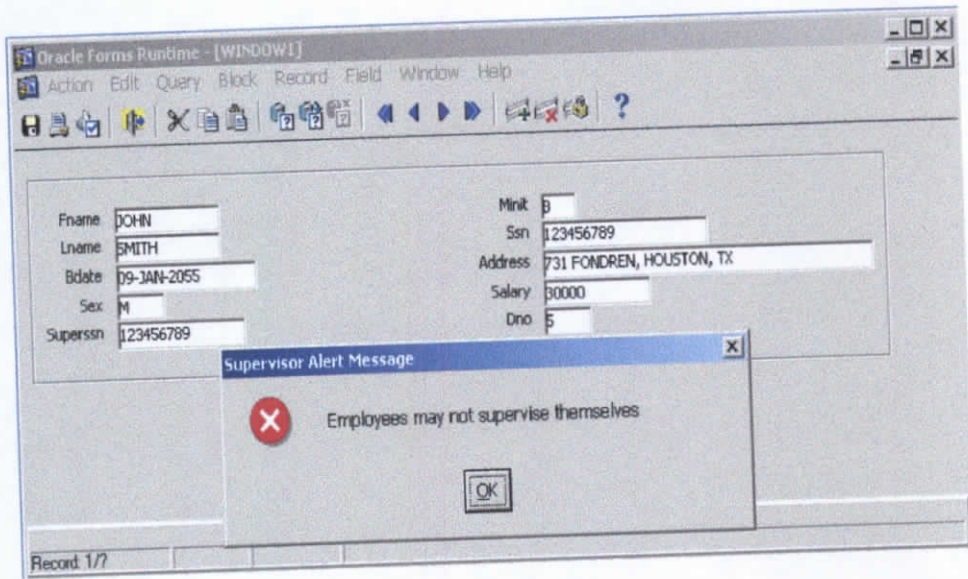


Fig. 3.66. Alertas

En general, las alertas pueden tener varios botones, por ejemplo, una forma podría invocar una alerta **¿Seguro que desea Salir?** con los botones **Si** y **No**; un trigger **Pre-Commit** podría invocar la alerta **¿Seguro que desea borrar el Empleado?** Con los botones **Si** y **No**, y sobre la base del valor de **alerta_retorno**, se podría hacer **commit** o **rollback**.

3.3.2.12.- Llenar items con valores por omisión

En muchos casos, se puede sugerir valores por omisión para ciertos items de datos, tomando como base valores ya suministrados en otros items. El método consiste en usar un trigger **Post-Item** o **When-Validate-Item** para llenar otro campo, sólo si el otro campo está en blanco. Por ejemplo, para llenar el ítem **SuperSsn** del bloque **Empleado** sobre la base de **MgrSsn** del departamento del empleado, el siguiente código de trigger **When-Validate-Item** se podría utilizar en el ítem **Dno**:

```
BEGIN
  IF :SuperSsn IS NULL THEN
    SELECT Departamento.MgrSsn
    INTO :Empleado.SuperSsn
    FROM Departamento
    WHERE Departamento.DNumero = :Empleado.Dno;
  END IF;
END;
```

La sentencia **IF** usa **:SuperSsn IS NULL** para verificar si cualquier valor ha sido suministrado para **SuperSsn**. Si no fue ingresado el ítem **SuperSsn**, entonces su valor es **NULL**. Si el usuario ingresa cualquier cosa en el ítem **SuperSsn**, entonces la sentencia **IF** se evaluará como falso y no se sugiere un valor.

3.3.2.13.- Funcionalidad de los botones de comando

Los botones de comando pueden ser adicionados en una forma para ejecutar funciones tales como salir de una forma, hacer commit y grabar los registros cambiados, etc. Un objeto botón, de comando básico, contiene propiedades para la etiqueta, y propiedades adicionales como atributos visuales para mostrar imágenes bitmap.

Una vez colocado un botón en la forma, la funcionalidad se adiciona con el trigger **When-Button-Pressed**, que se ejecuta cuando el usuario hace clic sobre él. Por ejemplo, para crear un botón para salir de la forma:

```
DECLARE
    alerta_retorno NUMBER;
BEGIN
    alerta_retorno := SHOW_ALERT ('EXIT_ALERT');
    IF (alerta_retorno = alert_button1) THEN
        EXIT_FORM;
    END IF;
END;
```

El código asume que existe una alerta llamada **Exit_Alert** con dos botones: **Yes** y **No**. Si el usuario hace clic en el botón **Yes** (primero), el procedimiento **EXIT_FORM** es llamado y la forma se cierra. Si el usuario hace clic en el otro botón, la alerta se cierra y la forma permanece abierta.

3.3.2.14.- Pasar parámetros entre formas

Muchas ocasiones se requiere el paso de datos de una forma a otra. Por ejemplo cuando se consulta cierto registro en una forma y se desea llamar a otra forma con el

mismo registro automáticamente recuperado. En general, los valores de datos pueden ser pasados entre formas usando variables **Globales** (:Global es tratado como un bloque disponible en todas las formas) o usar listas de parámetros.

Por ejemplo, con la forma **Empleado** y la forma master/detalle **Empleado/Dependiente**, colocar un botón en la forma **Empleado**, que al ser presionada, llame a la forma **Empleado/Dependiente** y automáticamente consulte los dependientes para el empleado de la forma **Empleado**.

- ☑ En la forma **Empleado**, crear el botón **Consultar Dependientes**. Crear un trigger **When-Button-Pressed** con el código:

```
DECLARE
    /* Declarar pl_id como identificador para la lista de parámetros */
    pl_id ParamList;
BEGIN
    /* Ver si la lista de parámetros existe. Si es así destruirla */
    pl_id := Get_Parameter_List('tmpdata');
    IF NOT Id_Null(pl_id) THEN
        Destroy_Parameter_List(pl_id);
    END IF;
    /* Crear una lista de parámetros */
    pl_id := Create_Parameter_List('tmpdata');
    /* Adicionar un nuevo parámetro a la lista EmpleadoSSN */
    /* Asignar este parámetro al valor actual del campo :SSN */
    Add_Parameter(pl_id, 'EMPLEADOSSN', TEXT_PARAMETER, :SSN);
    /* Ejecutar una nueva forma llamada empdepn (debe estar en el mismo */
    /* directorio que la forma Empleado que se está editando. */
    /* Pasar el identificador de la lista de parámetros (pl_id) */
    Run_Product(FORMS, 'empdepn', SYNCHRONOUS, RUNTIME,
        FILESYSTEM, pl_id, NULL);
END;
```

- ☑ Compilar y grabar esta forma.
- ☑ Abrir la forma **Empleado/Dependiente**.
- ☑ En el Navegador de Objetos, crear un nuevo parámetro llamado **EmpleadoSsn**, que tendrá automáticamente asignado un valor, sobre la base de una lista de parámetros pasados a la forma por el llamado al procedimiento **Run_Product**.
- ☑ Crear un trigger llamado **When-New-Form-Instance**, que se ejecuta cuando la forma es ejecutada.

```
DECLARE
    /* Declarar un identificador de bloque forma */
    blk_id Block;
BEGIN
```



```

/* Obtener la identificación del bloque Empleado. Este es el */
/* bloque Master en la forma master/detalle empdepn.          */
blk_id := Find_Block('EMPLEADO');
IF NOT Id_Null(blk_id) THEN
    /* Verificar, asegurar que el parámetro tiene un valor. Si esta forma */
    /* fue ejecutada por si misma, entonces el parámetro será nulo.      */
    /* Si es llamada desde Empleado, entonces el parámetro será */
    /* pasado y asignado a :PARAMETER.EmpleadoSSN */
    IF (:PARAMETER.EmpleadoSSN is not null) THEN
        /* Mientras se tenga un parámetro, usarlo para alterar WHERE */
        /* de forma que sea SSN=:PARAMETER.EmpleadoSSN */
        SET_BLOCK_PROPERTY(blk_id,DEFAULT_WHERE,'ssn
        =' || :PARAMETER.EMPLEADOssn);
        /* Pasar al bloque Empleado y ejecutar una consulta automática */
        GO_BLOCK('EMPLEADO');
        EXECUTE_QUERY;
    END IF;
END IF;
END;

```

- ☑ Compilar y grabar la forma.
- ☑ Ejecutar la forma **Empleado**, consultar un registro y hacer clic en el botón **Consultar Dependientes**, debe aparecer la forma **Empleado / Dependiente** con los dependientes automáticamente consultados.

3.3.3.- Oracle Reports

Otra herramienta Developer es **Oracle Reports**, que permite al desarrollador crear reportes sofisticados, con una variedad de modelos y muchas características que pueden ser personalizadas.

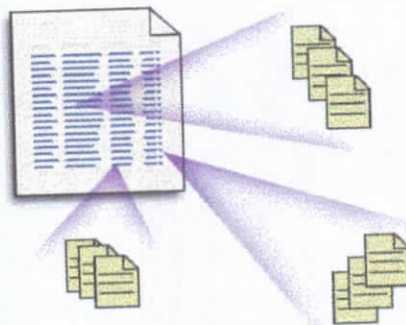


Fig. 3.67. Oracle Reports

Una instalación típica de **Developer** crea varios items principales para **Oracle Reports**:

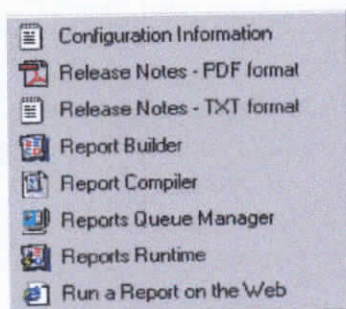


Fig. 3.68. Opciones Oracle Reports

Para ejecutar **Oracle Reports Builder** en Windows 95/98/NT/2000, escoger **Inicio | Programas | Oracle Reports | Report Builder**.

Similar a Oracle **Forms**, aparece una pantalla inicial:

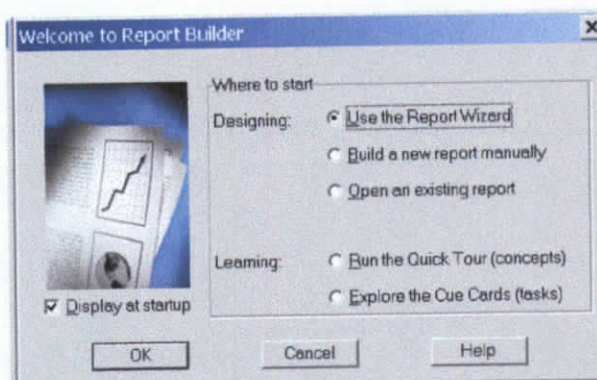


Fig. 3.69. Ventana Inicial de Oracle Reports

Hacer clic en el botón **Cancel** para ingresar directamente al Navegador de Objetos.

3.3.3.1.- Navegador de Objetos de Oracle Reports

La pantalla principal de **Oracle Reports**, igual que en **Forms** se llama **Navegador de Objetos**:

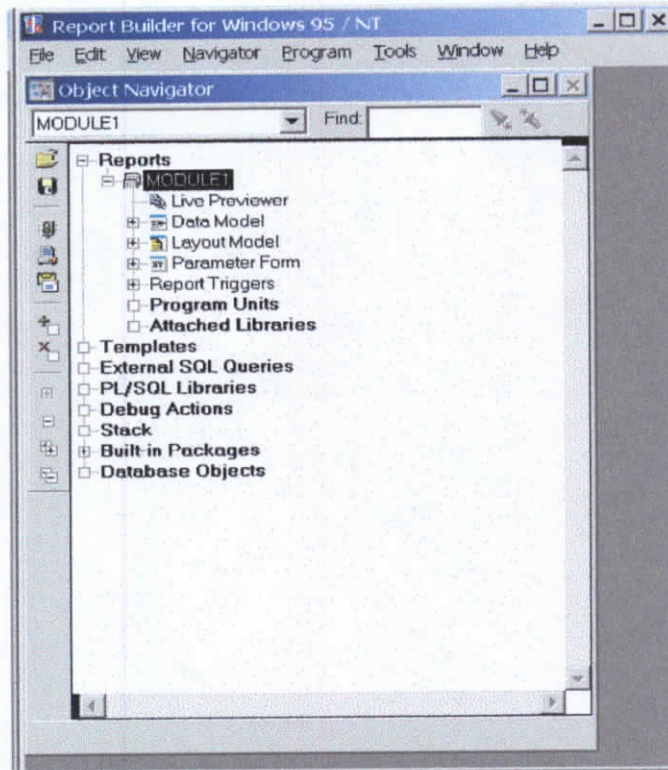


Fig. 3.70. Navegador de Objetos de Reportos

Las secciones principales son:

1. **Data Model:** Información acerca de las consultas usadas para un reporte.
2. **Layout:** Información del formato del reporte: cabeceras, pies, márgenes, tipos de letra, etc.
3. **Parameter Form:** Información de la pantalla principal que es visualizada cuando una forma es ejecutada.
4. **Report Triggers:** Código **PL/SQL** que puede ser ejecutado antes, durante o luego que un reporte ha sido ejecutado.

Como en **Forms**, el primer paso es conectarse a un servidor **Oracle**, seleccionar **File | Connect**.

Para crear reportes, se siguen cuatro pasos:

1. **Definir el Modelo de Datos:** especificar cuáles consultas se deben ejecutar en la base de datos, incluyendo: la forma en que múltiples consultas están relacionadas y cómo están agrupadas. Este paso debe ser

realizado manualmente. Consultas creadas en otra parte pueden ser importadas a Oracle Reports.

2. **Definir el Layout:** especificar el modelo del reporte, incluyendo la orientación de los resultados de la consulta y la supresión o repetición de grupos. Hay un número de modelos de reporte por omisión que pueden ser aplicados automáticamente a un modelo de datos.
3. **Crear y/o Personalizar la forma de Parámetros:** Si se requiere algún ingreso de usuario para ejecutar el reporte, se debe personalizar la forma de parámetros. Todos los reportes tienen una forma de parámetros por omisión.
4. **Crear disparadores o Unidades de Programa (opcional):** los cuales serán ejecutados con el reporte.

Oracle **Reports** dispone también de un asistente.

3.3.3.2.- Crear un Reporte de Tabla Simple

Para especificar el Modelo de Datos y el Layout, en el **Navegador de Objetos**, seleccionar **Tools | Report wizard...**

En la primera pantalla, seleccionar el estilo del reporte y su título:

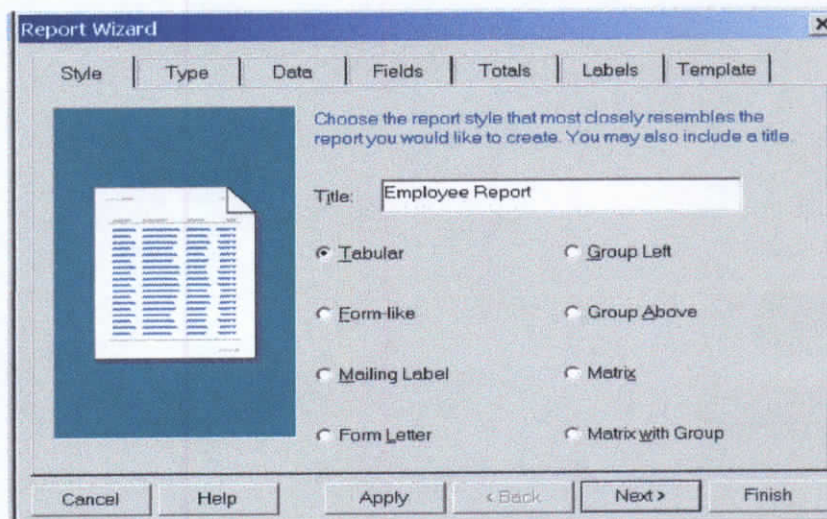


Fig. 3.71. Asistente Reports

Las opciones de layout son:

Tabular: Una tabla simple con cabeceras de columna en la parte superior, y los registros de datos es filas consecutivas hacia abajo.

Form-Like: Las cabeceras de columna al lado izquierdo, con los valores de datos a la derecha de ellas.

Mailing Label: Sin cabeceras de columna y los registros agrupados en secciones repetitivas, para imprimir directamente en una hoja de etiquetas de correo.

Form Letter: Coloca arbitrariamente los datos dentro del cuerpo de un texto.

Group-Left: Estilo Master/Detalle/Detalle, donde los registros master son agrupados al lado izquierdo, con los detalles a la derecha.

Group-Above: Estilo Master/Detalle/Detalle, donde los registros master aparecen sobre los registros detalle.

Matrix: Etiquetas de columna al lado izquierdo y en la parte superior, con los valores de datos en la mitad. Similar a una hoja de cálculo.

Matrix with Group: Combinación de Master/Detalle (Grupo-Arriba) y Matrix.

Para el ejemplo, ingresar **Reporte de Empleados** como título, seleccionar **Tabular** y hacer clic en el botón **Next**.

Especificar el tipo de consulta en que se basa el reporte: consulta **SQL** o consulta **Oracle Express (OLAP)**, seleccionar **SQL Statement** y hacer clic en el botón **Next**.

Especificar la consulta que formará la base del reporte. Ingresar la siguiente consulta:

```
SELECT Nombre, Apellido, Ssn, Sueldo, Dno  
FROM Empleado;
```

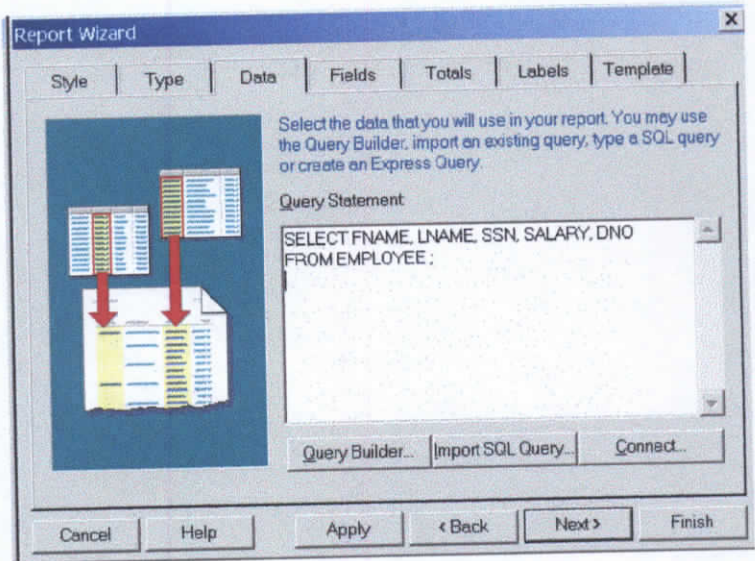


Fig. 3.72. Sentencia SQL para el Reporte

Hacer clic en el botón **Next**. Si no se está conectado a la base de datos, se solicita hacerlo.

Especificar cuáles columnas de la consulta **SQL** serán visualizadas en la salida del reporte, hacer clic en la doble flecha derecha para pasar todos los campos desde **Available Fields** a **Displayed Fields**, y luego hacer clic en **Next**.

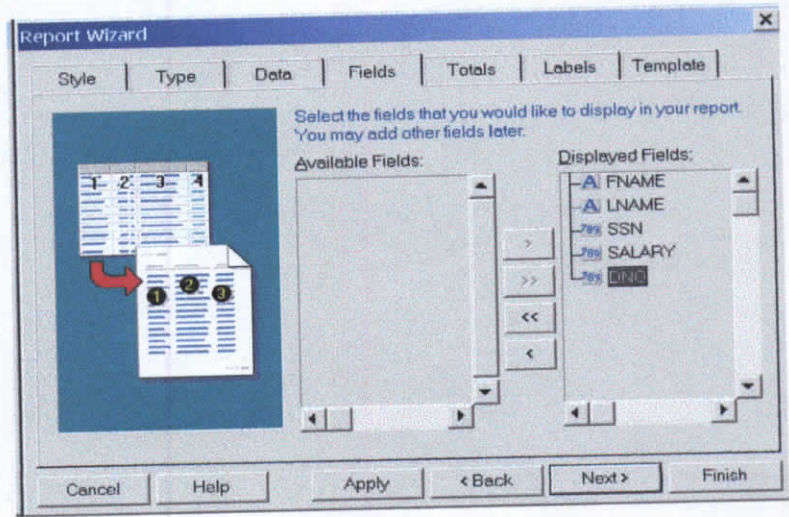


Fig. 3.73. Seleccionar Columnas para el Reporte

Agregar funciones aplicadas a los campos del reporte. Por ejemplo, para visualizar el salario promedio y el número de empleados, iluminar el campo **Sueldo** y hacer clic en el botón **Average**, iluminar el campo **Ssn** y hacer clic en el botón **Count**.

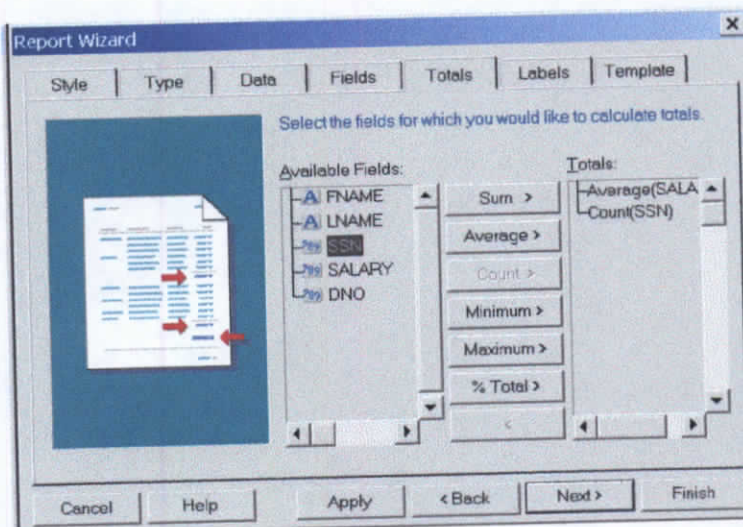


Fig. 3.74. Funciones Aplicadas a los Campos del Reporte

Cambiar las etiquetas de campo y ancho, para los campos en el reporte.

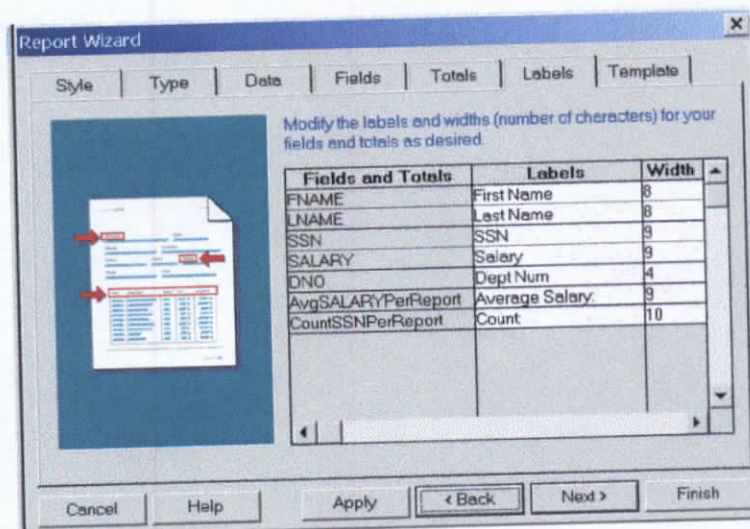


Fig. 3.75. Cambiar Etiquetas y Anchos en los Campos del Reporte

Finalmente, un modelo de diseño puede ser aplicado al reporte. Los modelos de diseño incluyen especificaciones para tipo de letra, colores y layout del reporte. Seleccionar un modelo y hacer clic en el botón **Finish**.

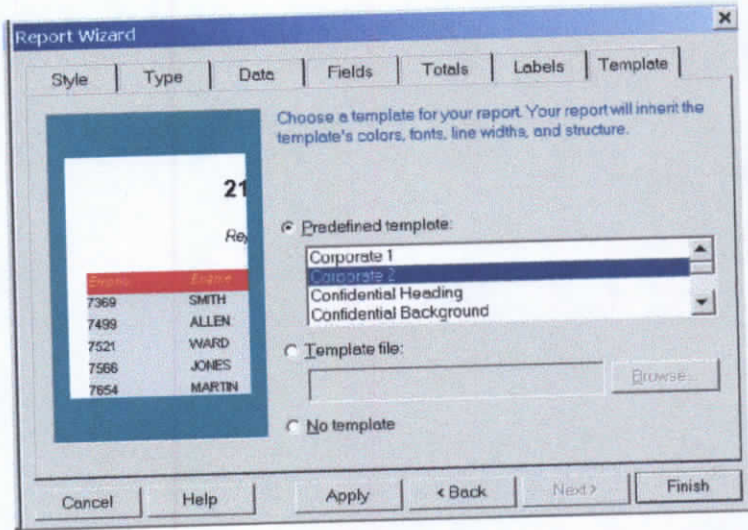


Fig. 3.76. Modelos de Reporte

Luego de una pausa corta, aparece una pre visualización del reporte, que muestra la ejecución del reporte.

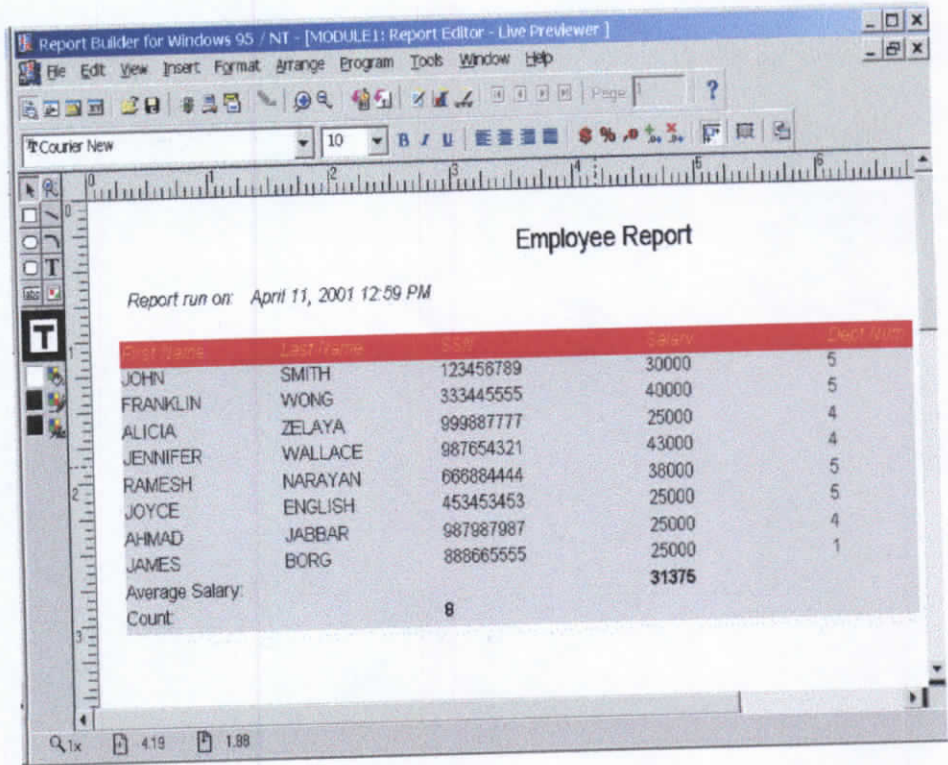


Fig. 3.77. Pre visualización del Reporte

Es posible hacer cambios en el reporte con la paleta de herramientas del lado izquierdo. Una vez finalizado, cerrar la pre visualización del reporte y retorna al

Navegador de Objetos. Para ver el reporte nuevamente, seleccionar **Program | Run Report**.

3.3.3.3.- Grabar, Generar y Ejecutar el Reporte

Para grabar el reporte, seleccionar **File | Save**, el código fuente de Oracle **Reports** se graba en archivos con la extensión **.rdf**, los reportes compilados y generados se graban con la extensión **.rep**.

Una vez grabado el reporte, puede ser ejecutado con **View | Runtime Preview**.

Mientras el reporte se está ejecutando, una pantalla de actividad aparece dando una indicación del procesamiento que se está desarrollando.

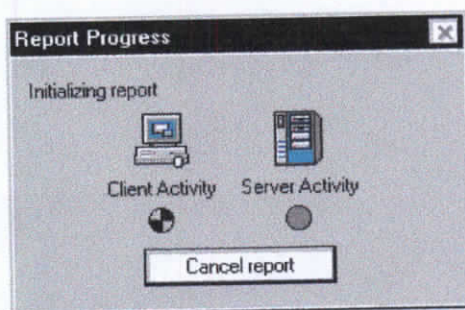


Fig. 3.78. Actividad de Generación del Reporte

La actividad se lleva a cabo en tres estados: **Client Activity** mientras las consultas son preparadas, **Server Activity** cuando las consultas son ejecutadas y finalmente **Client Activity** para formatear el reporte, que finalmente se ejecuta.

3.3.3.4.- Crear un Reporte Master-Detalle

Especificar el Modelo de Datos y Layout Master-Detalle

Como en el reporte para tabla simple, primero especificar el modelo de datos y layout para el reporte Master-Detalle, con el asistente, seleccionar **Tools | Report wizard**.

1. Para el Título del reporte usar: **Departamento y Empleado**
2. Para el Estilo del Reporte, escoger **Group Above**.
3. Escoger una sentencia **SQL**, e ingresar la consulta **SQL**:

```
SELECT DNombre, Nombre, Apellido, Sueldo
FROM Departamento, Empleado
WHERE Departamento.DNumero = Empleado.Dno
```
4. Designar al campo **DNombre** como campo de grupo (**Group field**) (Nivel 1). Hacer clic en el campo **DNombre** y luego en el botón flecha derecha.
5. Visualizar todos los campos en el reporte.
6. Escoger Promedio de Sueldo como campo **Totales**. Hacer clic en el campo **Sueldo** y luego en el botón **Average**.
7. Cambiar los campos y etiquetas:

<u>Campos y Totales</u>	<u>Etiquetas</u>
DNombre	Nombre Departamento
Nombre	Nombre
Apellido	Apellido
Sueldo	Sueldo
AvgSueldoperDNombre	Promedio:
AvgSueldoperRep	Promedio de todos los Empleados:
8. Utilizar la plantilla Corporate 2 , hacer clic en el botón Finish para visualizar el reporte previo:	

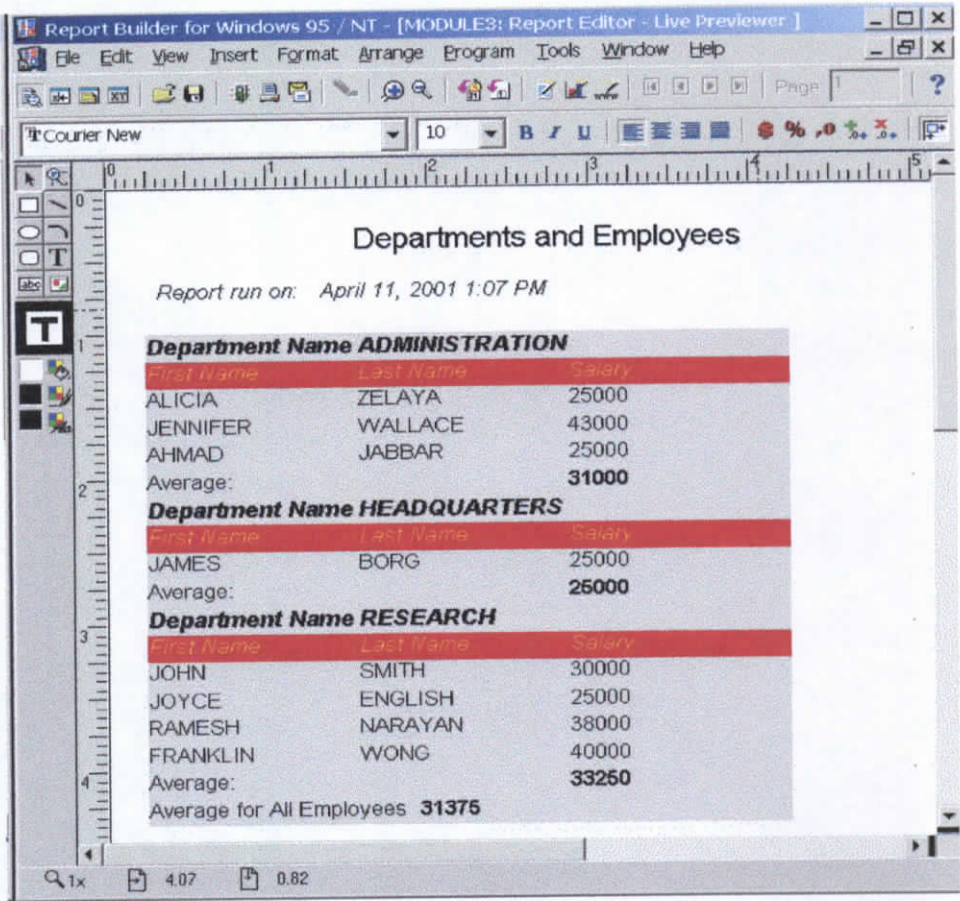


Fig. 3.79. Reporte Master/Detalle

Finalmente, grabar el reporte.

3.3.3.5.- Forma de Parámetro en Oracle Reports

Para cada reporte, todos los registros en las tablas respectivas han sido retornados. Pero hay situaciones donde sólo un grupo de registros de la base de datos son requeridos, se debe solicitar al usuario que ingrese algún criterio que será usado para filtrar el reporte, se conoce en **Oracle Reports** como **Forma de Parámetro**.

Las **Formas de Parámetros** son simples formas **Oracle**, que solicitan al usuario alguna información antes que el reporte sea ejecutado. Esta información puede ser parte de una cláusula **WHERE** en la consulta del reporte, y es usada para filtrar los registros retornados en el reporte. Los parámetros en **Oracle Reports** pueden ser

llenados por el usuario en una forma de parámetros, o pueden ser llenadas automáticamente pasando los parámetros desde una forma de **Oracle Forms**.

Para crear una forma de parámetro simple, que permita restringir una consulta en la tabla **Empleado**, y muestre solo los empleados en un número de departamento (**Dno**) suministrado por el usuario.

1. Crear una nuevo reporte con el asistente, con el título **Empleados** y un modelo **Tabular**.
2. Usar la siguiente consulta SQL.

```
SELECT *  
FROM Empleado  
WHERE Empleado.dno = :Parametro_Dno;
```

:Parametro_Dno es el nombre de un parámetro en tiempo de ejecución, cuando el reporte se esté ejecutando, se solicita al usuario un valor para el parámetro y así se completa las cláusula **WHERE** para la consulta. Hacer clic en el botón **Next** y se presenta la alerta:

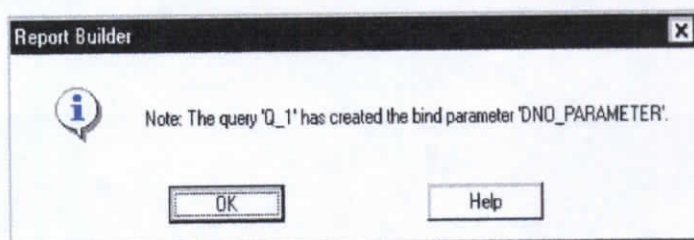


Fig. 3.80. Alerta para Forma de Parámetros

3. Escoger todas las columnas para visualizar en el reporte.
4. Cuando el asistente de reportes ha finalizado, el reporte y la forma de parámetros (basada en los parámetros dados en la consulta **SQL**) serán creados. Cuando se ejecuta la pre visualización, la forma de parámetros se visualiza:

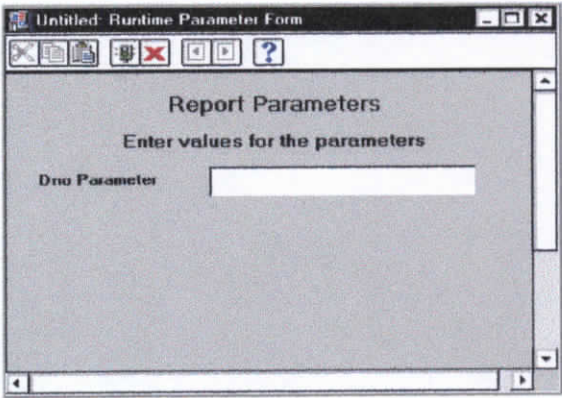


Fig. 3.81. Forma de Parámetros

3.3.3.6.- Construir una Forma con Parámetros Personalizados

Siempre que un reporte se ejecuta, una forma de parámetros por omisión que muestra **Parámetro_Dno** se visualiza. Sin embargo, es posible crear una forma de parámetros personalizada.

- 1. Pasar al navegador de Objetos, seleccionar **Tools | Parameter Form Builder**.

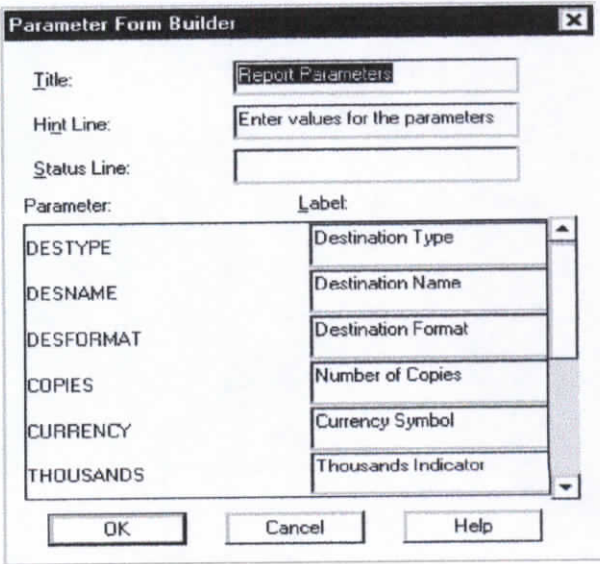


Fig. 3.82. Constructor de Forma de Parámetros

Un número de parámetros preconstruidos del sistema se visualizan, y pueden ser seleccionadas haciendo clic sobre ellos con el botón derecho

- del ratón. Al final de la lista se encuentra **Parametro_Dno** y debería estar seleccionado.
- Adicionalmente al **Parametro_Dno**, seleccionar los parámetros **DesType**, **DesName** y **Copies**. Luego hacer clic en el botón **Ok**, y se crea una nueva forma de parámetros automáticamente.

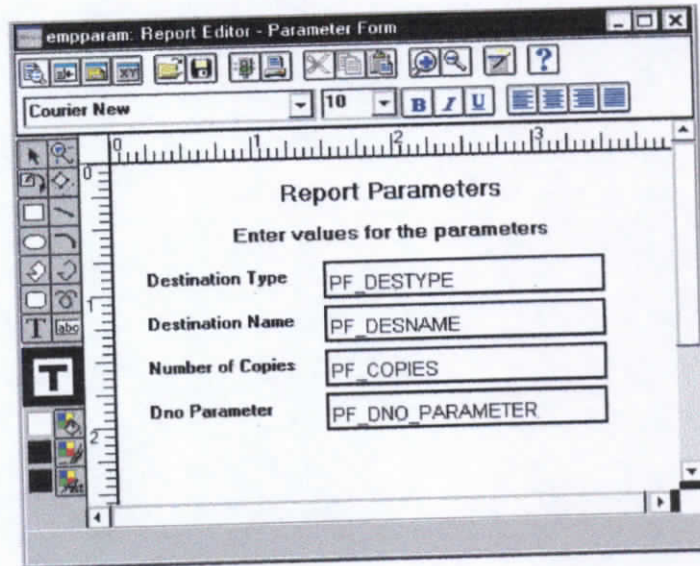


Fig. 3.83. Forma de Parámetros Personalizada

- Grabar el reporte. Cuando la forma sea ejecutada (**Program | Run Report**), la nueva forma de parámetros se visualiza:

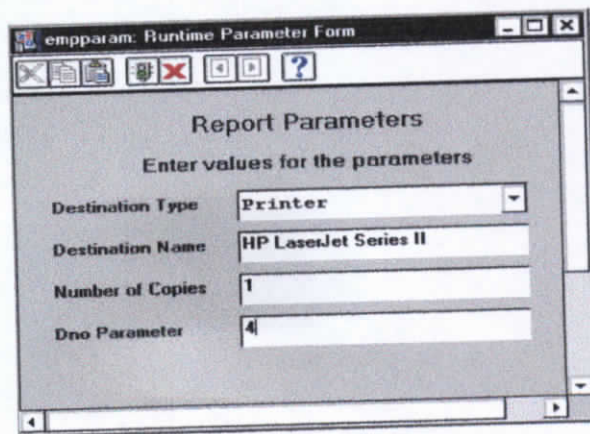


Fig. 3.84. Ejecución de la Forma de Parámetros

Con lo cuál, una copia del reporte será enviada a la impresora **HP LaserJet Series II**, y solo muestra los empleados en el departamento número 4.

Otras opciones de **Destination Type** incluyen: Screen, File y E-mail.

Las formas de parámetros pueden ser personalizadas para incluir listas de valores, indicaciones para el usuario entre otras.

3.3.4.- Opciones en Oracle Forms

En un sistema de base de datos completo, las aplicaciones consisten de muchas formas y reportes, que pueden ser usadas por un gran número de usuarios. El acceso a formas y reportes se realiza típicamente a través de opciones, que pueden ser contruidos en **Oracle Developer**, como parte del diseñador de **Forms**.

En Oracle **Forms**, cada forma se ejecuta con:

1. Las opciones por omisión son: **Action**, **Edit**, **Block**, **Field**, **Record**, **Query**, **Window**, **Help**, que no pueden ser cambiadas.



Fig. 3.85. Opciones por Omisión

2. Una opción personalizada se puede definir como un módulo separado, y luego enlazarla a la forma en tiempo de ejecución.
3. Sin opciones.

La opción por omisión es parte del módulo **Forms**, pero los módulos de opciones personalizados están separados de los módulos de forma, de forma que, cuando una aplicación de simple forma se ejecuta con una opción personalizada, hay dos archivos ejecutables:

- ☒ Un módulo de forma **.Fmx**
- ☒ Un módulo de opción **.Mmx**

En una aplicación multiforma, múltiples formas pueden compartir la misma opción, o cada forma puede invocar una opción diferente.

Para especificar explícitamente la opción por omisión en **Forms**:

1. Seleccionar el nombre de la forma/módulo en el Navegador de Objetos, hacer clic con el botón derecho del ratón y escoger **Properties**.
2. En la propiedad **Menu Module**, ingresar **Default&SmartBar** para obtener la opción por omisión y una barra de botones bajo ésta.

Si se desea que la forma se ejecute sin opciones, dejar la propiedad **Menu Module** en blanco.

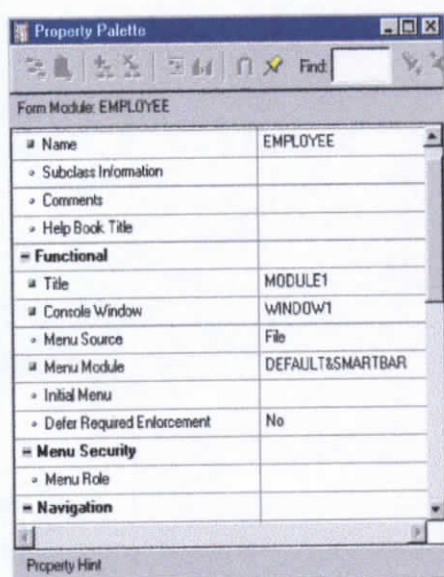


Fig. 3.86. Propiedad Menu Module

La opción por omisión es conveniente para el control sobre la forma actual que se está ejecutando, sin embargo, no contiene items de opciones personalizados que pertenezcan a una aplicación específica.

3.3.4.1.- Jerarquía de Opciones

La jerarquía de opciones **Oracle Forms**, contiene tres categorías de objetos:

1. **Menu modules:** Similar a los módulos de forma, puede contener items propios (opciones e items de opciones), unidades de programa y otros objetos.

2. **Menus:** Incluyendo opciones principales, opciones individuales, y subopciones
3. **Menu items**

Para diseñar una opción personalizada se requiere:

Crear un Módulo de Opciones

Hay varias formas para crear un nuevo módulo de opciones:

1. Seleccionar **File | New** y luego marcar **Menu**.
2. Navegar al ramal **Menus** en el Navegador de Objetos, seleccionar **Navigator | Create**.

Un nuevo módulo de opciones es creado, con un nombre por omisión, hacer clic en el nombre de la opción y asignarle un nombre significativo.

Para crear opciones e items de opciones, se utiliza **Editor Menu**, para activarlo, iluminar el nombre de la opción, luego **Tools | Menu Editor**.

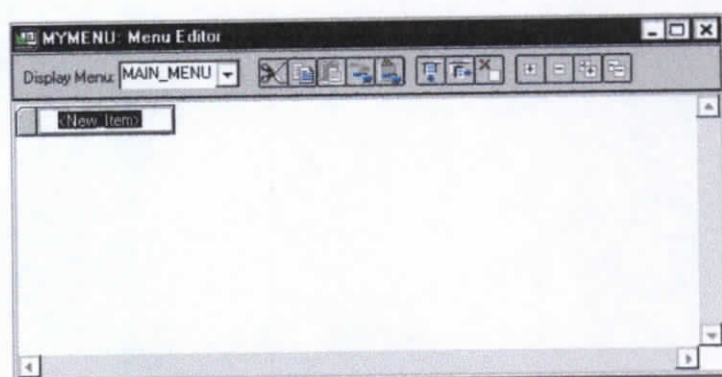


Fig. 3.87. Ventana Menu Editor

Los items de la opción principal van en la parte superior de la pantalla, para adicionar una nueva opción, hacer clic en el icono **Build Across** .

Las subopciones e items de opciones aparecen bajo cada opción. Para adicionar una nueva subopción o ítem de opción, hacer clic en el icono **Build Down** .

Para cambiar el nombre de una opción o ítem de opción, hacer clic en el objeto opción y sobrescribir el nombre que aparece.

Con los iconos, crear y cambiar las etiquetas las opciones:

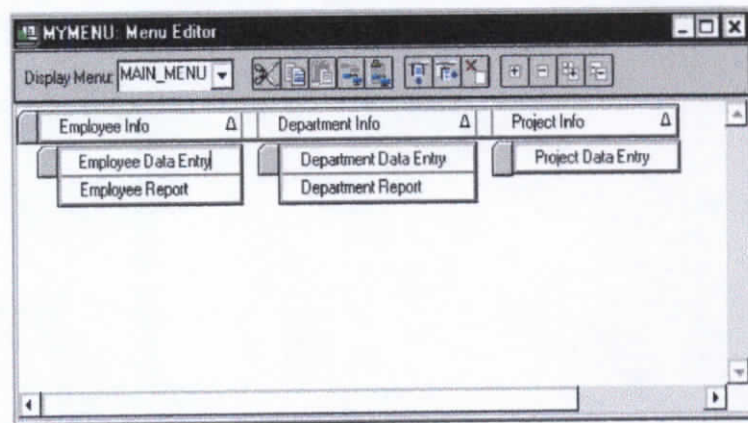


Fig. 3.88. Items de Opciones

Una vez completado, la estructura básica de opciones será colocada.

El comportamiento por omisión al hacer clic en cualquier opción, es que los items de opciones son visualizados. Para asignar comandos a cada ítem de opción, de forma que cuando un usuario haga clic, una forma, reporte o script **SQL*Plus** sea ejecutado.

Para asignar un comando a un ítem de opción:

1. Seleccionar el ítem de opción deseado en el Editor **Menu**, hacer clic con el botón derecho el ratón y escoger **Properties**.
2. Las propiedades de interés son:
 - **Command type:** Indica que tipo de comando debe ser ejecutado: Form, Report, PL/SQL, other menu. En general, el tipo de comando **PL/SQL** es usado para pasar a **Oracle Forms** y **Oracle Reports**.
 - **Menu Item Code** (o **Command Text**): El texto del comando actual a ejecutar.

Para ejecutar una forma desde el ítem de opción, escoger **Command Type PL/SQL**. Hacer doble clic en la propiedad **Menu Item Code** (o **Command Text**), en el editor ingresar:

```
BEGIN
    RUN_PRODUCT(FORMS, 'EMPLEADO',
    SYNCHRONOUS, RUNTIME, FILESYSTEM, ", ");
END;
```

Donde:

RUN_PRODUCT es una función preconstruida **PL/SQL** que ejecuta otro producto **Oracle Developer**, los parámetros son:

- **Producto:** FORMS o REPORTS
- **Documento/Módulo:** nombre del archivo entre apóstrofes
- **Modo de Comando:** SYNCHRONOUS o ASYNCHRONOUS
- **Modo Ejecutable:** RUNTIME
- **Ubicación:** FILESYSTEM o DATABASE
- **Lista de Parámetros:** Una lista de parámetros para pasar a la forma o NULL
- **Display:** Tipo de pantalla para mostrar el producto (NULL para indicar la actual).

Compilar el código con el botón **Compile**, luego cerrar el editor PL/SQL.

Repetir los pasos para cada ítem de opción.

Una opción final debería ser creada para salir de la aplicación, llamada: **Salir de Aplicación** e ingresar el código:

```
BEGIN
    EXIT_FORM;
END;
```

Donde: **EXIT_FORM** es un procedimiento preconstruido que existe en la forma actual.

Grabar y Generar el Módulo de Opciones

Una vez creada toda la estructura de opciones y los comandos para cada ítem de opción han sido especificados, el módulo debe ser grabado a un archivo (con la extensión **.mmx**), generado o compilado.

Para grabar el módulo de opciones, seleccionar **File | Save**.

Para generar o compilar el módulo de opciones, seleccionar **File | Administration | Generate**.

Enlazar un Módulo de Opciones a una Forma

Una opción personalizada debe ser enlazada a una forma para que sea visualizada. Típicamente, una forma sin entrada de datos se usa como la pantalla principal.

1. Crear una nueva forma sin bloques o ítems en ella. Seleccionar **File | New | Form**.

Hacer clic en el nombre de la forma en el **Navegador de Objetos** y renombrarla como **forma_blanco**.

2. Abrir la paleta de propiedades de la forma y colocar las propiedades:
Menu Source: **File** La información de opciones viene desde un archivo.
Menu Module: **mi_opcion.mmx** el nombre del módulo de archivo compilado, que puede ser dado por una ruta completa **c:\mi_opcion.mmx**

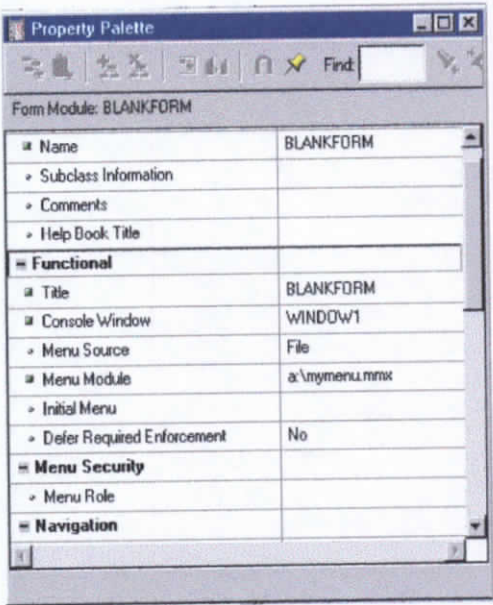


Fig. 3.89. Propiedades Forma en Blanco

- Grabar la forma como archivo **forma_blanco.fmb** y generarla. (Es recomendable grabar la forma en la misma unidad y directorio que el archivo de opciones).
- Ejecutar la forma.
 Cuando se ejecuta la forma en blanco, la opción por omisión es reemplazada por el Módulo de Opción **mi_opción**.

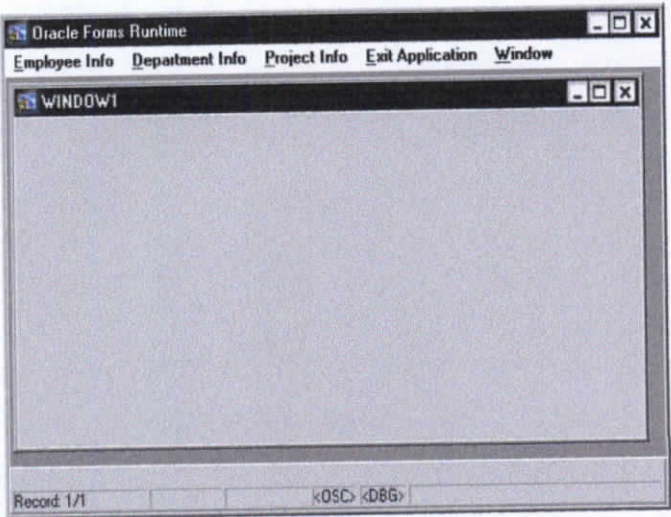


Fig. 3.90. Forma con Opciones Personalizadas

La forma en blanco usada como contenedor de la opción, no necesita estar en blanco, se puede usar el editor **Layout** para colocar algún gráfico o el título de la aplicación en un lienzo.

3.3.5.- Graphics y Chart

Graphics Builder se utiliza para construir aplicaciones, que permitan a los usuarios finales visualizar datos y gráficamente, integrarlos con una base de datos.

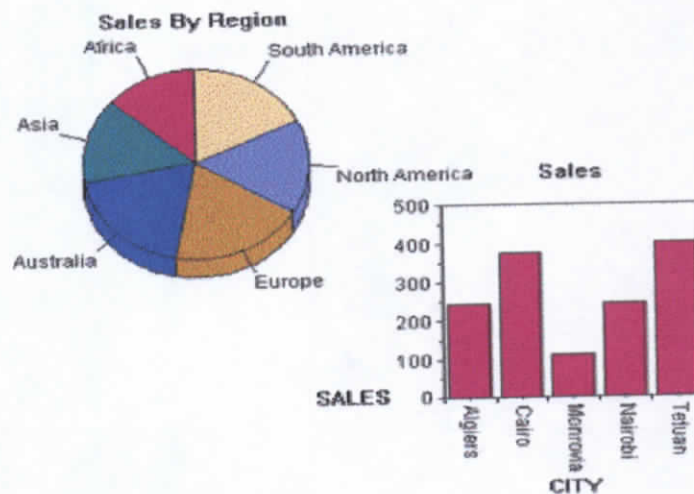


Fig. 3.91. Graphics Builder

El Diseñador **Oracle Graphics** puede ser usado para crear una variedad de gráficos y esquemas incluyendo esquemas de barras, dibujos de líneas y gráficos circulares. Típicamente un objeto gráfico diseñado con **Oracle graphics** es incluido en una forma o en un reporte. El diseñador **Oracle Forms** incluye un asistente que construye un objeto gráfico automáticamente con los datos de la forma, en un bloque de datos existente.

3.3.5.1- Adicionar un Chart a una Forma

Un objeto **Oracle Graphics** puede ser adicionado a una forma, por ejemplo, crear una nueva forma Master/Detalle con las tablas **Departamento (Dnumero y Dnombre)** como bloque master y **Empleado (Nombre, Apellido, Dno y Sueldo)** como bloque detalle. Generar un esquema que compare el **Sueldo** de los empleados en un departamento dado.

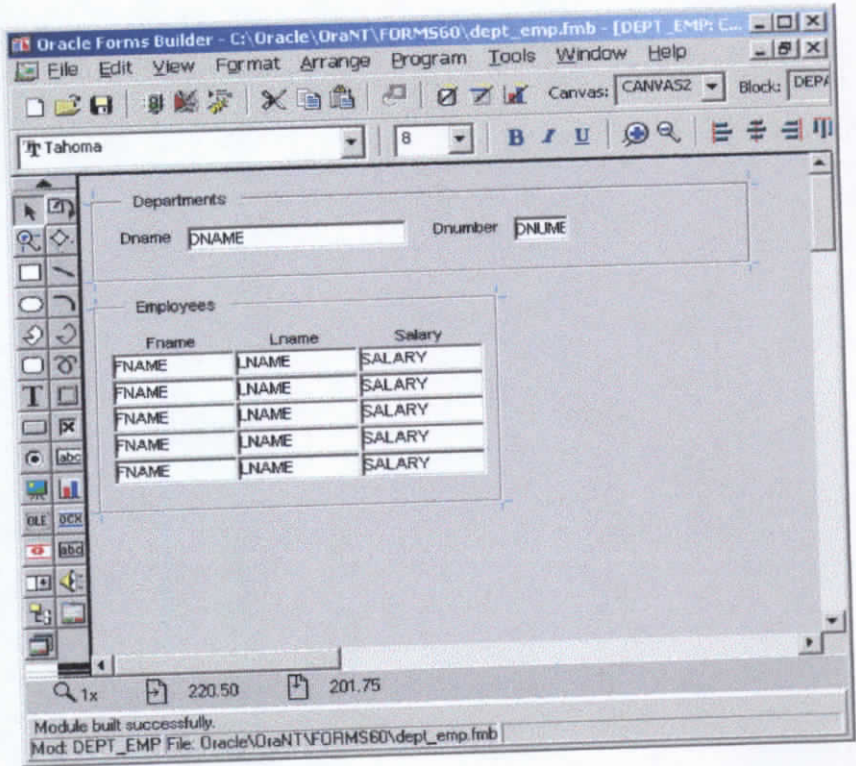


Fig. 3.92. Forma Master/Detalle Departamento-Empleados

Grabar la forma en disco antes de crear el esquema.

Para iniciar el Asistente **Chart**, hacer clic en un área de la forma en la vista layout, seleccionar **Tools | Chart Wizard**, y la primera pantalla del asistente aparece:

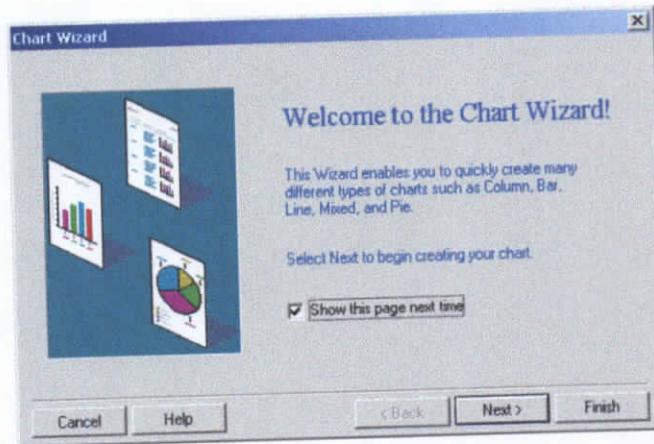


Fig. 3.93. Asistente Chart

Hacer clic en el botón **Next**, escoger el tipo de esquema (y subtipo si es aplicable) a ser creado. Seleccionar **Pie** y **Depth**, hacer clic en el botón **Next**:

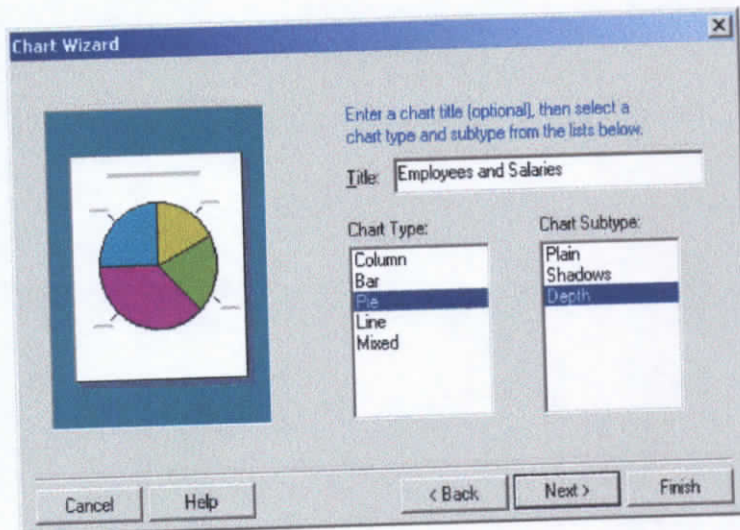


Fig. 3.94. Esquema y Subtipo de Chart

Escoger el bloque de datos que utilizará el esquema. Todos los bloques de datos en la forma actual se visualizan, escoger el bloque **Empleado** y hacer clic en el botón **Next**:

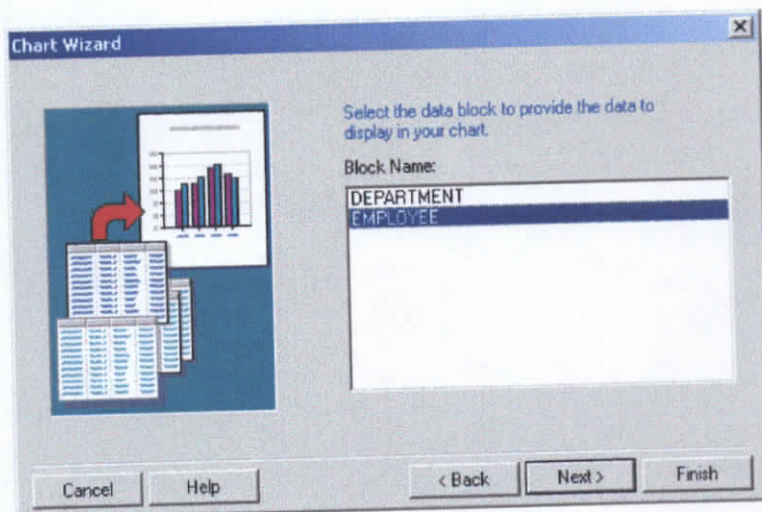


Fig. 3.95. Seleccionar Bloque de Datos para Chart

Escoger el campo en el bloque de datos que se usará para el eje X (o etiquetas de categoría) para el esquema, campo **Apellido**, hacer clic en el botón **Next**:

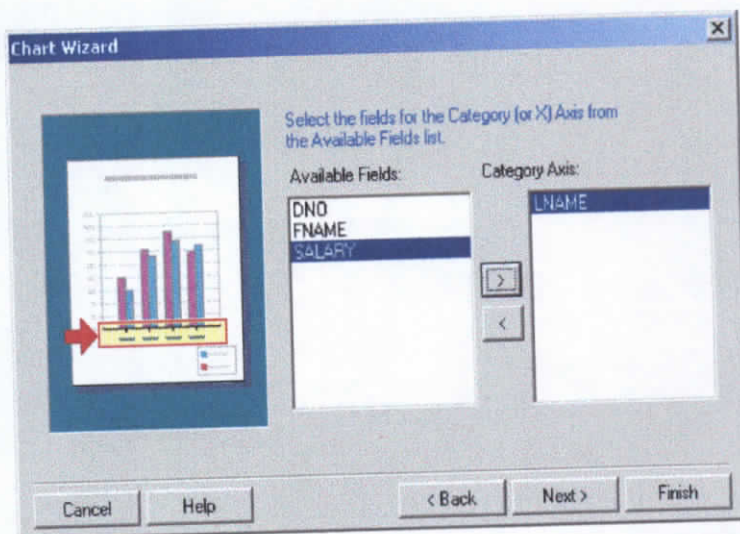


Fig. 3.96. Campos para el eje X del Chart

Escoger el campo en el bloque de datos que se usará como valores para el eje Y, escoger el campo **Sueldo** y hacer clic en el botón **Next**:

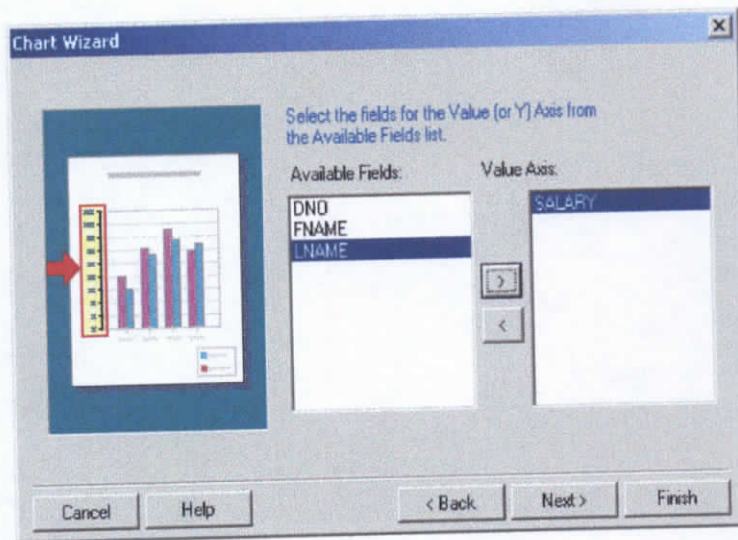


Fig. 3.97. Campos para el eje Y del Chart

Finalmente, el objeto debe ser grabado en un archivo con la extensión **.ogd** y presionar el botón **Finish**:

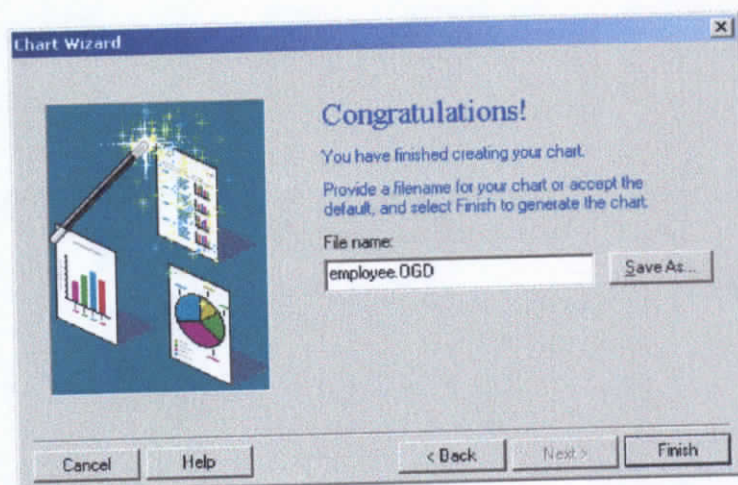


Fig. 3.98. Nombre del Archivo Chart

El objeto gráfico será creado y colocado en la forma, con el ratón se puede mover o cambiar el tamaño del objeto. Grabar la forma nuevamente y ejecutarla.

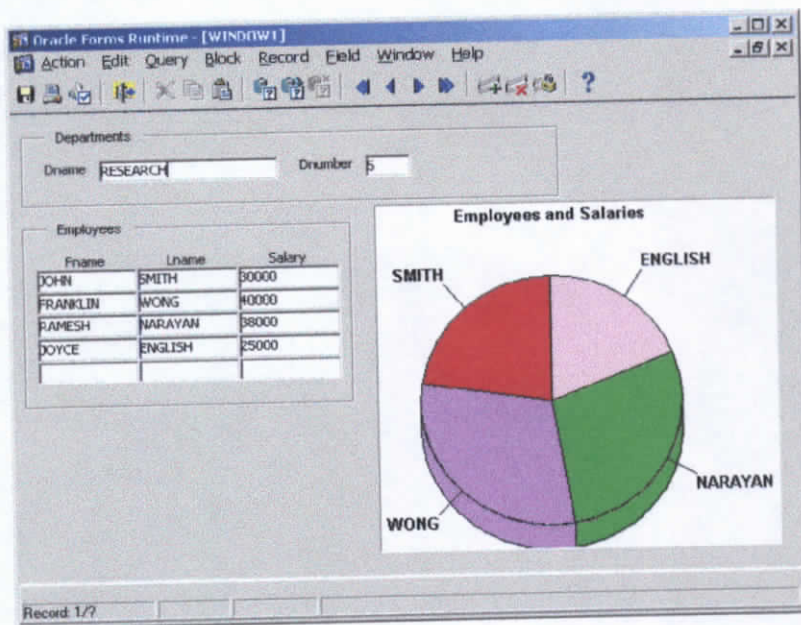


Fig. 3.99. Forma con Chart

Cuando se copia archivos de formas que utilicen **Oracle Graphics**, asegurar que se copien los archivos de objetos gráficos asociados.

3.3.6.- Oracle Schema Builder

Schema Builder provee una interfase de usuario gráfica fácil de usar, para definir y modificar tablas, vistas y otros objetos de esquema.

Ejecutar **Schema Builder**, y como en las otras herramientas **Oracle**, primero conectarse a la base de datos.

La pantalla principal de **Schema Builder** es:

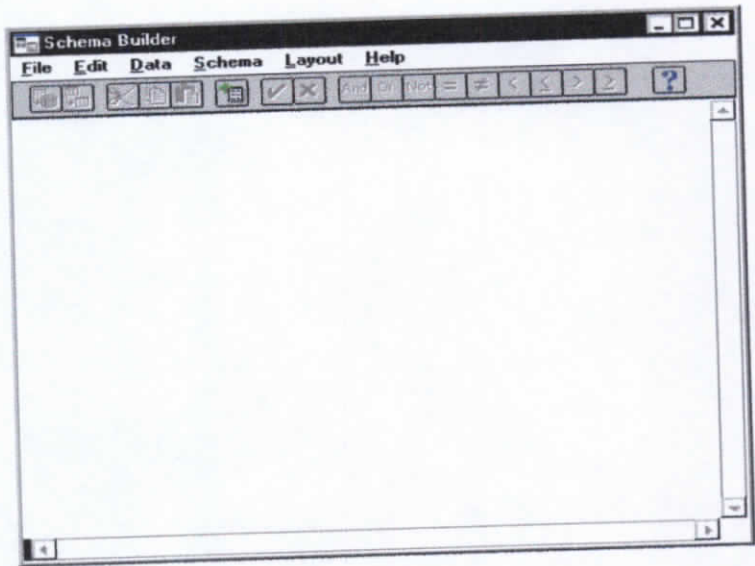


Fig. 3.100. Pantalla de Oracle Scheme

Crear una nueva tabla es un proceso directo: primero seleccionar **Schema | Create Table**, en la opción que aparece escoger **Table**. El apuntador del ratón cambia a una flecha con un signo +, hacer clic en un espacio de la pantalla principal de **Schema Builder** y se crea una nueva tabla.

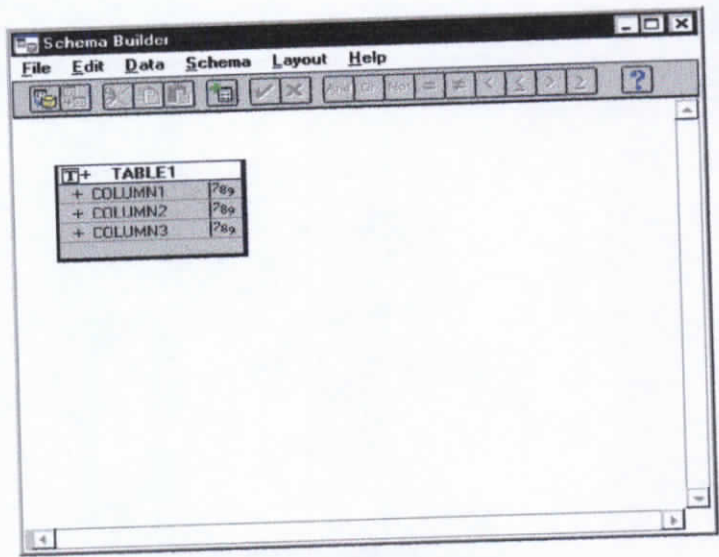


Fig. 3.101. Crear una Tabla en Schema Builder

La tabla tiene como nombre por omisión **Table1**, y varias columnas. Para cambiar el nombre de la tabla, hacer clic en el nombre actual e ingresar el nuevo nombre (sin espacios), para finalizar presionar **Enter**.

Para cambiar el nombre de una columna, hacer clic sobre un nombre de columna e ingresar su nuevo nombre.

Para adicionar más columnas a la tabla, simplemente hacer clic en la esquina inferior derecha de la tabla y arrastrarla hacia abajo, aparecen más columnas con nombres por omisión.

Para cambiar el tipo de dato y otras características de las columnas, hacer doble clic en el nombre de la columna, en el diálogo **Define Column** iluminar la columna que se desea modificar y realizar los cambios necesarios.

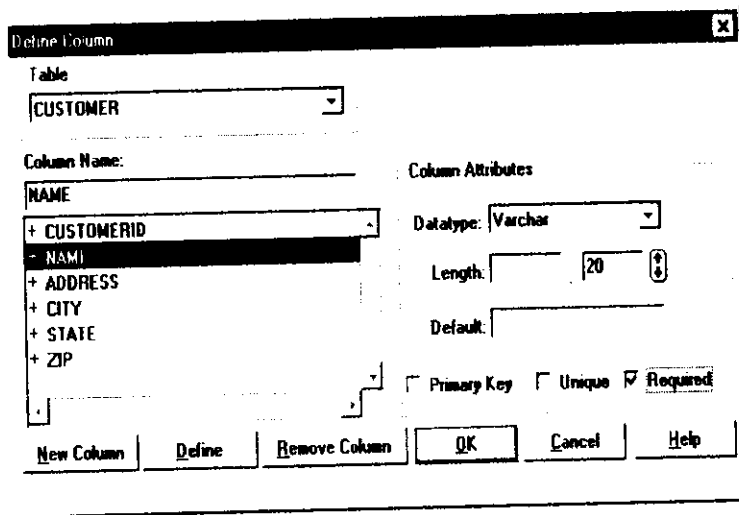


Fig. 3.102. Cambiar Características de Columnas

Definidas todas las columnas en la tabla **Cliente**:

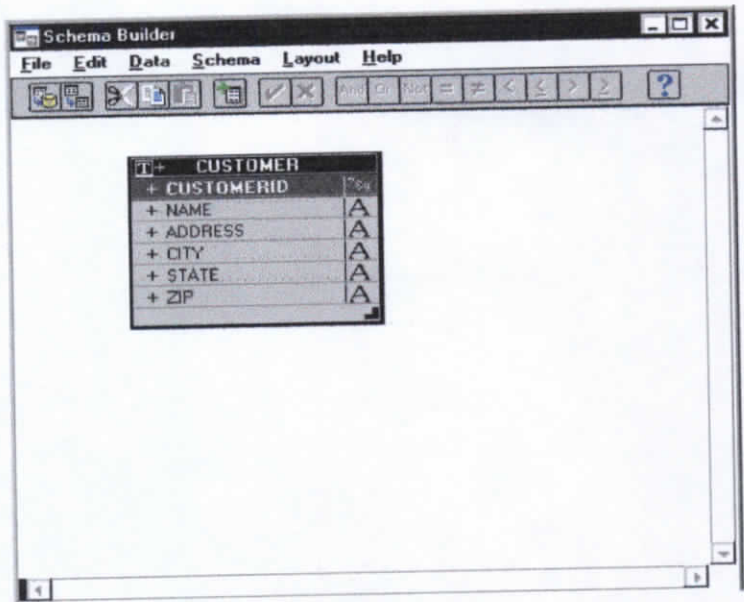


Fig. 3.103. Tabla Cliente en Schema Builder

Una vez que las columnas de la tabla han sido renombradas y definidas, debe ser grabada en la base de datos. Para ver las sentencias **SQL** que serán enviadas al servidor de base de datos, seleccionar **Schema | Show SQL**.

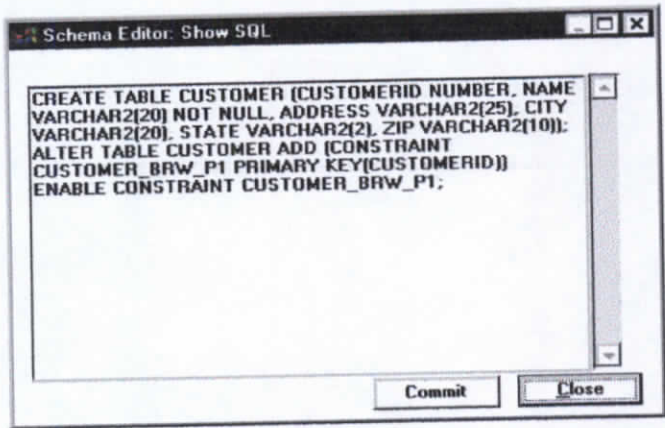


Fig. 3.104. Sentencia SQL Generada por Schema Builder

Las sentencias **SQL** que serán usadas para crear el esquema, pueden ser grabadas en un archivo de texto (con la extensión **.sql**) para futuros usos, seleccionar **File | Save SQL**.

Para hacer permanente el esquema en la base de datos, hacer clic en el botón **Commit** o, seleccionar **Schema | Commit Schema**, aparece una caja de diálogo que muestra el progreso.

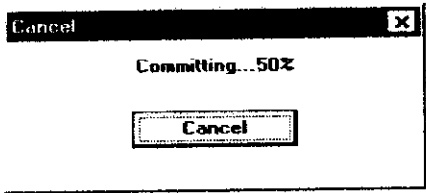


Fig. 3.105. Hacer Commit a la Base de Datos

Visualizar Objetos de Esquema Existente

Si el esquema tiene algunas tablas o vistas ya definidas, pueden ser visualizadas con **Data | Select Data Tables**, en la caja de diálogo:

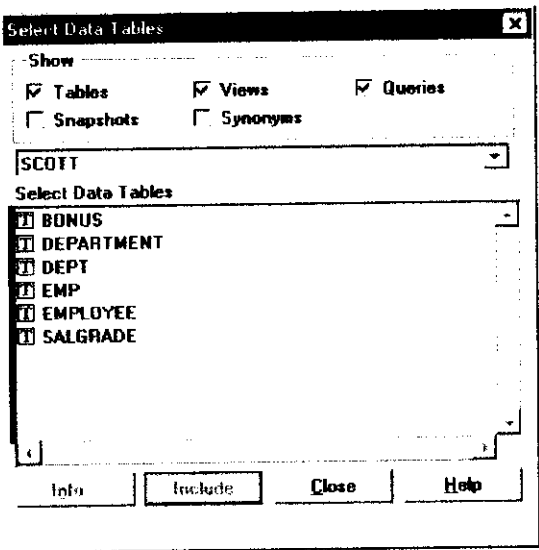


Fig. 3.106. Objetos del Esquema

Para incluir una tabla en la pantalla principal de **Schema Builder**, iluminar una tabla y hacer clic en el botón **Include**.

Al incluir las tablas **Empleado** y **Departamento**:

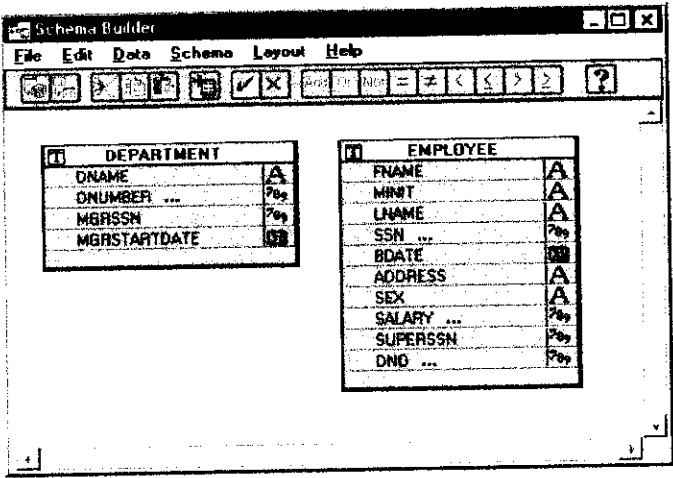


Fig. 3.107. Incluir Tablas Existentes en Schema Builder

Adicionar Relaciones entre Tablas

Una vez que las tablas han sido creadas y visualizadas en la pantalla principal de **Schema Builder**, se pueden definir las relaciones entre ellas, la columna **Dnumero** de la tabla **Departamento** está relacionada con la columna **Dno** en la tabla **Empleado**.

Para crear las restricciones apropiadas para la relación, seleccionar **Schema | Define Relationship**, en la caja de diálogo aparecen los espacios para las llaves Primaria y Foránea, llenarlas con **Foreign Key: Empleado.Dno** y **Primary Key: Departamento.Dnumero**:

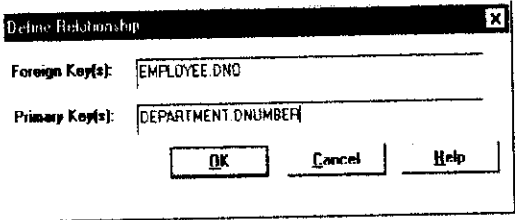


Fig. 3.108. Relacionar Tablas en Schema Builder

Hacer clic en el botón **Ok** para confirmar la relación, la visualización cambiará y aparece una línea que enlaza la Llave Primaria y la Llave Foránea.

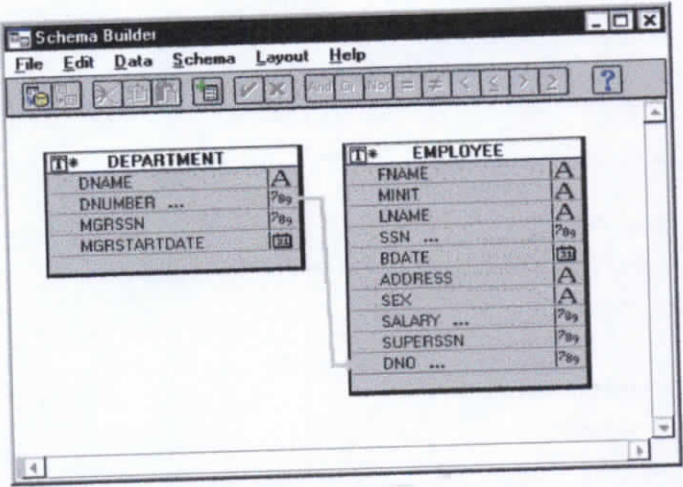


Fig. 3.109. Tablas Relacionadas en Schema Builder

Schema Builder provee una vista **SQL** que puede ser usada para crear las restricciones de llave foránea que implementarán la relación en el esquema de la base de datos. Para ver **SQL**, seleccionar **Schema | Show SQL**.

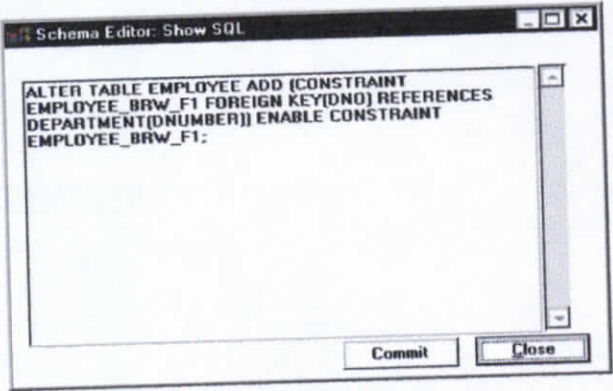


Fig. 3.110. SQL para Mostrar las Relaciones

Las sentencias SQL para crear las restricciones de llave foránea pueden ser grabadas en un archivo con **File | Save SQL**.

Finalmente, para hacer **commit** los cambios a la base de datos, seleccionar **Schema | Commit Schema**.

La próxima vez que se visualicen **Departamento** y **Empleado** en **Schema Builder**, la relación aparece automáticamente.

CAPITULO IV

DIAGNÓSTICO DEL SISTEMA ACTUAL

4.1.- Resultados de las técnicas de información

4.1.1.- Ubicación y Propietarios

La Hacienda El NILO y El ROSARIO se encuentra ubicada en la Parroquia **Marcos Espinel** del Cantón **Píllaro**, Provincia del **Tungurahua**, y sus propietarios son: Carlos Vargas y Betty Vargas.

4.1.2.- Sistema Actual

Todos los procesos que se llevan a cabo en la Hacienda El Nilo y El Rosario, principalmente en lo que se refiere al control de producción de leche, se realizan en forma manual, determinándose que no es el más adecuado y los propietarios consideran primordial la automatización de los mismos.

Toda la información del ganado y la producción de leche, se lleva por medio de tarjetas y hojas de producción diaria, respectivamente.

4.2.- Deficiencias del sistema

Por lo anotado, los propietarios han detectado errores en el registro tanto de animales como de producción, generando un perjuicio económico principalmente por el tiempo que se requiere para ubicar la información y los totales de producción diaria.

Del análisis realizado, se desprende que el proceso manual ha dejado de ser efectivo para el control y la toma oportuna de decisiones por parte de los propietarios de la hacienda.

4.3.- Análisis de requerimientos

Con el estudio minucioso de los problemas que se presentan en la hacienda El Nilo y El Rosario, con el propósito de agilizar, facilitar, y obtener confiabilidad en los datos y resultados, además de brindar las facilidades para el ingreso y obtención de información, los requerimientos o necesidades del software a desarrollar se pueden resumir en:

- ☒ Datos de Entrada
 - Ganado
 - Materia Prima
 - Registro de Clientes y Proveedores
 - Datos de Producción
 - Datos de Compra-Venta
 - Notas de Pedido
- ☒ Datos de Salida
 - Reportes de Ganado
 - Reportes de Producción (productos)
 - Reportes de Producción (leche)
 - Reportes de Clientes
 - Reportes de Proveedores
 - Facturas de Venta

Con la finalidad de:

- ☒ Garantizar la seguridad en los datos ingresados

5.1.1.- Planificación de las actividades

Nº	Actividad	Tiempo de Duración		
		Meses/Días	Inicio	Fin
1	Especificación de requerimientos	0.78 / 23	09/dic/2003	01/ene/2004
2	Análisis del sistema	1.56 / 47	02/ene/2004	17/feb/2004
3	Diseño	2.34 / 70	18/feb/2004	28/abr/2004
4	Codificación	2.34 / 70	29/abr/2004	07/jul/2004
5	Pruebas e Implantación	0.78 / 23	08/jul/2004	30/jul/2004
Duración del proyecto		7.80 / 234	09/dic/2004	31/jul/2004

Tabla 5.1. Planificación de Actividades

5.1.2.- Recursos del proyecto

El sistema propuesto será diseñado para trabajar bajo el sistema operativo Windows, por lo que se requiere los siguientes recursos:

Humanos

- ☒ Director de Tesis
- ☒ Asesores
- ☒ Desarrollador (Programador)

Hardware

- ☒ Computador servidor, Pentium IV o superior
- ☒ Computador cliente, Pentium III o superior
- ☒ Impresora
- ☒ HUB de 8 puertos

Software

- ☒ Sistemas Operativos NT (servidor) y Windows 98 (cliente)
- ☒ Base de Datos Oracle 8i o Superior
- ☒ Herramientas de Desarrollo Developer 6i
- ☒ Herramienta Case Erwin 3.5 o Superior

5.1.3.- Costo del sistema actual

Al no contar la hacienda con un sistema anterior, el costo del sistema actual está dado únicamente por las personas que trabajan en la hacienda (contadora y trabajadores).

5.1.4.- Costo del sistema propuesto

Para la estimación del costo real del proyecto se toma como base uno de los modelos COCOMO (Constructive Cost Model, Modelo Constructivo de Costos) de Barry Boehm¹:

- ☒ Modelo 1, **COCOMO Básico**, es invariable estático, que calcula el esfuerzo (y el costo) del desarrollo del software en función del tamaño del programa, expresado en líneas de código estimadas (**LDC**).
- ☒ Modelo 2, **COCOMO Intermedio**, calcula el esfuerzo del desarrollo de software en función del tamaño del programa y de un conjunto de **conductores de costo**, que incluyen la evaluación subjetiva del producto, del hardware, del personal y de los atributos del proyecto.
- ☒ Modelo 3, **COCOMO Avanzado**, incorpora todas las características de la versión intermedia, y lleva a cabo una evaluación del impacto de los conductores de costo en cada fase (análisis, diseño, etc.) del proceso de ingeniería del software.

Por las características del proyecto, se selecciona el método **COCOMO Básico**, con las siguientes ecuaciones:

☒ $E = a (KLDC)^b$

☒ $D = c (E)^d$

¹ PRESSMAN, Roger., Ingeniería del Software un Enfoque Práctico, Ed. MacGraw-Hill, Madrid España, 1994, págs. 90-93

Donde:

E = Esfuerzo personas – mes

KLDC = Miles de Líneas de Código

D = Tiempo de Desarrollo en meses

Los coeficientes **a**, **b**, **c** y **d**, se presentan en la siguiente tabla:

Proyecto de Software	a	b	c	d
Orgánico	2.4	1.05	2.5	0.38
Semilibre	3.0	1.12	2.5	0.35
Restringido	3.6	1.20	2.5	0.32

Tabla 5.2. Coeficientes a, b, c y d para las Ecuaciones

Como en el proyecto intervienen personas con variados niveles de experiencia, los requisitos son poco o medio rígidos y el nivel de complejidad y tamaño es intermedio, se toma el Modo de Proyecto **Semilibre**.

Se estima que el proyecto tendrá aproximadamente unas 1200 Líneas de Código, entonces **KLDC** = **1.2**, y:

$E = 3.0 (1.2)^{1.12}$
 $E = 3.68 \text{ personas/mes} = \mathbf{4 \text{ personas/mes}}$
 $D = 2.5 (3.68)^{0.35}$
 $D = \mathbf{3.94 \text{ meses}}$

Entonces: para el desarrollo del sistema se necesita 3.94 meses, trabajando 4 personas, pero como se cuenta con 2 personas (considerando al Director y Asesores como una persona, y al desarrollador del proyecto como otra), se estima un tiempo de duración de 7.8 meses, y la distribución de tiempo en las diferentes etapas del Ciclo de Vida del Proyecto son:

Actividad	Porcentaje	Meses
Especificación de Requisitos	10%	0.78
Análisis	20%	1.56
Diseño	30%	2.34
Codificación	30%	2.34
Pruebas e Implantación	10%	0.78
Tiempo de Duración del Proyecto		7.8 meses

Tabla 5.3. Ciclo de Vida del Proyecto

Para la estimación del costo total del proyecto (**CTP**), se tiene las fórmulas:

☒ $CTP = CD + CI$

- ☑ $CD = CFT + CTM + CMAT + OGD$
- ☑ $CI = 5\% \text{ de } CD$
- ☑ $CFT = TDP * SMCO * CTCO$
- ☑ $CTM = HTM * CHTM$
- ☑ $CMAT = \text{Suma}(\text{costo de materiales utilizados})$
- ☑ $OGD = \text{Suma}(\text{costo de pasajes, hospedaje, etc.})$

Donde:

CD = Costos Directos

CI = Costos Indirectos

CFT = Costo Fuerza de Trabajo

SMCO = Salario Mensual Categoría Ocupacional

CTCO = Cantidad de Trabajadores Categoría Ocupacional

CTM = Costo Tiempo Máquina

HTM = Horas Tiempo Máquina

CHTM = Costo HTM

CMAT = Costo de Materiales

OGD = Otros Gastos Directos

Entonces:

SMCO = \$ 200 + Beneficios = \$ 250

CFT = 7.8 meses * \$ 250 persona/mes * 2 personas = \$ 3900

HTM = Tiempos de (diseño + codificación + pruebas e implementación) *

Horas utilizadas al mes.

Horas utilizadas al mes = 20 días/mes * 4 horas/día = 80 horas/mes

HTM = (2.34 + 2.34 + 0.98) meses * 80 horas/mes = 452.8 horas

HTM = 453 horas

CTM = 453 horas * \$ 2 /hora = \$ 906

CMAT = \$ 100 (útiles de oficina)

OGD = \$ 50 (transporte)

CD = CFT + CTM + CMAT + OGD

CD = \$ 3900 + \$ 906 + \$ 100 + \$ 50 = \$ 4956

CTP = \$4956 + 5% (\$ 4956) = \$ 5203.80

CTP = \$ 5203.80

5.2.- Análisis de riesgos

Todo desarrollo de proyecto informático está sujeto a riesgos inevitables que harían tambalear su éxito, para lo cuál la Ingeniería de Software permite realizar un análisis de todos los riesgos posibles, para obtener un sistema fiable, económico y que funcione eficientemente.

Las actividades son:

☒ **Identificación del riesgo:** Consiste en enumerar los riesgos concretos del proyecto, dentro de las categorías:

- Riesgos del proyecto, identifican a potenciales problemas presupuestarios, de tiempo, de personal, de organización, de recursos, del cliente y de requisitos
- Riesgos técnicos, identifican a problemas del diseño, implementación, interfase, verificación y mantenimiento.
- Riesgos del negocio, se presentan los riesgos del mercado.

En la identificación, es conveniente establecer hasta donde pueden alcanzar los riesgos, de modo que se definan estrategias acordes, agendas que se adapten a cambios y otras medidas que permitan reducir el impacto de los riesgos.

☒ **Proyección o Impacto del riesgo:** Conocida como Estimación del riesgo, intenta evaluar cada riesgo de dos formas: la probabilidad que el riesgo sea real, y las consecuencias del mismo. El conjunto de riesgos generales que se pueden presentar en el proyecto, con su impacto sobre la base de la probabilidad que pueda darse:

- Falta de consistencia en la información que maneja el sistema
- Incompatibilidad con el equipo de hardware
- Cambios en las distintas tareas de la planificación de actividades
- Manejo inadecuado del sistema
- Alteraciones en la Base de Datos por personas no autorizadas
- Deterioro de disquetes del sistema, por accidentes o por el tiempo

La probabilidad que ocurran son:

- Improbable 10% a 20%
- Poco Probable 20% a 35%
- Moderado 35% a 50%
- Probable 50% a 70%
- Muy probable 70% a 90%

- ☑ **Evaluación del riesgo:** Intenta dar prioridad a los riesgos que no se han cubierto, para pensar la forma de controlarlos y/o prevenirlos.
En la mayoría de proyectos, el costo, el tiempo y el rendimiento son tres niveles típicos de referencia para el riesgo.
Para el proyecto, se considera que ninguno de los riesgos es probable o muy probable, por ser desarbolado en un ambiente seguro, libre de impacto inoportuno.
- ☑ **Gestión del riesgo:** La gestión y supervisión del riesgo toma como base las tres actividades anteriores, y permite dar un conjunto de pasos para minimizarlos o eliminarlos.

CAPITULO VI

DISEÑO DEL SISTEMA

6.1.- Análisis Estructurado

Se ha adoptado la metodología de **Análisis Estructurado**, que tiene como objetivo principal estructurar u organizar las tareas asociadas con la determinación de requerimientos, para lograr la comprensión completa y exacta de una situación dada. es decir, está orientado a la actividad de construcción de modelos.

Se necesita especificar lo que se requiere que realice el sistema o la aplicación. sin establecer como lo cumplirá, o la forma en que se implementará, permitiendo observar los elementos lógicos separados de los componentes físicos.

6.1.1.- Componentes

- ☑ **Símbolos Gráficos:** son las convenciones para identificar y describir los componentes de un sistema y las relaciones entre ellos.

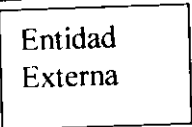
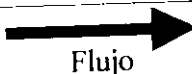
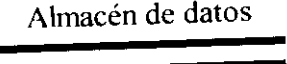
Símbolo	Descripción
 Entidad Externa	Productor o consumidor de información que reside fuera de los límites del sistema a ser modelado.
 Proceso	Transformador de información, reside dentro de los límites del sistema a ser modelado
 Flujo	Elemento o elementos de dato(s), indican el flujo de los datos
 Almacén de datos	Depósito de datos que se guardan para ser usados por uno o más procesos.

Tabla 6.1. Símbolos Gráficos para Componentes del Sistema

- ☑ **Diccionario de Datos:** contiene la descripción de todos los datos utilizados en el sistema. es un catálogo o depósito de elementos del sistema.
- ☑ **Descripción de procesos y procedimientos:** son declaraciones formales que emplean técnicas y lenguajes que permitan a los analistas describir actividades importantes que forman parte del sistema.
- ☑ **Reglas:** son los estándares para describir y documentar el sistema en forma correcta y completa.

6.1.2.- Análisis de Flujo

Es el empleo de los datos en cada actividad, documenta los hallazgos con diagramas de flujo de datos y en los diccionarios de datos que describen de manera formal los datos del sistema y los sitios donde son utilizados.

Contiene las características lógicas de los sitios donde se almacenan los datos del sistema, incluyendo sus nombres, descripción, alias, contenidos y organizaciones.

6.1.2.1.- Descripción del Diccionario de Datos

El diccionario de datos se desarrolla durante el análisis del flujo de datos y ayuda a los analistas que participan en la determinación de los requerimientos del sistema.

Los diccionarios de datos se utilizan para:

- ☑ Manejar los detalles en sistemas grandes
- ☑ Comunicar un significado común para todos los elementos del sistema
- ☑ Documentar las características del sistema
- ☑ Facilitar el análisis de los detalles con la finalidad de evaluar las características y determinar donde efectuar cambios en el sistema
- ☑ Localizar errores y omisiones en el sistema

Y contienen dos tipos de descripciones para el flujo de datos:

- ☑ Elemento de datos, bloques básicos para todos los demás datos del sistema.
- ☑ Estructura de datos, grupo de datos elementales que están relacionados con otros, y en conjunto describen un componente del sistema

6.1.2.2.- Diagrama de Flujo de Datos (DFD)

Permite simplificar la complejidad del sistema, representando los diferentes procesos de una manera sencilla, se fundamenta en las siguientes características:

- ☑ Representar gráficamente los límites del sistema en estudio.
- ☑ Mostrar el movimiento de los datos y las transformaciones a través del sistema.
- ☑ Diferenciar las restricciones físicas de las lógicas.

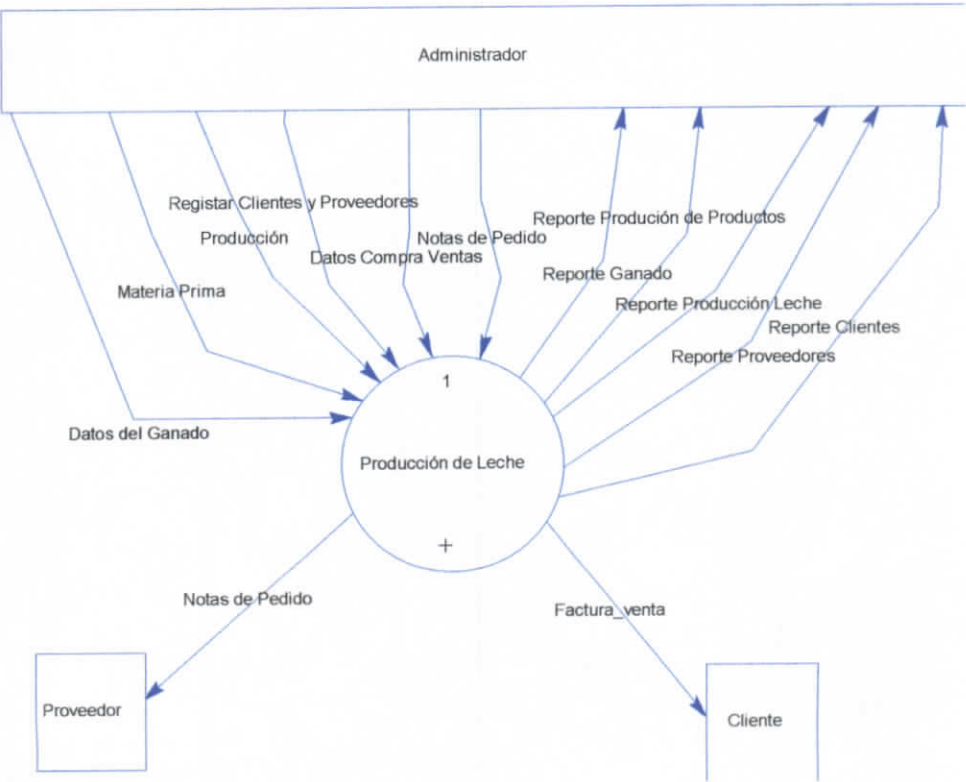


Fig. 6.1. DFD Nivel 0, Diagrama de Contexto

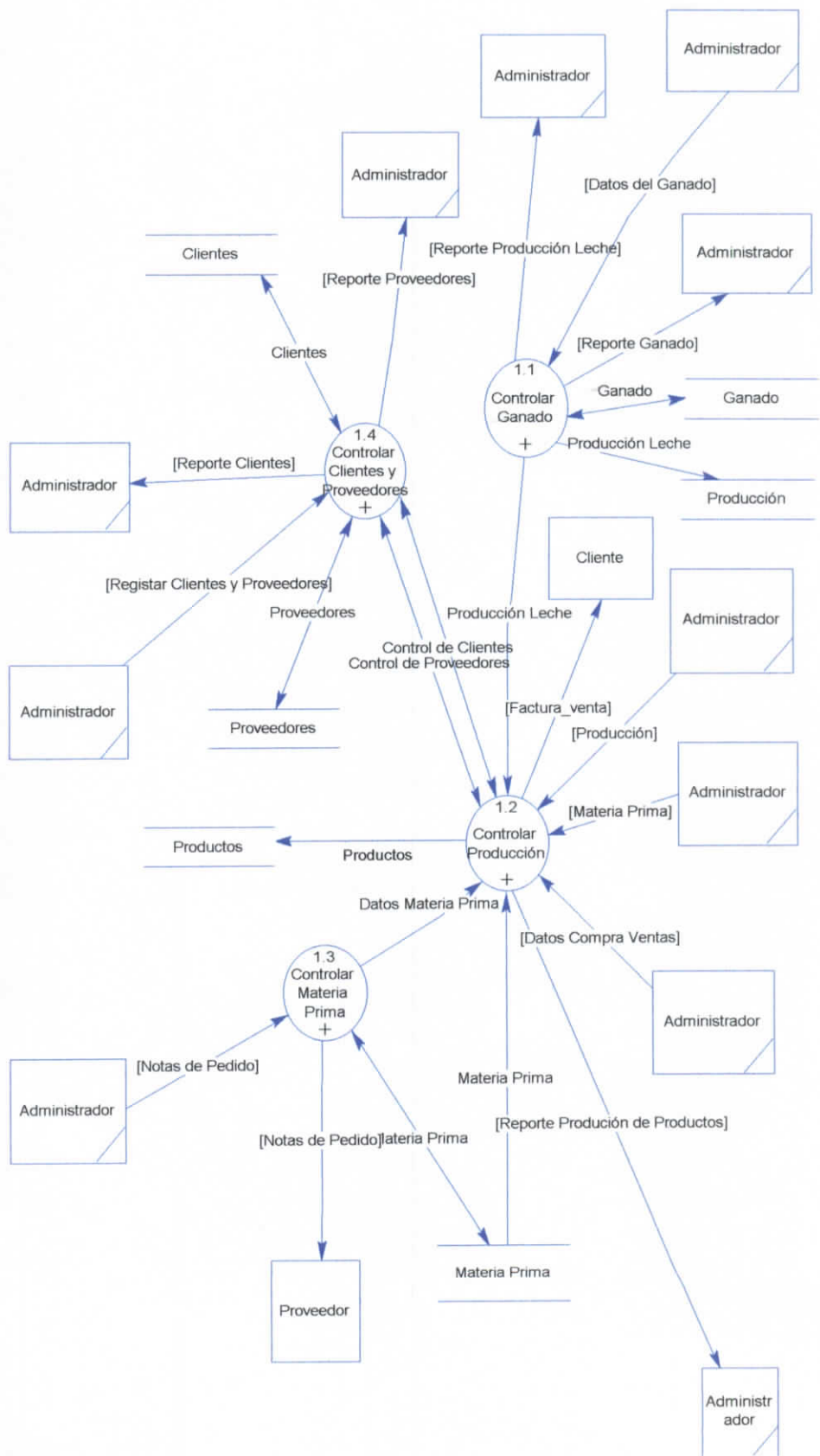


Fig. 6.2. DFD Nivel 1, Sistema Producción de Leche

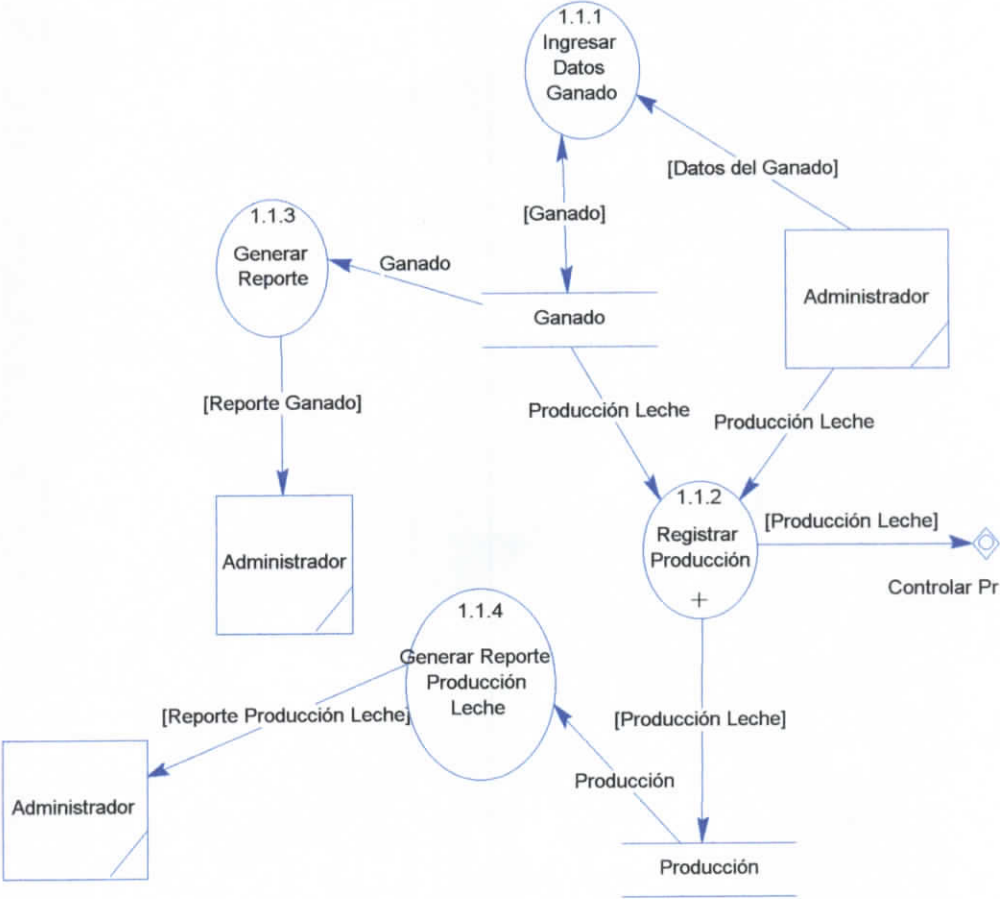


Fig. 6.3. DFD Nivel 2, Subproceso Producción de Leche

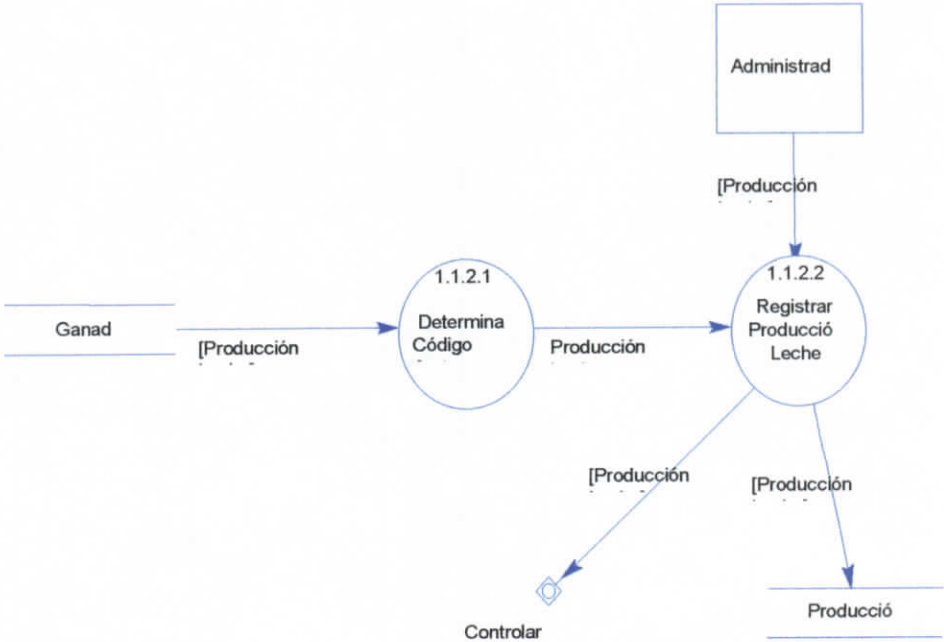


Fig. 6.4. DFD Nivel 3, Subproceso Registro de Producción de Leche

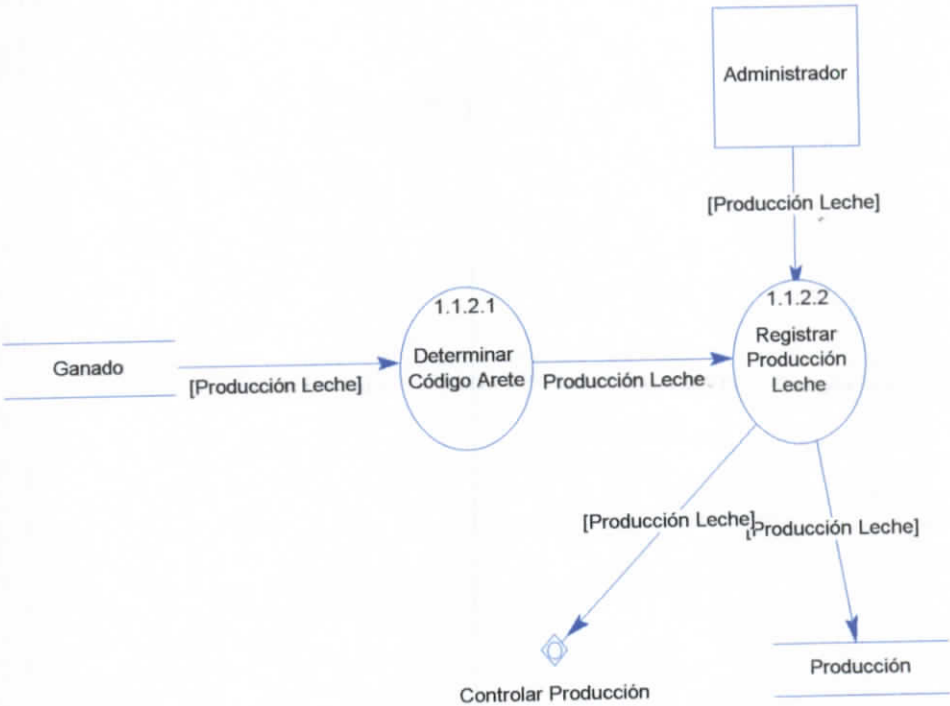


Fig. 6.4. DFD Nivel 3, Subproceso Registro de Producción de Leche



Fig. 6.5. DFD Nivel 2, Subproceso Producción

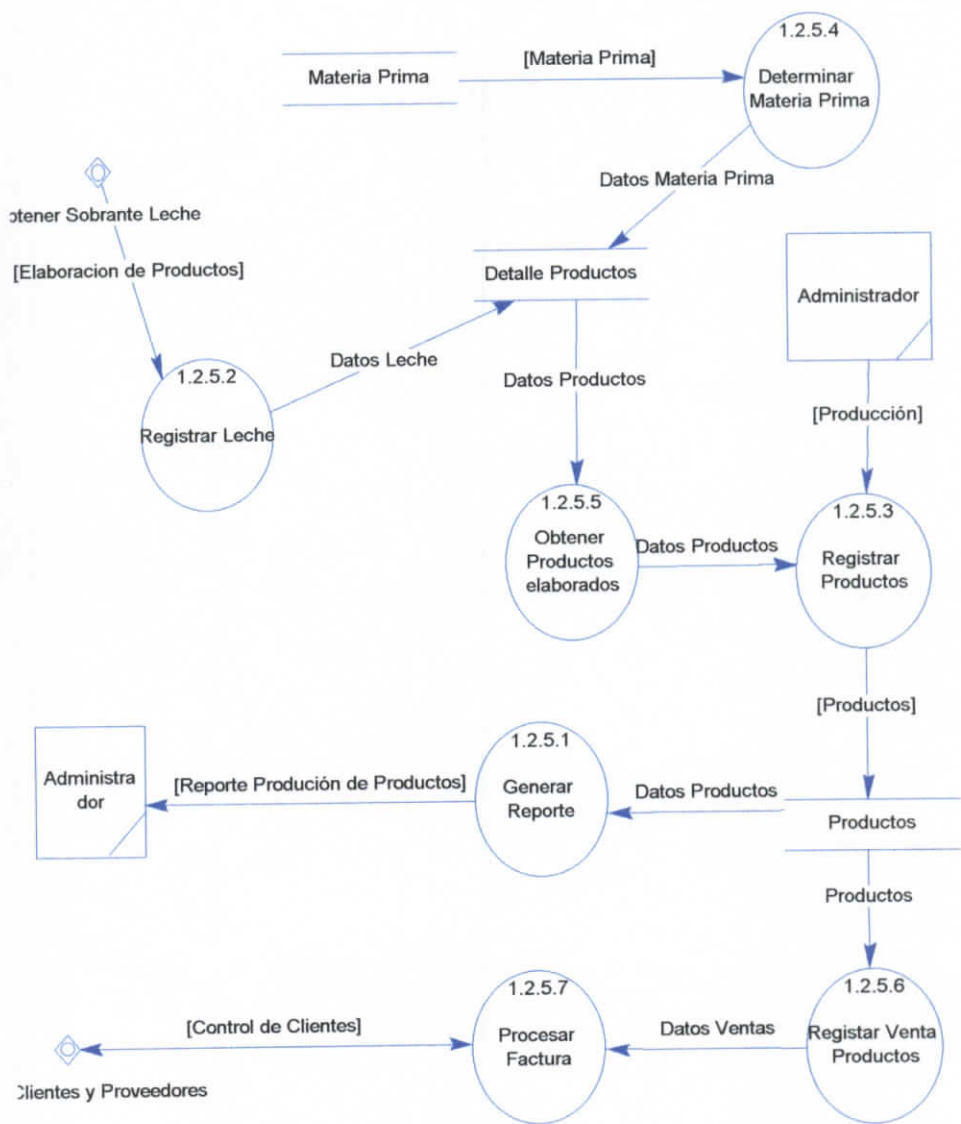


Fig. 6.6. DFD Nivel 3, Subproceso Registrar Productos

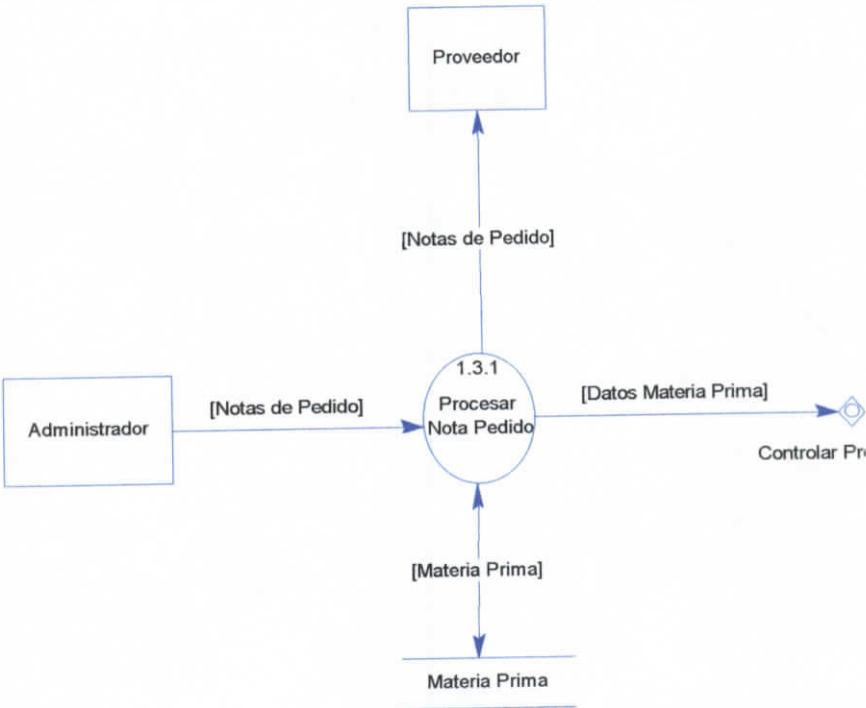


Fig. 6.7. DFD Nivel 2, Subproceso Pedidos

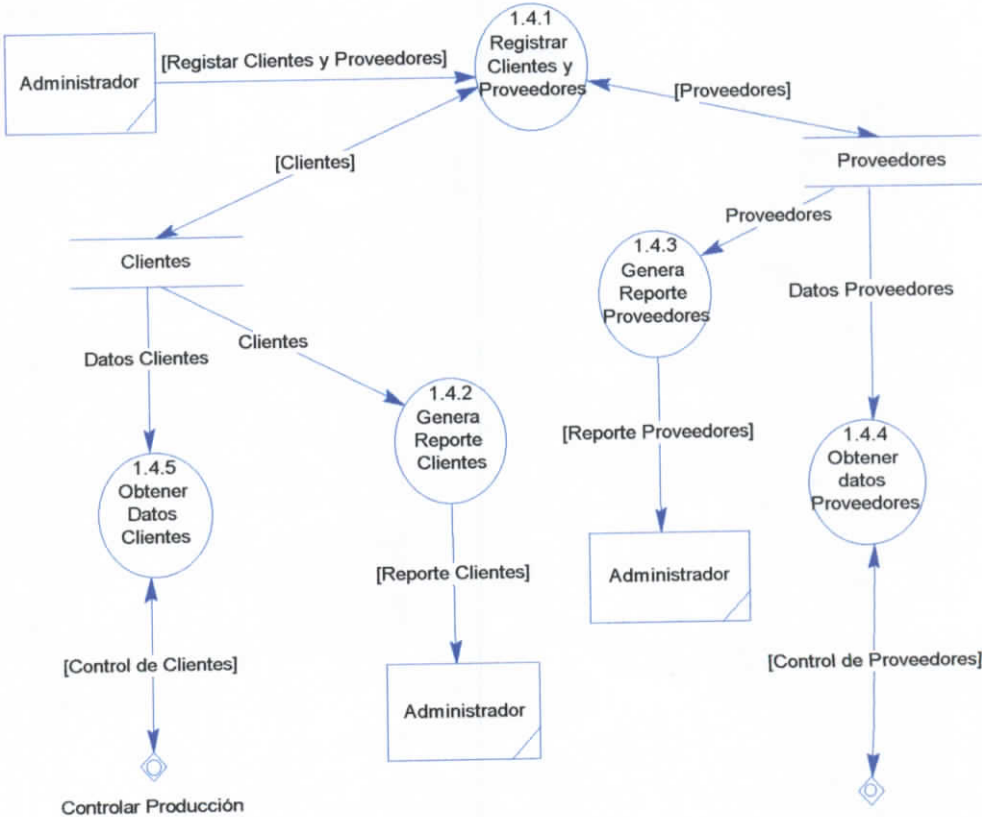


Fig. 6.8. DFD Nivel 2, Subproceso Clientes y Proveedores

6.1.2.3.- Jerarquía de Procesos

- Producción de Leche [1]
 - Controlar Ganado [1.1]
 - Ingresar Datos Ganado [1.1.1]
 - Registrar Producción [1.1.2]
 - Determinar Código Arete [1.1.2.1]
 - Registrar Producción Leche [1.1.2.2]
 - Generar Reporte [1.1.3]
 - Generar Reporte Producción Leche [1.1.4]
 - Controlar Producción [1.2]
 - Emitir Factura [1.2.1]
 - Procesar Pedidos Leche [1.2.2]
 - Obtener Sobrante Leche [1.2.3]
 - Registrar Materia Prima [1.2.4]
 - Elaborar Productos [1.2.5]
 - Generar Reporte [1.2.5.1]
 - Registrar Leche [1.2.5.2]
 - Registrar Productos [1.2.5.3]
 - Determinar Materia Prima [1.2.5.4]
 - Obtener Productos elaborados [1.2.5.5]
 - Registrar Venta Productos [1.2.5.6]
 - Procesar Factura [1.2.5.7]
 - Controlar Materia Prima [1.3]
 - Procesar Nota Pedido [1.3.1]
 - Controlar Clientes y Proveedores [1.4]
 - Registrar Clientes y Proveedores [1.4.1]
 - Genera Reporte Clientes [1.4.2]
 - Genera Reporte Proveedores [1.4.3]
 - Obtener datos Proveedores [1.4.4]
 - Obtener Datos Clientes [1.4.5]

6.1.2.4.- Referencia de procesos

Vía de Conexión	Conectado a	Fuente	Destino
Datos Compra Ventas	Administrador (Entidad Externa)		X
Datos del Ganado	Administrador (Entidad Externa)		X
Factura_venta	Cliente (Entidad Externa)	X	
Materia Prima	Administrador (Entidad Externa)		X
Notas de Pedido	Proveedor (Entidad Externa)	X	
Notas de Pedido	Administrador (Entidad Externa)		X
Producción	Administrador (Entidad Externa)		X
Registrar Clientes y Proveedores	Administrador (Entidad Externa)		X
Reporte Clientes	Administrador (Entidad Externa)	X	
Reporte Ganado	Administrador (Entidad Externa)	X	

Vía de Conexión	Conectado a	Fuente	Destino
Reporte Producción Leche	Administrador (Entidad Externa)	X	
Reporte Producción de Productos	Administrador (Entidad Externa)	X	
Reporte Proveedores	Administrador (Entidad Externa)	X	

6.1.2.5.- Entidades externas

Nombre	Código
Administrador	Administrador
Cliente	Cliente
Proveedor	Proveedor

6.1.2.6.- Almacenamientos

Nombre	Código
Clientes	Cliente
Detalle Productos	Detalle Producto
Ganado	Ganado
Materia Prima	Materia Prima
Producción	Producción
Productos	Productos
Proveedores	Proveedor

6.1.3.- Modelo Relacional

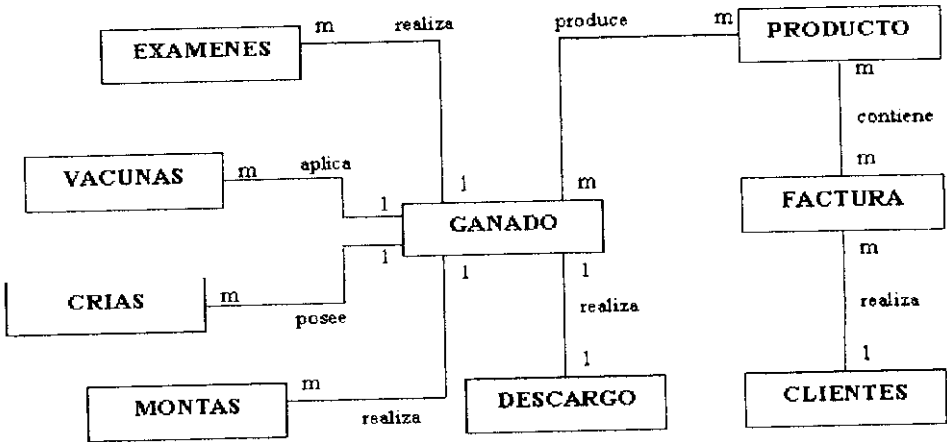


Fig. 6.9. Modelo Relacional

6.1.4.- Modelo Entidad Relación

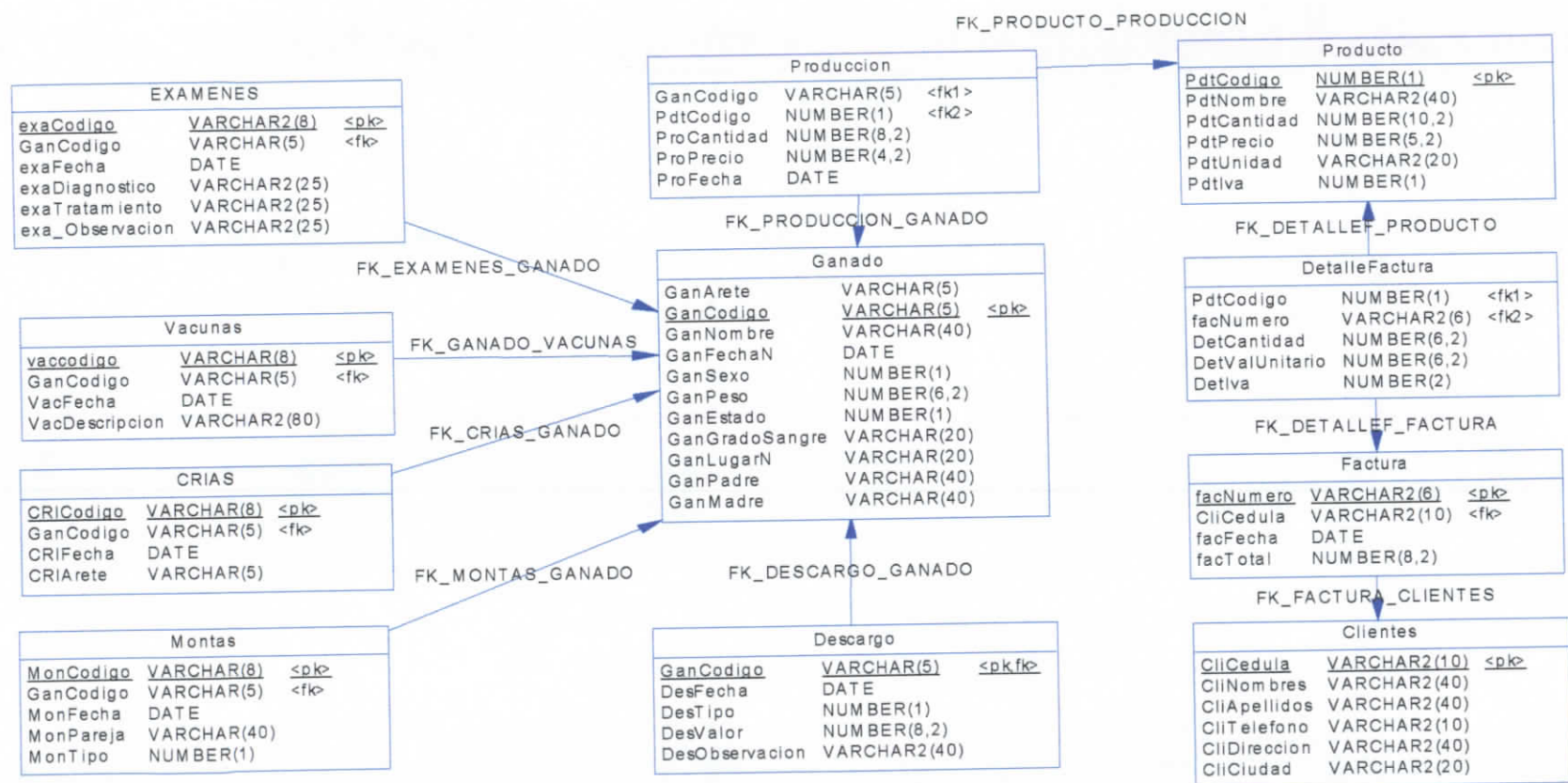


Fig. 6.10. Modelo Entidad-Relación

CAPITULO VII

DESARROLLO DEL SISTEMA

7.1.- Implementación

La implementación de la Base de Datos está dada por:

- ☒ Creación de las tablas representadas en el modelo entidad relación, descritas en SQL.
- ☒ Inserción de los datos requeridos.

7.1.1.- Creación de Tablas

```
CREATE TABLE CLIENTES
(
    CliCedula      VARCHAR2(10) NOT NULL,
    CliNombre      VARCHAR2(40) NULL,
    CliApellidos   VARCHAR2(40) NULL,
    CliTelefono    VARCHAR2(10) NULL,
    CliDireccion   VARCHAR2(40) NULL,
    CliCiudad      VARCHAR2(20) NULL,
);
ALTER TABLE CLIENTES
ADD ( PRIMARY KEY (CliCedula) );
CREATE TABLE CRIAS
(
    GanCodigo      VARCHAR2(5) NOT NULL,
    CriCodigo      VARCHAR2(8) NOT NULL,
    CriFecha       DATE NULL,
    CriArete       VARCHAR2(5) NULL,
);
ALTER TABLE CRIAS
ADD ( PRIMARY KEY (GanCodigo, CriCodigo) );
CREATE TABLE DESCARGO
(
    GanCodigo      VARCHAR2(5) NOT NULL,
    DesFecha       DATE NULL,
    DesTipo        NUMBER(1) NULL,
    DesValor       NUMBER(8,2) NULL,
    DesObservacion VARCHAR2(40) NULL,
);
ALTER TABLE DESCARGO
```

```

        ADD ( PRIMARY KEY (GanCodigo) ) ;
CREATE TABLE DETALLEFACTURA
(
    FacNumero      VARCHAR2(6) NOT NULL,
    CliCedula      VARCHAR2(10) NOT NULL,
    PdtCodigo      NUMBER(1) NOT NULL,
    DetCantidad    NUMBER(6,2) NULL,
    DetValUnitario NUMBER(6,2) NULL,
    DetIVA         NUMBER(2) NULL
);
ALTER TABLE DETALLEFACTURA
    ADD ( PRIMARY KEY (FacNumero, CliCedula, PdtCodigo) ) ;
CREATE TABLE EXAMENES
(
    GanCodigo      VARCHAR2(5) NOT NULL,
    exaCodigo      VARCHAR2(8) NOT NULL,
    exaFecha      DATE NULL,
    exaDiagnostico VARCHAR2(25) NULL,
    exaTratamiento VARCHAR2(25) NULL,
    exa_Observacion VARCHAR2(25) NULL
);
ALTER TABLE EXAMENES
    ADD ( PRIMARY KEY (GanCodigo, exaCodigo) ) ;
CREATE TABLE FACTURA
(
    CliCedula      VARCHAR2(10) NOT NULL,
    FacNumero      VARCHAR2(6) NOT NULL,
    FacFecha      DATE NULL,
    FacTotal      NUMBER(8,2) NULL
);
ALTER TABLE FACTURA
    ADD ( PRIMARY KEY (CliCedula, FacNumero) ) ;
CREATE TABLE GANADO
(
    GanCodigo      VARCHAR2(5) NOT NULL,
    GanArete      VARCHAR2(5) NULL,
    GanNombre      VARCHAR2(40) NULL,
    GanFechaN      DATE NULL,
    GanSexo        NUMBER(1) NULL,
    GanPeso        NUMBER(6,2) NULL,
    GanEstado      NUMBER(1) NULL,
    GanGradoSangre VARCHAR2(20) NULL,
    GanLugarN      VARCHAR2(20) NULL,
    GanPadre       VARCHAR2(40) NULL,
    GanMadre       VARCHAR2(40) NULL
);
ALTER TABLE GANADO
    ADD ( PRIMARY KEY (GanCodigo) ) ;
CREATE TABLE MONTAS
(
    GanCodigo      VARCHAR2(5) NOT NULL,
    MonCodigo      VARCHAR2(8) NOT NULL,
    MonFecha      DATE NULL,
    MonParcja     VARCHAR2(40) NULL,
    MonTipo        NUMBER(1) NULL
);

```



```
ALTER TABLE MONTAS
  ADD ( PRIMARY KEY (GanCodigo, MonCodigo) );
CREATE TABLE PRODUCCION
(
  PdtCodigo      NUMBER(1) NOT NULL,
  GanCodigo      VARCHAR2(5) NOT NULL,
  ProCantidad    NUMBER(8,2) NULL,
  ProPrecio      NUMBER(4,2) NULL,
  ProFecha       DATE NULL
);
ALTER TABLE PRODUCCION
  ADD ( PRIMARY KEY (PdtCodigo, GanCodigo) );
CREATE TABLE PRODUCTO
(
  PdtCodigo      NUMBER(1) NOT NULL,
  PdtNombre      VARCHAR2(40) NULL,
  PdtCantidad    NUMBER(10,2) NULL,
  PdtPrecio      NUMBER(5,2) NULL,
  PdtUnidad      VARCHAR2(20) NULL,
  PdtIVA         NUMBER(1) NULL
);
ALTER TABLE PRODUCTO
  ADD ( PRIMARY KEY (PdtCodigo) );
CREATE TABLE VACUNAS
(
  GanCodigo      VARCHAR2(8) NOT NULL,
  VacCodigo      VARCHAR2(5) NOT NULL,
  VacFecha       DATE NULL,
  VacDescripcion VARCHAR2(80) NULL
);
ALTER TABLE VACUNAS
  ADD ( PRIMARY KEY (GanCodigo, VacCodigo) );
ALTER TABLE CRIAS
  ADD ( FOREIGN KEY (GanCodigo)
        REFERENCES GANADO );
ALTER TABLE DESCARGO
  ADD ( FOREIGN KEY (GanCodigo)
        REFERENCES GANADO );
ALTER TABLE DETALLEFACTURA
  ADD ( FOREIGN KEY (CliCedula, FacNumero)
        REFERENCES FACTURA );
ALTER TABLE DETALLEFACTURA
  ADD ( FOREIGN KEY (PdtCodigo)
        REFERENCES PRODUCTO );
ALTER TABLE EXAMENES
  ADD ( FOREIGN KEY (GanCodigo)
        REFERENCES GANADO );
ALTER TABLE FACTURA
  ADD ( FOREIGN KEY (CliCedula)
        REFERENCES CLIENTES );
ALTER TABLE MONTAS
  ADD ( FOREIGN KEY (GanCodigo)
        REFERENCES GANADO );
ALTER TABLE PRODUCCION
  ADD ( FOREIGN KEY (GanCodigo)
        REFERENCES GANADO );
```

```

ALTER TABLE PRODUCCION
  ADD ( FOREIGN KEY (PdtCodigo)
        REFERENCES PRODUCTO ) ;
ALTER TABLE VACUNAS
  ADD ( FOREIGN KEY (GanCodigo)
        REFERENCES GANADO ) ;
create trigger tD_CLIENTES after DELETE on CLIENTES for each row
-- DELETE trigger on CLIENTES
declare numrows INTEGER;
begin
  /* CLIENTES realiza FACTURA ON PARENT DELETE RESTRICT */
  select count(*) into numrows
    from FACTURA
   where
     /* %JoinFKPK(FACTURA.:%Old." - ":" and") */
     FACTURA.CliCedula = :old.CliCedula;
  if (numrows > 0)
  then
    raise_application_error
      -20001,
      'Cannot DELETE CLIENTES because FACTURA exists.'
  );
  end if;
end;
/
create trigger tU_CLIENTES after UPDATE on CLIENTES for each row
-- UPDATE trigger on CLIENTES
declare numrows INTEGER;
begin
  /* CLIENTES realiza FACTURA ON PARENT UPDATE RESTRICT */
  if
    /* %JoinPKPK(:%Old.:%New." <> ":" or ") */
    :old.CliCedula <> :new.CliCedula
  then
    select count(*) into numrows
      from FACTURA
     where
      /* %JoinFKPK(FACTURA.:%Old." - ":" and") */
      FACTURA.CliCedula = :old.CliCedula;
    if (numrows > 0)
    then
      raise_application_error
        -20005,
        'Cannot UPDATE CLIENTES because FACTURA exists.'
    );
    end if;
  end if;
end;
/
create trigger tI_CRIAS after INSERT on CRIAS for each row
-- INSERT trigger on CRIAS
declare numrows INTEGER;
begin
  /* GANADO posee CRIAS ON CHILD INSERT RESTRICT */
  select count(*) into numrows
    from GANADO

```

```

where
  /* %JoinFKPK(:%New.GANADO." = "." and") */
  :new.GanCodigo = GANADO.GanCodigo;
if (
  /* %NotNullFK(:%New." is not null and") */
  numRows = 0
)
then
  raise_application_error
    -20002,
    'Cannot INSERT CRIAS because GANADO does not exist.'
);
end if;
end;
/

create trigger tU_CRIAS after UPDATE on CRIAS for each row
-- UPDATE trigger on CRIAS
declare numRows INTEGER;
begin
  /* GANADO posee CRIAS ON CHILD UPDATE RESTRICT */
  select count(*) into numRows
    from GANADO
    where
      /* %JoinFKPK(:%New.GANADO." = "." and") */
      :new.GanCodigo = GANADO.GanCodigo;
  if (
    /* %NotNullFK(:%New." is not null and") */
    numRows = 0
  )
  then
    raise_application_error
      -20007,
      'Cannot UPDATE CRIAS because GANADO does not exist.'
  );
  end if;
end;
/

create trigger tI_DESCARGO after INSERT on DESCARGO for each row
-- INSERT trigger on DESCARGO
declare numRows INTEGER;
begin
  /* GANADO realiza DESCARGO ON CHILD INSERT RESTRICT */
  select count(*) into numRows
    from GANADO
    where
      /* %JoinFKPK(:%New.GANADO." = "." and") */
      :new.GanCodigo = GANADO.GanCodigo;
  if (
    /* %NotNullFK(:%New." is not null and") */
    numRows = 0
  )
  then
    raise_application_error
      -20002,
      'Cannot INSERT DESCARGO because GANADO does not exist.'
  );

```

```

        end if;
    end;
/
create trigger tU_DESCARGO after UPDATE on DESCARGO for each row
-- UPDATE trigger on DESCARGO
declare numrows INTEGER;
begin
    /* GANADO realiza DESCARGO ON CHILD UPDATE RESTRICT */
    select count(*) into numrows
    from GANADO
    where
        /* %JoinFKPK(:%New.GANADO," = "," and") */
        :new.GanCodigo = GANADO.GanCodigo;
    if (
        /* %NotNullFK(:%New." is not null and") */
        numrows = 0
    )
    then
        raise_application_error(
            -20007,
            'Cannot UPDATE DESCARGO because GANADO does not exist.'
        );
    end if;
end;
/
create trigger tI_DETALLEFACTURA after INSERT on DETALLEFACTURA for each row
-- INSERT trigger on DETALLEFACTURA
declare numrows INTEGER;
begin
    /* FACTURA R/8 DETALLEFACTURA ON CHILD INSERT RESTRICT */
    select count(*) into numrows
    from FACTURA
    where
        /* %JoinFKPK(:%New.FACTURA," = "," and") */
        :new.CliCedula = FACTURA.CliCedula and
        :new.FacNumero = FACTURA.FacNumero;
    if (
        /* %NotNullFK(:%New." is not null and") */
        numrows = 0
    )
    then
        raise_application_error(
            -20002,
            'Cannot INSERT DETALLEFACTURA because FACTURA does not exist.'
        );
    end if;
    /* PRODUCTO R/7 DETALLEFACTURA ON CHILD INSERT RESTRICT */
    select count(*) into numrows
    from PRODUCTO
    where
        /* %JoinFKPK(:%New.PRODUCTO," = "," and") */
        :new.PdtCodigo = PRODUCTO.PdtCodigo;
    if (
        /* %NotNullFK(:%New." is not null and") */
        numrows = 0
    )

```

```

then
    raise_application_error
        -20002,
        'Cannot INSERT DETALLEFACTURA because PRODUCTO does not exist.'
);
end if;
end;
/
create trigger tU_DETALLEFACTURA after UPDATE on DETALLEFACTURA for each
row
-- UPDATE trigger on DETALLEFACTURA
declare numrows INTEGER;
begin
    /* FACTURA R/8 DETALLEFACTURA ON CHILD UPDATE RESTRICT */
    select count(*) into numrows
        from FACTURA
        where
            /* %JoinFKPK(:%New.FACTURA," = "," and") */
            :new.CliCedula = FACTURA.CliCedula and
            :new.FacNumero = FACTURA.FacNumero;
    if (
        /* %NotNullFK(:%New," is not null and") */
        numrows = 0
    )
    then
        raise_application_error
            -20007,
            'Cannot UPDATE DETALLEFACTURA because FACTURA does not exist.'
        );
    end if;

    /* PRODUCTO R/7 DETALLEFACTURA ON CHILD UPDATE RESTRICT */
    select count(*) into numrows
        from PRODUCTO
        where
            /* %JoinFKPK(:%New.PRODUCTO," = "," and") */
            :new.PdtCodigo = PRODUCTO.PdtCodigo;
    if (
        /* %NotNullFK(:%New," is not null and") */
        numrows = 0
    )
    then
        raise_application_error
            -20007,
            'Cannot UPDATE DETALLEFACTURA because PRODUCTO does not exist.'
        );
    end if;
end;
/
create trigger tI_EXAMENES after INSERT on EXAMENES for each row
-- INSERT trigger on EXAMENES
declare numrows INTEGER;
begin
    /* GANADO realiza EXAMENES ON CHILD INSERT RESTRICT */
    select count(*) into numrows
        from GANADO

```

```

where
/* %JoinFKPK(:%New.GANADO," = "," and") */
:new.GanCodigo = GANADO.GanCodigo;
if (
/* %NotNullFK(:%New," is not null and") */
numrows = 0
)
then
raise_application_error(
-20002,
'Cannot INSERT EXAMENES because GANADO does not exist.'
);
end if;
end;
/
create trigger tU_EXAMENES after UPDATE on EXAMENES for each row
-- UPDATE trigger on EXAMENES
declare numrows INTEGER;
begin
/* GANADO realiza EXAMENES ON CHILD UPDATE RESTRICT */
select count(*) into numrows
from GANADO
where
/* %JoinFKPK(:%New.GANADO," = "," and") */
:new.GanCodigo = GANADO.GanCodigo;
if (
/* %NotNullFK(:%New," is not null and") */
numrows = 0
)
then
raise_application_error(
-20007,
'Cannot UPDATE EXAMENES because GANADO does not exist.'
);
end if;
end;
/
create trigger tD_FACTURA after DELETE on FACTURA for each row
-- DELETE trigger on FACTURA
declare numrows INTEGER;
begin
/* FACTURA R/8 DETALLEFACTURA ON PARENT DELETE RESTRICT */
select count(*) into numrows
from DETALLEFACTURA
where
/* %JoinFKPK(DETALLEFACTURA.:%Old," = "," and") */
DETALLEFACTURA.CliCedula = :old.CliCedula and
DETALLEFACTURA.FacNumero = :old.FacNumero;
if (numrows > 0)
then
raise_application_error(
-20001,
'Cannot DELETE FACTURA because DETALLEFACTURA exists.'
);
end if;
end;

```



```

/
create trigger tI_FACTURA after INSERT on FACTURA for each row
-- INSERT trigger on FACTURA
declare numrows INTEGER;
begin
  /* CLIENTES realiza FACTURA ON CHILD INSERT RESTRICT */
  select count(*) into numrows
  from CLIENTES
  where
    /* %JoinFKPK(:%New.CLIENTES," = "," and") */
    :new.CliCedula = CLIENTES.CliCedula;
  if (
    /* %NotNullFK(:%New." is not null and") */
    numrows = 0
  )
  then
    raise_application_error(
      -20002,
      'Cannot INSERT FACTURA because CLIENTES does not exist.'
    );
  end if;
end;
/

create trigger tU_FACTURA after UPDATE on FACTURA for each row
-- UPDATE trigger on FACTURA
declare numrows INTEGER;
begin
  /* FACTURA R/8 DETALLEFACTURA ON PARENT UPDATE RESTRICT */
  if
    /* %JoinPKPK(:%Old,:%New," <> "," or ") */
    :old.CliCedula <> :new.CliCedula or
    :old.FacNumero <> :new.FacNumero
  then
    select count(*) into numrows
    from DETALLEFACTURA
    where
      /* %JoinFKPK(DETALLEFACTURA.:%Old," = "," and") */
      DETALLEFACTURA.CliCedula = :old.CliCedula and
      DETALLEFACTURA.FacNumero = :old.FacNumero;
    if (numrows > 0)
    then
      raise_application_error(
        -20005,
        'Cannot UPDATE FACTURA because DETALLEFACTURA exists.'
      );
    end if;
  end if;
  /* CLIENTES realiza FACTURA ON CHILD UPDATE RESTRICT */
  select count(*) into numrows
  from CLIENTES
  where
    /* %JoinFKPK(:%New.CLIENTES," = "," and") */
    :new.CliCedula = CLIENTES.CliCedula;
  if (
    /* %NotNullFK(:%New." is not null and") */
    numrows = 0
  )

```

```

)
then
  raise_application_error(
    -20007,
    'Cannot UPDATE FACTURA because CLIENTES does not exist.'
  );
end if;
end;
/
create trigger ID_GANADO after DELETE on GANADO for each row
-- DELETE trigger on GANADO
declare numrows INTEGER;
begin
  /* GANADO realiza DESCARGO ON PARENT DELETE RESTRICT */
  select count(*) into numrows
  from DESCARGO
  where
    /* %JoinFKPK(DESCARGO, :old.GanCodigo = :old.GanCodigo) */
    DESCARGO.GanCodigo = :old.GanCodigo;
  if (numrows = 0)
  then
    raise_application_error(
      -20001,
      'Cannot DELETE GANADO because DESCARGO exists.'
    );
  end if;
  /* GANADO R/11 PRODUCCION ON PARENT DELETE RESTRICT */
  select count(*) into numrows
  from PRODUCCION
  where
    /* %JoinFKPK(PRODUCCION, :old.GanCodigo = :old.GanCodigo) */
    PRODUCCION.GanCodigo = :old.GanCodigo;
  if (numrows > 0)
  then
    raise_application_error(
      -20001,
      'Cannot DELETE GANADO because PRODUCCION exists.'
    );
  end if;
  /* GANADO realiza MONTAS ON PARENT DELETE RESTRICT */
  select count(*) into numrows
  from MONTAS
  where
    /* %JoinFKPK(MONTAS, :old.GanCodigo = :old.GanCodigo) */
    MONTAS.GanCodigo = :old.GanCodigo;
  if (numrows > 0)
  then
    raise_application_error(
      -20001,
      'Cannot DELETE GANADO because MONTAS exists.'
    );
  end if;
  /* GANADO posee CRIAS ON PARENT DELETE RESTRICT */
  select count(*) into numrows
  from CRIAS
  where

```

```

/* %JoinFKPK(CRIAS.:%Old." = "." and") */
CRIAS.GanCodigo = :old.GanCodigo;
if (numrows > 0)
then
raise_application_error(
-20001,
'Cannot DELETE GANADO because CRIAS exists.'
);
end if;
/* GANADO aplica VACUNAS ON PARENT DELETE RESTRICT */
select count(*) into numrows
from VACUNAS
where
/* %JoinFKPK(VACUNAS.:%Old." = "." and") */
VACUNAS.GanCodigo = :old.GanCodigo;
if (numrows > 0)
then
raise_application_error(
-20001,
'Cannot DELETE GANADO because VACUNAS exists.'
);
end if;
/* GANADO realiza EXAMENES ON PARENT DELETE RESTRICT */
select count(*) into numrows
from EXAMENES
where
/* %JoinFKPK(EXAMENES.:%Old." = "." and") */
EXAMENES.GanCodigo = :old.GanCodigo;
if (numrows > 0)
then
raise_application_error(
-20001,
'Cannot DELETE GANADO because EXAMENES exists.'
);
end if;
end;
/
create trigger tU_GANADO after UPDATE on GANADO for each row
-- UPDATE trigger on GANADO
declare numrows INTEGER;
begin
/* GANADO realiza DESCARGO ON PARENT UPDATE RESTRICT */
if
/* %JoinPKPK(:%Old.:%New." <> "." or ") */
:old.GanCodigo <> :new.GanCodigo
then
select count(*) into numrows
from DESCARGO
where
/* %JoinFKPK(DESCARGO.:%Old." = "." and") */
DESCARGO.GanCodigo = :old.GanCodigo;
if (numrows > 0)
then
raise_application_error(
-20005,
'Cannot UPDATE GANADO because DESCARGO exists.'
);

```

```
);
end if;
end if;
/* GANADO R/I/ PRODUCCION ON PARENT UPDATE RESTRICT */
if
/* %JoinPKPK(:%Old.:%New." <> "," or ") */
:old.GanCodigo <> :new.GanCodigo
then
select count(*) into numrows
from PRODUCCION
where
/* %JoinFKPK(PRODUCCION.:%Old." = "," and") */
PRODUCCION.GanCodigo = :old.GanCodigo;
if (numrows > 0)
then
raise_application_error(
-20005,
'Cannot UPDATE GANADO because PRODUCCION exists.'
);
end if;
end if;
/* GANADO realiza MONTAS ON PARENT UPDATE RESTRICT */
if
/* %JoinPKPK(:%Old.:%New." <> "," or ") */
:old.GanCodigo <> :new.GanCodigo
then
select count(*) into numrows
from MONTAS
where
/* %JoinFKPK(MONTAS.:%Old." = "," and") */
MONTAS.GanCodigo = :old.GanCodigo;
if (numrows > 0)
then
raise_application_error(
-20005,
'Cannot UPDATE GANADO because MONTAS exists.'
);
end if;
end if;
/* GANADO posce CRIAS ON PARENT UPDATE RESTRICT */
if
/* %JoinPKPK(:%Old.:%New." <> "," or ") */
:old.GanCodigo <> :new.GanCodigo
then
select count(*) into numrows
from CRIAS
where
/* %JoinFKPK(CRIAS.:%Old." = "," and") */
CRIAS.GanCodigo = :old.GanCodigo;
if (numrows > 0)
then
raise_application_error(
-20005,
'Cannot UPDATE GANADO because CRIAS exists.'
);
end if;
```

```

end if;
/* GANADO aplica VACUNAS ON PARENT UPDATE RESTRICT */
if
  /* %JoinPKPK(:%Old,:%New," <> "," or ") */
  :old.GanCodigo <> :new.GanCodigo
then
  select count(*) into numrows
  from VACUNAS
  where
    /* %JoinFKPK(VACUNAS,:%Old," = "," and") */
    VACUNAS.GanCodigo = :old.GanCodigo;
  if (numrows > 0)
  then
    raise_application_error(
      -20005,
      'Cannot UPDATE GANADO because VACUNAS exists.'
    );
  end if;
end if;
/* GANADO realiza EXAMENES ON PARENT UPDATE RESTRICT */
if
  /* %JoinPKPK(:%Old,:%New," <> "," or ") */
  :old.GanCodigo <> :new.GanCodigo
then
  select count(*) into numrows
  from EXAMENES
  where
    /* %JoinFKPK(EXAMENES,:%Old," = "," and") */
    EXAMENES.GanCodigo = :old.GanCodigo;
  if (numrows > 0)
  then
    raise_application_error(
      -20005,
      'Cannot UPDATE GANADO because EXAMENES exists.'
    );
  end if;
end if;
end;
/
create trigger tl_MONTAS after INSERT on MONTAS for each row
-- INSERT trigger on MONTAS
declare numrows INTEGER;
begin
  /* GANADO realiza MONTAS ON CHILD INSERT RESTRICT */
  select count(*) into numrows
  from GANADO
  where
    /* %JoinFKPK(:%New.GANADO," = "," and") */
    :new.GanCodigo = GANADO.GanCodigo;
  if (
    /* %NotnullFK(:%New," is not null and") */
    numrows = 0
  )
  then
    raise_application_error(
      -20002,

```

```

        'Cannot INSERT MONTAS because GANADO does not exist.'
    );
end if;
end;
/
create trigger tU_MONTAS after UPDATE on MONTAS for each row
-- UPDATE trigger on MONTAS
declare numrows INTEGER;
begin
    /* GANADO realiza MONTAS ON CHILD UPDATE RESTRICT */
    select count(*) into numrows
    from GANADO
    where
        /* %JoinFKPK(:%New.GANADO." = "," and") */
        :new.GanCodigo = GANADO.GanCodigo;
    if (
        /* %NotNullFK(:%New." is not null and") */
        numrows = 0
    )
    then
        raise_application_error
        -20007,
        'Cannot UPDATE MONTAS because GANADO does not exist.'
    );
end if;
end;
/
create trigger tI_PRODUCCION after INSERT on PRODUCCION for each row
-- INSERT trigger on PRODUCCION
declare numrows INTEGER;
begin
    /* GANADO R/11 PRODUCCION ON CHILD INSERT RESTRICT */
    select count(*) into numrows
    from GANADO
    where
        /* %JoinFKPK(:%New.GANADO." = "," and") */
        :new.GanCodigo = GANADO.GanCodigo;
    if (
        /* %NotNullFK(:%New." is not null and") */
        numrows = 0
    )
    then
        raise_application_error
        -20002,
        'Cannot INSERT PRODUCCION because GANADO does not exist.'
    );
end if;
/* PRODUCTO R/10 PRODUCCION ON CHILD INSERT RESTRICT */
select count(*) into numrows
from PRODUCTO
where
    /* %JoinFKPK(:%New.PRODUCTO." = "," and") */
    :new.PdtCodigo = PRODUCTO.PdtCodigo;
if (
    /* %NotNullFK(:%New." is not null and") */
    numrows = 0

```



```

    )
    then
        raise_application_error(
            -20002,
            'Cannot INSERT PRODUCCION because PRODUCTO does not exist.'
        );
    end if;
end;
/
create trigger tU_PRODUCCION after UPDATE on PRODUCCION for each row
-- UPDATE trigger on PRODUCCION
declare numrows INTEGER;
begin
    /* GANADO R/11 PRODUCCION ON CHILD UPDATE RESTRICT */
    select count(*) into numrows
    from GANADO
    where
        /* %JoinFKPK(:%New.GANADO," = "," and") */
        :new.GanCodigo = GANADO.GanCodigo;
    if (
        /* %NotNullFK(:%New." is not null and") */
        numrows = 0
    )
    then
        raise_application_error(
            -20007,
            'Cannot UPDATE PRODUCCION because GANADO does not exist.'
        );
    end if;
    /* PRODUCTO R/10 PRODUCCION ON CHILD UPDATE RESTRICT */
    select count(*) into numrows
    from PRODUCTO
    where
        /* %JoinFKPK(:%New.PRODUCTO," = "," and") */
        :new.PdtCodigo = PRODUCTO.PdtCodigo;
    if (
        /* %NotNullFK(:%New." is not null and") */
        numrows = 0
    )
    then
        raise_application_error(
            -20007,
            'Cannot UPDATE PRODUCCION because PRODUCTO does not exist.'
        );
    end if;
end;
/
create trigger tD_PRODUCTO after DELETE on PRODUCTO for each row
-- DELETE trigger on PRODUCTO
declare numrows INTEGER;
begin
    /* PRODUCTO R/10 PRODUCCION ON PARENT DELETE RESTRICT */
    select count(*) into numrows
    from PRODUCCION
    where
        /* %JoinFKPK(PRODUCCION,:%Old," = "," and") */

```

```

        PRODUCCION.PdtCodigo = :old.PdtCodigo;
    if (numrows > 0)
    then
        raise_application_error(
            -20001,
            'Cannot DELETE PRODUCTO because PRODUCCION exists.'
        );
    end if;
    /* PRODUCTO R/7 DETALLEFACTURA ON PARENT DELETE RESTRICT */
    select count(*) into numrows
    from DETALLEFACTURA
    where
        /* %JoinFKPK(DETALLEFACTURA,:%Old," = "," and") */
        DETALLEFACTURA.PdtCodigo = :old.PdtCodigo;
    if (numrows > 0)
    then
        raise_application_error(
            -20001,
            'Cannot DELETE PRODUCTO because DETALLEFACTURA exists.'
        );
    end if;
end;
/
create trigger tU_PRODUCTO after UPDATE on PRODUCTO for each row
-- UPDATE trigger on PRODUCTO
declare numrows INTEGER;
begin
    /* PRODUCTO R/10 PRODUCCION ON PARENT UPDATE RESTRICT */
    if
        /* %JoinPKPK(:%Old,:%New," <> "," or ") */
        :old.PdtCodigo <> :new.PdtCodigo
    then
        select count(*) into numrows
        from PRODUCCION
        where
            /* %JoinFKPK(PRODUCCION,:%Old," = "," and") */
            PRODUCCION.PdtCodigo = :old.PdtCodigo;
        if (numrows > 0)
        then
            raise_application_error(
                -20005,
                'Cannot UPDATE PRODUCTO because PRODUCCION exists.'
            );
        end if;
    end if;
    /* PRODUCTO R/7 DETALLEFACTURA ON PARENT UPDATE RESTRICT */
    if
        /* %JoinPKPK(:%Old,:%New," <> "," or ") */
        :old.PdtCodigo <> :new.PdtCodigo
    then
        select count(*) into numrows
        from DETALLEFACTURA
        where
            /* %JoinFKPK(DETALLEFACTURA,:%Old," = "," and") */
            DETALLEFACTURA.PdtCodigo = :old.PdtCodigo;
        if (numrows > 0)

```

```

then
    raise_application_error(
        -20005,
        'Cannot UPDATE PRODUCTO because DETALLEFACTURA exists.'
    );
end if;
end if;
end;
/
create trigger tI_VACUNAS after INSERT on VACUNAS for each row
-- INSERT trigger on VACUNAS
declare numrows INTEGER;
begin
    /* GANADO aplica VACUNAS ON CHILD INSERT RESTRICT */
    select count(*) into numrows
    from GANADO
    where
        /* %JoinFKPK(:%New.GANADO." = ":" and") */
        :new.GanCodigo = GANADO.GanCodigo;
    if (
        /* %NotNullFK(:%New." is not null and") */
        numrows = 0
    )
    then
        raise_application_error(
            -20002,
            'Cannot INSERT VACUNAS because GANADO does not exist.'
        );
    end if;
end;
/
create trigger tU_VACUNAS after UPDATE on VACUNAS for each row
-- UPDATE trigger on VACUNAS
declare numrows INTEGER;
begin
    /* GANADO aplica VACUNAS ON CHILD UPDATE RESTRICT */
    select count(*) into numrows
    from GANADO
    where
        /* %JoinFKPK(:%New.GANADO." = ":" and") */
        :new.GanCodigo = GANADO.GanCodigo;
    if (
        /* %NotNullFK(:%New." is not null and") */
        numrows = 0
    )
    then
        raise_application_error(
            -20007,
            'Cannot UPDATE VACUNAS because GANADO does not exist.'
        );
    end if;
end;
/

```

7.1.2.- Insertar datos

```
INSERT INTO Clientes VALUES  
( '1801552694', 'Carlos', 'Perez Ruiz', '2873167', 'Calle Rumiñahui, Ciudad Nueva', 'Pillaro')  
INSERT INTO Clientes VALUES  
( '1801624692', 'Ana María', 'Rios Lara', '2873142', 'Calle Bolivar y Quevedo', 'Pillaro')  
INSERT INTO Clientes VALUES  
( '1801214459', 'Luis Ivan', 'Arias Torres', '2863154', 'Calle Olmedo y Primera  
Imprenta', 'Salcedo')
```

7.2.- Pruebas

Las pruebas se realizaron mientras se realizaba la codificación, y durante cada una de las etapas de desarrollo.

Una vez concluida la codificación, se realizaron pruebas con datos reales de la hacienda, para comprobar la eficiencia del funcionamiento, velocidad de proceso y entrega de resultados confiables y precisos, los mismos que fueron comparados con los obtenidos de los procesos manuales.

CONCLUSIONES

- ☒ La utilización del software para el desarrollo del proyecto, satisface las necesidades para el manejo de la información, Oracle 8i como Base de Datos y Developer 6i para el desarrollo de las interfaces, permitiendo al usuario interactuar de manera fácil y segura, con un costo bastante accesible.
- ☒ El uso de las herramientas RAD de Oracle, al trabajar en forma independiente, permite que, mientras se va desarrollando el proyecto, se vayan realizando las pruebas sobre los componentes sobre los que se está trabajando.
- ☒ El medio ambiente de trabajo RAD de Oracle, se facilita con el conocimiento previo de un ambiente de trabajo visual, como Visual Basic, ya que los conceptos como objeto, propiedad, método y evento, son comunes.
- ☒ La utilización de herramientas de desarrollo RAD, facilitan el desarrollo de aplicaciones, principalmente las medianas y grandes, por la facilidad de integración.
- ☒ Los resultados obtenidos, durante el desarrollo y durante las pruebas del sistema, satisfacen las expectativas, tanto del personal encargado del control de la información, como de los propietarios de la hacienda El Nilo y el Rosario.

RECOMENDACIONES

- ☒ Para ampliaciones o actualizaciones futuras del sistema, se recomienda mantener el software utilizado, para garantizar una mejor integración y confiabilidad.
- ☒ Se requiere una actualización permanente en las Herramientas para el Desarrollo de Sistemas.
- ☒ La Universidad Católica, debería implementar en su pénsum regular y cursos permanentes de Herramientas de Desarrollo actuales.
- ☒ Para los propietarios de la Hacienda el Nilo y el Rosario, se recomienda automatizar otras actividades que se llevan a cabo en la Hacienda.
- ☒ Para el desarrollo de nuevas aplicaciones, es recomendable utilizar y aprovechar al máximo las herramientas de desarrollo RAD de Oracle.
- ☒ Un limitante, para el trabajo con la Base de Datos y Herramientas ORACLE, esta dada principalmente por los requerimientos de hardware necesarios (espacio en disco y memoria).

BIBLIOGRAFÍA

- CRONIN, Daniel, "Mastering Oracle", Hayden Books
- <http://users.servicios.retecal.es/sapivi>
- JARAMILLO C. Fabián, "Aplicaciones Pedagógicas del Computador".
- JARAMILLO C. Fabián, "Metodología de la Educación".
- KOCH, George, "Manual de Referencia", Osborne7MacGraw-Hill
- KOCHHAR Neena, KRAMER Debby, "Introduction to Oracle: SQL and PL/SQL Usig Procedure Builder".
- ORACLE Corporation, "Developer 2000 Getting Started", 1997
- ORACLE Corporation, "Developer 2000 Guidelines for Building Applications", 1997
- ORACLE Corporation, "Graphics 2.5 Developer's Guide", 1995
- ORACLE Corporation, "Oracle Forms Developer and Oracle Reports Developer", 2000
- ORACLE Corporation, "Oracle Forms Developer's Guide", 1996
<http://mates.ms.mff.cuni.cz/oracle/doc/forms45/dg/>
- ORACLE Corporation, "Oracle Procedure Builder Reference", 1999
- ORACLE Corporation, "Oracle Reports Developer", 2000
- ORACLE Corporation, "Oracle Reports Reference", 1999
- ORACLE Corporation, "Procedure Builder Developer's Guide", 1995

- Oracle Corporation, Revista bimestral ORACLE
- Oracle Corporation, Revista PROFIT
- QUEZADA S. Sosa, "Clasificación y Diagnóstico del Aprendizaje por Computadora".
- SCHWINN Ulrike, VENKATACHALAM Vijayanandan, "Oracle 8: Database Administration".
- www.bd.cesma.usb.ve/ci3391/manual/blocks.html
- www.bd.cesma.usb.ve/ci3391/manual/functions.html
- www.bd.cesma.usb.ve/ci3391/manual/procedures.html
- www.bd.cesma.usb.ve/ci3391/manual/scripts.html
- www.ciberia.ya.com/alexcasan/intplsql.htm
- www.ei.uvigo.es/~evali/abd/cursoPLSQL.pdf
- www.galileo.edu/wp/display/111/

GLOSARIO

Acción Predefinida	Acción disponible para el usuario, propia de cualquier herramienta de Project Builder
Acciones	Son cadenas de comandos que se aplican a un tipo determinado
Bloque	Representación de una entidad en una forma
Caja de Diálogo	Ventana usada para que el usuario ingrese la información necesaria para completar una acción específica
Campo	Elemento de la interfase que visualiza información y/o acepta ingreso de datos del usuario
DBA	DataBase Administrator, Administrador de la Base de Datos
DBMS	DataBase Management System, Sistemas Manejadores de Bases de Datos
DCL	Data Control Language, Lenguaje de Control de Datos
DDL	Data Definition Language, Lenguaje de Definición de Datos
DML	Data Manipulation Language, Lenguaje de Manipulación de Datos
Entidad	Una cosa que tiene significado para el usuario, una entidad puede incluir varios bloques
FK	Foreign Key, Llave Foránea
Grupo	Colección de items relacionados, disponibles a través de subopciones, fuera del lanzador
GUI	Graphical User Interface, Interfase Gráfica de Usuario
Lanzador	Barra de Herramientas secundaria, que permite acceder a otras herramientas de desarrollo
Lienzo	Superficie donde se dibujan los items e indicadores de la interfase
Macros	Son variables que se pueden usar para modificar acciones
Máscara de Formato	Posición que define la apariencia del valor de un campo
Master-Detalle	Relación entre dos entidades, que indica una jerarquía de la información
Navegador del Proyecto	Interfase de estilo navegador o explorador, que permite ver los módulos de una aplicación
ODBC	Open DataBase Connecivity, Conectividad Abierta a las Bases de Datos
OOP	Object Oriented Programming, Programación Orientada a Objetos
ORACLE_HOME	Variable del Medio Ambiente, que indica la raíz del servidor Oracle
Paleta de Propiedades	Permite examinar y modificar las propiedades de los items seleccionados
PK	Primary Key, Llave Primaria
PL/SQL	Procedural Language/Structured Query Language.

	Lenguaje Procedural/Lenguaje de Consulta Estructurado
Proyecto	Grupo de apuntadores a módulos y archivos parte de una aplicación
RAD	Rapid Application Development, Desarrollo Rápido de Aplicaciones
Registro de Usuario	Contiene los nuevos tipos, tipos existentes predefinidos y acciones o macros definidas por el usuario
Registro Global	Contiene los tipos predefinidos de Project Builder
SQL	Structured Query Language, Lenguaje de Consulta Estructurado
Tipo	Un tipo es la base de un ítem, y controla las acciones que están disponibles en Project Builder
UID	Unique Identifier, Identificador Único

ANEXOS

Anexo 1.- Manual de Usuario

Anexo 2.- Manual del Administrador

**Sistema: Control de Producción de Leche en la
Hacienda “El Nilo y El Rosario”
MANUAL DE USUARIO**

Contenido

Contenido 1

Índice de Gráficos 1

1.- Ingreso al Sistema 3

2.- Opciones Principales del Sistema 3

3.- Actualización de Datos 5

4.- Control de Producción 7

5.- Facturación 9

6.- Control 10

7.- Reportes 14

Índice de Gráficos

Fig. A1 Ingreso al Sistema 3

Fig. A2 Ventana Principal del Sistema 3

Fig. A3 Opciones de Archivo 4

Fig. A4 Opción de Producción 4

Fig. A5 Opción de Facturación 4

Fig. A6 Opción de Control 4

Fig. A7 Opción de Reportes 4

Fig. A8 Botones de Navegación 5

Fig. A9. Opción Archivo | Ganado 5

Fig. A10. Opción Archivo | Productos 6

Fig. A11. Opción Archivo | Clientes 7

Fig. A12. Opción Archivo | Parámetros 7

Fig. A13. Opción Producción | Ingresar Producción 8

Fig. A14. Agregar Producción 8

Fig. A15. Opción Facturación | Factura 9

Fig. A16. Ejemplo de factura 10

Fig. A17. Opción Control | Vacunas 11

Fig. A18. Opción Control | Exámenes 11

Fig. A19. Opción Control | Montas..... 12

Fig. A20. Opción Control | Crías 12

Fig. A21. Ingreso de Crías..... 13

Fig. A22. Opción Control | Descargo..... 13

Fig. A23. Listado de Facturas General..... 14

Fig. A24. Parámetro: Fecha de Facturas 14

Fig. A25. Parámetro: Número de Factura 15

Fig. A26. Listado de Factura por Número 15

Fig. A27. Listado Total de Ganado..... 15

Fig. A28. Listado de Producción por Ejemplar 16

Fig. A29. Listado de Vacunas del Ganado..... 16

Fig. A30. Listado de Montas del Ganado..... 16

Fig. A31. Listado de Partos del Ganado..... 17

Fig. A32. Listado de Descargos de Ganado 17

1.- Ingreso al Sistema

Para ingresar al sistema, hacer clic en el acceso directo de PRODUCCIÓN. Luego de lo cual, aparece una ventana de Conexión (Fig. A1):

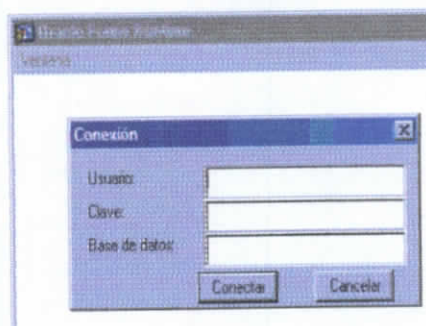


Fig. A1 Ingreso al Sistema

Ingresar la siguiente información, suministrada por el DBA:

- Usuario: nombre de usuario (Ejm. Administrador),
- Clave: clave del usuario (Ejm. Administrador),
- Base de Datos: nombre de la base de datos o cadena de conexión (Ejm. Sistema.World)

Una vez ingresados los datos correctos, se presenta la ventana principal del sistema (Fig. A2):

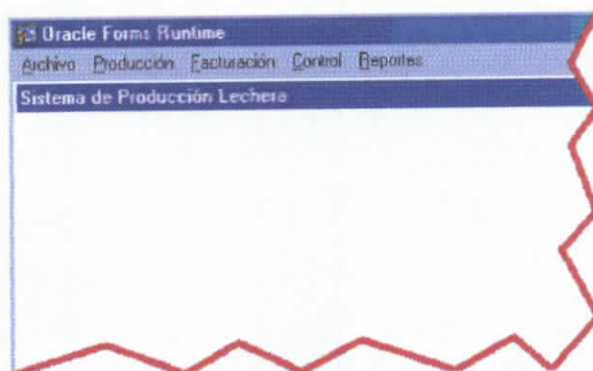


Fig. A2 Ventana Principal del Sistema

2.- Opciones Principales del Sistema

Opción **Archivo**: Permite ingresar, modificar, eliminar datos de: Ganado, Productos, Clientes y Parámetros, y adicionalmente Salir para abandonar el sistema (Fig. A3):

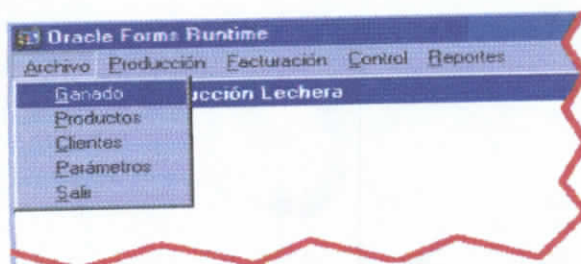


Fig. A3 Opciones de Archivo

Opción **Producción**: Permite ingresar datos de producción (Fig. A4):

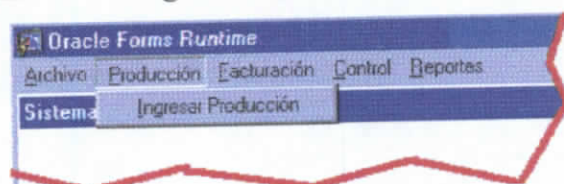


Fig. A4 Opción de Producción

Opción **Facturación**: Permite realizar una factura (Fig. A5):

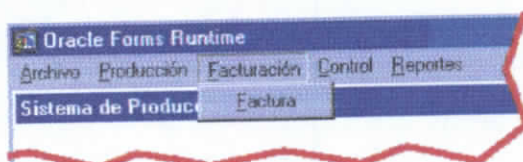


Fig. A5 Opción de Facturación

Opción **Control**: Permite manipular información de: Vacunas, Exámenes, Montas, Crías y Descargo (Fig. A6):

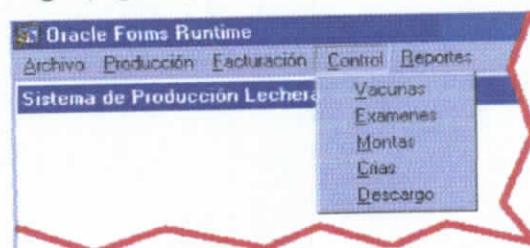


Fig. A6 Opción de Control

Opción **Reportes**: Permite obtener los diferentes tipos de listados, requeridos por la gerencia: Facturas, Ganado, Producción, Control del Ganado, Montas del Ganado, Partos del Ganado, y Descargo (Fig. A7):

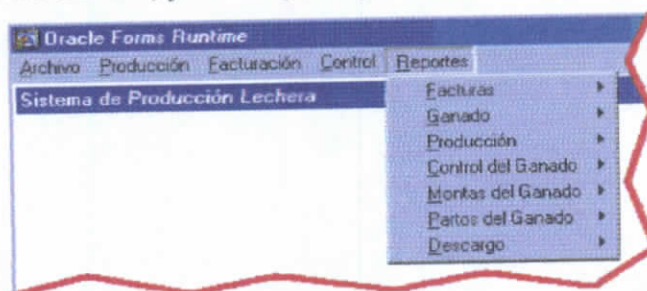


Fig. A7 Opción de Reportes

3.- Actualización de Datos

Al ingresar a las opciones de **Archivo**: Ganado, Productos y Clientes, se mantiene una barra con botones común, en la parte inferior (Fig. A8):

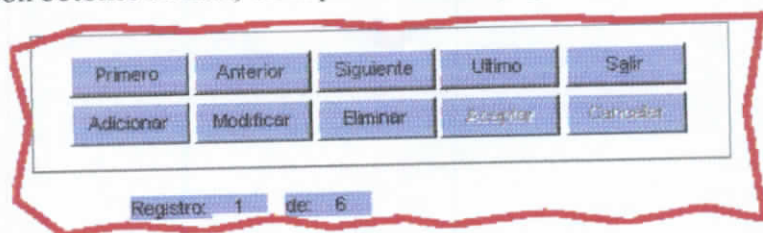


Fig. A8 Botones de Navegación

- **Primero**: para ubicase en el primer registro de la tabla
- **Anterior**: para regresar al registro anterior en la tabla
- **Siguiente**: para avanzar al registro siguiente de la tabla
- **Ultimo**: para ubicarse en el último registro de la tabla
- **Salir**: para abandonar la opción y retornar a Opciones Principales
- **Adicionar**: para ingresar un nuevo registro en la tabla
- **Modificar**: para editar el registro actual
- **Eliminar**: para borrar el registro actual
- **Aceptar**: para grabar el ingreso o modificación realizado
- **Cancelar**: para no hacer efectivos los cambios realizados

Además del número del registro donde se encuentra ubicado, y el número total de registros existentes en la tabla correspondiente, en la parte inferior.

Opción **Archivo** | **Ganado**: permite manipular los datos siguientes, correspondientes a cada una de las cabezas de ganado (Fig. A9):

Fig. A9. Opción Archivo | Ganado

- **Registro:** número de registro (asignado automáticamente)
- **Arete:** número de arete asignado
- **Nombre:** nombre del ganado
- **Sexo:** sexo del ganado (macho o hembra)
- **Estado:** estado en que se encuentra (ternera, señorita, vientre, vaca lechera, o descargo)
- **Peso:** peso en Kg.
- **Fecha Nac:** fecha de nacimiento
- **Lugar Nacimiento:** lugar de nacimiento
- **Grado de Sangre:** grado de sangre
- **Nombre Padre:** nombre del padre
- **Nombre Madre:** nombre de la madre

Opción **Archivo** | **Productos:** permite manipular los datos siguientes, correspondientes a cada producto que se generan en la hacienda (Fig. A10):

Productos

Productos

* Código: 1

* Nombre: LECHE

Cantidad: 17

* Precio: 58

* Unidad: LITROS

* Incluye IVA: NO

Fig. A10. Opción Archivo | Productos

- **Código:** número de código asignado al producto
- **Nombre:** nombre del producto (Ejm: leche, abono, zamora, etc)
- **Cantidad:** cantidad en existencia
- **Precio:** precio de venta por unidad
- **Unidad:** unidad de medida del producto
- **Incluye IVA:** se incluye o no el IVA (Si / No)

Opción **Archivo** | **Clientes:** permite manipular los datos siguientes, correspondientes a cada uno de los clientes de la hacienda (Fig. A11):

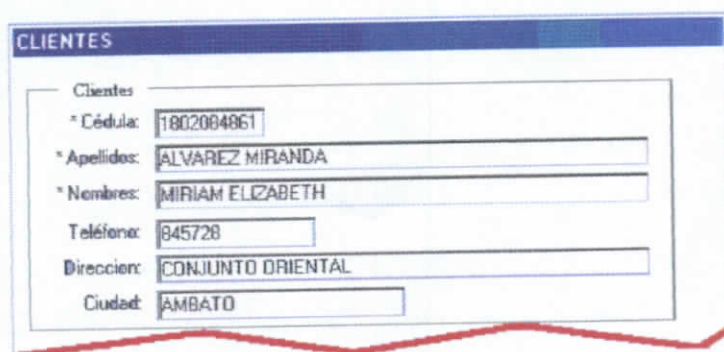


Fig. A11. Opción Archivo | Clientes

- **Cédula:** número de cédula del cliente
- **Apellidos:** apellidos del cliente
- **Nombres:** nombres del clientes
- **Teléfono:** número e teléfono del cliente
- **Dirección:** dirección del cliente
- **Ciudad:** lugar de residencia del cliente

Opción **Archivo** | **Parámetros:** permite manipular el dato siguiente, correspondiente al valor de IVA (Fig. A12):

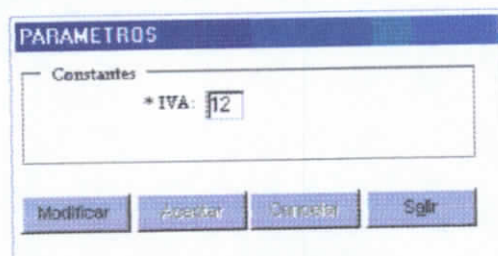


Fig. A12. Opción Archivo | Parámetros

- **IVA:** valor de IVA, que será asignado o no a ciertos productos

Contiene únicamente los botones Modificar, Aceptar, Cancelar y Salir.

4.- Control de Producción

Opción **Producción** | **Ingresar Producción:** permite manipular los datos siguientes, correspondiente a los valores de producción (Fig. A13):

PRODUCCIÓN

Ganado

Código: 00006 Arete: 00449

Nombre: LIDA

* Sexo: Hembra Estado: Vaca Lechera

Arete a buscar:

Buscar

Agregar Producción

Detalle Producción

Código	Producto	Cantidad	Fecha
1	LECHE	32	19/04/2005
1	LECHE	6	19/04/2005
2	ZAMORA	12	19/04/2005
1	LECHE	12	19/04/2005

Primero Anterior Siguiente Ultimo Salir

Registro: 1 de 2

Fig. A13. Opción Producción | Ingresar Producción

- **Ganado:** despliega información del ganado (Código, Arete, Nombre, Sexo y Estado).
- **Detalle Producción:** despliega información del detalle de producción del ganado seleccionado en la sección **Ganado**.
- **Arete a Buscar:** permite acceder a cierto ganado, luego de ingresar el número de arete y presionar el botón **Buscar**.

Para agregar detalle de producción para el ganado seleccionado, presionar el botón **Agregar Producción**, luego de lo cuál, aparece la siguiente ventana (Fig. A14):

Detalle Producción

* Fecha: 05/05/2005

* Producto:

* Cantidad:

Aceptar Cancelar

Fig. A14. Agregar Producción

- **Fecha:** Por omisión, se presenta la fecha actual, se puede modificar únicamente con fechas anteriores.
- **Producto:** de la lista, seleccionar el nombre del producto.
- **Cantidad:** ingresar la cantidad del producto.

5.- Facturación

Opción **Facturación | Factura**: permite generar las facturas, correspondiente a los productos vendidos (Fig. A15):

FACTURA

Factura
Número: 000008 Fecha: 10/05/2005
* Cliente: [dropdown]

Detalle

Producto	Cant.	ValU.	IVA	Sub Total	Eliminar
[dropdown]	[input]	[input]	[input]	[input]	X
[dropdown]	[input]	[input]	[input]	[input]	X
[dropdown]	[input]	[input]	[input]	[input]	X
[dropdown]	[input]	[input]	[input]	[input]	X
[dropdown]	[input]	[input]	[input]	[input]	X
[dropdown]	[input]	[input]	[input]	[input]	X
[dropdown]	[input]	[input]	[input]	[input]	X
[dropdown]	[input]	[input]	[input]	[input]	X
[dropdown]	[input]	[input]	[input]	[input]	X
[dropdown]	[input]	[input]	[input]	[input]	X

Sub Total: (\$) [input]
Sub Total: (\$) [input]
Total a Pagar: (\$) [input]

Nueva Factura
Aceptar
Cancelar
Nuevo
Eliminar
Salir

Fig. A15. Opción Facturación | Factura

Para generar una nueva factura, presionar el botón **Nueva Factura**, el número de la Factura se genera en forma automática.

Seleccionar de la lista, el nombre del cliente, luego de la lista de productos, seleccionar uno a uno los productos, e ingresar la cantidad adquirida por el cliente.

Los Subtotales, IVA y Valor a Pagar, se generan en forma automática.

Para Eliminar un registro, hacer clic en el botón marcado con una X. (Fig. A16):

FACTURA

Factura

Número: 000008

Fecha: 10/05/2005

* Cliente: ALVAREZ MIRANDA MIRIAM ELIZABETH

Detalle

	Producto	Cant.	ValU.	IVA	Sub Total	Eliminar
1	LECHE	10	58	0	5,8	X
						X
						X
						X
						X
						X
						X
						X
						X
						X

Sub Total: (\$)

5,8

Sub Total: (\$)

0

Total a Pagar: (\$)

5,8

Nueva Factura

Aceptar

Cancelar

Imprimir

Reservar

Salir

Fig. A16. Ejemplo de factura

Una vez finalizada una Factura, presionar el botón **Aceptar**, luego de lo cuál, se puede enviar a la impresora con el botón **Imprimir**.

6.- Control

Opción **Control | Vacunas**: permite llevar el registro de vacunas de cada una de las cabezas de ganado de la Hacienda (Fig. A17):

VACUNAS

Ganado

Código: 000008

Arete: 00500

Nombre: TURCO

* Sexo: Macho

Estado:

Arete a buscar:

Buscar

Grabar

Cancelar

Vacunas

Fecha	Descripción	Eliminar
19/04/2005	INDIAVIT 2MILIMETROS	X
10/04/2005	FIEBRE APTOSA 3 MIL	X
19/04/2005	LEPTOSPIROSIS	X
		X
		X
		X
		X
		X
		X
		X

Primero

Anterior

Siguiente

Ultimo

Salir

Registro: 5 de 6

Fig. A17. Opción Control | Vacunas

Ingresa la fecha, y la descripción de la vacuna suministrada al ganado, cuya información aparece en la sección **Ganado**.

Se puede ubicar a una determinada cabeza de ganado, ingresando su número de arete, en la caja **Arete a Buscar**, y luego presionar el botón **Buscar**.

Luego de realizar cambios en la información, presionar el botón **Grabar**.

Opción **Control | Exámenes**: permite llevar el registro de exámenes y tratamientos de cada una de las cabezas de ganado de la Hacienda (Fig. A18):

Fecha	Diagnóstico	Tratamiento	Observación	Eliminar
10/05/2005	Parasitosis	Inyección de Pota-sulfato	una cada dos días	X
				X
				X
				X
				X
				X
				X
				X

Fig. A18. Opción Control | Exámenes

Ingresa la fecha, la descripción del examen y las observaciones de cada cabeza de ganado, cuya información aparece en la sección **Ganado**.

Se puede ubicar a una determinada cabeza de ganado, ingresando su número de arete, en la caja **Arete a Buscar**, y luego presionar el botón **Buscar**.

Luego de realizar cambios en la información, presionar el botón **Grabar**.

Opción **Control | Montas**: permite llevar el registro de las montas realizadas a cada cabezas de ganado de la Hacienda (Fig. A19):

Ganado		Arete a buscar:
Código: 00006	Arete: 00449	Buscar
Nombre: LIDA		
* Sexo: Hembra	Estado: Vaca Lechero	Grabar Cancelar

Registro de Montas			
Fecha	Nombre del Toro	Tipo	Estatus
22/03/1994	ROCKET	Monta	X
10/05/2005	BABALLU	Inseminación	X

Primero	Anterior	Siguiente	Ultimo	Salir
---------	----------	-----------	--------	-------

Registro: 0 de 2

Fig. A19. Opción Control | Montas

Ingresar la fecha, nombre del toro y seleccionar el tipo de monta, realizado a cada cabeza de ganado, cuya información aparece en la sección **Ganado**.

Se puede ubicar a una determinada cabeza de ganado, ingresando su número de arete, en la caja **Arete a Buscar**, y luego presionar el botón **Buscar**.

Luego de realizar cambios en la información, presionar el botón **Grabar**.

Opción **Control** | **Crías**: permite llevar el registro de las crías de cada una de las vacas de la Hacienda (Fig. A20):

CRIAS			
Nombre de la Madre: 00007 MERY			
Presione F9 para ver lista			
Ganado			
* Arete:	00147		
* Nombre:	TANIA		
* Sexo:	Hembra	Estado:	Descargo
		Peso:	0
Fecha Nac.(DD/MM/AAAA):	Lugar Nacimiento:		
19/09/1997	CUNCHIBAMBA		
Grado de Sangre:	PURA		
Nombre Padre:	ROCKET		
Adicionar Aceptar Cancelar Salir			

Fig. A20. Opción Control | Crías

Seleccionar el nombre de la madre de una lista que aparece al presionar la tecla **F9**, de la cuál aparece la información de sus crías.

Para adicionar información de otra cría, presionar el botón **Adicionar** (Fig. A21):

Fig. A21. Ingreso de Crías

Opción **Control | Descargo**: permite llevar el registro de los descargos de ganado y las razones por las que ocurre (Fig. A22):

Fig. A22. Opción Control | Descargo

Ubicar al ganado, o ingresar el número de arete en la caja **Arete a Buscar**, y luego presionar el botón **Buscar**.

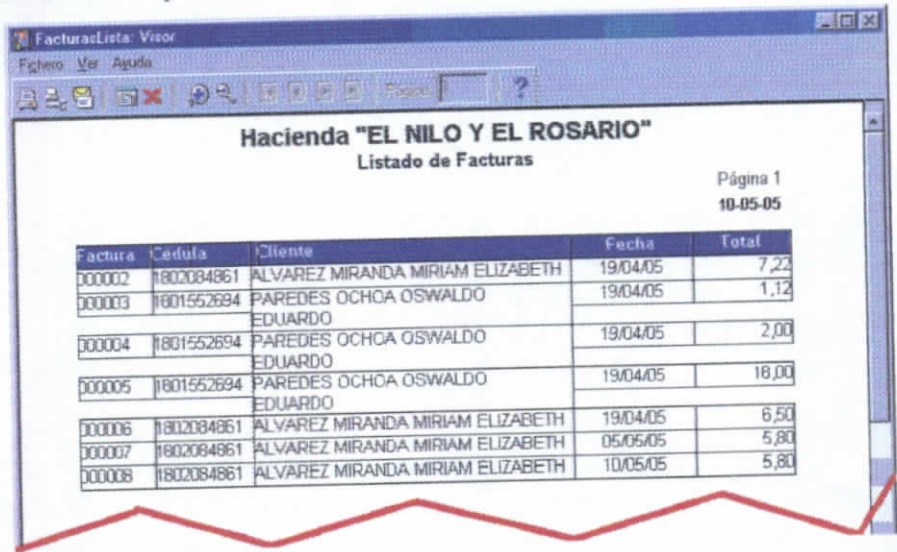
Una vez ubicado el ganado, presionar el botón Descargo, ingresar la fecha, seleccionar el motivo por el cuál se descarga el ganado, el valor y la observación, luego presionar el botón **Aceptar**.

7.- Reportes

Las reportes, en general, tienen una misma forma de funcionamiento, dependiendo de la opción seleccionada, y pueden ser enviados a la impresora.

Opción **Reportes** | **Facturas** (Listado General, Listado por Fecha, Factura por Número).

El Listado General presenta todas las facturas (Fig. A23):

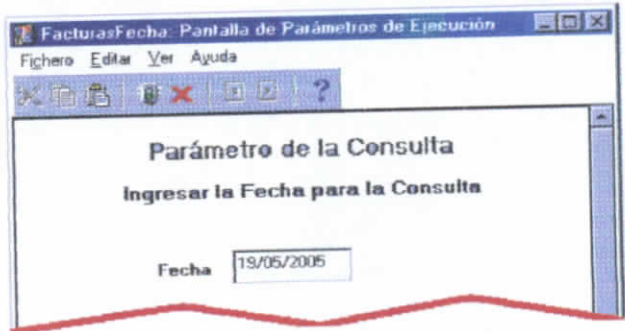


The screenshot shows a window titled 'FacturasLista: Vncor' with a menu bar (Fichero, Ver, Ayuda) and a toolbar. The main content area is titled 'Hacienda "EL NILO Y EL ROSARIO" Listado de Facturas'. It indicates 'Página 1' and the date '10-05-05'. Below this is a table with 5 columns: Factura, Cedula, Cliente, Fecha, and Total. The table contains 8 rows of invoice data.

Factura	Cedula	Cliente	Fecha	Total
000002	1802084861	ALVAREZ MIRANDA MIRIAM ELIZABETH	19/04/05	7,22
000003	1801552694	PAREDES OCHOA OSWALDO	19/04/05	1,12
		EDUARDO		
000004	1801552694	PAREDES OCHOA OSWALDO	19/04/05	2,00
		EDUARDO		
000005	1801552694	PAREDES OCHOA OSWALDO	19/04/05	16,00
		EDUARDO		
000006	1802084861	ALVAREZ MIRANDA MIRIAM ELIZABETH	19/04/05	6,50
000007	1802084861	ALVAREZ MIRANDA MIRIAM ELIZABETH	05/05/05	5,80
000008	1802084861	ALVAREZ MIRANDA MIRIAM ELIZABETH	10/05/05	5,80

Fig. A23. Listado de Facturas General

El listado por fecha, presenta las facturas de una determinada fecha, que debe ser ingresada en la ventana (Fig. A24), con el mismo formato que la Fig. A23:



The screenshot shows a window titled 'FacturasFecha: Pantalla de Parámetros de Ejecución' with a menu bar (Fichero, Editar, Ver, Ayuda) and a toolbar. The main content area is titled 'Parámetro de la Consulta' and 'Ingresar la Fecha para la Consulta'. There is a label 'Fecha' followed by a text input field containing the date '19/05/2005'.

Fig. A24. Parámetro: Fecha de Facturas

El listado Factura por Número (Fig. A26), presenta la información de una factura, que debe ser ingresada en la siguiente ventana (Fig. A25):

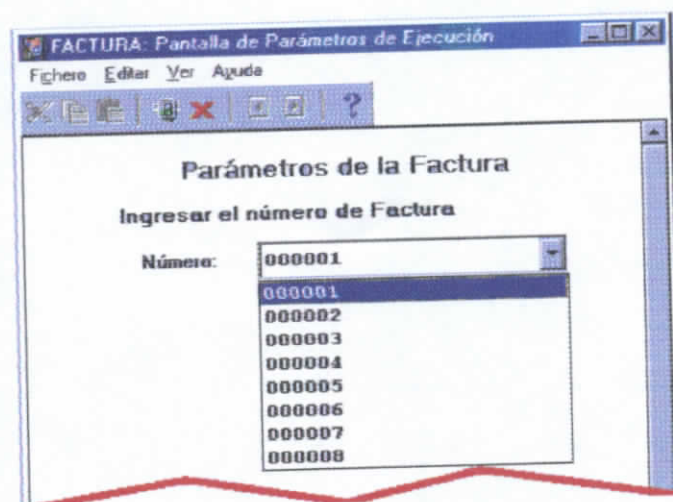


Fig. A25. Parámetro: Número de Factura

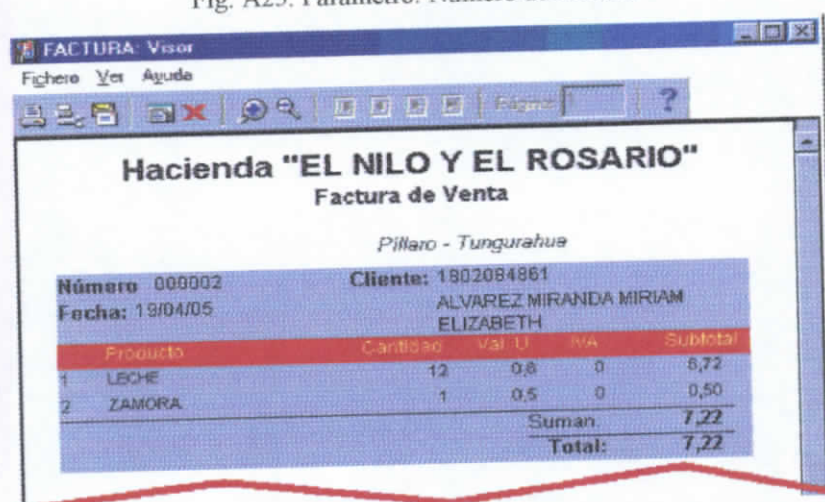


Fig. A26. Listado de Factura por Número

Opción **Reportes** | **Ganado** (Todos, Información por Ejemplar, Por Estado, y Por Sexo) (Fig. A27):

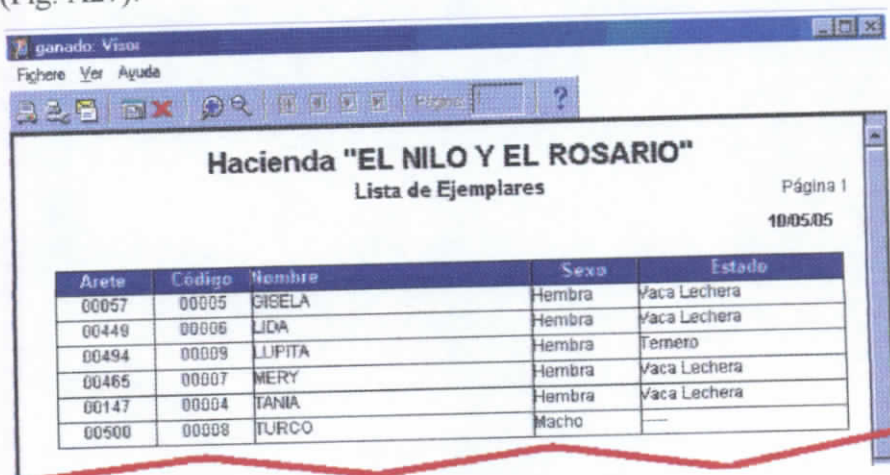


Fig. A27. Listado Total de Ganado

Opción **Reportes** | **Producción** (Por Ejemplar, Máxima, y Por Fecha) (Fig. A28):

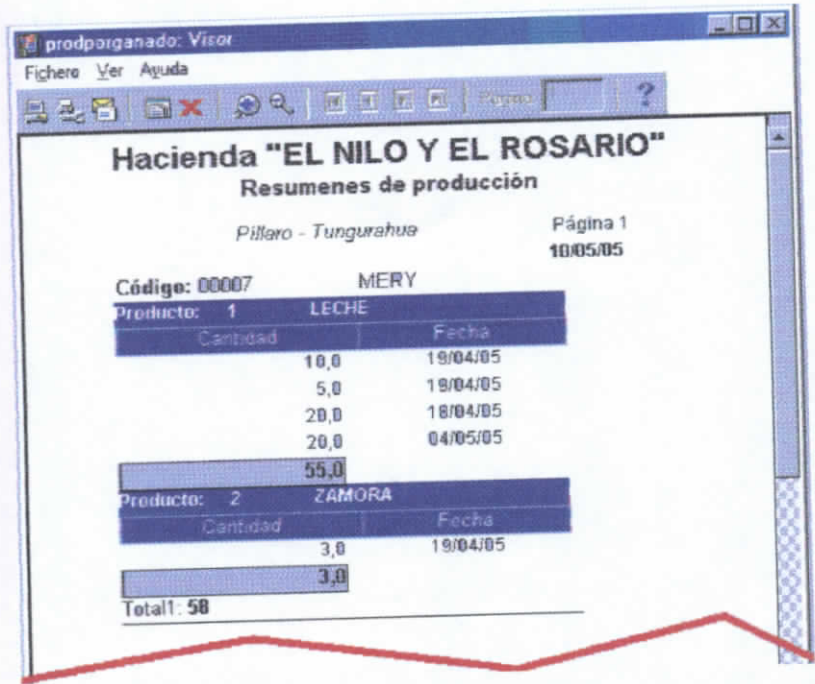


Fig. A28. Listado de Producción por Ejemplar

Opción Reportes | Control de Ganado (Vacunas y Exámenes) (Fig. A29):

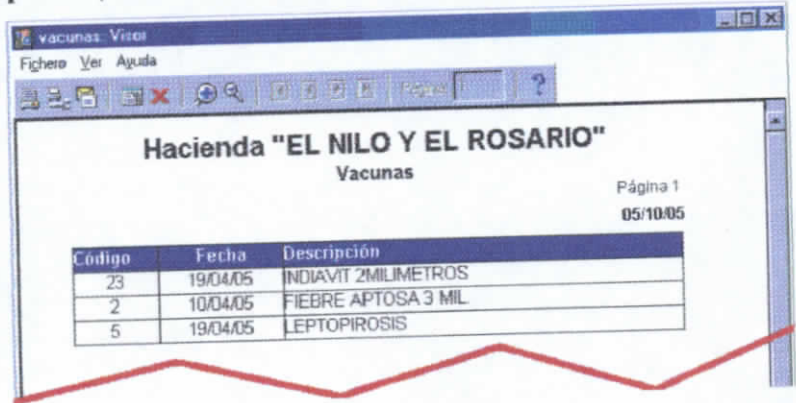


Fig. A29. Listado de Vacunas del Ganado

Opción Reportes | Montas del Ganado (Todas y Por Ejemplar) (Fig. A30):

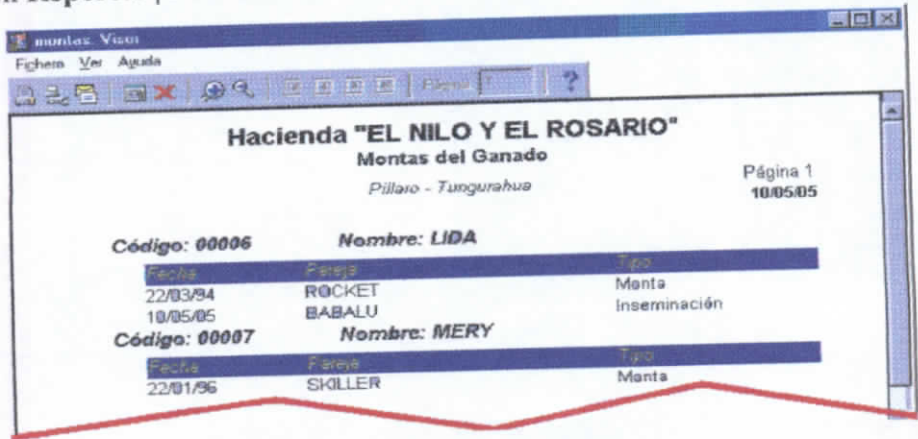


Fig. A30. Listado de Montas del Ganado

Opción **Reportes | Partos del Ganado** (Todas y Por Madre) (Fig. A31):

Hacienda "EL NILO Y EL ROSARIO"			
Listado de Crías			
			Página 1
			05/10/05
Código	Fecha	Arete	Cría
1	29/06/94	00494	LUPITA

Fig. A31. Listado de Partos del Ganado

Opción **Reportes | Descargos** (Todas y Por Ejemplar) (Fig. A32):

Hacienda "EL NILO Y EL ROSARIO"								
Descargo								
Arete	Sexo	Estado	Nombre	Fecha	Descargo	Valor	Otros	
00057	Hembra	Vaca Lechera	GISELA	10/05/05	Muerte			
00465	Hembra	Vaca Lechera	MERY	10/05/05	Otra		Cambio por Toro	
00147	Hembra	Vaca Lechera	TANIA	19/04/05	Remate	500		

Fig. A32. Listado de Descargos de Ganado

**Sistema: Control de Producción de Leche en la
Hacienda “El Nilo y El Rosario”
MANUAL DEL ADMINISTRADOR**

Contenido

Contenido 1

Índice de Gráficos 1

1.- Requisitos 2

2.- Instalación 2

3.- Creación de las Tablas del Sistema 5

Índice de Gráficos

Fig. B1 Archivos para Instalación 2

Fig. B2 Presentación de la Instalación 3

Fig. B3 Directorio de Instalación 3

Fig. B4 Proceso de Instalación 4

Fig. B5 Fin del Proceso de Instalación 4

Fig. B6 Grupo de Programas del Sistema 5

Fig. B7 Ingreso a SQL como Administrador 5

Fig. B8 Crear Tablas del Sistema 6

1.- Requisitos

Para la instalación y ejecución del sistema **Control de Producción de Leche en la Hacienda “El Nilo y El Rosario”**, se requiere:

En el servidor:

- Sistema Operativo Windows NT/2000/XP
- Servidor de Base de Datos Oracle 8 o superior

En el cliente:

- Sistema Operativo Windows 98/XP
- Forms & Reports Runtime
- Cliente de Base de Datos Oracle 8 o superior

2.- Instalación

Para instalar el sistema, se requiere de los siguientes archivos en un dispositivo de instalación (Ejm: E:), o en una carpeta del equipo (Ejm: C:\Sistema): Fig. B1



Fig. B1 Archivos para Instalación

Los archivos son: el primero el de instalación, y el segundo para la creación de la Base de Datos para el sistema.

Ejecutar el archivo de instalación **Hacienda**: aparece la siguiente información: (Fig. B2)

Hacienda EL NILO y EL ROSARIO

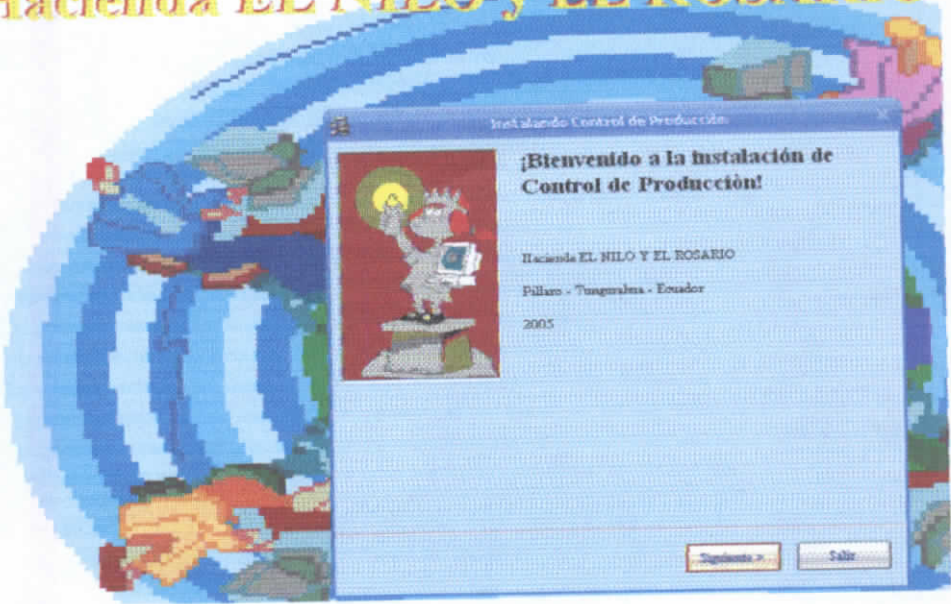


Fig. B2 Presentación de la Instalación

Para continuar presionar el botón **Siguiente**, para abandonar la instalación presionar el botón **Salir**.

Luego de presionar el botón Siguiente, aparece la ventana: (Fig. B3)

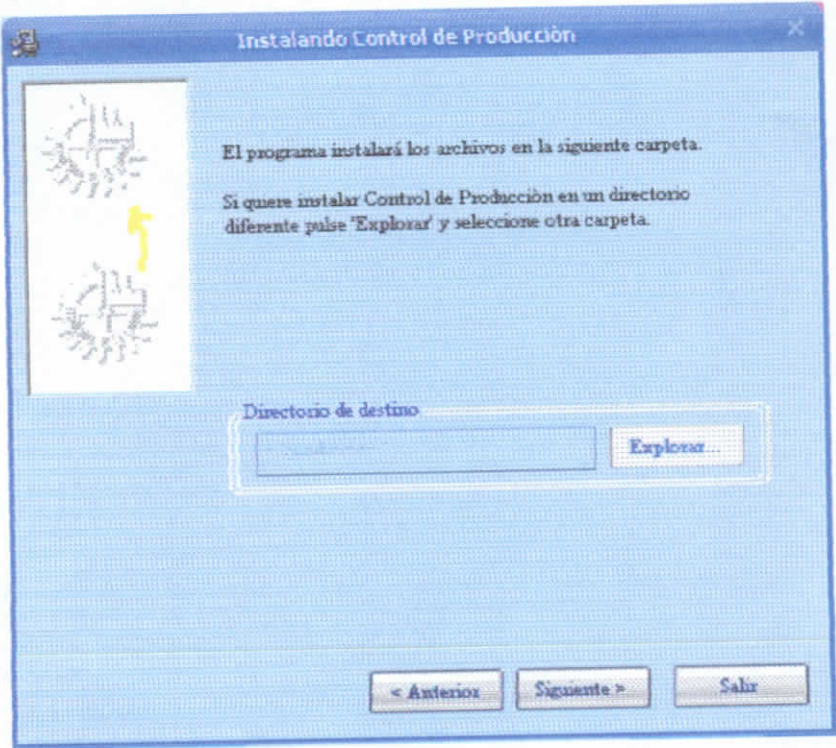


Fig. B3 Directorio de Instalación

El sistema se instalará en el directorio **C:\Produccion**, presionar el botón Siguiente, para iniciar el proceso de instalación: (Fig. B4)

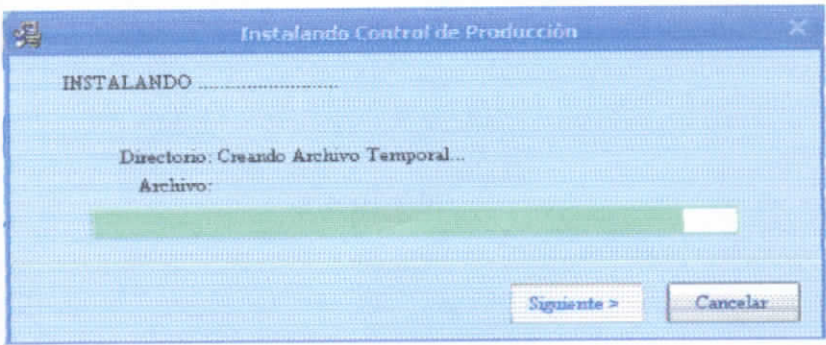


Fig. B4 Proceso de Instalación

Una vez finalizada la instalación, aparecen las ventanas: (Fig. B5)

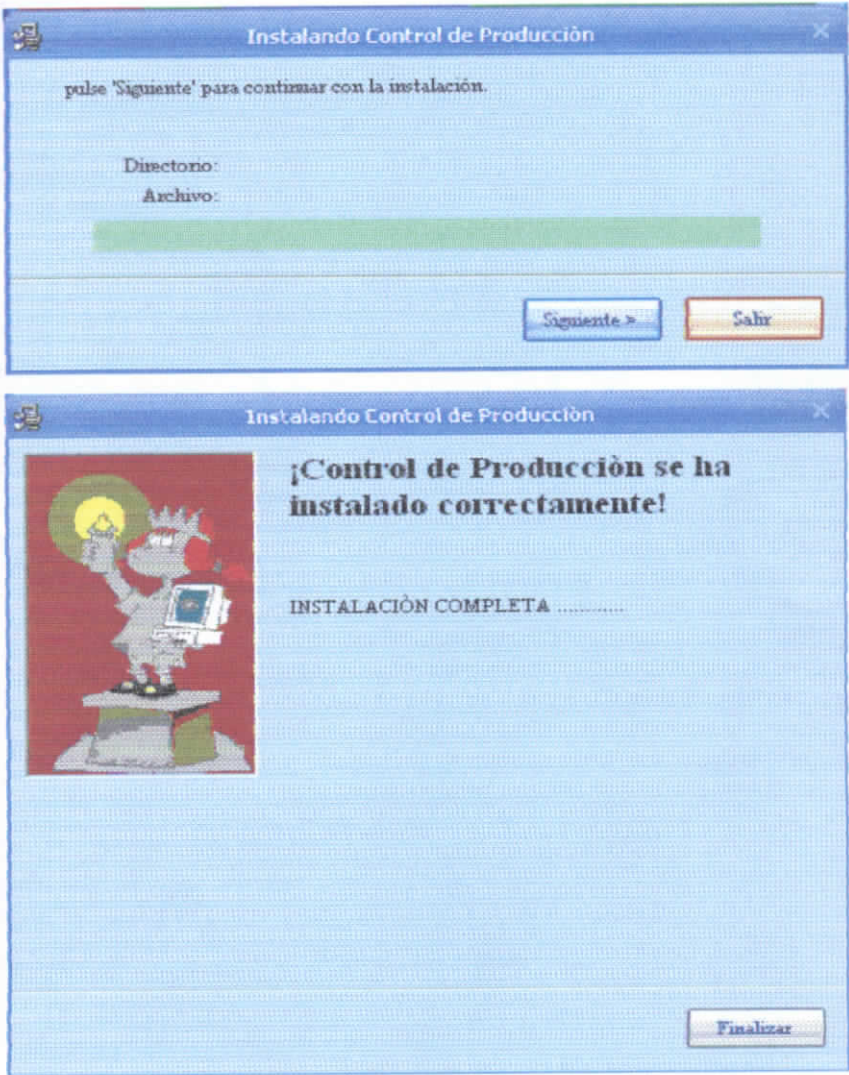


Fig. B5 Fin del Proceso de Instalación

Con lo cual se crea el grupo de programas **Hacienda**, con los archivos de ingreso al sistema llamado **Hacienda** y el archivo de Desinstalación llamado **Desinstalar**. (Fig. B6)

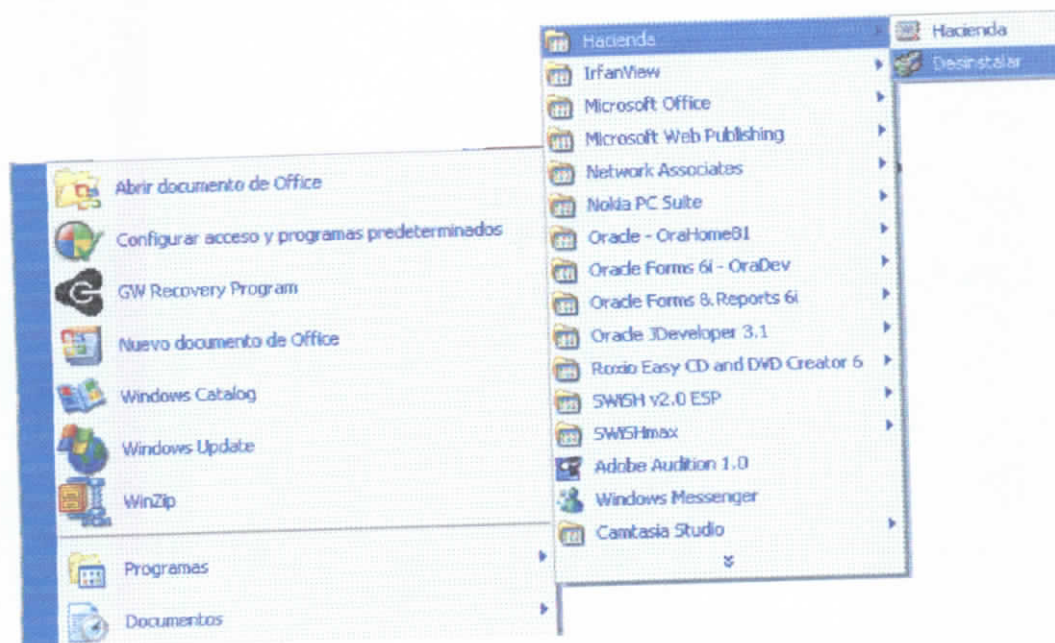


Fig. B6 Grupo de Programas del Sistema

3.- Creación de las Tablas del Sistema

Una vez instalada la aplicación, se requiere crear el usuario y las tablas en la Base de Datos, para lo cual se debe seguir los siguientes pasos:

a.- Ingresar a SQL como administrador: (Fig. B7)

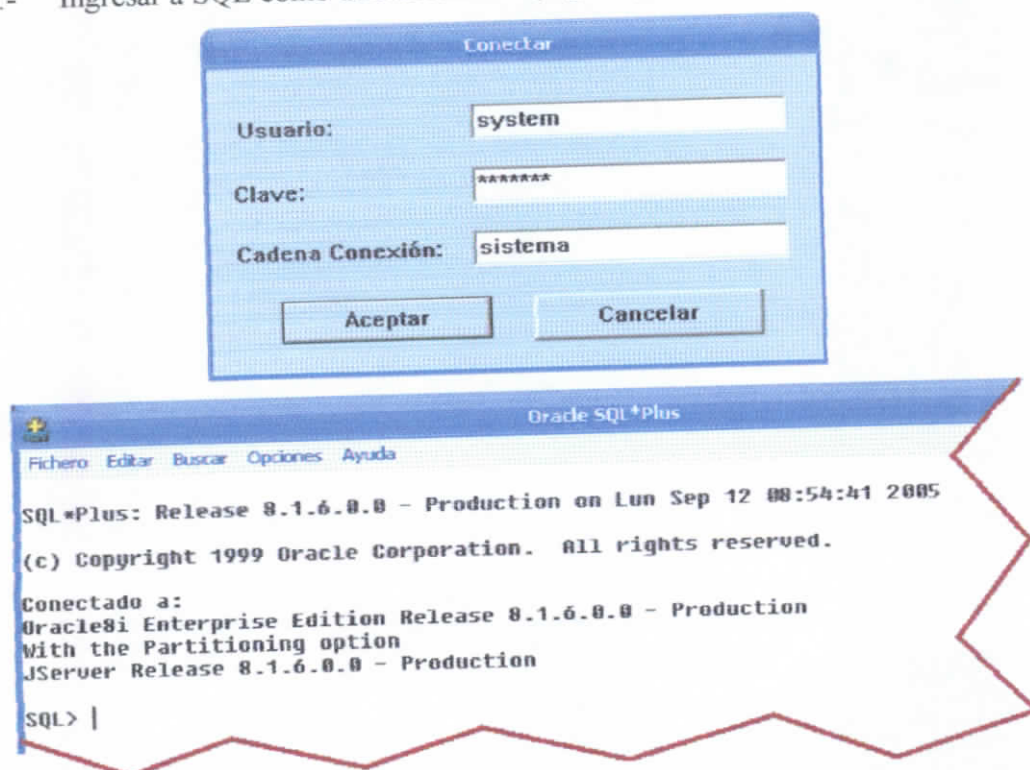


Fig. B7 Ingreso a SQL como Administrador

- b.- Ejecutar el archivo **Hacienda.sql**, desde la unidad o directorio de instalación: (Fig. B8)



The image shows a screenshot of a SQL command prompt window. The text is enclosed in a red, hand-drawn, star-like border. The command entered is 'SQL> start c:\Sistema\hacienda.sql' and the output is 'Construyendo las Tablas del Sistema'.

```
SQL> start c:\Sistema\hacienda.sql  
Construyendo las Tablas del Sistema
```

Fig. B8 Crear Tablas del Sistema

VALIDACIÓN



HACIENDA "EL NILO & EL ROSARIO"

Dir: Píllaro – Tungurahua / Telf: 2873119

CERTIFICACION

JUAN FRANCISCO VARGAS CONTRERAS, Administrador de las Haciendas "El Rosario-El Nilo" de la Parroquia Marcos Espinel del Cantón Píllaro, tengo a bien en certificar en honor a la verdad que en las Haciendas se utiliza el Sistema para el Control de la Producción de Leche y utilización de las Herramientas RAD (Rapid Application Development) DE ORACLE, el mismo que esta aplicándose y funcionando perfectamente.

Agradezco infinitamente a la Tecnóloga MIRIAM ELIZABETH ALVAREZ MIRANDA y de manera especial a la Pontificia Universidad Católica de Ambato.

Es todo cuento certificar en honor a la verdad.

Píllaro 3 de Octubre del 2.005.

Atentamente,

AB. JUAN FRANCISCO VARGAS CONTRERAS
Administrador de las Haciendas " El Rosario-El Nilo"
Píllaro-Tungurahua

Ambato octubre 12, 2005

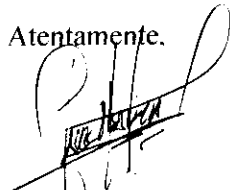
Ingeniero
Telmo Viteri
DIRECTOR ESCUELA DE INGENIERIA DE SISTEMAS
Presente

De mi consideración:

Tengo a bien indicar que he revisado la parte práctica de la Disertación de Grado titulado "ESTUDIO DE LAS HERRAMIENTAS RAD (Rapid Application Development) DE ORACLE PARA EL DESARROLLO DE APLICACIONES", correspondiente a la Sra. Miriam Elizabeth Álvarez Miranda, el cual se encuentra funcionando correctamente, por lo que sugiero se proceda con el trámite de defensa de la misma.

Particular que pongo en su conocimiento, para los fines pertinentes.

Atentamente.



Ing. Patricio Medina
VALIDADOR DE LA DISERTACIÓN

PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR
Sede Ambato
ESCUELA DE INGENIERIA DE SISTEMAS
SECRETARIA

Recibido: Meja

Fecha: 13-0d-2005 Hora: 12h00

Ambato octubre 13, 2005

Ingeniero
Telmo Viteri
DIRECTOR ESCUELA DE INGENIERIA DE SISTEMAS
Presente

De mi consideración:

Tengo a bien indicar que he revisado la parte práctica de la Disertación de Grado titulado "ESTUDIO DE LAS HERRAMIENTAS RAD (Rapid Application Development) DE ORACLE PARA EL DESARROLLO DE APLICACIONES", correspondiente a la Sra. Miriam Elizabeth Álvarez Miranda, el cual se encuentra funcionando correctamente, cumpliendo con los objetivos planteados; por lo se puede proceder a la defensa respectiva.

Particular que pongo en su conocimiento, para los fines pertinentes.

Atentamente,



Ing. Darío Robayo
VALIDADOR DE LA DISERTACIÓN

PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR
Sede Ambaro
ESCUELA DE INGENIERIA DE SISTEMAS
SECRETARIA

Recibido: 
Fecha: 13-Oct-2005 Hora: 10h40

