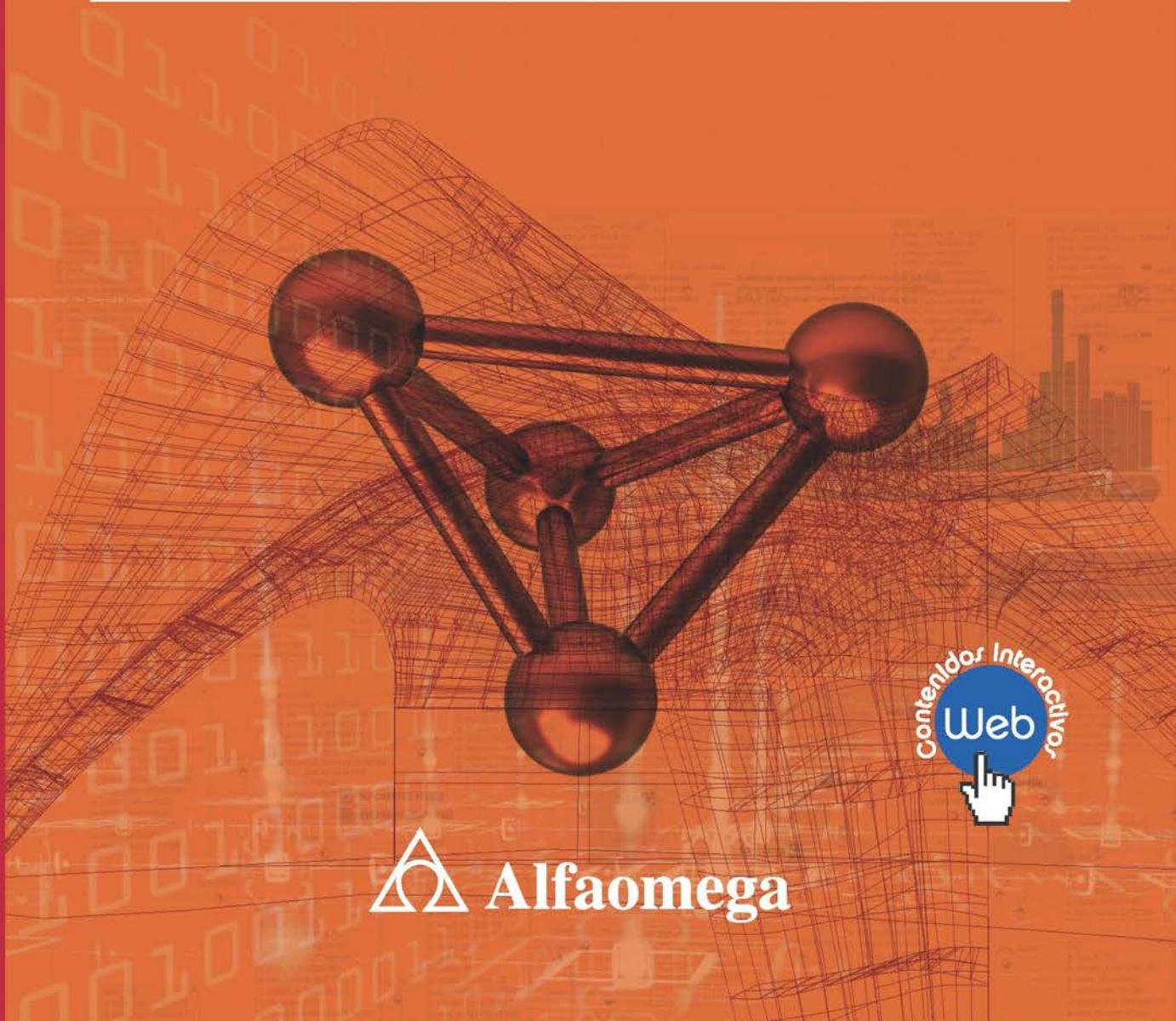


# BASES DE DATOS

---

ENRIQUE JOSÉ REINOSA - CALIXTO ALEJANDRO MALDONADO - ROBERTO MUÑOZ -  
LUIS ESTEBAN DAMIANO - MAXIMILIANO ADRIÁN ABRUTSKY

---



libroweb



Alfaomega







# BASES DE DATOS



# **BASES DE DATOS**

**Enrique José Reinosa - Calixto Alejandro Maldonado - Roberto Muñoz -  
Luis Esteban Damiano - Maximiliano Adrián Abrutsky**



Buenos Aires • Bogotá • México DF • Santiago de Chile

Reinosa, Enrique José  
Bases de Datos / Enrique José Reinosa ; Calixto Alejandro Maldonado ; Roberto Muñoz ; Luis Esteban Damiano ; Maximiliano Adrián Abrutsky. - 1a ed.  
- Buenos Aires : Alfaomega Grupo Editor Argentino, 2012.  
384 p. ; 24x21 cm.

ISBN 978-987-1609-31-4

1. Informática. 2. Base de datos. I. Maldonado, Calixto II. Muñoz, Roberto  
III. Título  
CDD 005.3

Queda prohibida la reproducción total o parcial de esta obra, su tratamiento informático y/o la transmisión por cualquier otra forma o medio sin autorización escrita de Alfaomega Grupo Editor Argentino S.A.

Edición: Damián Fernández

Corrección de estilo: Juan Arana, Silvia Mellino y Vanesa García

Diagramación de interiores: Diego Linares y Patricia Baggio

Revisión de armado: Juan Micán

Revisión técnica: Claudia Deco, Cristina Bendec, Calixto Alejandro Maldonado y Roberto Muñoz

Diseño de tapa: Diego Linares

Internet: <http://www.alfaomega.com.mx>

Todos los derechos reservados © 2012, por Alfaomega Grupo Editor Argentino S.A.

Paraguay 1307, PB, oficina 11

ISBN 978-987-1609-31-4

Queda hecho el depósito que prevé la ley 11.723

**NOTA IMPORTANTE:** La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. Alfaomega Grupo Editor Argentino S.A. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Los nombres comerciales que aparecen en este libro son marcas registradas de sus propietarios y se mencionan únicamente con fines didácticos, por lo que Alfaomega Grupo Editor Argentino S.A. no asume ninguna responsabilidad por el uso que se dé a esta información, ya que no infringe ningún derecho de registro de marca. Los datos de los ejemplos y pantallas son ficticios, a no ser que se especifique lo contrario.

Los hipervínculos a los que se hace referencia no son necesariamente administrados por la editorial, por lo que no somos responsables de sus contenidos o de su disponibilidad en línea.

Empresas del grupo:

Argentina: Alfaomega Grupo Editor Argentino, S.A.  
Paraguay 1307 P.B. "11", Buenos Aires, Argentina, C.P. 1057  
Tel.: (54-11) 4811-7183 / 0887  
E-mail: [ventas@alfaomegagroupeditor.com.ar](mailto:ventas@alfaomegagroupeditor.com.ar)

México: Alfaomega Grupo Editor, S.A. de C.V.  
Pitágoras 1139, Col. Del Valle, México, D.F., México, C.P. 03100  
Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396  
E-mail: [atencionalcliente@alfaomega.com.mx](mailto:atencionalcliente@alfaomega.com.mx)

Colombia: Alfaomega Colombiana S.A.  
Carrera 15 No. 64 A 29, Bogotá, Colombia  
PBX (57-1) 2100122 – Fax: (57-1) 6068648  
E-mail: [cliente@alfaomega.com.co](mailto:cliente@alfaomega.com.co)

Chile: Alfaomega Grupo Editor, S.A.  
Doctor La Sierra 1437 – Providencia, Santiago, Chile  
Tel.: (56-2) 235-4248 – Fax: (56-2) 235-5786  
E-mail: [aechile@alfaomega.cl](mailto:aechile@alfaomega.cl)

“Dedicado a mi familia por el tiempo restado a ellos para la generación de este libro y a mis compañeros docentes y ayudantes por su colaboración en el proceso educativo de nuestros alumnos”.

**Enrique José Reinoso**

“A Alicia, mi compañera por veinte años y a Ludmila, Emilio y Sofía, mis tesoros, les dedico el esfuerzo y el resultado”.  
**Calixto Alejandro Maldonado**

“A mis padres, por inculcarme el valor del estudio. A Eugenia, Alfonsina y Amparo por el estímulo permanente en todo lo que emprendo”.  
**Roberto Muñoz**

“A los intelectualmente generosos, quienes ayudan a la construcción de un conocimiento para todos”.

“A mi familia, a Valentina, Bruno y Andrea”.

**Luis Esteban Damiano**

“A Kela, mi madre, por inculcarme el esfuerzo y sus principios morales que me conducen en la vida, a Marcos, hermano y mejor amigo, y a Roberto un ejemplo de profesional y persona, sin ellos este aporte no hubiese sido posible”.

**Maximiliano Adrián Abrutsky**



## **Mensaje del Editor**

Los conocimientos son esenciales en el desempeño profesional. Sin ellos es imposible lograr las habilidades para competir laboralmente. La universidad o las instituciones de formación para el trabajo ofrecen la oportunidad de adquirir conocimientos que serán aprovechados más adelante en beneficio propio y de la sociedad. El avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos. Cuando se toma la decisión de embarcarse en una vida profesional, se adquiere un compromiso de por vida: mantenerse al día en los conocimientos del área u oficio que se ha decidido desempeñar.

Alfaomega tiene por misión ofrecerles a estudiantes y profesionales conocimientos actualizados dentro de lineamientos pedagógicos que faciliten su utilización y permitan desarrollar las competencias requeridas por una profesión determinada. Alfaomega espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Alfaomega hace uso de los medios impresos tradicionales en combinación con las tecnologías de la información y las comunicaciones (IT) para facilitar el aprendizaje. Libros como éste tienen su complemento en una página Web, en donde el alumno y su profesor encontrarán materiales adicionales, información actualizada, pruebas (test) de autoevaluación, diapositivas y vínculos con otros sitios Web relacionados.

Esta obra contiene numerosos gráficos, cuadros y otros recursos para despertar el interés del estudiante, y facilitarle la comprensión y apropiación del conocimiento.

Cada capítulo se desarrolla con argumentos presentados en forma sencilla y estructurada claramente hacia los objetivos y metas propuestas. Cada capítulo concluye con diversas actividades pedagógicas para asegurar la asimilación del conocimiento y su extensión y actualización futuras.

Los libros de Alfaomega están diseñados para ser utilizados dentro de los procesos de enseñanza-aprendizaje, y pueden ser usados como textos guía en diversos cursos o como apoyo para reforzar el desarrollo profesional.

Alfaomega espera contribuir así a la formación y el desarrollo de profesionales exitosos para beneficio de la sociedad.

**Enrique José Reinosa**

- Analista Universitario de Sistemas de la UTN – FRBA.
- Ingeniero en Sistemas de Información de la UTN – FRBA.
- Posgrado en Ingeniería Gerencial (MBA) en la UTN – FRBA.
- Docente Investigador Categorizado en la UTN – FRBA.
- Profesor Titular Ordinario y Jefe de las Cátedras de:
  - Gestión de Datos
  - Paradigmas de Programación
  - Técnicas Gráficas por Computadora
  - Técnicas Avanzadas de Programación
  - Tecnologías Avanzadas en la Construcción de Software
  - Administración de Recursos
- Coautor del Plan de Estudios 1995 de la Carrera Ingeniería en Sistemas de Información de la UTN.
- Profesor Tutor de Tesis de Títulos Intermedios de Analista Universitario de Sistemas.
- Director de Desarrollo de Software de la UTN – FRBA.
- Director de Desarrollo de Software del Instituto Nacional de Educación Técnica (INET).
- Consultor Independiente en Desarrollo de Software y Bases de Datos.

**Calixto Alejandro Maldonado**

- Ingeniero en Sistemas de Información de la UTN-FRC.
- Profesional certificado Oracle Developer y DBA-OCP.
- Doctorando de Ingeniería en software basado en componentes reutilizables, en la Universidad de Vigo.
- Docente e Investigador en la UTN-FRC y la Universidad Empresarial 21.
- Consultor DBA independiente.

**Roberto Muñoz**

- Ingeniero en Sistemas de Información de la UTN-FRC.
- Especialista en Docencia Universitaria de la UTN-FRC.
- Maestrando en Ingeniería en Sistemas de Información de la UTN-FRC.
- Jefe de la cátedra de Gestión de Datos de la UTN-FRC.
- Investigador en la UTN-FRC.
- Coordinador de la Diplomática Superior en Administración y Exploración de Bases de Datos en la UTN-FRC.
- Coordinador universitario del Programa Nacional “Plan +” de “Becas Control-F” en la UTN-FRC.
- Integrante del Centro de Investigación y Desarrollo de Sistemas (CIDS) en la UTN-FRC.

### **Luis Esteban Damiano**

- Analista de Sistemas de Computación de la UBP-ISP, Argentina.
- Licenciado en Tecnología Educativa de la UTN-FRC.
- Docente de las cátedras de Gestión de datos y Programación de aplicaciones visuales I en la UTN-FRC.
- Analista de Sistemas en la Dirección de Sistemas de Gobierno en Córdoba, Argentina.
- Capacitador del Programa Nacional “Plan+” de “Becas Control-F” (años 2008, 2009, 2010).
- Diseñador de sistemas independiente.

### **Maximiliano Adrián Abrutsky**

- Ingeniero en Sistemas de Información de la UTN-FRC.
- Docente, Investigador y Maestrando (desarrollando Tesis) en la UTN-FRC.
- Docente en la UNC - Fac. Cs. Económicas.
- Socio gerente en Liricus Soluciones Informáticas.
- Java Developer Senior - DBA certificado DB2 - SQL Server.

### **Revisores técnicos:**

Claudia Deco:

- Licenciada en Matemáticas de la Universidad Nacional de Rosario, Argentina.
- Magíster en Informática de la Universidad de la República, Uruguay.
- Doctora en Ingeniería de la Universidad Nacional de Rosario, Argentina.

Cristina Bendec:

- Ingeniera Electrónica de la Universidad de Rosario, Argentina.
- Magíster en Informática de la Universidad de la República, Uruguay.
- Especialista en Gestión de Sistemas de Información de la Universidad Católica Argentina.

Calixto Alejandro Maldonado

Roberto Muñoz

Agradecemos las sugerencias y comentarios realizados por:

- Lucila Patricia Arellano Mendoza, Departamento de Computación de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México.
- Nancy Ocotilla Rojas, Docente del Departamento de Ingeniería en Sistemas Computacionales en la ESCOM – IPN.
- Euler Hernández Contreras, Jefe de la asignatura de Bases de Datos de la carrera de Ingeniería en Sistemas en la Escuela Superior de Cómputo del Instituto Politécnico Nacional.



## Contenido

|  |            |
|--|------------|
| Prólogo .....  | XIX        |
| <b>Capítulo 1</b>  |            |
| <b>Estructura y tipos de bases de datos.....</b>                             | <b>1</b>   |
| <b>1.1 Introducción.....</b>   | <b>2</b>   |
| <b>1.2 Definición de base de datos.....</b>                                  | <b>3</b>   |
| <b>1.3 Sistema de Gestión de Bases de Datos.....</b>                         | <b>4</b>   |
| <b>1.4 Usuarios de la base de datos.....</b>                                 | <b>5</b>   |
| <b>1.5 Seguridad de las bases de datos.....</b>                              | <b>7</b>   |
| 1.5.1 Creación de usuarios.....  | 8          |
| 1.5.2 Otorgar privilegios de sistema.....                                    | 8          |
| 1.5.3 Usar roles para administrar los accesos de base de datos.....          | 9          |
| 1.5.4 Otorgar privilegios de objetos.....                                    | 10         |
| 1.5.5 El cambio de contraseñas.....  | 11         |
| 1.5.6 Uso de sinónimos para la transparencia de base de datos.....           | 11         |
| <b>1.6 Funciones y responsabilidades de un DBA.....</b>                      | <b>12</b>  |
| <b>1.7 Arquitectura ANSI/SPARC.....</b>                                      | <b>13</b>  |
| <b>1.8 Modelos de datos.....</b>   | <b>17</b>  |
| 1.8.1 Modelo de datos orientado a objetos.....                               | 17         |
| 1.8.2 Modelos de datos basados en registros.....                             | 19         |
| <b>1.9 Resumen.....</b>  | <b>23</b>  |
| <b>1.10 Contenido de la página Web de apoyo.....</b>                         | <b>23</b>  |
| 1.10.1 Mapa conceptual del capítulo.....                                     | 23         |
| 1.10.2 Autoevaluación.....   | 23         |
| 1.10.3 Presentaciones.....   | 23         |
| <b>Capítulo 2</b>  |            |
| <b>Modelo de datos relacional.....</b>                                       | <b>25</b>  |
| <b>2.1 Introducción.....</b>   | <b>26</b>  |
| 2.1.1 Conceptos básicos en el modelo relacional .....                        | 27         |
| 2.1.2 Propiedades de las relaciones.....                                     | 29         |
| 2.1.3 Reglas de Codd.....  | 29         |
| 2.1.4 Atributos claves.....  | 31         |
| 2.1.5 Condiciones de las claves candidatas.....                              | 31         |
| 2.1.6 Claves primarias.....  | 32         |
| 2.1.7 Clave foránea.....   | 33         |
| 2.1.8 Marcador de valores desconocidos.....                                  | 34         |
| 2.1.9 Correspondencias.....  | 35         |
| 2.1.10 Reglas de integridad.....   | 36         |
| 2.1.11 Diccionario de datos.....   | 37         |
| 2.1.12 Diseño de software utilizando bases de datos relacionales.....        | 38         |
| 2.1.13 El acceso a las bases de datos con aplicaciones escritas en JAVA..... | 39         |
| 2.1.14 Acceso a las bases de datos a través de servicios Web.....            | 40         |
| <b>2.2 Álgebra relacional.....</b>   | <b>41</b>  |
| 2.2.1 Operaciones de conjuntos.....  | 44         |
| 2.2.2 Operaciones relacionales.....  | 48         |
| 2.2.3 Reunión.....   | 50         |
| 2.2.4 División.....  | 54         |
| 2.2.5 Renombrar.....   | 55         |
| 2.2.6 Funciones.....   | 56         |
| 2.2.7 Agrupación.....  | 57         |
| 2.2.8 Relaciones temporales.....   | 58         |
| 2.2.9 Cálculo relacional.....  | 61         |
| <b>2.3 Normalización.....</b>  | <b>63</b>  |
| 2.3.1 Fundamentos de la normalización.....                                   | 63         |
| 2.3.2 A qué se refiere la normalización.....                                 | 64         |
| 2.3.3 A qué no se refiere la normalización.....                              | 65         |
| 2.3.4 Normalización aspecto formal.....                                      | 66         |
| 2.3.5 Concepto de normalizar.....  | 66         |
| 2.3.6 Objetivos de la normalización.....                                     | 67         |
| <b>2.4 Origen de los datos.....</b>  | <b>67</b>  |
| 2.4.1 Los datos.....   | 67         |
| 2.4.2 Problemática asociada al origen de datos.....                          | 68         |
| 2.4.3 Los resultados.....  | 69         |
| <b>2.5 Las formas normales.....</b>  | <b>69</b>  |
| 2.5.1 las normas.....  | 69         |
| 2.5.2 Dominio (extensión conceptual).....                                    | 70         |
| 2.5.3 Fundamento teórico de las normas.....                                  | 70         |
| 2.5.4 Enumeración de las normas.....   | 72         |
| 2.5.5 Interpretación y aplicación de las formas normales                     | 73         |
| <b>2.6 Las estructuras.....</b>  | <b>92</b>  |
| 2.6.1 El modelo y sus estructuras.....                                       | 92         |
| <b>2.7 Un caso de estudio.....</b>   | <b>99</b>  |
| <b>2.8 Resumen.....</b>  | <b>101</b> |
| <b>2.9 Contenido de la página Web de apoyo.....</b>                          | <b>101</b> |
| 2.9.1 Mapa conceptual del capítulo.....                                      | 101        |
| 2.9.2 Autoevaluación.....  | 101        |
| 2.9.3 Presentaciones*.....   | 101        |

|   |            |
|---|------------|
| <b>Capítulo 3</b>   |            |
| <b>SQL.....</b>   | <b>103</b> |
| <b>3.1 Introducción.....</b>  | <b>104</b> |
| <b>3.2 Algo de historia del lenguaje SQL.....</b>                             | <b>104</b> |
| <b>3.3 Características del lenguaje SQL.....</b>                              | <b>105</b> |
| <b>3.4 El lenguaje SQL y su sublenguaje de definición de datos o DDL.....</b> | <b>106</b> |
| <b>3.5 Sentencias del sublenguaje DML.....</b>                                | <b>112</b> |
| <b>3.5.1 Funciones de fila simple.....</b>                                    | <b>113</b> |
| <b>3.5.2 Funciones numéricas.....</b>   | <b>113</b> |
| <b>3.5.3 Funciones generales de comparación.....</b>                          | <b>115</b> |
| <b>3.5.4 Funciones de conversión.....</b>                                     | <b>115</b> |
| <b>3.5.5 Funciones de grupo.....</b>  | <b>115</b> |
| <b>3.5.6 La cláusula FROM.....</b>  | <b>118</b> |
| <b>3.5.7 La cláusula WHERE.....</b>   | <b>121</b> |
| <b>3.5.8 La cláusula GROUP BY.....</b>  | <b>122</b> |
| <b>3.5.9 La cláusula HAVING.....</b>  | <b>122</b> |
| <b>3.5.10 La cláusula ORDER BY.....</b>                                       | <b>123</b> |
| <b>3.6 Sentencias del sublenguaje DML. INSERT, UPDATE, DELETE.....</b>        | <b>123</b> |
| <b>3.6.1 Sentencia INSERT.....</b>  | <b>123</b> |
| <b>3.6.2 Sentencia UPDATE.....</b>  | <b>124</b> |
| <b>3.6.3 Sentencia MERGE.....</b>   | <b>124</b> |
| <b>3.6.4 Sentencia DELETE.....</b>  | <b>125</b> |
| <b>3.7 Sentencias del sublenguaje TCL de control de transacciones.....</b>    | <b>125</b> |
| <b>3.8 Procesamiento de consultas.....</b>                                    | <b>127</b> |
| <b>3.8.1 Plan de consultas.....</b>   | <b>128</b> |
| <b>3.8.2 Optimización de consultas.....</b>                                   | <b>129</b> |
| <b>3.9 Resumen.....</b>   | <b>129</b> |
| <b>3.10 Contenido de la página Web de apoyo.....</b>                          | <b>129</b> |
| <b>3.10.1 Mapa conceptual del capítulo.....</b>                               | <b>129</b> |
| <b>3.10.2 Autoevaluación.....</b>   | <b>129</b> |
| <b>3.10.3 Presentaciones*.....</b>  | <b>129</b> |
| <b>Capítulo 4</b>   |            |
| <b>Lenguaje procedimental como extensión de SQL.....</b>                      | <b>131</b> |
| <b>4.1 Introducción.....</b>  | <b>132</b> |
| <b>4.2 Vista general de PL/SQL.....</b>                                       | <b>132</b> |
| <b>4.3 Uso del PL/SQL para acceder a la base de datos Oracle.....</b>         | <b>132</b> |
| <b>4.4 Programas con PL/SQL.....</b>  | <b>133</b> |
| <b>4.4.1 Modularidad.....</b>   | <b>133</b> |
| <b>4.4.2 Procedimientos, funciones, triggers y paquetes ....</b>              | <b>133</b> |
| <b>4.5 Componentes de un bloque PL/SQL.....</b>                               | <b>134</b> |
| <b>4.5.1 Las construcciones lógicas y de control de bucle ....</b>            | <b>135</b> |
| <b>4.6 Cursor.....</b>  | <b>135</b> |
| <b>4.7 Errores.....</b>   | <b>136</b> |
| <b>4.8 El desarrollo de un bloque PL/SQL.....</b>                             | <b>136</b> |
| <b>4.8.1 Los tipos de datos de la base de datos.....</b>                      | <b>136</b> |
| <b>4.8.2 Tipos de datos que son propios del lenguaje PL/SQL.....</b>          | <b>137</b> |
| <b>4.9 Interacción con la base de datos Oracle.....</b>                       | <b>138</b> |
| <b>4.10 El tratamiento de transacciones en PL/SQL.....</b>                    | <b>139</b> |
| <b>4.11 Sentencias de control de bucle.....</b>                               | <b>141</b> |
| <b>4.11.1 Usando loops.....</b>   | <b>142</b> |
| <b>4.12 Manejo de cursor.....</b>   | <b>144</b> |
| <b>4.13 Manejo de errores.....</b>  | <b>148</b> |
| <b>4.13.1 Excepciones comunes.....</b>  | <b>150</b> |
| <b>4.13.2 Codificando en la sección de excepciones . . .</b>                  | <b>150</b> |
| <b>4.14 Construyendo procedimientos y funciones con PL/SQL.....</b>           | <b>151</b> |
| <b>4.14.1 Creando paquetes con PL/SQL.....</b>                                | <b>155</b> |
| <b>4.14.2 Creando triggers con PL/SQL.....</b>                                | <b>157</b> |
| <b>4.15 Resumen.....</b>  | <b>159</b> |
| <b>4.16 Contenido de la página Web de apoyo.....</b>                          | <b>159</b> |
| <b>4.16.1 Mapa conceptual del capítulo.....</b>                               | <b>159</b> |
| <b>4.16.2 Autoevaluación.....</b>   | <b>159</b> |
| <b>4.16.3 Presentaciones.....</b>   | <b>159</b> |
| <b>Capítulo 5</b>   |            |
| <b>Bases de datos multidimensionales y tecnologías OLAP ...</b>               | <b>161</b> |
| <b>5.1 Introducción.....</b>  | <b>162</b> |
| <b>5.2 Bases de datos multidimensionales.....</b>                             | <b>162</b> |
| <b>5.2.1 Evolución de las bases de datos.....</b>                             | <b>162</b> |
| <b>5.2.2 Concepto.....</b>  | <b>164</b> |
| <b>5.2.3 Estructura de almacenamiento.....</b>                                | <b>166</b> |
| <b>5.2.4 Dispersión de datos.....</b>   | <b>167</b> |
| <b>5.3 Tecnologías OLAP.....</b>  | <b>169</b> |
| <b>5.3.1 Introducción.....</b>  | <b>169</b> |
| <b>5.3.2 Concepto de OLAP.....</b>  | <b>169</b> |
| <b>5.3.3 Características de OLAP.....</b>                                     | <b>169</b> |
| <b>5.3.4 Comparación entre el modelo OLTP y el modelo OLAP.....</b>           | <b>170</b> |
| <b>5.4 Integración entre bases de datos y herramientas OLAP .....</b>         | <b>173</b> |
| <b>5.5 Arquitecturas OLAP y OLTP.....</b>                                     | <b>174</b> |
| <b>5.6 OLAP: multidimensional contra relacional.....</b>                      | <b>175</b> |
| <b>5.6.1 OLAP multidimensional (MOLAP).....</b>                               | <b>177</b> |
| <b>5.6.2 OLAP relacional (ROLAP).....</b>                                     | <b>178</b> |
| <b>5.6.3 OLAP híbrido (HOLAP).....</b>  | <b>182</b> |
| <b>5.7 Evaluación de servidores y herramientas OLAP.....</b>                  | <b>182</b> |
| <b>5.7.1 Características y funciones.....</b>                                 | <b>182</b> |

|   |            |
|---|------------|
| 5.7.2 Motores de servicios OLAP.....  | 182        |
| 5.7.3 Administración.....   | 183        |
| 5.7.4 Arquitectura global.....  | 183        |
| <b>5.8 Desarrollo de aplicaciones OLAP.....</b>                                       | <b>184</b> |
| <b>5.9 Áreas de aplicación de las tecnologías OLAP.....</b>                           | <b>185</b> |
| 5.9.1 Análisis de ventas.....   | 185        |
| 5.9.2 Gestión de informes financieros.....  | 186        |
| <b>5.10 Ventajas y desventajas de OLAP.....</b>                                       | <b>186</b> |
| <b>5.11 Resumen.....</b>  | <b>187</b> |
| <b>5.12 Contenido de la página Web de apoyo.....</b>                                  | <b>188</b> |
| 5.12.1 Mapa conceptual del capítulo.....  | 188        |
| 5.12.2 Autoevaluación.....  | 188        |
| 5.12.3 Presentaciones.....  | 188        |
| <b>Capítulo 6</b>   |            |
| <b>Almacén de datos.....</b>  | <b>189</b> |
| <b>6.1 Introducción.....</b>  | <b>190</b> |
| <b>6.2 Inteligencia de negocio.....</b>   | <b>191</b> |
| 6.2.1 Datos, Información y Conocimiento.....  | 192        |
| <b>6.3 Sistemas de almacén de datos.....</b>  | <b>193</b> |
| <b>6.4 Concepto de almacén de datos.....</b>  | <b>194</b> |
| <b>6.5 Características de un almacén de datos.....</b>                                | <b>194</b> |
| <b>6.6 Arquitectura del almacén de datos.....</b>                                     | <b>196</b> |
| 6.6.1 Bases de datos fuentes.....   | 196        |
| 6.6.2 Base de datos con datos resumidos.....  | 197        |
| 6.6.3 Interfaces orientadas al usuario.....   | 197        |
| <b>6.7 Funcionalidades y objetivo.....</b>  | <b>197</b> |
| 6.7.1 Acceso a fuentes.....   | 198        |
| 6.7.2 Carga.....  | 198        |
| 6.7.3 Almacenamiento.....   | 199        |
| 6.7.4 Consultas.....  | 199        |
| 6.7.5 Metadatos.....  | 199        |
| <b>6.8 Almacén de datos y Data Mart.....</b>  | <b>200</b> |
| <b>6.9 La integridad de los datos.....</b>  | <b>202</b> |
| 6.9.1 Concepto de Integridad.....   | 203        |
| 6.9.2 La perspectiva del usuario final.....   | 203        |
| 6.9.3 La perspectiva del Sistema de Información.....                                  | 203        |
| 6.9.4 Controles de integridad de datos.....   | 203        |
| <b>6.10 Costos y valor del almacén de datos.....</b>                                  | <b>208</b> |
| 6.10.1 Costos de un almacén de datos.....   | 208        |
| 6.10.2 Valor del almacén de datos.....  | 210        |
| 6.10.3 Balance entre los costos y el valor.....                                       | 210        |
| <b>6.11 Impactos de la implementación de un almacén de datos.....</b>                 | <b>210</b> |
| 6.11.1 Recursos Humanos.....  | 211        |
| 6.11.2 Impactos organizacionales.....   | 211        |
| 6.11.3 Impactos técnicos del almacén de datos.....                                    | 212        |
| 6.11.4 Consideraciones finales.....   | 212        |
| <b>6.12 Estrategia recomendada para la implementación de un almacén de datos.....</b> | <b>213</b> |
| 6.12.1 Prototipo.....   | 213        |
| 6.12.2 Piloto.....  | 213        |
| 6.12.3 Prueba del concepto tecnológico.....   | 214        |
| 6.12.4 Arquitectura de un almacén de datos.....                                       | 214        |
| 6.12.5 Acceso a datos de usuario finales.....   | 215        |
| 6.12.6 Factores de riesgo.....  | 215        |
| <b>6.13 Resumen.....</b>  | <b>216</b> |
| <b>6.14 Contenido de la página Web de apoyo.....</b>                                  | <b>216</b> |
| 6.14.1 Mapa conceptual del capítulo.....  | 216        |
| 6.14.2 Autoevaluación.....  | 216        |
| 6.14.3 Presentaciones*.....   | 216        |
| <b>Capítulo 7</b>   |            |
| <b>Minería de datos.....</b>  | <b>217</b> |
| <b>7.1 Introducción.....</b>  | <b>218</b> |
| <b>7.2 Concepto.....</b>  | <b>218</b> |
| <b>7.3 Características de la minería de datos.....</b>                                | <b>219</b> |
| <b>7.4 Capacidades de la minería de datos.....</b>                                    | <b>220</b> |
| <b>7.5 Herramientas algorítmicas de la minería de datos.....</b>                      | <b>221</b> |
| 7.5.1 Redes neuronales artificiales.....  | 222        |
| 7.5.2 Algoritmos Genéticos.....   | 224        |
| 7.5.3 Árboles de decisión.....  | 226        |
| <b>7.6 Modelado de la minería de datos.....</b>                                       | <b>228</b> |
| <b>7.7 Integración entre almacén de datos y minería de datos.....</b>                 | <b>229</b> |
| <b>7.8 Ventajas de la minería de datos.....</b>                                       | <b>230</b> |
| <b>7.9 Diferencias entre el análisis estadístico y la minería de datos.....</b>       | <b>230</b> |
| <b>7.10 Aplicaciones de la minería de datos.....</b>                                  | <b>232</b> |
| <b>7.11 Resumen.....</b>  | <b>233</b> |
| <b>7.12 Contenido de la página Web de apoyo.....</b>                                  | <b>233</b> |
| 7.12.1 Mapa conceptual del capítulo.....  | 233        |
| 7.12.2 Autoevaluación.....  | 233        |
| 7.12.3 Presentaciones.....  | 233        |
| <b>Capítulo 8</b>   |            |
| <b>Bases de Datos Orientadas a Objetos.....</b>                                       | <b>235</b> |
| <b>8.1 Introducción.....</b>  | <b>236</b> |
| <b>8.2 Historia y origen de las BDOO.....</b>   | <b>237</b> |
| <b>8.3 Conceptos fundamentales.....</b>   | <b>238</b> |
| <b>8.4 Bases de la orientación a objetos.....</b>                                     | <b>239</b> |

|  |            |   |            |
|--|------------|---|------------|
| <b>8.5 Características de las Bases de Datos Orientadas a Objetos.....</b>               | <b>240</b> | <b>9.4 Contenido de la página Web de apoyo.....</b> | <b>296</b> |
| 8.5.1 Modelo conceptual.....   | 240        | 9.4.1 Mapa conceptual del capítulo.....             | 296        |
| 8.5.2 Modelo de datos orientado a objetos.....   | 241        | 9.4.2 Autoevaluación.....                           | 296        |
| 8.5.3 Persistencia de los datos.....   | 241        | 9.4.3 Presentaciones.....                           | 296        |
| 8.5.4 Almacenamiento y acceso de los objetos persistentes en una BDOO.....               | 242        |   |            |
| 8.5.5 Manifesto de Atkinson.....   | 242        |   |            |
| 8.5.6 Intento de estandarización ODMG.....   | 244        |   |            |
| 8.5.7 Enfoques para la construcción de BDOO.....   | 251        |   |            |
| <b>8.6 Sistema de gestión de BDOO (SGBDOO).....</b>                                      | <b>252</b> |   |            |
| 8.6.1 Concepto de un SGBDOO.....   | 252        |   |            |
| 8.6.2 Objetivo.....  | 253        |   |            |
| 8.6.3 Características de los SGBDOO.....   | 253        |   |            |
| 8.6.4 Estructura de un SGBDOO.....   | 255        |   |            |
| <b>8.7 Rendimiento de las BDOO.....</b>  | <b>256</b> |   |            |
| <b>8.8 Ventajas de las BDOO.....</b>   | <b>256</b> |   |            |
| <b>8.9 Desventajas de las BDOO.....</b>  | <b>257</b> |   |            |
| <b>8.10 Bases de Datos Objeto-Relacionales.....</b>                                      | <b>258</b> |   |            |
| 8.10.1 Concepto.....   | 258        |   |            |
| 8.10.2 Características de las BDOR.....  | 258        |   |            |
| 8.10.3 Implementación en Oracle.....   | 259        |   |            |
| <b>8.11 Resumen.....</b>   | <b>273</b> |   |            |
| <b>8.12 Contenido de la página Web de apoyo.....</b>                                     | <b>273</b> |   |            |
| 8.12.1 Mapa conceptual del capítulo.....   | 273        |   |            |
| 8.12.2 Autoevaluación.....   | 273        |   |            |
| 8.12.3 Presentaciones.....   | 273        |   |            |
| <b>Capítulo 9</b>  |            |   |            |
| <b>Implementando el modelo en Oracle Express Edition XE ..</b>                           | <b>275</b> |   |            |
| <b>9.1. Introducción.....</b>  | <b>276</b> |   |            |
| 9.1.1 Obteniendo el software e inicializando la instalación del Motor Oracle 10g XE..... | 276        |   |            |
| 9.1.2. Descarga.....   | 276        |   |            |
| 9.1.3 Instalación.....   | 277        |   |            |
| <b>9.2 Creando y probando nuestro modelo de Estudiante – Universidad.....</b>            | <b>283</b> |   |            |
| <b>9.3 Creación de las tablas y sus relaciones.....</b>                                  | <b>285</b> |   |            |
| 9.3.1 Ejecución de script.....   | 285        |   |            |
| 9.3.2 Modelo de datos.....   | 285        |   |            |
| 9.3.3 Script (sentencias DDL).....   | 286        |   |            |
| 9.3.4 Datos de prueba.....   | 290        |   |            |
| 9.3.5 Función escalar definida por el usuario.....                                       | 292        |   |            |
| 9.3.6 Procedimiento almacenado definido por el usuario.....                              | 293        |   |            |
| 9.3.7 Ejemplos de ejecución.....   | 295        |   |            |
| <b>Capítulo 10</b>   |            |   |            |
| <b>Implementando el modelo en IBM DB2.....</b>   | <b>297</b> |   |            |
| <b>10.1 Introducción.....</b>  | <b>298</b> |   |            |
| <b>10.2 Instalando e inicializando el servidor.....</b>                                  | <b>298</b> |   |            |
| 10.2.1 Download.....   | 298        |   |            |
| 10.2.2 Instalación.....  | 298        |   |            |
| <b>10.3 Creando y probando nuestro modelo de Estudiante – Universidad.....</b>           | <b>299</b> |   |            |
| 10.3.1 Creación de la base de datos.....   | 299        |   |            |
| 10.3.2 Creación de las tablas y sus relaciones.....                                      | 303        |   |            |
| 10.3.3 Modelo de datos.....  | 304        |   |            |
| 10.3.4 Datos de prueba.....  | 308        |   |            |
| <b>10.4 Contenido de la página Web de apoyo.....</b>                                     | <b>310</b> |   |            |
| 10.4.1 Mapa conceptual del capítulo.....   | 310        |   |            |
| 10.4.2 Autoevaluación.....   | 310        |   |            |
| 10.4.3 Presentaciones.....   | 310        |   |            |
| <b>Capítulo 11</b>   |            |   |            |
| <b>Implementando el modelo en SQL Server 2005.....</b>                                   | <b>311</b> |   |            |
| <b>11.1 Introducción.....</b>  | <b>312</b> |   |            |
| <b>11.2 Inicializando el servidor.....</b>   | <b>312</b> |   |            |
| 11.2.1 Download.....   | 312        |   |            |
| 11.2.2 Instalación.....  | 312        |   |            |
| 11.2.3 Inicialización.....   | 316        |   |            |
| <b>11.3 Inicializando el cliente.....</b>  | <b>317</b> |   |            |
| 11.3.1 Download.....   | 317        |   |            |
| 11.3.2 Instalación.....  | 317        |   |            |
| 11.3.3 Utilizando el SSMS.....   | 317        |   |            |
| <b>11.4 Creando y probando nuestro modelo de Estudiante–Universidad.....</b>             | <b>318</b> |   |            |
| 11.4.1 Creación de la base de datos.....   | 319        |   |            |
| 11.4.2 Creación de las tablas y sus relaciones.....                                      | 320        |   |            |
| 11.4.3 Programación de la base de datos.....   | 327        |   |            |
| <b>11.5 Contenido de la página Web de apoyo.....</b>                                     | <b>334</b> |   |            |
| 11.5.1 Mapa conceptual del capítulo.....   | 334        |   |            |
| 11.5.2 Autoevaluación.....   | 334        |   |            |
| 11.5.3 Presentaciones.....   | 334        |   |            |
| <b>Capítulo 12</b>   |            |   |            |
| <b>Implementando el modelo en MySQL 5.1.....</b>   | <b>335</b> |   |            |
| <b>12.1 Introducción.....</b>  | <b>336</b> |   |            |

|  |            |  |            |
|--|------------|--|------------|
| <b>12.2 Inicializando el servidor.....</b>                                       | <b>336</b> | 12.4.1 Creación de una conexión al servidor.....                         | 344        |
| 12.2.1 Download.....   | 336        | 12.4.2 Creación de la base de datos, las tablas<br>y sus relaciones..... | 345        |
| 12.2.2 Instalación.....  | 337        |  |            |
| 12.2.3 Inicialización.....   | 341        |  |            |
| <b>12.3 Inicializando el cliente.....</b>  | <b>342</b> | <b>12.5 Contenido de la página Web de apoyo.....</b>                     | <b>358</b> |
| 12.3.1 Download.....   | 342        | 12.5.1 Mapa conceptual del capítulo.....                                 | 358        |
| 12.3.2 Instalación.....  | 342        | 12.5.2 Autoevaluación.....   | 358        |
| 12.3.3 Utilizando el MySQL Workbench.....  | 343        | 12.5.3 Presentaciones.....   | 358        |
| <b>12.4 Creando y probando nuestro modelo<br/>de Estudiante–Universidad.....</b> | <b>343</b> | <b>Bibliografía.....</b>   | <b>359</b> |
|  |            | <b>Indice analítico.....</b>   | <b>361</b> |

## Información del contenido de la página Web

El material marcado con asterisco (\*) solo está disponible para docentes.

### Capítulo 1.

#### Estructura y tipos de bases de datos

- Mapa conceptual
- Autoevaluación
- Presentaciones\*
- Vínculos de interés

### Capítulo 2.

#### Modelo de datos relacional

- Mapa conceptual
- Autoevaluación
- Presentaciones\*

### Capítulo 3.

#### SQL

- Mapa conceptual
- Autoevaluación
- Presentaciones\*

### Capítulo 4.

#### Lenguaje procedimental como extensión de SQL

- Mapa conceptual
- Autoevaluación
- Presentaciones\*

### Capítulo 5.

#### Bases de datos multidimensionales y tecnologías OLAP

- Mapa conceptual
- Autoevaluación
- Presentaciones\*

### Capítulo 6.

#### Almacén de datos

- Mapa conceptual
- Autoevaluación
- Presentaciones\*

### Capítulo 7.

#### Minería de datos

- Mapa conceptual
- Autoevaluación
- Presentaciones\*

### Capítulo 8.

#### Bases de datos orientadas a objetos

- Mapa conceptual
- Autoevaluación
- Presentaciones\*

### Capítulo 9.

#### Implementando el modelo en Oracle Express Edition XE

- Mapa conceptual
- Autoevaluación
- Presentaciones\*

### Capítulo 10.

#### Implementando el modelo en IBM DB2

- Mapa conceptual
- Autoevaluación
- Presentaciones\*

**Capítulo 11.****Implementando el modelo en SQL Server 2005**

- **Mapa conceptual**
- **Autoevaluación**
- **Presentaciones\***

- **Lecturas complementarias**
- **Código fuente de los ejemplos**
- **Vínculos de interés**

Incluye hipervínculos a la descarga de las versiones gratuitas de MySQL 5.1, SQL Server 2005, Oracle Express Edition XE e IBM DB2.

**Capítulo 12.****Implementando el modelo en MySQL 5.1**

- **Mapa conceptual**
- **Autoevaluación**
- **Presentaciones\***

**Registro en la Web de apoyo**

Para tener acceso al material de la página Web de apoyo del libro:

1. Ir a la página <http://virtual.alfaomega.com.mx>
2. Registrarse como usuario del sitio y propietario del libro.
3. Ingresar al apartado de inscripción de libros y registrar la siguiente clave de acceso

4. Para navegar en la plataforma virtual de recursos del libro, usar los nombres de Usuario y Contraseña definidos en el punto número dos. El acceso a estos recursos es limitado. Si quiere un número extra de accesos envíe un correo electrónico a [webmaster@alfaomega.com.mx](mailto:webmaster@alfaomega.com.mx)

Estimado profesor: Si desea acceder a los contenidos exclusivos para docentes por favor contacte al representante de la editorial que lo suele visitar o envíenos un correo electrónico a [webmaster@alfaomega.com.mx](mailto:webmaster@alfaomega.com.mx)

**Convenciones utilizadas en el texto**

|   |  |
|---|--|
|  | Conceptos para recordar: bajo este ícono se encuentran definiciones importantes que refuerzan lo explicado en la página. |
|  | Comentarios o información extra: este ícono ayuda a comprender mejor o ampliar el texto principal.                       |
|  | Contenidos interactivos: indica la presencia de contenidos extra en la Web.  |

## Prólogo

En los últimos 20 años, el manejo de Base de Datos y el conocimiento de los Motores de Base de Datos existentes en el mercado han tomado un protagonismo absoluto en el desarrollo de software. Si bien este tema siempre se analizó dentro del área informática, antes solo era considerado para el desarrollo de sistemas de mediana y gran envergadura, pero actualmente no se justifica la construcción de una página Web sin tener como respaldo una base de datos para manejar la persistencia de los datos, aunque sean mínimos.

Por esta razón, el objetivo de este libro es introducir al lector en este tema mediante el concepto de composición de un Motor de Base de Datos, la forma de interactuar con él a través de los lenguajes asociados y los diferentes modelos de construcción lógica de datos.

En el Capítulo 1, se desarrollan los conceptos generales de las bases de datos y los Sistemas de Gestión de las mismas. Además, analiza conceptos como la seguridad de las mismas, la administración de usuarios y las funciones y las responsabilidades de un DBA (por sus siglas en inglés, *Data Base Administrator* ).

En el Capítulo 2, se especifican los conceptos fundamentales del modelo relacional, trabajando el concepto de relaciones entre tablas, claves, integración, etcétera.

En el Capítulo 3, se especifican los comandos y la utilización del lenguaje de Consulta Estructurado (SQL), de forma tal que el lector conozca la forma de interactuar con un Sistema de Gestión de Base de Datos.

El Capítulo 4 extiende lo tratado en el capítulo anterior, pero también agrega el concepto de los lenguajes procedimentales considerando una extensión de SQL (PL/SQL) para poder agregar objetos creados por el diseñador a una base de datos específica.

En el Capítulo 5, se especifican las diferencias implementadas por las bases de datos multidimensionales y las distintas variantes de modelado aplicadas por las tecnologías OLAP.

Los Capítulos 6 y 7 establecen una aproximación a dos temas vinculados con las bases de datos como son el almacenamiento y la minería de datos. Si bien el objetivo del libro no es profundizar estos conceptos, se brinda una visión general de los mismos para conocimiento del lector.

En el Capítulo 8, se desarrolla el concepto de objetos aplicado a la persistencia de los datos administradas por un Sistema de Gestión. Además, se tratan las extensiones de bases de datos consideradas orientadas a objetos y las Bases de datos objeto-relacionales.

Por último, para poder trabajar sobre la práctica real, se desarrolló un modelo de datos que se implementa en Oracle Express Edition XE, IBM DB2, SQL Server 2005 y MySQL 5.1, en los Capítulos 9, 10, 11 y 12 respectivamente.



# 1

## Estructura y tipos de bases de datos

### Contenido

|  |    |
|--|----|
| 1.1 Introducción.....                            | 2  |
| 1.2 Definición de base de datos.....             | 3  |
| 1.3 Sistema de Gestión de Bases de Datos.....    | 4  |
| 1.4 Usuarios de la base de datos.....            | 5  |
| 1.5 Seguridad de las bases de datos.....         | 7  |
| 1.6 Funciones y responsabilidades de un DBA..... | 12 |
| 1.7 Arquitectura ANSI/SPARC.....                 | 13 |
| 1.8 Modelos de datos.....                        | 17 |
| 1.9 Resumen.....                                 | 23 |
| 1.10 Contenido de la página Web de apoyo.....    | 23 |

### Objetivos

- Conocer la teoría de bases de datos.
- Identificar los modelos de datos anteriores y actuales para el almacenamiento persistente de grandes volúmenes de datos.
- Distinguir los tipos de usuarios y las funciones de un administrador de bases de datos.

## 1.1 Introducción



Entre 1960 y 1970, las computadoras tenían capacidades limitadas de procesamiento y almacenamiento.



Los datos que era necesario almacenar se ubicaban en archivos por orden físico.

Entre 1960 y 1970, las computadoras tenían capacidades limitadas de procesamiento y de almacenamiento. Si bien las aplicaciones no eran tan exigentes como las actuales, había que trabajar demasiado para que se lograran los resultados esperados.

Los datos, que era necesario almacenar, se ubicaban en archivos por un orden físico, cuya determinación se efectuaba de acuerdo con su orden de ingreso o por otro orden previamente indicado. Esto generaba mucho trabajo, ya que algún símbolo separaba los campos que conformaban los registros que, luego, se debían detectar para saber dónde finalizaba uno y empezaba el otro.

La lectura secuencial de los archivos era muy común y se debían recorrer todos los registros para encontrar el deseado; esta situación acarreaba problemas en la velocidad de respuesta.

Muchas veces, los distintos sectores de una empresa contaban con sus propios datos para resolver los problemas de información que se presentaban. Como no había ni políticas ni reglas claras, tampoco había personal informático que centralizara este tipo de situaciones, entonces, se repetían los datos.

Por esta razón, se necesitaba un esfuerzo mayor que el actual en la programación de aplicaciones para acceder a los datos, puesto que el almacenamiento se centraba en pocos tipos de datos, por ejemplo, numéricos y de texto. Esto limitaba las capacidades de respuesta a las necesidades de las empresas y exigía algoritmos de programación para un sinfín de situaciones, como el control y el almacenamiento de una determinada fecha.

En la actualidad, todas las personas interactúan con datos que, de alguna manera, se almacenan en un medio físico y se asocian a un sistema informático que los registra y permite su acceso. Por ejemplo: si desde su casa o desde un teléfono móvil, una persona marca un número de teléfono, lo que se hace es utilizar los datos registrados dentro de una empresa telefónica, que reconoce el teléfono, el origen de la llamada y el número del receptor. Eso implica una serie de acciones que desencadenan controles y registros en una base de datos. Con respecto al control, se deduce que éste debe verificar la existencia del número receptor o que, por ejemplo, no se encuentre ocupado; de lo contrario, es una llamada equivocada. Una vez establecida la comunicación, el sistema informático registrará mínimamente su número, el número destino, la hora de inicio y final de la llamada, al momento de dar por finalizada la comunicación. Dichos registros son los que permiten obtener la facturación de los clientes y de todos los reportes que se soliciten.

Lo mismo sucede cuando el cliente de un banco interactúa con el cajero automático para retirar dinero: el cliente debe ingresar su tarjeta, que se leerá para, luego, cruzar la identificación de la tarjeta con la clave que el usuario ingresa. La base de datos mantiene los datos del cliente y sus cuentas asociadas, mientras que una aplicación informática hará controles y mostrará las opciones disponibles al usuario.

En alguna sección de un sistema de gestión de alumnos de una universidad, el docente publica el programa de estudios de su materia, las fechas de los exámenes parciales y las notas de los estudiantes de su curso. También, publica los mensajes y los archivos para sus alumnos. El personal de la universidad que carga los resultados de

los exámenes indica las notas de los alumnos presentes y registra los ausentes. Otros usuarios acceden al sistema para cargar asistencias e inasistencias a clase de los alumnos. Por su parte, los alumnos ingresan con su legajo y su clave a la autogestión que presenta el sistema —por intranet o por Internet—, para ver la información de su estado académico, conformado por las materias que cursaron y las que cursarán, los resultados de sus exámenes —parciales y finales—, la cantidad de faltas que poseen e, incluso, se pueden inscribir en los exámenes de las materias que ya regularizaron.

Entonces, ¿qué es una base de datos? Algunos podrían pensar que son solo datos almacenados en una computadora como, por ejemplo, una planilla de cálculos. Sin embargo, es mucho más que eso.

## 1.2 Defnición de base de datos

Cuando una empresa decide la implementación de un sistema informático, para cubrir necesidades de información específicas, el equipo de análisis y diseño inicia el proceso de desarrollo del software, basándose en las especificaciones con las que trabaja el cliente que solicita el sistema para, de esta manera, responder a las necesidades de la empresa.

Durante este proceso, se definen los archivos, como el de “Estudiantes”, para almacenar los datos de todos los alumnos de la universidad; el de los “Profesores”, para almacenar los datos de contacto necesarios; el de las “Carreras” que propone la universidad; y el de las materias que conforman cada plan de estudios vigente, o no, de cada carrera, etcétera.

Pero esos archivos nunca quedan aislados de otros datos; es decir: mantienen relaciones que, normalmente, son transparentes para los usuarios y que permiten su aprovechamiento para obtener información, a través de aplicaciones programadas para tales fines. Por ejemplo, es necesario tener algún archivo que mantenga los datos de la inscripción de cada alumno —en las carreras y en las materias que se cursarán—: esto permitirá que se refejen los datos de resultados de los exámenes parciales, las asistencias, etcétera.

Por lo que hasta aquí se expresó, se puede ensayar una primera definición: “una base de datos es un conjunto de datos estructurados y definidos a través de un proceso específico, que busca evitar la redundancia, y que se almacenará en algún medio de almacenamiento masivo, como un disco”.

Cuando se hace referencia al término ‘redundancia’, se alude a la repetición que puede producirse en el momento de definir los almacenamientos de datos. Es incorrecta la pretensión de que los datos —como el apellido y la dirección— se encuentren definidos, a la vez, en los almacenamientos de estudiantes y en los relacionados con los exámenes finales. Un cambio en la dirección de un alumno podría significar que en uno de los dos almacenamientos no se realice la modificación y que provoque inconvenientes a la hora de devolver la información de un alumno.



Una base de datos es un conjunto de datos estructurados y definidos a través de un proceso específico, que busca evitar la redundancia, y que se almacenará en algún medio de almacenamiento masivo, como un disco.

### 1.3 Sistema de Gestión de Bases de Datos



Al SGBD se lo puede pensar como una capa de software que controla todos los accesos a la base de datos.

Cuando el alumno desea información de las notas que obtuvo durante el último año cursado, y que está almacenada en una base de datos, no accede directamente a ellos. Cada alumno de la universidad utiliza aplicaciones que ya han sido desarrolladas para fines específicos, como, por ejemplo, la obtención de un reporte de notas.

Por su parte, las aplicaciones que usa interactúan con un conjunto de programas aglutinados en lo que se denomina “el Sistema de Gestión de Bases de Datos (SGBD)”, “Database Management System” (DBMS) e, incluso, “Motor de Base de Datos”. A este Motor de Base de Datos, se lo puede pensar —de manera simplificada— como una capa de software que controla todos los accesos a la base de datos. Cabe aclarar, que un DBMS, no se crea para una situación específica de una empresa, sino que se desempeñará tanto para fines de un sistema de gestión de alumnos como para un sistema bancario, una empresa telefónica, comercial, etcétera.

El DBMS puede implementar instrucciones dadas por los distintos usuarios, que se describen en las próximas páginas, y que tienen distintos efectos en una base de datos. Las instrucciones se agrupan mínimamente en: DDL (Lenguaje de Definición de Datos) y DML (Lenguaje de Manipulación de Datos), aunque también suele reconocerse al DCL (Lenguaje de Control de Datos). A manera introductoria, se indican algunas instrucciones que conforman estos grupos:

**DDL:** es el conjunto de órdenes que permiten definir la estructura de una base de datos. Por ejemplo, para crear la base de datos en una empresa se utiliza una orden de este tipo, acompañada de los parámetros necesarios e indicados en cada DBMS. Lo mismo sucede si se desea modificar la estructura de un objeto de la base de datos, como por ejemplo cuando en un archivo de CLIENTES se desea agregar un dato nuevo no considerado al ser creada la estructura. Además incluye una instrucción para eliminar objetos obsoletos en la base de datos. No son instrucciones a incluir en las aplicaciones.

**DML:** las instrucciones que conforman este grupo son las que están incluidas en las aplicaciones y se usan para alterar el contenido de un archivo de datos. Por ejemplo, cuando se desea insertar un nuevo cliente o producto, modificar la dirección de un cliente o el precio de un producto, o eliminar un producto que fue cargado por error. Nótese la diferencia respecto al DDL que actuaba sobre la estructura para almacenar, mientras que estas órdenes afectan al contenido de los archivos de datos.

**DCL:** son órdenes que se utilizan para implementar seguridad en la base de datos, como por ejemplo indicar que privilegios —insertar, modificar, etc.— tiene cada usuario respecto a los distintos objetos de la base de datos. Incluso pueden ser retirados privilegios a usuarios existentes.

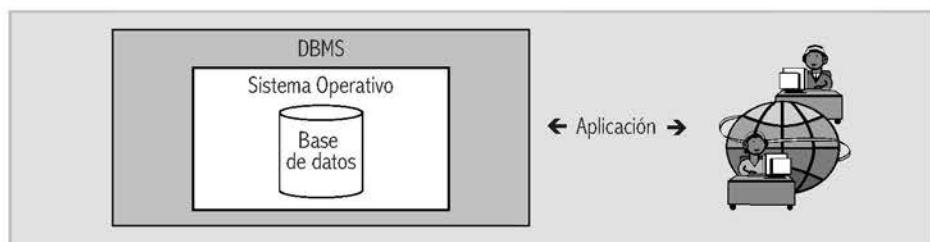


Fig. 1.1 Representación gráfica de una base de datos.

Si bien las aplicaciones poseen controles de seguridad, al solicitar un nombre de usuario y contraseña cuando el usuario ingresa al sistema, siempre hay una segunda instancia de seguridad definida por personal que implementa la base de datos, donde se definen al DBMS perfiles de usuarios y privilegios que posee respecto a los datos, por ejemplo si un usuario Carlos puede o no agregar nuevos datos, o modificar datos almacenados en la base de datos.

Teniendo en cuenta lo expresado, ahora podemos pensar en una definición que amplíe el concepto de bases de datos enunciado en el punto 1.2. Para ello, se utilizará una cita —muy utilizada por otros autores— que abarca las nociones hasta aquí enunciadas. James Martin en su obra de 1975 Computer Data-base Organization define base de datos como:

“Colección de datos interrelacionados almacenados en conjunto sin redundancias perjudiciales o innecesarias; su finalidad es servir a una aplicación o más, de la mejor manera posible; los datos se almacenan de modo que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluir nuevos datos y para modificar o extraer los datos almacenados”.

Esta definición incluye el concepto de independencia de datos, que significa que las aplicaciones deben ser independientes de las modificaciones que pudieran sufrir los datos, en lo lógico y en lo físico. Sería deseable que esto se lograra en las bases de datos, aunque, en algunas situaciones, resulte más o menos complejo.

Es decir que si, por ejemplo, en el almacenamiento “Estudiantes” de la base de datos se agrega un dato —como la cuenta de correo electrónico o el número de teléfono móvil—, no implica que en los programas que usan el almacenamiento se deba hacer cambios. Eso es lo que se denomina independencia lógica. En otras épocas, las aplicaciones incluían la definición de la estructura de datos y había una sección en la que figuraban todos los datos que poseía cada uno de los archivos de datos, a los que se accedía por dicho programa. Esto significaba que, ante un cambio en la estructura, el agregado de un nuevo dato o la alteración de la dimensión de un dato existente, las aplicaciones —que podían ser muchas— debían refejar el cambio en la sección correspondiente.

En cuanto a la independencia física, el almacenamiento “Estudiantes” puede definirse por la indexación, que es el acceso rápido por el legajo del estudiante. Si esta forma de acceso se cambiara para mejorar la performance del sistema, no debería determinar un cambio en las aplicaciones y, si se decidiera la utilización de otro almacenamiento físico para la base de datos, tampoco se deberían generar cambios en las mencionadas aplicaciones.

## 1.4 Usuarios de la base de datos

Hay diversas clasificaciones de los usuarios que actúan en un entorno de bases de datos, como las planteadas por grandes autores como C.J. Date en Sistemas de Bases de Datos, Korth en Fundamentos de Bases de Datos y Ramez Elmasri y Shamkant Navathe en Sistemas de Bases de Datos: conceptos fundamentales.

Se estudiarán distintos momentos de un sistema con bases de datos y se irán detectando los perfiles que, actualmente, se repiten en las empresas, a los que les asignaremos una denominación y, de esta manera, quedarán clasificados.

Cuando se desea informatizar un nuevo sistema de información, es imprescindible que participe un equipo de análisis y diseño de sistemas, conformado por ingenieros



Independencia de datos significa que las aplicaciones deben ser independientes de las modificaciones que pudieran sufrir los datos, en lo lógico y en lo físico.



Christopher J. Date. Investigador principal del modelo relacional de bases de datos de Edgar Codd.



En la Web de apoyo, encontrará el vínculo a la página Web personal de Shamkant B. Navathe.

y analistas de sistemas. Este equipo es el que toma los requerimientos del cliente mediante entrevistas, cuestionarios, etc., y plasma —en diagramas y descripciones— la interpretación de ese relevamiento. Luego entrará en un intenso proceso ingenieril, que incluirá el análisis, la creación y el diseño. El equipo planteará una propuesta al cliente —con los alcances y los límites bien determinados— para que la revise y mejore. Si se acepta la propuesta, se mejorará el nivel de detalle del diseño, se intercambiarán opiniones con los usuarios futuros con respecto a las interfaces y los reportes del sistema, para estipular cuáles serán las entidades que participarán en el desarrollo de las aplicaciones, cuáles serán las estructuras de cada una, quiénes serán los usuarios y sus respectivos roles, etcétera.



#### Usuarios de la base de datos:

1. Administrador de la base de datos.
2. Programador de aplicaciones.
3. Usuarios final.

De esta manera, se resume un proceso muy importante, que llevará días o meses de trabajo, según la complejidad del sistema; pero la intención es indicar que este equipo no interactúa necesariamente con el sistema en producción, sino que lo hace en la etapa previa a la construcción. Por esta razón, no se lo considera usuario del sistema.

Después de reconocer las entidades y relaciones entre ellas, se pueden detectar usuarios que interactúan con la base de datos de distinta manera y con distintos objetivos. La clasificación que se propone a continuación no es la única que suele expresarse, muchos autores indican otras con diferentes roles en la relación del usuario con la base de datos.

- Administrador de la base de datos: comúnmente es el profesional–ingeniero o analista— con perfil técnico que, en el ambiente informático, se denomina DBA (Data Base Administrator). Este profesional, muy importante para la empresa, recibe las especificaciones del Equipo de Análisis y Diseño para su implementación en un Sistema de Gestión de Base de Datos, como por ejemplo, Oracle, SQL Server, DB2, etcétera. El DBA tiene múltiples funciones y responsabilidades que se detallarán más adelante.
- Programador de aplicaciones: conoce los casos que se desarrollarán —escritos e identificados por el Equipo de Análisis y Diseño—, los prototipos de interfaces y las estructuras de los almacenamientos que se manipularán. El programador genera las aplicaciones necesarias en el sistema —con el lenguaje de programación que se le indica y conoce—, para la obtención de las entradas de datos que alimentarán la base de datos y, también, para lograr las salidas —como las pantallas de resultados o reportes—, que se plantearon en la propuesta de solución. Es conveniente aclarar que no hace falta que este usuario conozca toda la estructura de la base de datos, sino solo lo que necesita para programar. Normalmente, el programador de aplicaciones trabaja en un equipo de desarrollo, cuyo tamaño dependerá de la envergadura del sistema.
- Usuario final: es el personal que interactúa con las aplicaciones programadas por el usuario mencionado en el párrafo precedente y es, de entre todos los usuarios, el que menos conocimiento técnico posee. Si bien hay perfiles distintos entre los usuarios finales que interactúan con el sistema, esta persona no conoce ni los detalles técnicos ni los de la base de datos. Por ejemplo: un alumno que accede a su autogestión vía Internet para conocer su estado académico no necesita conocer la estructura del sistema, solo utiliza las opciones que se le presentan en el menú, que están asociadas con un perfil definido

para todos los alumnos. Otro usuario final del Sistema de Gestión de Alumnos es el profesor de un curso, quien verá las opciones definidas para ese perfil, con la posibilidad de cargar las notas de los parciales, enviar mensajes a todo el curso o imprimir la lista de sus alumnos.

Durante la vida del sistema, seguramente, se presentarán requerimientos que cambiarán y otros que no fueron considerados cuando se diseñó el sistema. En este último caso, es lógico que determinados usuarios no tengan los programas que necesitan. Por ejemplo, es muy común que el personal directivo especifique una necesidad como, por ejemplo, "Obtener los datos de los alumnos que cursan el último año de la carrera, trabajan, tienen promedio superior a 9 (nueve) y que no han rendido materias en los últimos dos turnos de examen". Este tipo de solicitud no programada suele dar lugar a que ciertos usuarios tengan permisos para acceder a la base de datos mediante consultas que ellos mismos arman en pantalla, en las que indican los almacenamientos que usarán, las condiciones que cumplirán, los datos que mostrarán y los cálculos que realizarán. Si esta solicitud se repitiera en el tiempo, se la podría incluir como una opción del sistema.

## 1.5 Seguridad de las bases de datos

La seguridad en la base de datos es un tema muy importante. En la mayoría de las organizaciones, es un conjunto de funciones manejadas por el Administrador de la Base de datos (o DBA) o, más apropiadamente, por un administrador de seguridad. Este rol es responsable de la creación de los nuevos usuarios y de la accesibilidad de los objetos de la base de datos. Como regla general, cuanto más grande es la organización, más sensible es la información y más probable es que la seguridad sea manejada por un Administrador de seguridad. Sin embargo, es importante que los desarrolladores, los DBA y los usuarios entiendan las opciones disponibles en el modelo de seguridad.

La base de datos más segura es aquella que no tiene usuarios, pero esta situación carece de sentido. Por esta razón, se debe llegar a un balance entre el permiso de acceso a los usuarios y el control de lo que se les permite hacer cuando establecen una sesión a través de una conexión.

Para ello, existe en cada base de datos un modelo de seguridad. Por razones de síntesis, nos dedicaremos a estudiar el que ofrece la base de datos Oracle. Se recomienda visitar la página de documentación oficial de la empresa en:

<http://www.oracle.com/technetwork/database/focus-areas/security/index.html>

En Oracle, el modelo de seguridad se basa en los siguientes elementos:

- Privilegios de sistema.
- Uso de roles para administrar el acceso a los datos.
- Privilegios de objeto.
- Contrasenas modificables.
- Otorgar y revocar los privilegios de objetos y de sistema.
- Uso de sinónimos para la transparencia de base de datos.

El modelo de seguridad Oracle consiste en dos partes:

1. Autenticación de contraseñas, que puede basarse en el Sistema Operativo que aloja al motor o usar el propio sistema de autenticación de la base. Cuando la última opción es la elegida por el DBA, la información de contraseñas se almacena en un formato encriptado en el diccionario de la base de datos.
2. Control de qué objeto de la base de datos se puede acceder y por cuál grupo de usuarios. Esta autoridad se transmite a través de los privilegios que se le asignan a los usuarios directamente o por medio de roles a grupos de usuarios.

### 1.5.1 Creación de usuarios

El inicio del proceso —que es el que permite el acceso a la base de datos— es la creación de usuarios con el comando SQL:

`CREATE USER <nombre> IDENTIFIED BY <contraseña>`

Usualmente se completa con algunas cláusulas de almacenamiento y opciones de usos de la base de datos, pero que no se relacionan con el tema que trataremos aquí.

### 1.5.2 Otorgar privilegios de sistema

Los privilegios de sistema permiten al usuario que los recibe la creación, modificación y eliminación de los objetos de la base de datos que almacenan los datos de las aplicaciones.

De hecho, para poder realizar alguna operación en la base de datos, el usuario necesita un privilegio de sistema para ingresar, denominado *CREATE SESSION*.

Dentro del alcance de los privilegios de sistema hay dos categorías:

- La primera de ellas es el conjunto de privilegios de sistema que se relacionan con el manejo de los objetos como tablas, índices, disparadores o *triggers*, secuencias, vistas, paquete, procedimientos y funciones almacenadas. Con estos privilegios se pueden realizar las siguientes acciones: crear, alterar su definición y eliminarlos en general y, para las tablas y programas, tenemos los privilegios de creación de índices, hacer referencia a la clave primaria con una clave foránea y ejecutar un programa almacenado.
- La segunda categoría de los privilegios se refiere a la habilidad de un usuario para realizar actividades especiales a lo largo del sistema. Estas actividades incluyen funciones como: actividades de auditoría, generación de estadísticas para soportar el funcionamiento del optimizador basado en costos y permitir el acceso a la base de datos solamente a los usuarios con un privilegio de sistemas especial denominado “sesión restringida” o *restricted session*. Estos privilegios usualmente se deberían otorgar solo a los usuarios que realizarán tareas de administración de alto nivel.

El otorgamiento de los privilegios de sistemas se ejecuta con la sentencia *GRANT* y, para otorgar un privilegio de sistema a otro usuario, el que lo transmite (*GRANTOR*) debe haber recibido el mismo privilegio con el agregado *WITH ADMIN OPTION*, para poder retransmitirlo a otros usuarios.



Los privilegios de sistema permiten al usuario que los recibe la creación, modificación y eliminación de los objetos de las bases de datos que almacenan los datos de las aplicaciones.

Los usuarios pueden crear objetos dentro de su esquema y si fuera necesario que un usuario deba crear objetos para cualquier otro esquema, deberá recibir el privilegio con la palabra *ANY* como en el siguiente ejemplo:

```
GRANT CREATE ANY TABLE TO miusuario
```

La anulación de un privilegio otorgado se realiza con la sentencia *REVOKE*. Si un usuario recibió un privilegio de sistema y lo ha transmitido con la opción *WITH ADMIN OPTION*, el hecho de perderlo no afectará a los usuarios que lo recibieron.

Por ejemplo: el usuario PEPE recibió el privilegio *CREATE ANY TABLE WITH ADMIN OPTION* realizó su trabajo creando cinco tablas y le pasó el mismo privilegio a dos usuarios más, a uno con la opción *WITH ADMIN* y a otro sin ella. Si otro usuario le revoca este privilegio al usuario PEPE, esta acción no tiene efecto sobre las tablas ni sobre los dos usuarios, ya que no pierden su privilegio de creación de tablas en cualquier esquema.

### 1.5.3 Usar roles para administrar los accesos de base de datos

Cuando la cantidad de objetos y usuarios va creciendo, la administración de accesos y privilegios se torna complicada y trabajosa, porque se van agregando aplicaciones que traen entre cientos y miles de objetos y se incorporan nuevos usuarios o los que están adquieren nuevos privilegios de acceso sobre estos objetos en forma acorde con la operativa.

La forma de simplificar elegida por los Sistemas de Gestión de Base de Datos ha sido la creación de una capa de abstracción llamada ROL, que agrupa los privilegios y, generalmente, define un nombre de rol que describe un tipo de tarea o función. Además de simbolizar un tipo de tarea o actividad general, con el ROL se establece la idealización de un usuario que necesita un grupo de privilegios para un cierto trabajo, como ESTUDIANTE, PROFESOR, VENDEDOR, OPERADOR, DATAENTRY, SUPERVISOR, OPERADOR\_BATCH, OPERADOR\_BACKUP. Así, todos los accesos a los objetos se pueden agrupar en estos roles de base de datos para ser administrados y otorgados entre los usuarios y entre otros ROLES, ya que es posible asignar ROLES a otros ROLES.



La forma de simplificar elegida por los Sistemas de Gestión de Bases de Datos ha sido la creación de una capa de abstracción llamada ROL.

Para usar ROLES es necesario realizar dos tareas:

1. Agrupar lógicamente los privilegios, como los de creación de tablas, índices y procedimientos. También se puede definir una contraseña para un ROL para, de esta manera, proteger el acceso a los privilegios con la cláusula idéntica a la creación de usuarios. Veamos el ejemplo:

```
CREATE ROLE creador_procs IDENTIFIED BY soycreador
```

```
GRANT create any procedure TO creador_procs WITH ADMIN OPTION
```



Para usar ROLES es necesario realizar dos tareas:

1. Agrupar lógicamente los privilegios.
2. Agrupar a los usuarios de cada aplicación que manejan necesidades similares.

Esta forma de hacer niveles de ROLES como capas intermedias, para distribuir privilegios establecidos, facilita la administración, puesto que bastará con determinar el tipo de usuario que corresponde o la clase de tarea que se le habilitará para encontrar uno o dos ROLES. De esta manera, el usuario recibirá todos los privilegios necesarios.

En el ejemplo, ilustramos esta secuencia de tareas:

```
CREATE ROLE desarrollador  
GRANT CREATE TABLE TO desarrollador  
GRANT SELECT ANY TABLE TO desarrollador  
GRANT DROP USER TO desarrollador  
GRANT desarrollador TO juan  
GRANT desarrollador TO pedro
```

Los roles se pueden modificar para que soporten el requerimiento de contraseña usando la siguiente sentencias:

```
ALTER ROLE desarrollador IDENTIFIED BY abrepuestas
```

Los roles se eliminan con la ejecución de la sentencia *DROP ROLE*. Esta acción requiere que el administrador tenga los siguientes privilegios: *CREATE ANY ROLE*, *ALTER ANYROLE* y *DROP ANYROLE*, o que sea dueño del ROL.

Los roles se pueden otorgar a otros roles con *GRANT* y se quitan de los usuarios o roles con *REVOKE*.

El hecho de que un ROL tenga contraseña, nos hace preguntar en qué momento y cómo se activa para ejercer los privilegios asociados. Al inicio de la sesión, se activan los roles fijados como *default* para el usuario. Normalmente, el estado de un ROL es *ENABLED* o activado, a menos que haya sido expresamente desactivado. Estas acciones se realizan con las siguientes sentencias:

```
ALTER USER juan DEFAULT ROLE ALL (Activa todos los roles asignados a juan)  
ALTER USER juan DEFAULT ROLE ALL EXCEPT sysdba  
ALTER USER juan DEFAULT ROLE desarrollador, operador_backup  
ALTER USER juan DEFAULT ROLE NONE (ningún ROL queda activado al iniciar la sesión)
```

#### 1.5.4 Otorgar privilegios de objetos

Una vez que se ha creado, un objeto puede ser administrado por su creador o por el usuario que tenga el privilegio *GRANT ANY PRIVILEGE*.

La administración consiste en otorgar los privilegios sobre el objeto que permitirán manipularlo: agregando filas, cambiando su estructura o viendo los datos de sus columnas.

Los privilegios de objetos se agrupan en dos grupos: los que permiten ver, agregar, modificar e insertar filas y los de objetos que dejan eliminar otros objetos o usarlos como el privilegio *REFERENCE* y *EXECUTE*, que permiten crear claves foráneas en una tabla que referencia a una clave primaria de otra tabla sobre la que se ha dado este privilegio de referencia. El privilegio *EXECUTE* otorga a un usuario la facultad de

Los privilegios de objetos se agrupan en dos grupos: los que permiten ver, agregar, modificar e insertar filas y los de objetos que dejan eliminar otros objetos o usarlos como el privilegio *REFERENCE* y *EXECUTE*.

ejecutar una función, procedimiento o paquete almacenado. Otros privilegios administran la modificación y creación de objetos de base de datos, como *ALTER TABLE*, *ALTER SEQUENCE* y el privilegio que permite crear un índice sobre una tabla de otro usuario es *INDEX TABLE*.

Estos privilegios de objetos pertenecen al usuario que los creó o que los posea con ANY. Se otorgan también con una opción que facilita su transmisión a otros usuarios, como es el parámetro *WITH GRANT OPTION*.

A diferencia de los privilegios de sistemas, un privilegio de objeto otorgado por un usuario que lo recibió con el parámetro *WITH GRANT OPTION* se revocará automáticamente si a este usuario GRANTOR se le anulara este privilegio. Es decir: hay un efecto cascada en el *REVOKE* para los privilegios de objeto que no ocurre con los privilegios de sistema.

### 1.5.5 El cambio de contraseñas

Una vez creado, el usuario puede cambiar su contraseña con la sentencia:

```
ALTER USERjuan IDENTIFIED BYabretesesamo
```

### 1.5.6 Uso de sinónimos para la transparencia de base de datos

Los objetos de base de datos son de los usuarios que los crean y solo estarán disponibles para él, a menos que se otorguen explícitamente privilegios a otros usuarios o roles.

Sin embargo, y pese a que otro usuario tenga privilegios para usar estos objetos de otro esquema, al hacer uso de ellos debe respetar los límites del esquema; esto es: debe llamar al objeto con su denominación completa en la base, es decir, el nombre del esquema precederá al nombre del objeto. Por ejemplo: si Juan desea usar la tabla clientes del esquema Pedro, usará el nombre del esquema como prefijo del nombre de la tabla. Si Juan ya tiene permiso sobre la tabla "Materias" de Pedro, incluirá el nombre del esquema de esta manera:

```
SELECT * FROM pedro.materias
```

Sin la información del nombre del esquema al que pertenece el objeto, el motor de la base de datos no entenderá de qué tabla Materias se trata.

Este requisito de anteponer el esquema produce la pérdida de transparencia que debe resolverse usando "sinónimos", ya que permite a Juan ver la tabla Clientes sin necesidad de anteponer explícitamente el nombre del esquema.

Hay dos tipos de sinónimos:

- Privados: tienen una aplicación más limitada.
- Públicos: se usan para poner a disposición de los objetos a todos los usuarios sin el requisito de calificarlos con el formato "esquema.nombre\_del\_objeto".

Si se realiza la siguiente operación:

```
CREATE PUBLIC SYNONYM materias FOR pedro. materias
```



El requisito de anteponer el esquema produce la pérdida de transparencia que debe resolverse usando "sinónimos".

Ahora Juan (y cualquier otro usuario) podrá referirse a la tabla Materias de Pedro solamente con el sinónimo del objeto, sin prefijo del esquema.

```
SELECT * FROM materias
```

Los privilegios privados se crean con el mismo fin, pero tienen alcance limitado para cada usuario en particular, como:

```
CREATE SYNONYM materias FOR pedro. materias
```

Ahora solamente Juan podrá hacer esto con la tabla de Pedro:

```
SELECT * FROM materias
```

Con estos componentes del modelo de seguridad de la base de datos, el usuario podrá entender la mayoría de los modelos de otras bases, ya que implementan esencialmente las mismas características, pero deben observarse las diferencias que pudieran existir en cada implementación.

## 1.6 Funciones y responsabilidades de un DBA



El DBA indica cuáles son los datos de cada entidad, los tipos de datos, la dimensión de cada dato, las relaciones entre ellos, sus claves identificadas, las vistas para los usuarios finales, etc.

Como se afirmó en los párrafos precedentes, el DBA tiene múltiples funciones técnicas y responsabilidades con el sistema. En algunas organizaciones, suele trabajar solo, pero es muy común que necesite personal a su cargo para realizar tareas operacionales y para cubrir la alta demanda de tiempo; por ejemplo, para atender los requerimientos de empresas con servicios internacionales y con uso del sistema en 7días x 24horas, es decir, todas las horas de todos los días de la semana del año, como en el caso de un sistema de vuelos internacionales.

Entre sus funciones principales se pueden mencionar:

- Especificación lógica de la base de datos: el DBA especifica —mediante la interfaz del DBMS elegido o las sentencias de definición de datos— la estructura de la base de datos que recibió del Equipo de Análisis y Diseño. Indica cuáles son los datos de cada entidad, los tipos de datos, la dimensión de cada dato, las relaciones entre ellos, sus claves identificadas, las vistas para los usuarios finales, etcétera. Las sentencias que utiliza el DBA se agrupan en lo que se denomina Lenguaje de Definición de Datos (DDL) y está formada por las instrucciones para trabajar en la estructura y, también, para crear objetos, para modificar las estructuras de los objetos o para eliminar los que ya no se necesiten en la base de datos.
- Especificación física: el DBA define el medio físico que almacenará a la base de datos; por ejemplo, el disco y su partición en el servidor y, también, cómo se almacenarán los archivos de datos y cómo se accederá a ellos para lograr una mejor performance.
- Definición de seguridad: define grupos de usuarios y usuarios individuales, con los perfiles para cada uno, e indica los archivos a los que pueden acceder y los derechos que poseen de manera individual. Por ejemplo: el perfil "Estudiante" permite el derecho de consultar los archivos que contienen los datos de sus

evaluaciones; de consultar y enviar mensajes a los docentes; de modificar su contraseña. Sin embargo, no tendrá la posibilidad de ver los datos personales de sus docentes o el derecho a modificar las notas de sus evaluaciones.

- Definir procedimiento de respaldo: es responsabilidad del DBA asegurar que los datos estén respaldados para evitar inconvenientes ante algún tipo de incidente (rotura de medio físico, errores de procedimientos en actualizaciones, robo de hardware, incendio, etc.). Por lo tanto, deberá definir la periodicidad de los respaldos, el o los medios para mantenerlos (back-up); si copiará todo el contenido de la base de datos cada vez que inicie el respaldo, o si será parcial, etcétera. Esto no significa que el DBA será el que realice este trabajo, sino que es el que define el procedimiento que ejecutará el personal de soporte en la empresa. Del mismo modo, designará a los encargados del procedimiento para la recuperación de datos.
- Implementar reglas de integridad: el Equipo de Análisis y Diseño especifica ciertas limitaciones a los datos que se almacenarán, determinados por el mundo real de la organización o por reglas lógicas propias de las bases de datos. Por ejemplo: en determinados países, la nota que un estudiante puede obtener está entre el valor 0 (cero) y el 10 (diez), lo que implica que el dato "Nota" nunca estará fuera de esos límites. Otro ejemplo de integridad se observa en un archivo de "exámenes", en el cual al estudiante que se registra, debe ser alumno regular de la universidad y, la materia que rinda, parte integrante del plan de estudios de la carrera que cursa. Si se analiza un sistema bancario, también tiene las restricciones propias del negocio al igual que una empresa de telefonía. Estas reglas se definen durante el proceso de creación de las bases de datos y de los archivos que la conforman; incluso, se pueden agregar algunas nuevas que surjan durante el uso del sistema y mientras los datos lo permitan, ya que algunos, seguramente, existen.
- Monitorear la performance de la base de datos: el DBA debe monitorear el rendimiento de la base de datos, detectando los procesos que generen demoras en la devolución de información, para mejorar, permanentemente, su performance general. En algunos casos, el resultado es que los desarrolladores deben revisar las aplicaciones o la arquitectura del sistema; en otras situaciones, se pueden mejorar los métodos de acceso; incluso, la ubicación de los archivos de datos o las capacidades del hardware o de la conectividad que infuyen en el rendimiento. El DBA resuelve determinadas situaciones pero, en otras, solo se informará a quienes tienen el poder de decisión, para que encuentren las soluciones posibles.

## 1.7 Arquitectura ANSI/SPARC

Como las funciones de los distintos usuarios mencionados de una base de datos son distintas, lo que implica que cada uno puede interactuar de diversas formas con los datos almacenados, surge la Arquitectura ANSI/SPARC para estandarizar los conceptos y permitir una mejor lectura de la independencia de datos, lo que permitirá que se ubique a cada usuario de una base de datos en función de su relación con ella, ya que no todos poseen la misma visión, aunque los datos almacenados son únicos.

# S

ANSI es el instituto que "supervisa la creación, promulgación y el uso de miles de normas y directrices que impactan directamente en casi todos los sectores de las empresas: desde dispositivos acústicos hasta los equipos de construcción, desde la producción lechera y ganadera hasta la distribución de energía, y muchos más".

ANSI (American National Standards Institute) tuvo mucha influencia en la definición de esta arquitectura. Fundado el 19 de octubre de 1918, se autodefine, en su página Web ([www.ansi.org](http://www.ansi.org)), como el instituto que "supervisa la creación, promulgación y el uso de miles de normas y directrices que impactan directamente en casi todos los sectores de las empresas: desde dispositivos acústicos hasta los equipos de construcción, desde la producción lechera y ganadera hasta la distribución de energía, y muchos más".

La arquitectura, que se presentará en tres niveles, tiene su antecedente en 1971, cuando CODASYL (Conference on Data Systems Languages). Además, este comité, formado por personas de industrias informáticas, tiene el objetivo de regular el desarrollo de un lenguaje de programación estándar, de donde surge el lenguaje COBOL, reconoce la necesidad de una arquitectura en dos niveles: un esquema —vista de sistema— y un subesquema —vista de usuario.

Posteriormente, ANSI se dedica al tema, como lo dice en el libro *Databases, role and structure: an advanced course*, escrito por P. M. Stocker, P. M. D. Gray y M. P. Atkinson, en el cual indica que cuenta con un comité denominado X3, que se dedica a la computación y al procesamiento de información. El SPARC (Standards Planning and Requirements Committee) de ANSI/X3 fija un grupo de estudio de DBMS para especificar los modelos para la estandarización de las bases de datos. En 1972, este comité produce un reporte provisional, seguido por otro final, en 1977, el cual divide la vista de sistema —destacada por CODASYL— en vista de empresa y de almacenamiento, que es lo que actualmente se conoce como:

- Nivel externo: lo conforman las múltiples vistas de los datos almacenados en la base de datos y se presentan a los distintos usuarios de múltiples formas, adecuándolas a las necesidades de información que tiene cada uno. A este nivel también se lo denomina nivel de visión, que se define con el lenguaje de manipulación de datos.
- Nivel conceptual: es la estructura lógica global, que representa las estructuras de datos y sus relaciones. Hay una única vista en este nivel y se la define con el lenguaje de definición de datos.
- Nivel interno: se define como el conjunto de datos que están almacenados físicamente y, como no se accede a ellos, también se lo denomina nivel físico.

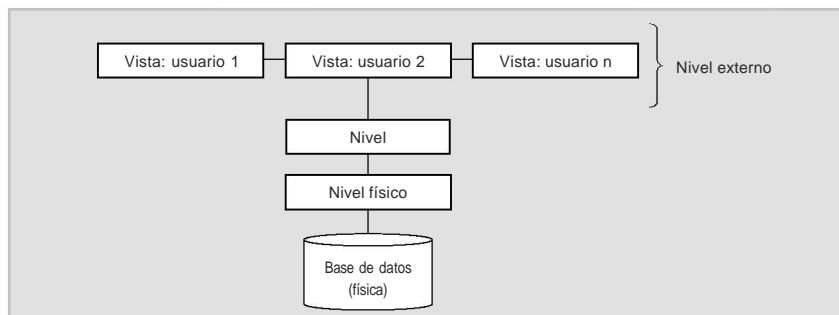


Fig. 1.2 Niveles de la base de datos.

A continuación, se verá la relación de los usuarios con los niveles de abstracción expresados en la Fig. 1.2. Los usuarios finales y los programadores de aplicaciones no

necesitan conocer cómo están almacenados los datos físicamente, porque tienen una visión alejada del nivel físico y personalizada para cubrir las necesidades de cada uno. Por ejemplo, un cajero de una entidad bancaria no precisa los mismos datos que un gerente, ni un estudiante universitario las opciones que usa un profesor de la universidad. Un reporte presentado a un gerente, puede ser el resultado de un largo proceso en el cual se combinaron los datos provenientes de distintos archivos y se aplicaron múltiples cálculos. Es lo que necesita, ya que no conoce ni le interesan estos detalles técnicos.

Otro usuario, el DBA, está en condiciones y es el único que debería conocer el nivel físico, para modificarlo si hiciera falta. Por ejemplo: podría cambiar el medio de almacenamiento de la base de datos o la forma de acceso a los archivos, manteniendo siempre inalterables las vistas para los demás usuarios.

Solo el DBA puede introducir cambios en el nivel conceptual, pero sí es necesario que mantenga las vistas inalteradas hacia los usuarios finales y los programadores de aplicaciones. Por ejemplo: agregar una columna a un archivo de datos cuando éste no se lo tuvo en cuenta inicialmente en el sistema o agregar un archivo de datos con las relaciones que corresponda a los ya existentes.

Un archivo de datos de “Productos” y otro de “Proveedores”, de una base de datos de una empresa que vende insumos y artículos informáticos, podrían tener este aspecto:

Ejemplo:

| Tabla Productos |                            |           |         |       |       |     |
|-----------------|----------------------------|-----------|---------|-------|-------|-----|
| CodProd         | DesProd                    | PrecioVta | CodProv | Stock | StMin |     |
| 108             | IMP. MULTIF. TX410         | 699,80    | 19      | 2     | 6     |     |
| 230             | MON. 19 PULG. LCD W1943C   | 925,50    | 32      | 12    | 10    |     |
| 110             | IMP. MULTIF. C5580-CH.TIN. | 769       | 21      | 3     | 6     |     |
| ...             | ...                        | ...       | ...     | ...   | ...   | ... |

| Tabla Proveedores |             |          |          |
|-------------------|-------------|----------|----------|
| CodProv           | RazónSocial | Contacto | Teléfono |
| 21                | HP Center   | ...      | ...      |
| 19                | Moura EPSON | ...      | ...      |
| 32                | Asoc. LG    | ...      | ...      |
| ...               | ...         | ...      | ...      |

Físicamente, estos archivos de la base de datos se almacenan de una única manera que, en definitiva, es un conjunto de bytes que no son visibles para ningún tipo de usuario y que se administra a través del sistema operativo. Pero, el DBA, si utiliza



Los archivos de la base de datos se almacenan de una única manera que, en definitiva, es un conjunto de bytes que no son visibles para ningún tipo de usuario y que se administra a través del sistema operativo.

herramientas e instrucciones del DBMS, lo puede organizar e indicará en qué disco se almacena la base de datos (disco local o remoto, en uno u otro servidor) y definirá para cada archivo, su método de acceso. Por ejemplo: mediante la generación de un índice por código de producto en el archivo "Productos" y, de esta manera, agilizará su búsqueda.

En el nivel conceptual o de estructura lógica, el DBA puede crear o modificar las estructuras de estos archivos. Por ejemplo: si indica que el dato "Código del Producto" es numérico, con cuatro dígitos posibles (0 a 9999), se autogenera cada vez que se inserta un nuevo producto, sin que pueda repetirse. Aquí, también, es necesario identificar la relación entre los archivos de "Productos" y de "Proveedores" y determinará cómo se produce una búsqueda o qué sucede cuando un proveedor quiere que se lo elimine de la base de datos.

En el nivel externo, cada usuario verá lo que necesita y tiene habilitado; por ejemplo:

- El vendedor verá, en su pantalla, un listado ordenado por descripción del producto, con cabeceras de listado distintas a los nombres de los campos de datos en el archivo origen. A él no le interesa ni quién es el proveedor ni cuál es el stock mínimo de cada producto. Tampoco sabe si todos estos datos provienen de un solo almacenamiento o de una combinación.

| Producto | Descripción                | Precio | Stock actual |
|----------|----------------------------|--------|--------------|
| 108      | IMP. MULTIF. TX410         | 699,80 | 2            |
| 110      | IMP. MULTIF. C5580-CH.TIN. | 769    | 3            |
| 230      | MON. 19 PULG. LCD W1943C   | 925,50 | 12           |
| ...      | ...                        | ...    | ...          |

- El encargado de compras visualizará un listado con combinación de datos de los dos archivos, con cabeceras más descriptivas, que incluirá el resultado de un cálculo e indicará lo que debería comprar, como mínimo, para cubrir el stock por producto:

| Stock    |                            |        |        |        |         |             |
|----------|----------------------------|--------|--------|--------|---------|-------------|
| Producto | Descripción                | Precio | Actual | Mínimo | Comprar | Proveedor   |
| 108      | IMP. MULTIF. TX410         | 699,80 | 2      | 6      | 4       | Moura EPSON |
| 230      | MON. 19 PULG. LCD W1943C   | 925,50 | 12     | 10     | 0       | Asoc. LG    |
| 110      | IMP. MULTIF. C5580-CH.TIN. | 769    | 3      | 6      | 3       | HP Center   |
| ...      | ...                        | ...    | ...    | ...    | ...     | ...         |

- El gerente de la empresa, seguramente, necesitará un informe con una mayor complejidad de proceso, que incluirá los datos estadísticos resultantes de los cálculos que resumirá una situación, en la que se podría utilizar hasta un gráfico en pantalla; pero, en definitiva, los obtendrá de los datos almacenados en los mismos archivos con una vista distinta.

## 1.8 Modelos de datos

En los ambientes de desarrollo existen lenguajes de programación estructurados, orientados a aspectos, orientados a objetos, etc.; donde cada uno brinda conceptos y ventajas en determinados tipos de aplicaciones.

En los ambientes de base de datos también existen modelos de datos para almacenar los datos de la empresa, algunos que históricamente dieron las bases para el desarrollo de los demás, otros que están actualmente en uso y los que están en proceso de investigación.

Un modelo de datos brinda distintos conceptos y permite definir las reglas y las estructuras para el almacenamiento de datos para, después, manipularlos.

Los modelos de datos se clasifican de diversas formas. Sin embargo, en la literatura especializada, existe cierta coincidencia en utilizar —para su taxonomía— los conceptos que conforman la base de cada uno de ellos. Por ejemplo: hay modelos de datos basados en objetos y otros en registros; también el modelo físico de datos que apunta al almacenamiento de los datos en un nivel bajo de abstracción, relacionado con el formato y con el camino de acceso a los datos.

En los modelos de datos basados en registros, los datos se estructuran en un conjunto de campos que conforman un registro. Tal es el caso de un registro “Estudiante” conformado por campos como legajo, apellido, nombres, fecha y lugar de nacimiento.

En el caso de los modelos basados en objetos, se han aprovechado los conceptos utilizados en el paradigma de programación orientada a objetos, como sucede en las áreas de ingeniería de software. Solo que, en las bases de datos orientadas a objetos, existe la posibilidad de crear objetos para que tengan un almacenamiento que persista en el tiempo y para que puedan ser utilizados.

### 1.8.1 Modelo de datos orientado a objetos

Este modelo nace con el objetivo de permitir el almacenamiento de datos para aplicaciones complejas que —como dicen Elisa Bertino y Lorenzo Martino en Sistemas de bases de datos orientadas a objetos— no se orientan a las del área comercial y administrativa, sino a las de las ingenierías, tales como CAD/CAM, CASE, CIM, sistemas multimediales y sistemas de gestión de imágenes, en las que la estructura de los datos es compleja y las operaciones se definen en función de la necesidad de las aplicaciones.

El grupo de gestión de bases de datos orientadas a objetos (ODMG, Object Database Management Group) —conformado por usuarios y productores de sistemas de gestión de bases de datos orientados a objetos (OODBMS, Object-Oriented Database Management System)— es el encargado de estandarizar el modelo de datos, el lenguaje de definición de objetos (ODL, Object Definition Language) y el lenguaje de consulta de objetos (OQL, Object Query Language).

Los datos se almacenan de manera persistente en objetos, que se identifican, únicamente, por un OID (identificador de objeto), generado por el sistema.

Los objetos se definen con una estructura (OID, Constructor\_Tipo, Estado) en la que:

- OID: identificador único en el sistema; incluso, cuando se elimina un objeto, es conveniente que el OID no se reutilice.



Un modelo de datos brinda distintos conceptos y permite definir las reglas y las estructuras para el almacenamiento de datos para, luego, manipularlos.



Los datos se almacenan de manera persistente en objetos, que se identifican, únicamente, por un OID, generado por el sistema.

- Constructor\_Tipo: es la definición de la construcción del estado del objeto.
- Estado: se define a través del Constructor\_Tipo.

Por ejemplo: si el Constructor\_Tipo es de tupla, significa que el estado tendrá la estructura, donde es un nombre de atributo y es un OID.

Si el Constructor\_Tipo es átomo, significa que el estado será un valor.

De esta manera, el Constructor también puede ser un conjunto, una lista, una bolsa y un array y determina que el estado se refiera a OID de objetos. Solo el átomo se puede referir a un valor, aunque éste puede, también, referirse a otro objeto mediante su OID.

Con estos constructores, se producen anidamientos y se logra una estructura de alta complejidad, además de la posibilidad de incorporar conceptos propios de la programación orientada a objetos: como clase, herencia, polimorfismo, etcétera.

La definición en ODL de una clase "Personas" se puede expresar de la siguiente manera:

```
Class Personas
{
    attribute string apellido,
    attribute string nombres,
    attribute enum PosiblesSexo {Masculino, Femenino} sexo,
    attribute struct Domicilio
        {string calle, short numero, string ciudad,
        attribute date fecha_nac;
        short edad();
    };
}
```

Para consultar la base de datos, se utiliza el lenguaje OQL propuesto por ODMG. Las sentencias de este lenguaje son similares a las de SQL y pueden ser insertadas en programas desarrollados con lenguajes de la programación orientada a objetos, como en el caso de C++ o Java. El resultado es un conjunto de objetos que pueden ser manipulados directamente en esos lenguajes y otros que sigan el paradigma.

Una consulta OQL puede ser:

```
SELECT p.apellido
FROM p in Personas
WHERE p.sexo="Masculino"
```

En esta expresión, aparece "p" como variable iteradora que asume los valores de cada objeto y la consulta devuelve una bolsa *string*, porque es una colección de apellidos que pueden repetirse.

Entre las ventajas de este modelo, se pueden distinguir: la posibilidad de manipular objetos complejos con buen rendimiento, la integración de la persistencia de datos a la programación orientada a objetos y un menor costo y esfuerzo en el desarrollo de las aplicaciones y en el mantenimiento.

La aparición de este tipo de sistemas de gestión de bases de datos exigió a los RDBMS (sistemas de gestión de bases de datos relacionales), que incorporaran objetos en sus estructuras y flexibilizaran las posibilidades de almacenamiento; de ello, surge el concepto actual de sistemas de gestión de bases de datos objeto-relacionales.

En la actualidad, se conocen prototipos y productos de gestión de bases de datos orientadas a objetos como, por ejemplo:

- Jasmine: tecnología orientada a objetos que permite tratar datos multimedia y complejas informaciones relacionadas. Se trata de un producto Java que se puede utilizar en entornos Intranet, Internet, Cliente/Servidor y a través de cualquier navegador.
- GemStone: introducido en 1987, es el ODBMS comercial con más tiempo en el mercado. Soporta acceso concurrente de múltiples lenguajes, que incluye al Smalltalk-80, Smalltalk/V, C++ y C.
- Versant Object Database (V/OD): ofrece características importantes a los desarrolladores C++, Java o .NET, el apoyo a la concurrencia masiva y a grandes conjuntos de datos.

Éste es un tema tan interesante como extenso; por ello, se dejará su tratamiento para otro capítulo y se empezará con la explicación de las bases de conceptos para llegar al modelo relacional.

### 1.8.2 Modelos de datos basados en registros

Estos modelos de datos tienen sus inicios con los modelos de red y jerárquicos, hoy obsoletos desde el punto de vista tecnológico.

Ambos modelos, red y jerárquicos, compartían el concepto de registro de datos y los enlaces físicos entre ellos, pero diferían en sus restricciones y en su representación.

Posteriormente, surgió el Modelo Relacional, donde se mantiene el concepto de registro de datos y las relaciones entre las entidades del mundo real son lógicas, dando origen a entidades que permiten establecer relaciones, como también a otros temas propios del modelo. Por la importancia que tiene el tema en sí mismo, se decidió dedicar otros capítulos al Modelo Relacional.

#### 1.8.2.1 Características del modelo de datos en red

Los datos se representan como una colección de registros y las relaciones entre ellos mediante enlaces, que se implementan como campos que almacenan punteros.

Cada registro es una colección de campos y cada campo almacena un dato.

Gráficamente, se los representaba con un diagrama, en el que la estructura de datos estaba formada por cajas y líneas. Las cajas eran los registros y las líneas indicaban sus relaciones.



Los datos se representan como una colección de registros y las relaciones entre ellos mediante enlaces, que se implementan como campos que almacenan punteros.

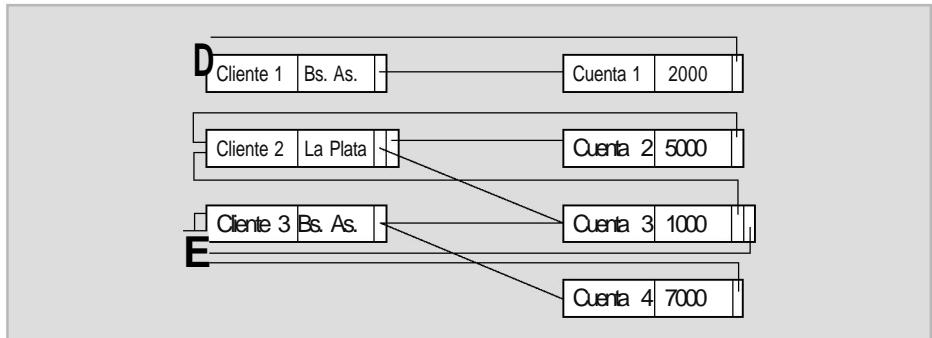


Fig. 1.3 Modelo de datos en red.

La Fig. 1.3 representa a tres registros de clientes y a cuatro registros de cuentas bancarias donde:

- Cliente 1 posee una sola cuenta con un saldo de \$2000. Se observa que este registro tiene un campo para un puntero del Cliente a la Cuenta 1 y, en el registro Cuenta 1, se visualiza un campo con un puntero hacia el Cliente 1.
- Cliente 2 tiene una Cuenta 2 a su nombre y comparte la Cuenta 3 con el Cliente 3. En este caso, en Cliente 2 hay dos campos desde donde parten dos punteros.
- Cliente 3 comparte la Cuenta 3 con el Cliente 2 y, además, posee su propia cuenta que es la 4.

Se ve que en el registro de la Cuenta 3 hay dos campos, porque tiene un puntero hacia el Cliente 2 y otro, hacia el 3.

Los campos que almacenaban punteros hacían que los registros fueran de longitud variable, porque en algunos casos se necesitaba solo uno y en otros, más.

La flexibilidad del modelo hizo que la estructura de datos fuera compleja, ya que no había restricciones, en la que se puede observar cierta estructura Padre-Hijo.

Las acciones de manipulación de datos en la programación implicaban órdenes que significaban movimientos físicos. Ejemplo:

- *Find*: localiza un registro, según alguna condición y el puntero actual, en el archivo, queda apuntando a ese registro.
- *Get*: copia registro señalado por el puntero actual en el área de trabajo-buffer de memoria.
- *Store*: crea un registro nuevo, en un archivo, con valores de datos ubicados en la memoria que maneja el usuario.
- *Modify*: modifica contenido. Implica encontrarlo, subirlo a la memoria y, solo ahí, modificarlo.
- *Erase*: elimina registro, con su búsqueda previa.
- *Connect*: conecta un registro con un conjunto de registros de un archivo.
- *Disconnect*: desconecta un registro de un conjunto archivo.

- **Reconnect:** mueve un registro de un conjunto a otro. Por ejemplo: cambia una cuenta de una sucursal a otra.

Al crear los conjuntos archivos, se debe indicar qué hacer ante las siguientes acciones:

- **Inserción:** si la inserción es manual, debe existir en el programa una instrucción connect para enlazar el nuevo registro y, si no, puede ser automática, donde al ejecutar store el registro se almacena en el conjunto y se enlaza.
- **Eliminación:** depende si se definió como:
  - Opcional, en el cual pueden existir registros sin conexión al conjunto (elimina padre y desconecta todos los hijos).
  - Fijo, en el cual no puede haber registros de miembros sin conexión al grupo y, además, no pueden reconectarse con otra ocurrencia de un conjunto (elimina padre y elimina todos los hijos).
  - Obligatorio, tampoco puede haber registros sin conexión al grupo, pero sí pueden moverse de un conjunto a otro (no deja eliminar el registro).
- **Ordenación:** en el conjunto “archivo” debe definirse qué se hará al añadir un nuevo registro. Las opciones son que, al insertarlo, pase a alguna de estas posiciones en el conjunto:
  - Primero del conjunto.
  - Último.
  - Siguiente al registro actual del conjunto.
  - Anterior al registro actual.
  - Arbitrario, según determinación del sistema de gestión.
  - Ordenado según algún criterio y el sistema de gestión debe mantener el orden.

#### 1.8.2.2 Características del modelo de datos jerárquico

Este modelo se puede considerar una implementación particular del modelo en red, en el cual la estructura física también se basa en registros y en punteros, pero con forma de árbol invertido y con restricciones en las relaciones.

Las relaciones se representan mediante enlaces implementados como campos que almacenan punteros.

Los grafos se organizan como árboles con un nodo raíz.

Un registro es una colección de campos y cada uno de esos campos almacena un dato. A la colección de registros, se la denomina tipo de registro, que son los nodos en el árbol.

Las relaciones que se mantienen entre tipos de registros son del tipo 1: N, en la que se distinguen un registro “Padre” y n registros Hijos.

En el diagrama jerárquico, se muestra la estructura de árbol, con cajas que representan los tipos de registros y líneas que representan las relaciones “Padre-Hijo”.

Los niveles del árbol no tienen limitación y el nodo raíz está en el nivel 0 (cero).



Un registro es una colección de campos y cada uno de esos campos almacena un dato.

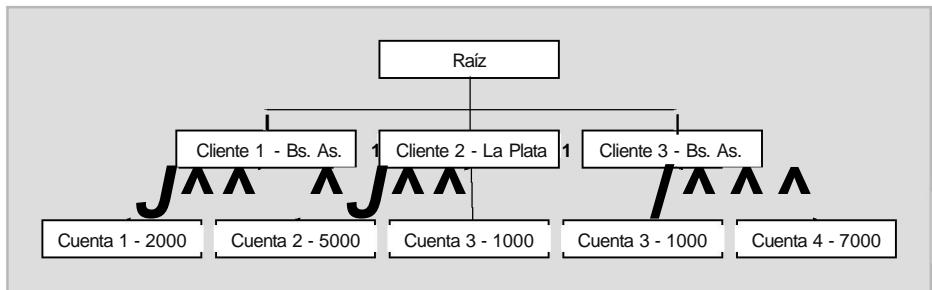


Fig. 1.4 Modelo de datos jerárquico.

Algunas restricciones del modelo son:

- El nodo raíz no participa como hijo en ninguna relación y los tipos de registros son siempre hijos de un solo tipo de registros.
- Un nodo padre puede tener un número ilimitado de hijos, pero un nodo hijo puede y debe tener solo un nodo padre.
- Si un tipo de registro "Padre" posee más de un tipo de registro "Hijo", los tipos de registros "Hijo" estarán ordenados de izquierda a derecha.
- Se puede eliminar un registro "Hijo" y el registro "Padre" puede quedar con 0 (cero) hijos, pero si se desea eliminar en el registro "Padre" no puede quedar un registro "Hijo" sin padre, por lo que se elimina el "Padre" y todos los hijos asociados.

Para que un registro "Hijo" tenga más de un registro "Padre", debe duplicarse con la consiguiente redundancia de datos. Algunas implementaciones evitan esto mediante uso de registros virtuales con punteros.

Ese tipo de situaciones muestran una desventaja del modelo jerárquico: la poca flexibilidad que posee para representar estructuras que, en la vida real, hacen falta. Que dos o más clientes de un banco comparten una cuenta es muy común, como también en una estructura de productos existen productos que son piezas de otro producto (estructura recursiva de datos).

Algunas acciones que se pueden mencionar en la manipulación de datos de este modelo:

- *Get*: localiza y copia un registro al área de trabajo las condiciones de búsqueda se agregan con la cláusula WHERE. Con un ciclo, se podía recorrer toda una rama del árbol, leyendo todos los registros al recorrerla, sin que importe la altura del árbol.
- *Insert*: añade un registro al árbol. Primero se localiza a su registro "Padre" y luego, se emite la orden de inserción.
- *Replace*: modifica uno o más campos de un registro que, primeramente, se ubicará como el registro actual de la base de datos.
- *Delete*: elimina un registro de la base de datos que esté como registro actual.

La primera implementación de este modelo fue IMS (Information Management System). Se trata de un diseño de IBM y otros colaboradores, realizado en 1966, para el Programa Apolo de la NASA. En abril de 1968, fue instalado en la División Espacial de la NASA Rockwell en Downey, California.

## 1.9 Resumen

De lo expuesto en este capítulo, podemos concluir que actualmente todas las personas interactúan con datos de alguna manera y en circunstancias diferentes. Estos datos se almacenan en un medio físico y luego, se vinculan a un sistema informático que los registra y habilita su acceso.

En el pasado, esta tarea de almacenamiento de datos requería de un esfuerzo mucho mayor a la hora de acceder a la información guardada, dado que este almacenamiento se focalizaba en tipos de datos muy reducidos lo cual limitaba las posibilidades de respuesta sin satisfacer por completo las necesidades de las empresas.

Fueron presentados los conceptos fundamentales para introducir al lector en los entornos de bases de datos, incluyendo la descripción de elementos componentes de un sistema con bases de datos, como son el DBMS, los distintos usuarios que interactúan con los datos y las definiciones de seguridad a considerar cuando se asignan privilegios.

También se caracterizaron modelos de datos que hacen posible el almacenamiento de datos, incluyendo la introducción mínima al Modelo de Datos Relacional.

Encontrará más información sobre IMS (*Information Management System*) en la publicación de IBM: <http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.imsintro.doc.intro/ip0ind0011003710.htm>

## 1.10 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) solo está disponible para docentes.

### 1.10.1 Mapa conceptual del capítulo

### 1.10.2 Autoevaluación

### 1.10.3 Presentaciones\*



# 2

## Modelo de datos relacional

### Contenido

|  |     |
|--|-----|
| 2.1 Introducción.....                        | 26  |
| 2.2 Álgebra relacional.....                  | 41  |
| 2.3 Normalización.....                       | 63  |
| 2.4 Origen de los datos.....                 | 67  |
| 2.5 Las formas normales.....                 | 69  |
| 2.6 Las estructuras.....                     | 92  |
| 2.7 Un caso de estudio.....                  | 99  |
| 2.8 Resumen.....                             | 101 |
| 2.9 Contenido de la página Web de apoyo..... | 101 |

### Objetivos

- Conocer los conceptos básicos del Modelo Relacional y los relacionados con la consistencia e integridad en una base de datos.
- Aplicar los operadores del Álgebra y del Cálculo Relacional.
- Lograr el aprendizaje en la escritura de expresiones para relacionar datos.
- Conocer la normalización.

## 2.1 Introducción

El modelo de datos relacional —perteneciente al grupo de modelos de datos orientados a registros— es hoy el modelo de mayor uso y difusión en los distintos tipos de organizaciones, aunque con importantes cambios y adecuaciones realizados a través del tiempo.

El Dr. Edgar Frank Codd produjo —tras la publicación de uno de sus principales trabajos— el gran cambio en los modelos de datos existentes, en el cual expresaba conceptos que no perdieron su vigencia y que constituyen un pilar de conocimientos.

A partir de 1960, este matemático inglés —que emigró a EE. UU. para trabajar como investigador en IBM en 1949—, concentró su atención en la gestión de bases de datos.

Su amigo y reconocido escritor, Christopher Date —autor, entre otras, de la obra *Introducción a los Sistemas de Bases de Datos*— hace un tributo después de la muerte del Dr. Codd diciendo: “El Modelo Relacional es ampliamente reconocido como una de las grandes innovaciones del siglo xx”, y continúa: “El modelo relacional proporcionó un marco teórico en el que una variedad de problemas importantes podrían ser atacados de una manera científica. Ted describió por primera vez su modelo en 1969 en un informe de investigación de IBM: “Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks” (IBM Research Informe RJ599 - 19 de agosto 1969).

En junio de 1970, el Dr. Codd publicó un paper —que es el más conocido— con el título *A Relational Model of Data for Large Shared Data Banks*, que significa: “Un modelo relacional de datos para grandes bancos de datos compartidos”. Uno de los objetivos centrales de este artículo se expresó de la siguiente manera: “Los futuros usuarios de los grandes bancos de datos deben ser protegidos de tener que saber cómo se organizan los datos en la máquina (la representación interna)”.

De su afirmación se podría interpretar que deseaba alejar a los usuarios del nivel físico; es decir: elevar el nivel de abstracción con el que debían interactuar los usuarios y que no dependieran de los detalles técnicos para usar los datos almacenados.

Por su parte, Date consideró que “Ted no solo inventó el modelo relacional en particular, él inventó el concepto de modelo de datos en general”. (Ver su paper *Data Models in Database Management*, ACM SIGMOD Record 11 , No. 2 February 1981).

El modelo relacional logró la adhesión inicial de otros expertos, porque apuntaba a resolver el problema de grandes bases de datos compartidas.

Un ejemplo claro es el trabajo de Michael Stonebraker, que puede visualizarse en el paper *Retrospection on a Database System*. En el cual se describe la historia de la implementación de Ingres, desde marzo de 1973, con la primera implementación en septiembre de 1975 y cómo continuaron. Dicho paper está disponible en:

<http://ece.ut.ac.ir/classpages/f84/AdvancedDatabase/Paper/p225-stonebraker.pdf>

Edgar “Ted” Frank Codd. (23 de agosto de 1923 - 18 de abril de 2003). Científico informático inglés conocido por sus aportes a la teoría de bases de datos relacionales.

Michael Stonebreaker. Científico especializado en la base de datos de investigación y desarrollo, reconocido por su colaboración en la creación de la mayoría de las bases de datos relacionales del mercado. Además, es fundador de Ingres, Cohera, StreamBase, Vertica, VoltDB, SciDB, entre otros.

### 2.1.1 Conceptos básicos en el modelo relacional

El concepto fundamental, en el Modelo Relacional, es que los datos se representan de una sola manera, en el nivel de abstracción que es visible al usuario, y es, específicamente, como una estructura tabular —conformada por filas y columnas— o como una tabla con valores.

| Relación Estudiantes |          |                |         |          |             |           |
|----------------------|----------|----------------|---------|----------|-------------|-----------|
| Legajo               | Apellido | Nombres        | TipoDoc | NroDoc   | IdLocalidad | Sexo      |
| 11904                | González | Ana Carolina   | DNI     | 35900342 | 31          | Femenino  |
| 13789                | López    | Hugo Sebastián | DNI     | 29762007 | 27          | Masculino |
| <b>1</b>             |          |                |         |          |             |           |

Columnas

A esta estructura se la denomina formalmente relación aunque, en lo informal y en la práctica, se la conoce con el nombre de tabla. La diferencia está en el grado de abstracción ya que la relación cuenta con ciertas propiedades —que se verán más adelante— y la relación ya es la implementación, en la que no necesariamente se cumple con todas las propiedades de la relación, porque depende del diseño que se propone y que se implementa.

Los conceptos básicos que se utilizarán, a partir de este momento, son propios del modelo y se resumen en:

- Relación: no es más que una representación en dos dimensiones, o de doble entrada, constituida por filas, o tuplas y columnas o atributos. Es el concepto primitivo y fundamental del modelo. La relación debe cumplir con un conjunto de restricciones o propiedades que ya han sido mencionadas. Dentro del diseño de la base de datos, las relaciones representan a las entidades que se modelaron. Las entidades poseen atributos que las distinguen y cada uno de ellos está ligado a un dominio en particular.
- Fila o tupla: es un hecho en la relación que contiene datos de la realidad. Por ejemplo: los datos de un alumno en particular forman una tupla de la relación "Estudiantes"; los datos de un artículo pueden ser una tupla en una relación de la base de datos de una "Empresa de Ventas o Producción", etcétera. La tupla en una relación es una colección ordenada de elementos diferentes. Cada tupla es también una única combinación de estos elementos; por ende, las tuplas no se repiten dentro de la misma relación.
- Cuerpo: al conjunto de tuplas de una relación se lo denomina cuerpo de la relación. El cuerpo de la relación "Estudiantes" tiene dos tuplas.



Los datos se representan como una estructura tabular —conformada por filas y columnas— o como una tabla con valores.

- Columna o atributo: es una propiedad que caracteriza a cada entidad, como el color o tamaño de un artículo, el apellido o el legajo de un estudiante, etcétera. Los elementos que componen la estructura de la entidad se denominan atributos y serán irrepetibles en la entidad.
- Cabecera: el conjunto de atributos de una relación conforma la cabecera de dicha relación. La cabecera de la relación "Estudiantes" es:

| Legajo | Apellido | Nombres | TipoDoc | NroDoc | IdLocalidad | Sexo |
|--------|----------|---------|---------|--------|-------------|------|
|--------|----------|---------|---------|--------|-------------|------|

- Dato: es la mínima unidad que se almacena en una relación, indivisible en el concepto original del modelo almacenado en la intersección de una fila y de una columna. Por ejemplo: 35900342 es un número de documento de un alumno en la relación "Estudiantes".
- Grado: se llama así al número de columnas que conforman la relación. Es estático en el tiempo, aunque se modificará ante una necesidad de cambio dado por la realidad de la organización. En la relación "Estudiantes", el grado es 7 (siete).
- Cardinalidad: así se denomina al número de tuplas o filas de una relación. Es dinámica, ya que depende del agregado o la eliminación de filas a través del tiempo. La relación "Estudiantes" tiene cardinalidad 2 (dos).
- Dominio: es el conjunto de valores posibles de un atributo en la relación. Ejemplos:
  - Sexo: en la relación "Estudiantes" asume dos valores posibles (Femenino, Masculino).
  - Legajo se puede definir como un dato que puede asumir valores numéricos enteros positivos, sin incluir el cero.

La fecha de factura en una determinada relación, que puede asumir valores tipo fecha y que nunca será menor a la fecha de creación de la empresa que factura o de la sucursal.

Cada atributo se representa en un único dominio. La definición de cada dominio es conceptual en el momento de la definición lógica y se refiere a la cualidad del atributo en relación con la entidad.

El dominio es la restricción de lo que se quiere que el atributo represente dentro de la entidad. En esta instancia conceptual, el dominio se refiere a una cualidad o conceptualización del dato que se conoce y se quiere representar dentro de la entidad. Por ejemplo: cuando se establece dentro de la entidad "Personas" un atributo "Número de Documento", se está definiendo un atributo que refleja esta información; es decir: solo el número de documento. Este dominio abarca los números enteros en un rango que va de uno a infinito. En esta etapa, no interesa de qué forma se almacenará y qué tipo de limitaciones tendrá para esta acción; es decir: se quita la importancia al formato de almacenamiento, que será materia de análisis en una instancia posterior al modelado de una base de datos relacional. La definición del tipo de valor alojado en la columna, ya que se refiere al modelo físico, también será denominada dominio, pero en otra perspectiva de análisis y contexto.



El dominio es la restricción de lo que se quiere que el atributo represente dentro de la entidad. En esta instancia conceptual, el dominio se refiere a una cualidad o conceptualización del dato que se conoce y se quiere representar dentro de la entidad.

## 2.1.2 Propiedades de las relaciones

Estas propiedades fijan limitaciones en el modelo, que son las que diferencian las relaciones de las tablas implementadas. Tienen su origen en la Matemática de Conjuntos. El Dr. Codd las definió en su publicación *A Relational Model of Data for Large Shared Data Banks*.

1. Orden de las tuplas en una relación: como una relación es un conjunto de tuplas y el concepto en Matemática indica que los conjuntos no son ordenados, las tuplas en la relación no están ordenadas. "Codd dice que el orden de las filas es inmaterial". Aquí, se marca una diferencia con las relaciones, ya que, una vez creadas, comienzan a almacenar las filas con algún orden y por algún criterio. En las relaciones hay una primera y una última fila; en las relaciones no se contempla esta posibilidad.
2. Orden de los atributos: como se dijo más arriba, la cabecera de una relación es un conjunto de atributos y, como elementos de un conjunto, no pueden estar ordenados, ni de izquierda a derecha ni de derecha a izquierda. En la práctica, suelen ubicarse en primera instancia los atributos que conforman la identificación de las relaciones —lo que más adelante se verá como claves—, aunque no es un mandato que se deberá cumplir, sino una costumbre o coincidencia de los diseñadores de bases de datos.
3. Todas las filas son distintas: así enuncia Codd a esta propiedad que surge, también, de la teoría de conjuntos matemáticos, puesto que los elementos de un conjunto no están duplicados: son todos distintos. En las relaciones —marcando otra diferencia con las relaciones— puede haber duplicidad de filas, pero por un error comúnmente dado por mala selección de esos elementos, que se llaman claves.
4. Un valor único en la intersección de fila y columna: en la definición original del modelo relacional, se dijo que toda relación está normalizada —concepto que se asocia a la técnica de diseño de bases de datos y que veremos más adelante—, con lo que no era aceptada la idea de que dicho valor pudiera ser una estructura divisible. En la actualidad, hay productos comerciales que dejan de lado esta propuesta y ofrecen la posibilidad de incluir relaciones dentro de una relación, con la intención de permitir almacenar estructuras de datos más complejas.

## 2.1.3 Reglas de Codd

Entre los grandes aportes que hizo el Dr. Codd —desde la creación del modelo relacional— se encontraban una serie de reglas, denominadas Reglas de Codd, que buscaban fijar conceptos para asegurar que un DBMS fuera relacional o RDBMS.

En 1985, Codd publicó —en la revista *Computer World*— dos artículos: "Is Your DBMS Really Relational?" (14 de octubre de 1985) y "Does Your DBMS Run By the Rules?" (21 de octubre de 1985). Allí, sostuvo que escribió las mencionadas reglas porque veía que muchas empresas creaban un DBMS como si gestionaran bases de datos y no resguardaban los conceptos originales.

Si bien se habla de las doce Reglas de Codd, existen trece reglas numeradas por otros autores, de la 0 (cero) a la 12 (doce).



### Propiedades de las relaciones:

1. Orden de las tuplas en una relación.
2. Orden de los atributos.
3. Todas las filas son distintas.
4. Un valor en la intersección de fila y columna.

El Departamento de Ciencias de la Computación e Ingeniería de la Universidad del Estado de Ohio publica:

- **Regla 0: Regla de fundación**  
Un sistema de gestión de base de datos debe gestionar sus datos almacenados utilizando solo sus capacidades relacionales.
- **Regla 1: Regla de la información**  
Toda la información en la base de datos debe estar representada en una y solo una forma: como valores en una relación.
- **Regla 2: Regla de acceso garantizado**  
Todos y cada uno de los datos (valor atómico) están garantizados para ser lógicamente accesibles recurriendo a una combinación de nombre de la relación, valor de clave primaria —atributo cuyas características serán desarrolladas a continuación de este tema— y nombre de la columna.
- **Regla 3: Tratamiento sistemático de valores nulos**  
Los valores nulos (concepto distinto de cadena de caracteres vacía o de una cadena de caracteres en blanco y distinto de cero, o de cualquier otro número) se soportan en el DBMS relacional para la representación de la información que falta en una manera sistemática, independiente del tipo de dato. Los valores nulos serán tratados en el Modelo Relacional y en SQL.
- **Regla 4: Catálogo dinámico en línea basado en el modelo relacional**  
La descripción de la base de datos está representada, en el nivel lógico, de la misma manera que los datos ordinarios, de modo que los usuarios autorizados puedan aplicar el mismo lenguaje relacional para consultar que el aplicado a los datos regulares. El concepto de catálogo o diccionario de datos será tratado entre los siguientes temas.
- **Regla 5: Regla del sublenguaje de datos completa**  
Un sistema relacional puede admitir varios lenguajes y modos de uso de terminales. Sin embargo, debe haber al menos un lenguaje cuyas sentencias se puedan expresar por una sintaxis bien definida, como una cadena de caracteres y que soporte lo siguiente:
  - a. Definición de datos.
  - b. Definición de vista.
  - c. Manipulación de datos (interactiva y por programa).
  - d. Restricciones de integridad.
  - e. Autorización.
  - f. Límites de la transacción (iniciar —*BEGIN*—, realizar —*COMMIT*— y deshacer —*ROLLBACK*—).
- **Regla 6: Regla de actualización de vistas**  
Por el sistema se deben actualizar todas las vistas que son teóricamente actualizables.
- **Regla 7: Inserción, actualización y borrado de alto nivel**  
La capacidad de manejar una relación base o derivada como un simple operando, se aplica no solo a la recuperación de datos sino también a la inserción, la actualización y el borrado de datos.
- **Regla 8: Independencia Física de Datos**  
Los programas de aplicación y las actividades de terminal permanecen

lógicamente inalterables, siempre que se realicen cambios en las representaciones de almacenamiento o de métodos de acceso.

- **Regla 9: Independencia lógica de datos**  
Los programas de aplicación y las actividades de terminal permanecen lógicamente inalterables cuando los cambios preservan la información en las relaciones bases.
- **Regla 10: Independencia de integridad**  
Las reglas de Integridad, específicas de las bases de datos relacionales, se deben definir en un sublenguaje de datos y almacenar en el catálogo o diccionario de datos, no en los programas de aplicación. Estas reglas se definen y ejemplifican con los conceptos de claves.
- **Regla 11: Independencia de distribución**  
El sublenguaje de manipulación de datos de un DBMS relacional debe permitir que los programas de aplicación y actividades de terminal permanezcan lógicamente intactos, cuando los datos están físicamente centralizados o distribuidos.
- **Regla 12: Regla de no subversión**  
Si un sistema relacional tiene o soporta un lenguaje de bajo nivel (un registro a la vez), ese lenguaje no se puede usar para subvertir o saltar las reglas de integridad o las reglas expresadas en el lenguaje relacional de alto nivel (múltiples registros por vez).

#### 2.1.4 Atributos claves

En todas las relaciones de una base de datos relacional, hay atributos que cumplen determinadas condiciones y, en función de ellas, se clasifican.

A continuación, se describirán las condiciones y los tipos de claves que se conocen, aunque algunos son solo conceptos que no se implementan.

#### 2.1.5 Condiciones de las claves candidatas

Para que un atributo o conjunto de atributos de una relación sea clave candidata, deben cumplirse las siguientes condiciones:

- **Unicidad:** indica la no repetición del valor de un atributo en la vida de una relación. Vale aclarar que la clave candidata puede ser compuesta por más de un atributo; entonces, en ese caso, debe buscarse la combinación de atributos que, en conjunto, nunca asuman valores repetidos, es decir, que sean únicos. Si está bien elegida, permite que nunca haya una fila duplicada en una relación.
- **Minimalidad:** se llama así a la condición que se debe considerar cuando se elige una clave candidata compuesta por más de un atributo y que indica que será la mínima combinación de atributos posibles que cumpla con la condición de unicidad.

Para ejemplificar estos conceptos, se verá la cabecera de la relación presentada en la relación “Estudiantes”:

| Legajo | Apellido | Nombres | TipoDoc | NroDoc | IdLocalidad | Sexo |
|--------|----------|---------|---------|--------|-------------|------|
|--------|----------|---------|---------|--------|-------------|------|



Para que un atributo o conjunto de atributos de una relación sea clave candidata, deben cumplirse dos condiciones:

1. Unicidad.
2. Minimalidad.

Algunas situaciones que surgen son:

- El atributo “Legajo” del Estudiante cumple con la condición de unicidad, ya que no debería haber en la universidad dos alumnos con igual número de legajo. No se puede decir lo mismo de “Apellido” o de Nombres. También se debe asegurar que sea mínima —que, en este caso, se cumple—, puesto que no se la puede dividir. Entonces, “Legajo” es una clave candidata.
- La combinación “TipoDoc” y “NroDoc” puede cumplir también con la condición de unicidad, si se confía plenamente en los organismos emisores y de control que deben asegurar que esa combinación sea única. ¿Es mínima? Sí, también lo es, porque, si bien es compuesta, no se pueden separar los atributos, sin perder la condición de unicidad. Entonces, se está frente a una clave candidata compuesta por dos atributos.
- La combinación de “Legajo” y “NroDoc” cumplen con la condición de Unicidad, pero no es mínima, porque si se quitara el “NroDoc”, seguiría cumpliendo con la condición de unicidad.

Es decir que esta relación tiene dos claves candidatas: “Legajo” y “TipoDoc+NroDoc”.

Hay casos en los que todos los atributos de la relación combinados conforman la clave candidata, para cumplir con la condición de unicidad.



La clave primaria es la clave candidata que el diseñador elige para identificar a cada entidad que se almacenará en una base de datos relacional, dentro de las opciones que pudieran presentarse al analizar los datos.

#### 2.1.6 Claves primarias

Estas claves asumen una gran importancia en el diseño de la base de datos, dado que, si no se determinan bien los datos almacenados, no se identificarán de manera inequívoca y provocarán la repetición de tuplas, que generarán problemas demasiado costosos para resolver en un sistema en producción.

La clave primaria es la clave candidata que el diseñador elige para identificar a cada entidad que se almacenará en una base de datos relacional, dentro de las opciones que pudieran presentarse al analizar los datos. Es decir que la clave primaria cumple con las condiciones exigidas a la clave candidata, esto es unicidad y minimalidad.

La decisión de cuál clave candidata se elegirá como clave primaria depende, normalmente, del criterio y de la experiencia del diseñador y, también, del “mundo real” de la organización en la cual se utilizará la base de datos.

En el ejemplo que se analizó en el punto anterior, se debe decidir entre dos claves: la candidata simple “Legajo” y la candidata compuesta “TipoDoc+NroDoc”.

La clave primaria es la forma o mecanismo de identificación de una instancia dentro de una entidad. Expresado de otra forma, es la forma de identificación de una tupla dentro de una relación. Definida como un atributo, o conjunto de ellos, permite establecer la localización inequívoca de la tupla dentro de la relación, o de una instancia dentro de una entidad. Esta definición es constante para toda la entidad y es la misma para todas las ocurrencias posibles de tuplas. Si, por algún motivo, esta definición no satisficiera la condición de combinación irrepetible, no sería, entonces, la clave primaria.

Si la clave primaria es el atributo—o una colección de éstos—, que permite identificar, en forma inequívoca, a la tupla dentro de la relación; entonces, ningún atributo de los que componen la clave primaria puede provocar incertidumbre. Todos los valores que toman los atributos de la clave primaria deben ser ciertos.

El conjunto de atributos que componen la clave primaria establece una relación única con el conjunto de atributos no clave.

La clave candidata que no se elige suele denominarse clave alternativa o alterna, pero no se implementa; a lo sumo, se la define de manera tal que, en una relación, no se repitan los valores. Por ejemplo, si se eligiera “Legajo” como clave primaria, sería deseable que se defniera a la combinación “TipoDoc+NroDoc” de forma tal que nunca se repitieran ambos valores de manera simultánea.

Cabe aclarar que una relación nunca podrá tener más de una clave primaria y, salvo casos excepcionales, no cambiará en toda la vida del sistema.

#### 2.1.7 Clave foránea

La clave foránea o ajena es también muy importante en un sistema relacional, ya que es un atributo, o combinación de atributos, que permite la combinación de datos de las distintas relaciones del sistema.

Por ejemplo: el atributo “IdLocalidad”, en la relación “Estudiantes”, puede ser clave foránea. Esto significa que existirá una relación “Localidades” a la que referencia y, dicha relación, deberá contener un identificador de localidades —clave primaria— para establecer la relación necesaria.

Si la clave foránea fuera compuesta por más de un atributo, deberá referenciar a la misma composición en la relación referenciada.

| Relación Localidades |                     |             |
|----------------------|---------------------|-------------|
| IdLocalidad          | Descripción         | IdProvincia |
| 27                   | Ciudad Buenos Aires | 4           |
| 31                   | Córdoba             | 7           |
| 30                   | Villa María         | 7           |
| 33                   | Godoy Cruz          | 2           |
| 25                   | Villa Mercedes      | 8           |
| 20                   | Villa Dolores       | 7           |
| 36                   | Mar del Plata       | 4           |
| 22                   | Cruz del Eje        | 7           |
| 21                   | San José            | 9           |
| 34                   | Las Heras           | 2           |
| 29                   | Oberá               | 9           |
| 32                   | Tunuyán             | 2           |
| 37                   | La Plata            | 4           |

Las relaciones “Estudiantes” y “Localidades” se relacionan a través del atributo “IdLocalidad”. Asimismo, en “Localidades” existe el atributo “IdProvincia”, con un número. Para obtener la descripción de la provincia, deberá referenciar a otra relación, en este caso, “Provincias”, lo que hace que el atributo “IdProvincia” de la relación “Localidades” sea clave foránea o ajena.



La clave candidata que no se elige suele denominarse clave alternativa o alterna, pero no se implementa; a lo sumo, se la define de manera tal que, en una tabla, no se repitan los valores.



La clave foránea o ajena es un atributo, o combinación de atributos, que permite la combinación de datos de las distintas relaciones del sistema.

La clave foránea es el mecanismo que establece la dependencia de una entidad con otra.

| Relación Estudiantes |          |                   |         |          |             |           |
|----------------------|----------|-------------------|---------|----------|-------------|-----------|
| Legajo               | Apellido | Nombres           | TipoDoc | NroDoc   | IdLocalidad | Sexo      |
| 11904                | González | Ana Carolina      | DNI     | 35900342 | 31          | Femenino  |
| 13789                | López    | Hugo Sebastián    | DNI     | 29762007 | 27          | Masculino |
| 25784                | Acuña    | Juan Martín       | DNI     | 33457821 | 30          | Masculino |
| 23892                | Salinas  | Rubén             | DNI     | 32890128 | 31          | Masculino |
| 19098                | Romero   | María Amparo      | DNI     | 28089727 | 27          | Femenino  |
| 24561                | Moreno   | Santiago Rubén    | DNI     | 30892109 | 31          | Masculino |
| 13490                | Funes    | Elizabeth         | DNI     | 32782340 | 33          | Femenino  |
| 25801                | Sosa     | Marcelo Alejandro | DNI     | 29872301 | 27          | Masculino |
| 12802                | López    | Analía            | DNI     | 27001287 | 31          | Femenino  |

Esta dependencia es desde la entidad en la que se define la clave foránea hacia la entidad que posee una clave primaria que satisface la dependencia.

Por lo expresado anteriormente, toda clave foránea de una entidad requiere de la existencia de una clave primaria en otra entidad.

| Relación Provincias |              |
|---------------------|--------------|
| IdProvincia         | Descripcion  |
| 4                   | Buenos Aires |
| 7                   | Córdoba      |
| 2                   | Mendoza      |
| 9                   | Misiones     |
| 5                   | Entre Ríos   |
| 8                   | San Luis     |

# S

La clave foránea es el atributo o combinación de atributos, que existe en una relación y que es la clave primaria en la otra relación.

La clave foránea, entonces, es el atributo o combinación de atributos, que existe en una relación y que es la clave primaria en la otra relación.

Al contrario de lo expuesto en la definición de clave primaria, una clave foránea sí puede poseer un valor desconocido. Esto significa que la relación no se encuentra establecida y no causa distorsión. Esta dependencia podría reanudarse en cualquier momento.

Es conveniente aclarar que una relación puede tener una, muchas o ninguna clave foránea.

## 2.1.8 Marcador de valores desconocidos

En la Regla 3, Codd menciona los valores nulos o desconocidos.

No debe relacionarse el valor 0 (cero), de las Matemáticas, con los valores desconocidos o también denominados valores nulos. Cero es un valor matemático, mientras

que nulo es un marcador que indica la ausencia del dato en una base de datos relacional. Por ejemplo, la nota de un estudiante puede ser 0 (cero), que no es lo mismo que no conocerla.

El concepto fue introducido por el Dr. Codd para representar la ausencia de información en el almacenamiento de datos.

Por ejemplo:

- Un empleado nuevo puede no tener su domicilio en la ciudad donde está instalada la empresa. El valor para el atributo “Domicilio”, en la relación “Empleados”, no puede tomar algún valor, por lo que se asume que ese valor es desconocido por un determinado tiempo.
- La fecha de nacimiento del nuevo empleado también se puede desconocer en una primera instancia.
- El número de sección a la que se asigna un nuevo empleado, también se puede desconocer en una instancia inicial, hasta que se incluya al empleado en un área o sección, de acuerdo con sus conocimientos.

En los casos mencionados más arriba, la intención fue ejemplificar, a través de diferentes tipos de datos —como carácter, fecha y número—, que el valor desconocido no depende del tipo de dato en la base de datos.

La clave foránea de una relación puede asumir un valor desconocido, por ejemplo: cuando un estudiante ingresa en la universidad podría suceder que no presentara su partida o registro de nacimiento. El operador, que carga sus datos en la base de datos, no determinará la localidad donde nació y, en ese caso, el atributo “IdLocalidad” almacenará un valor desconocido.

En SQL manearemos NULL para representar un valor desconocido en la implementación de una base de datos relacional.

### 2.1.9 Correspondencias

En virtud de lo expresado en los párrafos anteriores, se hará la siguiente asociación de correspondencia de términos entre los distintos momentos o etapas en el diseño de una base de datos relacional:

| Relación Momentos o etapas en el diseño de una base de datos relacional |                       |                          |
|---|-----------------------|--------------------------|
| Diseño conceptual   | Modelo relacional     | Base de datos relacional |
| Entidad   | Relación              | Tabla o relación         |
| Instancia   | Tupla                 | Tupla o fila             |
| Atributo  | Atributo              | Atributo o columna       |
| Dominio   | Dominio               | Dominio                  |
| Identificación inequívoca   | Clave primaria        | Clave primaria           |
| Dependencia funcional   | Clave foránea o ajena | Clave foránea o ajena    |
| Valor desconocido   | Valor desconocido     | Valor nulo               |



La regla de integridad de las entidades se basa en las claves primarias de cada tabla y de todas las tablas del modelo relacional.

### 2.1.10 Reglas de integridad

Una de las principales ventajas que ofreció el modelo relacional —y que se implementó desde el principio en los RDBMS— fue la utilización de las reglas de integridad. Por esta razón, el Dr. Codd las incluyó en la Regla 10, para asegurar que todo RDBMS brindara las herramientas necesarias para defñirlas.

Se puede afirmar que son restricciones que se aplican a los datos, en función de los conceptos de las bases de datos relacionales y de las organizaciones en las que se implementan. Además, aseguran que la base de datos no almacene valores inválidos para una organización que implementa un sistema informático basado en el modelo relacional.

Existen tres reglas que se definen en el modelo relacional: la regla de integridad de las entidades, la regla de integridad referencial y las reglas de negocios o comerciales.

**Regla de integridad de las entidades:** se basa en las claves primarias de cada relación y de todas las relaciones en el modelo relacional. Exige que la clave primaria no asuma, nunca, un valor desconocido. Ya que, la clave primaria —que se elige para identificar a cada entidad almacenada— no puede ser desconocida.

A esta regla —y, también, las que se tratarán a continuación—, se la define desde la creación de la relación con el DBMS elegido. Una de las tareas del DBA es la de realizar la especificación lógica de la base de datos, que se definirá al especificar cuál será la clave primaria de cada relación creada.

Por ejemplo, en la relación “Estudiantes”, se quiere significar que, si se eligiera a “Legajo” como clave primaria, dicho atributo nunca asumirá un valor desconocido —esto es tan lógico como importante—, puesto que, al ingresar un nuevo estudiante tendrá un legajo para que lo identifique, de manera única, en la universidad. Ese legajo lo acompañará y se asociará con todas las actividades que desempeñe como alumno, por ejemplo: en los exámenes, en el retiro de libros de la biblioteca, en las sanciones, en los pagos, etcétera.

**Regla de integridad referencial:** esta regla se define sobre la base de las claves foráneas y restringe las relaciones entre relaciones.

Indica que cada valor de una clave foránea debe existir como valor en la clave primaria de la otra relación o ser desconocido.

Por ejemplo:

- En el momento de registrar el lugar de nacimiento de un estudiante en la relación “Estudiantes”, ese valor ya debe existir de antemano como valor en la relación “Localidades”.
- Una empresa solo puede comercializar productos que tiene registrados; entonces, en una factura no debe haber un producto que no esté registrado en la base de datos. Lo registrado en una relación, como “Productos” de la base de datos, es un refejo fiel del depósito de productos físicos.

Esto también es muy lógico y se especifica al crear las tablas, cuando se indica cuál es el atributo que conforma la clave foránea y a qué tabla y a qué atributo de ella referencia. A partir de allí, se dejará “en manos” del DBMS el control de lo citado.



La regla de integridad referencial se define sobre la base de las claves foráneas y restringe las relaciones entre tablas.

Reglas de negocios o comerciales: cada organización define sus reglas de negocio, según su realidad y en función de los objetivos que persigue, por lo que estas reglas son específicas de cada empresa y suelen denominarse, también, como comerciales o específicas.

El Equipo de Análisis y Diseño identifica estas reglas y el DBA las implementa.

Por ejemplo:

- Una entidad bancaria define los montos mínimos para que un cliente abra y mantenga una cuenta en caja de ahorros o de plazo fijo. En un banco, el mínimo para abrir una caja de ahorro puede ser \$1000 y en otro \$5000.
- Una provincia, país o entidad define cuál es la nota que determina la aprobación, o no, de una instancia de evaluación en su sistema educativo. Por ejemplo: cuatro puede ser la nota de aprobación mínima en una universidad y seis en otra. En todos los casos, es un valor numérico; pero, algunas entidades almacenan valores con decimales y otros solo con enteros.
- Se puede definir que la fecha de ingreso de un empleado nunca sea menor a la fecha actual o, al menos, que no sea inferior a la fecha de creación de la empresa.
- Los tipos de documentos de las personas, para cualquier tipo de sistema, son un conjunto bien determinado. Ese conjunto se define para ese dato en la base de datos. De esta manera, se asegurará que nunca habrá un tipo de documento fuera de ese dominio.

En resumen, se puede decir que las tres reglas de integridad se definen con la creación de cada tabla, en la que se especifican los atributos —con los valores válidos para cada uno de ellos— y los tipos de datos. Las dos primeras, la regla de integridad de las entidades y la regla de integridad referencial se asumen para todas las bases de datos relacionales, sin que importe la realidad de la empresa. Cuando se especifican las claves primarias de las relaciones y las claves foráneas, en la creación de la base de datos, el RDBMS comienza a aplicarlas.

En la sección dedicada a SQL, se verán ejemplos concretos al crear la base de datos.

En otras épocas, este trabajo estaba solo en manos de los programadores, quienes debían incorporar las reglas en la programación. Si lo olvidaban en alguno de los programas desarrollados, generaban errores en la base de datos. También estaba el esfuerzo de la programación que se multiplicaba a medida que se incorporaban más archivos de datos y múltiples programas en los que se agregaban estos controles antes de acceder a los datos.

En la actualidad, las aplicaciones que se desarrollan controlan el ingreso de datos, con rutinas propias; pero, por su lado, los RDBMS aseguran al propietario de la empresa que, si las reglas están bien definidas, no habrá datos inválidos en su base de datos.

### 2.1.11 Diccionario de datos

El diccionario de datos, también conocido como catálogo del sistema, es una base de datos que es permanentemente administrada y modificada por el DBMS. Esta base de datos se construye con todas las características del modelo relacional;

# S

El diccionario de datos es una base de datos que es permanentemente administrada y modificada por el DBMS. Se construye con todas las características del modelo relacional; es un conjunto de relaciones que poseen claves primarias, foráneas, restricciones, etcétera.

es decir que es un conjunto de relaciones y éstas poseen claves primarias, foráneas, restricciones, etcétera.

Cuando el DBA modifica la estructura de una tabla, perteneciente a alguna base de datos, el DBMS refleja estos cambios en el diccionario de datos. Es decir que toda sentencia escrita en Lenguaje de Definición de Datos (DDL) modifica su contenido.

Este diccionario asegura el buen funcionamiento del DBMS y contiene información muy importante para un DBA, por ejemplo, la relacionada con:

- Bases de datos que contienen información sobre quién la creó y cuándo, etcétera.
- Objetos que conforman cada base de datos (vistas, tablas, índices, procedimientos almacenados, etc.), que incluyen la fecha de creación, la definición estructural de cada uno, las relaciones con otros objetos, el propietario del objeto, etcétera.
- Restricciones y reglas de integridad definidas por el DBA, que incluyen su definición, los datos sobre los que actúa, las claves definidas, los dominios de los datos, etcétera.
- Usuarios que pueden acceder a cada base de datos, clave, objetos con los que puede trabajar y los privilegios sobre esos almacenamientos (como modificar, o solo leer).

La Regla 4 de Codd dice que los usuarios autorizados deben poder aplicar un lenguaje relacional para consultar el contenido del diccionario, puesto que toda la información está en relaciones. Normalmente, solo el DBA accede con sentencias de consulta sobre los datos, ya que el diccionario es solo lectura.

Entre los distintos DBMS del mercado, hay diferencias en cuanto a la implementación del diccionario, por ejemplo, en los nombres de las relaciones que conforman la base de datos. Sin embargo, comparten la estructura general del diccionario y el tipo de información que se almacenará.

## 2.1.12 Diseño de software utilizando bases de datos relacionales

Las bases de datos relacionales han tomado el liderazgo de las soluciones de almacenamiento y gestión de la información para las aplicaciones.

Las ventajas que ofrecen —seguridad, consistencia, gestión del acceso concurrente por parte de cientos y miles de usuarios simultáneamente, acceso remoto, interoperabilidad, entre otras— hacen que sea la elección casi obligada para contener la información de un sistema informático. Distinto es el caso de lenguajes y arquitecturas para construir los programas de los sistemas que atiendan los requerimientos de información de una empresa.

De esta manera, surge, como primera opción, el modelo o arquitectura Cliente/Servidor, cuyo código de aplicación se ejecuta en una máquina diferente al servidor de la base de datos. Las llamadas a la base de datos, con las sentencias SQL transmitidas por el cliente, son procesadas por el servidor quien devuelve el resultado, que consiste en flas o una señal que avisa que la base se ha actualizado con las sentencias enviadas. En el cliente, las flas o la señal de la tarea realizada es procesada por programas escritos en lenguajes como C, C++, Java, COBOL, y otros.

Este modelo de ejecución de aplicaciones se suele denominar de dos capas o two-tier.

El otro modelo de ejecución de aplicaciones consiste en almacenar la lógica en la misma base de datos y que se ejecuten automáticamente a través de disparadores o con funciones y procedimientos llamados explícitamente por servidores Web de aplicaciones desde páginas de interfaz de usuarios escritas en lenguajes como PHP, JAVA, en su extensión JSP, u otros. Este modelo es denominado de tres capas o three-tier porque se agrega la capa correspondiente al servidor Web nombrado anteriormente, que envía los requerimientos de datos o acciones sobre los datos a la base de datos y reenvía el resultado de parte de ésta para las aplicaciones. Además, este servidor puede agregar funciones avanzadas, como el uso de memoria temporal (caché) y balance de carga de trabajo, que son importantes en aplicaciones Web, las cuales pueden atender a cientos o a miles de sesiones simultáneamente, como en sitios Web que atienden a esta cantidad de visitantes en un portal de comercio electrónico o de noticias. Esta capa intermedia recibe un peso importante del proceso requerido, y deja al cliente la tarea de presentación de la información y, como esta carga es más liviana que la que realiza el cliente en el modelo Cliente/Servidor, se lo denomina como cliente delgado o thin client. Éste requiere solamente un Web browser para enviar los requerimientos sobre los protocolos TCP/IP o HTTP con los que se vincula al resto de las capas. Las interfaces de usuario de la aplicación que se mostrarán al usuario dependen de la tecnología que soporta el funcionamiento de todo el modelo y pueden estar soportadas por HTML y Java.

En las aplicaciones tradicionales Cliente/Servidor, la aplicación puede mantener un registro de las acciones del usuario y usar esta información para llevar a cabo una o más sesiones, como, por ejemplo, algunas opciones elegidas anteriormente se pueden presentar en un menú, de manera que no deben ser ingresadas nuevamente. Si la aplicación puede almacenar esta información se considera de estado completo o stateful. En cambio, las aplicaciones Web o de cliente delgado —que son sin estado o stateless— son más fáciles de desarrollar y, generalmente, recopilan toda la información requerida, la procesan usando la base de datos y reanudan el contacto con el siguiente usuario. Ésta es una estrategia común para procesar pantallas simples, como el registro de un estudiante nuevo.

Si es necesario obtener un componente stateful en una aplicación Web —que generalmente es stateless por defecto, como por ejemplo en un formulario de ingreso de una página—, puede pasar información a las páginas siguientes, permitiendo una interfaz del estilo asistente o wizard que va recordando la información cargada por el usuario en los distintos pasos. En este caso, pueden usarse las cookies, que son pequeñas porciones de información que se almacenan en un área de memoria del Web browser en la máquina cliente para ser recuperadas por el sitio Web. También se utilizan usualmente porciones de código almacenados en el servidor, denominados servlets, para mantener la sesión de la base de datos abierta y almacenar variables que soporten los pedidos del mismo usuario durante la interacción con la interfaz de programa.

#### 2.1.13 El acceso a las bases de datos con aplicaciones escritas en JAVA

Java es un lenguaje de propósitos generales que puede conectarse eficientemente a bases de datos relacionales de la misma forma que realiza otras tareas; esto es: usando clases predefinidas que permiten conectarse con servidores de bases de



Java es un lenguaje de propósitos generales que puede conectarse eficientemente a bases de datos relacionales de la misma forma que realiza otras tareas.

datos, enviarles sentencias SQL para que sean procesadas y recibir el resultado de la acción solicitada.

Estas clases se agrupan como una interfaz de programas o API creada por SUN, autor del lenguaje, denominada JDBC, siglas que significan Conectividad con Bases de Datos para Java. Esta forma de conectarse de Java ha sido adoptada por la mayoría de los motores de base de datos, que permiten que las aplicaciones que usan bases de datos sean portables a distintos motores con mínimos cambios sobre el código original. La función de JDBC es un conjunto de clases abstractas que contienen métodos como `executeQuery()`, que realizan la tarea de enviar sentencias y recibir datos, respectivamente.

Como alternativa a JDBC, existe también otra forma de incluir código SQL en los programas fuentes JAVA, que se denomina SQLJ, y que se puede usar del lado del cliente o del servidor.

#### 2.1.14 Acceso a las bases de datos a través de servicios Web

Los servicios Web se definen como un paradigma de computación distribuido para desarrollar aplicaciones Java, que se puede considerar como una alternativa a las anteriores opciones JDBC y SQLJ. Permite la interacción entre aplicaciones a través del lenguaje de marcación extendido XML (eXtended Markup Language) y protocolos Web. Por ejemplo: un vendedor de partes electrónicas publica en la Web una interfaz de programa para enviar pedidos a sus proveedores y para el mantenimiento del inventario de las mismas. Este programa, denominado servicio Web, puede realizar pedidos de reaprovisionamiento a sus proveedores a medida que su stock lo requiera y a medida que se consuman los productos.

Algunos aspectos clave de la tecnología usada en los servicios Web son:

## S

Las reglas de integridad son restricciones que se aplican a los datos, en función de los conceptos de las bases de datos relacionales y de las organizaciones en las que se implementan.

- El lenguaje de descripción de servicios Web, o su sigla en inglés WSDL, que es un formato para crear un documento XML. En éste se describe qué puede hacer un servicio Web y cómo se debe invocar, además de la ubicación, protocolo de transporte y el estilo de invocación.
- La mensajería del protocolo simple de acceso al objeto o Simple Object Access Protocol (SOAP), que es un protocolo de mensajes basado en XML que usa el servicio.
- UDDI (Universal Description, Discovery and Integration) que lista los servicios Web disponibles en Internet y otra información.

Con todos estos elementos descriptos, y otros, se pueden diseñar aplicaciones que almacenen información en tablas relacionales, que usan diversos lenguajes de programación para construir los componentes que conforman un sistema de aplicación como: menús, pantallas de carga de información, programas que piden parámetros de selección al usuario y luego listar la información en forma de pantallas, reportes que responderán a los requerimientos de información de una operatoria dada, etcétera.

## 2.2 Álgebra relacional

Después de haber visto los conceptos relacionados con el modelo relacional, entre los que se incluyó a las claves y a las reglas de integridad, se verá una serie de operadores que forman el Álgebra relacional y que permiten operar con los datos de las relaciones del modelo.

Conviene aclarar que los RDBMS no implementan a los operadores del Álgebra relacional de manera directa, o como serán usados a continuación, sino que las sentencias SQL incorporan los conceptos y la lógica es común.

El objetivo de la enseñanza del Álgebra relacional es facilitar el aprendizaje de la escritura de sentencias SQL, dado por la manipulación de datos de las relaciones con operaciones.

Como concepto inicial, se dirá que el resultado de aplicar una operación del Álgebra Relacional es otra relación. Esto permite, entonces, la combinación de operaciones y su anidamiento, ya que la entrada de una operación podría ser el resultado generado por otra operación anterior.

Las operaciones se clasifican comúnmente en dos grupos:

- Operaciones de conjuntos: unión, diferencia, intersección y producto cartesiano. Resultan familiares porque se las estudió en la teoría de conjuntos de las Matemáticas, y las relaciones son conjuntos.
- Operaciones relacionales: selección, proyección, reunión, división, agrupación y funciones. Estas operaciones y funciones se crean, específicamente, para la manipulación de datos en el modelo relacional.

Hay una operación auxiliar, que se denomina renombrar ( $\rho$ ), que no se incluye en ninguno de los dos grupos, pero existen situaciones en las que debe utilizarse. Esta operación se describirá y habrá ejemplos más adelante.

En la relación “Resultado” siempre se analiza cómo quedan la cabecera y el cuerpo, pero, también, es necesario controlar que las relaciones de entrada cumplan con las exigencias propias de cada operación.

La simbología en el Álgebra Relacional no es única, y se podrán ver, en otros autores, otras notaciones. En esta obra, se adoptarán los símbolos más utilizados para cada operación.

A modo de ejemplo, se continuará trabajando con las relaciones “Estudiantes” y “Localidades”, y se agregarán las relaciones “Exámenes”, “Materias”, “Docentes” y “NoDocentes”.

Algunas consideraciones antes de comenzar a usar estas relaciones:

- En “Exámenes” está el legajo del alumno y del docente, que es el presidente de la mesa de examen.
- La nota de aprobación, en este ejemplo, es 4 (cuatro).
- No se incluyen las relaciones “Turnos” ni “Carreras”, para simplificar el problema.



El objetivo de la enseñanza del Álgebra relacional es facilitar el aprendizaje en la escritura de sentencias SQL, dado por la manipulación de datos de las relaciones con operaciones.

| Relación Estudiantes |          |                   |         |          |             |           |
|----------------------|----------|-------------------|---------|----------|-------------|-----------|
| Legajo               | Apellido | Nombres           | TipoDoc | NroDoc   | IdLocalidad | Sexo      |
| 11904                | González | Ana Carolina      | DNI     | 35900342 | 31          | Femenino  |
| 13789                | López    | Hugo Sebastián    | DNI     | 29762007 | 27          | Masculino |
| 25784                | Acuña    | Juan Martín       | DNI     | 33457821 | 30          | Masculino |
| 23892                | Salinas  | Rubén             | DNI     | 32890128 | 31          | Masculino |
| 19098                | Romero   | María Amparo      | DNI     | 28089727 | 27          | Femenino  |
| 22941                | Folmer   | Alfonsina         | DNI     | 31892170 | Nulo        | Femenino  |
| 24561                | Moreno   | Santiago Rubén    | DNI     | 30892109 | 31          | Masculino |
| 13490                | Funes    | Elizabeth         | DNI     | 32782340 | 33          | Femenino  |
| 25801                | Sosa     | Marcelo Alejandro | DNI     | 29872301 | 27          | Masculino |
| 12802                | López    | Analía            | DNI     | 27001287 | 31          | Femenino  |

| Relación Localidades |                     |             |
|----------------------|---------------------|-------------|
| IdLocalidad          | Descripción         | IdProvincia |
| 27                   | Ciudad Buenos Aires | 4           |
| 31                   | Córdoba             | 7           |
| 30                   | Villa María         | 7           |
| 33                   | Godoy Cruz          | 2           |
| 25                   | Villa Mercedes      | 8           |
| 20                   | Villa Dolores       | 7           |
| 36                   | Mar del Plata       | 4           |
| 22                   | Cruz del Eje        | 7           |
| 21                   | San José            | 9           |
| 34                   | Las Heras           | 2           |
| 29                   | Oberá               | 9           |
| 32                   | Tunuyán             | 2           |
| 37                   | La Plata            | 4           |

| Relación Provincias |              |
|---------------------|--------------|
| IdProvincia         | DescripProv  |
| 4                   | Buenos Aires |
| 7                   | Córdoba      |
| 2                   | Mendoza      |
| 9                   | Misiones     |
| 5                   | Entre Ríos   |
| 8                   | San Luis     |

| Relación Materias |                                   |         |     |
|-------------------|-----------------------------------|---------|-----|
| IdMateria         | Descripción                       | Carrera | Año |
| 333               | Álgebra y Geometría               | K       | 1   |
| 205               | Análisis de Sistemas              | K       | 2   |
| 306               | Sistemas y Organizaciones         | K       | 1   |
| 290               | Análisis Matemático I             | K       | 1   |
| 459               | Algoritmo y Estructuras de Datos  | K       | 1   |
| 207               | Física I                          | K       | 1   |
| 304               | Matemática Discreta               | K       | 1   |
| 451               | Análisis Matemático II            | K       | 2   |
| 330               | Sintaxis y Semántica del Lenguaje | K       | 2   |
| 289               | Física II                         | K       | 2   |
| 277               | Diseño de Sistemas                | K       | 3   |
| 453               | Gestión de Datos                  | K       | 3   |
| 224               | Matemática Superior               | K       | 3   |

| Relación Exámenes |        |       |      |          |      |        |
|-------------------|--------|-------|------|----------|------|--------|
| IdMateria         | Legajo | Turno | Año  | Fecha    | Nota | LegDoc |
| 306               | 25801  | 9     | 2009 | 07/12/09 | 2    | 22654  |
| 459               | 13789  | 10    | 2009 | 22/12/09 | 8    | 21823  |
| 290               | 19098  | 8     | 2009 | 27/11/09 | 5    | 21722  |
| 333               | 25801  | 1     | 2010 | 03/02/10 | 4    | 21987  |
| 306               | 19098  | 1     | 2010 | 05/02/10 | 4    | 22654  |
| 333               | 13789  | 2     | 2010 | 10/02/10 | 7    | 21987  |
| 306               | 25801  | 2     | 2010 | 12/02/10 | 2    | 22654  |
| 333               | 19098  | 2     | 2010 | 10/02/10 | 7    | 21987  |
| 333               | 24561  | 10    | 2009 | 21/12/09 | 8    | 21987  |
| 306               | 12802  | 1     | 2010 | 05/02/10 | 10   | 22654  |
| 207               | 11904  | 2     | 2010 | 08/02/10 | 2    | 20871  |
| 290               | 25801  | 3     | 2010 | 24/02/10 | 6    | 21722  |
| 207               | 19098  | 2     | 2010 | 08/02/10 | 3    | 20871  |

| Relación Docentes |           |                   |         |          |             |           |
|-------------------|-----------|-------------------|---------|----------|-------------|-----------|
| Legajo            | Apellido  | Nombres           | TipoDoc | NroDoc   | IdLocalidad | Sexo      |
| 23875             | Ríos      | Adela             | DNI     | 22098231 | 32          | Femenino  |
| 22654             | Ruiz      | María Eugenia     | DNI     | 17656209 | 37          | Femenino  |
| 21987             | López     | Edgardo Antonio   | DNI     | 19001291 | 21          | Masculino |
| 20871             | Castro    | Javier            | DNI     | 20605892 | 36          | Masculino |
| 21823             | Funes     | Elizabeth         | DNI     | 19807223 | 27          | Femenino  |
| 21802             | Venier    | Marcelo Ricardo   | DNI     | 21876341 | 22          | Masculino |
| 21805             | Soria     | Liliana del Valle | DNI     | 22898231 | 34          | Femenino  |
| 21722             | Rodríguez | Mario César       | DNI     | 17872349 | 31          | Masculino |

| Relación NoDocentes |          |                   |         |          |             |           |
|---------------------|----------|-------------------|---------|----------|-------------|-----------|
| Legajo              | Apellido | Nombres           | TipoDoc | NroDoc   | IdLocalidad | Sexo      |
| 25002               | Asís     | Jorge Antonio     | DNI     | 20907237 | 31          | Masculino |
| 25409               | Sosa     | Santiago Martín   | DNI     | 19802287 | 31          | Masculino |
| 25675               | Bustos   | Magdalena         | DNI     | 18762108 | 25          | Femenino  |
| 21805               | Soria    | Liliana del Valle | DNI     | 22898231 | 34          | Femenino  |

## 2.2.1 Operaciones de conjuntos

Son las operaciones binarias más simples. Significa que, en la expresión del Álgebra relacional, se incluyen dos relaciones de entrada.

Otra característica de las operaciones de conjuntos es que se consideran primivas, ya que no se pueden escribir combinando otras operaciones del Álgebra relacional.

### 2.2.1.1 Unión

Esta operación recibe dos relaciones de entrada: se simboliza con **u** y exige que dichas relaciones sean compatibles respecto a la unión.

Compatibles respecto a la unión significa que ambas relaciones deben tener cabeceras idénticas, es decir, con el mismo número de atributos y los atributos correspondientes definidos sobre el mismo dominio.

Si la relación A posee dos atributos (x1, x2) y B también dos atributos (x1, x2), se puede implementar la unión de A y B solo si A:x1 y B:x1 están definidos sobre el mismo dominio pero, también, A:x2 y B:x2 están definidos sobre el mismo dominio.

El resultado es una nueva relación, en la que la cabecera es la de A o la de B (x1, x2) y el cuerpo está formado por todas las flas de A y de B, sin repetición de flas.

Que no haya flas repetidas en el resultado se debe a que es una relación y la tercera propiedad de las relaciones —definida anteriormente— dice “todas las flas son distintas”.



Las operaciones de conjuntos son las operaciones binarias más simples.

### Ejemplo:

Hacer la unión de las relaciones de estudiantes y docentes implica escribir la siguiente expresión del Álgebra relacional y obtener el siguiente resultado, correspondiente a los datos de todos los estudiantes y docentes de la universidad.

Estudiantes **U** Docentes

| Relación Estudiantes <b>U</b> Docentes |           |                   |         |          |             |           |
|--|-----------|-------------------|---------|----------|-------------|-----------|
| Legajo                                 | Apellido  | Nombres           | TipoDoc | NroDoc   | IdLocalidad | Sexo      |
| 11904                                  | González  | Ana Carolina      | DNI     | 35900342 | 31          | Femenino  |
| 13789                                  | López     | Hugo Sebastián    | DNI     | 29762007 | 27          | Masculino |
| 25784                                  | Acuña     | Juan Martín       | DNI     | 33457821 | 30          | Masculino |
| 23892                                  | Salinas   | Rubén             | DNI     | 32890128 | 31          | Masculino |
| 19098                                  | Romero    | María Amparo      | DNI     | 28089727 | 27          | Femenino  |
| 22941                                  | Folmer    | Alfonsina         | DNI     | 31892170 | nulo        | Femenino  |
| 24561                                  | Moreno    | Santiago Rubén    | DNI     | 30892109 | 31          | Masculino |
| 13490                                  | Funes     | Elizabeth         | DNI     | 32782340 | 33          | Femenino  |
| 25801                                  | Sosa      | Marcelo Alejandro | DNI     | 29872301 | 27          | Masculino |
| 12802                                  | López     | Analía            | DNI     | 27001287 | 31          | Femenino  |
| 23875                                  | Ríos      | Adela             | DNI     | 22098231 | 32          | Femenino  |
| 22654                                  | Ruiz      | María Eugenia     | DNI     | 17656209 | 37          | Femenino  |
| 21987                                  | López     | Edgardo Antonio   | DNI     | 19001291 | 21          | Masculino |
| 20871                                  | Castro    | Javier            | DNI     | 20605892 | 36          | Masculino |
| 21823                                  | Funes     | Elizabeth         | DNI     | 19807223 | 27          | Femenino  |
| 21802                                  | Venier    | Marcelo Ricardo   | DNI     | 21876341 | 22          | Masculino |
| 21805                                  | Soria     | Liliana del Valle | DNI     | 22898231 | 34          | Femenino  |
| 21722                                  | Rodríguez | Mario César       | DNI     | 17872349 | 31          | Masculino |

### 2.2.1.2 Diferencia

Esta operación también exige que las dos relaciones de entrada sean compatibles respecto a la unión.

Sea la relación A ( $x_1, x_2$ ) y B ( $x_1, x_2$ ) con A: $x_1$  y B: $x_1$ , definidas sobre el mismo dominio, y A: $x_2$  y B: $x_2$ , también definidas sobre el mismo dominio, se puede expresar que:

El resultado es una nueva relación, en la que la cabecera es la de A, o la de B, ya que son idénticas, y el cuerpo está formado por todas las flas de A, que no están en B.

### Ejemplo:

Obtener las flas con los datos de los Docentes que no se desempeñan como No docentes implica hacer la diferencia entre las relaciones “Docentes” y “No Docentes”. Dicha operación se expresa en el Álgebra relacional como a continuación, también se muestra el resultado.

Docentes — No Docentes

| Relación Docentes - No docentes |           |                 |         |          |             |           |
|---------------------------------|-----------|-----------------|---------|----------|-------------|-----------|
| Legajo                          | Apellido  | Nombres         | TipoDoc | NroDoc   | IdLocalidad | Sexo      |
| 23875                           | Ríos      | Adela           | DNI     | 22098231 | 32          | Femenino  |
| 22654                           | Ruiz      | María Eugenia   | DNI     | 17656209 | 37          | Femenino  |
| 21987                           | López     | Edgardo Antonio | DNI     | 19001291 | 21          | Masculino |
| 20871                           | Castro    | Javier          | DNI     | 20605892 | 36          | Masculino |
| 21823                           | Funes     | Elizabeth       | DNI     | 19807223 | 27          | Femenino  |
| 21802                           | Venier    | Marcelo Ricardo | DNI     | 21876341 | 22          | Masculino |
| 21722                           | Rodríguez | Mario César     | DNI     | 17872349 | 31          | Masculino |

En esta relación resultante, se ve que la fla

|       |       |                   |     |          |    |          |
|-------|-------|-------------------|-----|----------|----|----------|
| 21805 | Soria | Liliana del Valle | DNI | 22898231 | 34 | Femenino |
|-------|-------|-------------------|-----|----------|----|----------|

no se encuentra en el cuerpo, puesto que era la única fla que estaba en “Docentes y NoDocentes”, lo que significa que es alguien que desempeña las dos funciones en la universidad.

#### 2.2.1.3 Intersección

Esta operación se simboliza con  $\cap$  y también exige que las dos relaciones de entrada sean compatibles respecto a la unión.

Sean las relaciones A ( $x_1, x_2$ ) y B ( $x_1, x_2$ ) con A: $x_1$  y B: $x_1$ , definidas sobre el mismo dominio, y A: $x_2$  y B: $x_2$  también definidas sobre el mismo dominio se puede expresar que:

El resultado será una nueva relación, en la que la cabecera es la de A, o la de B, y el cuerpo está formado por todas las flas de A que están también en B.

Ejemplo:

Hacer la intersección entre las relaciones de docentes y no docentes implica escribir la siguiente expresión del Álgebra relacional y obtener el siguiente resultado.

Docentes  $\cap$  No Docentes

| Estudiantes $\cap$ Docentes |          |                   |         |          |             |          |
|-----------------------------|----------|-------------------|---------|----------|-------------|----------|
| Legajo                      | Apellido | Nombres           | TipoDoc | NroDoc   | IdLocalidad | Sexo     |
| 21805                       | Soria    | Liliana del Valle | DNI     | 22898231 | 34          | Femenino |

Ésta es la única fla común entre “Docentes” y en “NoDocentes”; significa que es alguien que, como decíamos en la operación anterior, desempeña las dos funciones en la universidad.

Esto es, se obtiene la lista de empleados de la universidad que trabajan como “Docentes” y “NoDocentes”.

### 2.2.1.4 Producto cartesiano

Esta operación se simboliza con  $\times$ , trabaja sobre dos relaciones y exige que ambas relaciones de entrada no sean compatibles respecto a la unión; muy por el contrario, deben ser compatibles respecto al producto, exigiendo que las cabeceras no posean atributos con igual nombre. Incluso, si hubiera algún atributo con igual nombre, habría que renombrarlo para luego escribir la operación.

Sea la relación A ( $x_1, x_2$ ) y B ( $x_3, x_4$ ) se puede expresar:

$$A \times B$$

El resultado será una nueva relación, en la que la cabecera es la de A más los atributos de B, y el cuerpo está formado por la combinación de todas las filas de A con todas las tuplas de B.

Ejemplo:

Hacer el producto cartesiano entre las relaciones “Materias” y “Provincias” implica escribir la siguiente expresión del Álgebra relacional:

$$\text{Materias} \times \text{Provincias}$$

El proceso inicia cuando se agrega la cabecera de “Provincias” a la cabecera de “Materias” y queda:

| Relación Materias x Provincias |             |         |     |             |             |
|--------------------------------|-------------|---------|-----|-------------|-------------|
| IdMateria                      | Descripción | Carrera | Año | IdProvincia | DescripProv |

Luego se combinan las tuplas de “Materias” con todas y cada una de las filas de “Provincias”.

| Relación Materias |                                   |         |     |
|-------------------|-----------------------------------|---------|-----|
| IdMateria         | Descripción                       | Carrera | Año |
| 333               | Álgebra y Geometría               | K       | 1   |
| 205               | Análisis de Sistemas              | K       | 2   |
| 306               | Sistemas y Organizaciones         | K       | 1   |
| 290               | Análisis Matemático I             | K       | 1   |
| 459               | Algoritmo y Estructuras de Datos  | K       | 1   |
| 207               | Física I                          | K       | 1   |
| 304               | Matemática Discreta               | K       | 1   |
| 451               | Análisis Matemático II            | K       | 2   |
| 330               | Sintaxis y Semántica del Lenguaje | K       | 2   |
| 289               | Física II                         | K       | 2   |
| 277               | Diseño de Sistemas                | K       | 3   |
| 453               | Gestión de Datos                  | K       | 3   |
| 224               | Matemática Superior               | K       | 3   |

| Relación Provincias |              |
|---------------------|--------------|
| IdProvincia         | DescripProv  |
| 4                   | Buenos Aires |
| 7                   | Córdoba      |
| 2                   | Mendoza      |
| 9                   | Misiones     |
| 5                   | Entre Ríos   |
| 8                   | San Luis     |

El resultado suele ser un gran conjunto de filas y la información que se obtiene no aporta demasiado, pero la operación es importante porque es la base de otra operación que se denomina “Reunión”.

| IdMateria | Descripción         | Carrera | Año | IdProvincia | DescripProv  |
|-----------|---------------------|---------|-----|-------------|--------------|
| 333       | Álgebra y Geometría | K       | 1   | 4           | Buenos Aires |
| 333       | Álgebra y Geometría | K       | 1   | 7           | Córdoba      |
| 333       | Álgebra y Geometría | K       | 1   | 2           | Mendoza      |
| 333       | Álgebra y Geometría | K       | 1   | 9           | Misiones     |
| 333       | Álgebra y Geometría | K       | 1   | 5           | Entre Ríos   |
| 333       | Álgebra y Geometría | K       | 1   | 8           | San Luis     |
| ...       | ...                 | ...     | ... | ...         | ...          |

En este ejemplo, se tomaron dos relaciones que cumplen las exigencias para realizar el Producto cartesiano, sin atributos comunes entre ellas, pero sin sentido para asegurar que se vea el volumen de datos que produce y el poco valor que se logra como información.

De las 78 tuplas del resultado que se obtendrá, se muestran 6 filas, que resultaron de trabajar con una de “Materias” contra todas las de “Provincias”; luego se debe continuar el proceso con el resto de las tuplas.

13 tuplas de Materias X 6 tuplas de Provincias = 78 tuplas en el resultado

## 2.2.2 Operaciones relacionales



Las operaciones relacionales brindan más potencial, incluso cuando se combinan con las operaciones de conjuntos y entre ellas mismas. No todas las operaciones relacionales son binarias y no todas son primitivas.

Estas operaciones brindan más potencial, incluso cuando se combinan con las operaciones de conjuntos y entre ellas mismas.

No todas las operaciones relacionales son binarias y, además, no todas son primitivas, ya que, si no existieran, se podrían expresar por la combinación de otras operaciones. Se aclararán estos aspectos para cada una de las operaciones.

### 2.2.2.1 Selección

Esta operación permite seleccionar un conjunto de tuplas con una única relación de entrada, que cumplan determinadas condiciones que se incluyen en la expresión.

Como la entrada es una sola relación, se dice que la selección es una operación unaria.

Se simboliza con  $\sigma$  (sigma) y solo es necesario expresar las condiciones si se considera el dominio de los datos involucrados.

$$\sigma \text{ condición } (A)$$

El resultado será una nueva relación, en la que la cabecera es la de A y el cuerpo está formado por todas las tuplas de A que cumplan las condiciones indicadas.

En la condición pueden incluirse signos u operadores de comparación, como son:

- \_ menor o igual
- \_ mayor o igual
- = igual
- <> distinto
- > mayor
- < menor
- distinto

Dichos signos se utilizan en la condición, armando una condición lógica, de alguna de las siguientes formas:

Forma 1:  $\sigma_{<\text{NombreDeAtributo}><\text{Signo}><\text{ValorConstante}>} (A)$

Forma 2:  $\sigma_{<\text{NombreDeAtributo}><\text{Signo}><\text{NombreDeAtributo}>} (A)$

Siendo NombreDeAtributo el nombre de un atributo en la relación A y ValorConstante un valor que corresponde al dominio del atributo mencionado.

Ejemplo:

Si se desean obtener todos los estudiantes de la Universidad que poseen legajo superior a 14 000, la expresión será:

$\sigma_{\text{Legajo} > 14000} (\text{Estudiantes})$

El resultado de la expresión es:

| Legajo | Apellido | Nombres           | TipoDoc | NroDoc   | IdLocalidad | Sexo      |
|--------|----------|-------------------|---------|----------|-------------|-----------|
| 25784  | Acuña    | Juan Martín       | DNI     | 33457821 | 30          | Masculino |
| 23892  | Salinas  | Rubén             | DNI     | 32890128 | 31          | Masculino |
| 19098  | Romero   | María Amparo      | DNI     | 28089727 | 27          | Femenino  |
| 22941  | Folmer   | Alfonsina         | DNI     | 31892170 | nulo        | Femenino  |
| 24561  | Moreno   | Santiago Rubén    | DNI     | 30892109 | 31          | Masculino |
| 25801  | Sosa     | Marcelo Alejandro | DNI     | 29872301 | 27          | Masculino |

Para combinar más de una condición en una expresión, es necesario trabajar con conectores lógicos:

- $\wedge$ : simboliza que deben cumplirse ambas condiciones, equivalente a AND.
- $\vee$ : exige el cumplimiento de una de las condiciones, equivale a OR.
- $\neg$  : es la negación de una expresión, equivale a A/O I

A estos conectores, se los puede complementar con paréntesis para fijar prioridad de aplicación de las operaciones.

Ejemplo:

Para obtener todos los datos de los exámenes aprobados por el alumno, cuyo legajo es 19098, la expresión podría ser:

$\sigma_{\text{Nota} > 4} \wedge \text{Legajo} = 19098 (\text{Exámenes})$

El resultado de la expresión es:

| IdMateria | Legajo | Turno | Año  | Fecha    | Nota | LegDoc |
|-----------|--------|-------|------|----------|------|--------|
| 290       | 19098  | 8     | 2009 | 27/11/09 | 5    | 21722  |
| 306       | 19098  | 1     | 2010 | 05/02/10 | 4    | 22654  |
| 333       | 19098  | 2     | 2010 | 10/02/10 | 7    | 21987  |

### 2.2.2.2 Proyección

Así como en el resultado de la selección se obtienen filas que cumplen determinadas condiciones, en este caso, lo que podemos elegir son columnas, que van separadas por coma.

Es decir que la operación, que se simboliza con  $\pi$  y es unaria, permite elegir cuáles serán las columnas de la relación entrante que formarán parte de la relación "Resultado".

$\pi_K \text{ListaDeColumnas } (A)$

El resultado será una nueva relación, en la que la cabecera estará formada solo por las columnas elegidas de A y el cuerpo estará formado por todas las tuplas de A, mientras no tengan una operación de selección.

Ejemplo:

Si se desean obtener Apellido y Nombres de todos los no docentes de la universidad, la expresión será:

$\pi_{\text{Apellido}, \text{Nombres}} (\text{NoDocentes})$

El resultado de la expresión es:

| Apellido | Nombres          |
|----------|------------------|
| Asís     | Jorge Antonio    |
| Sosa     | Santiago Martín  |
| Bustos   | Magdalena        |
| Soria    | Lilián del Valle |

### 2.2.3 Reunión

Esta operación binaria brinda la posibilidad de ampliar los datos de una relación con datos de otra relación, a través de un atributo en común que deben poseer las relaciones, que obtendrá una mayor información. Generalmente, el atributo común entre ambas es clave primaria en una relación y foránea en la otra; es decir, que se definirán en el mismo dominio.

Esta no es una operación primitiva, puesto que el mismo resultado puede lograrse aplicando producto cartesiano, selección y proyección y se simboliza con  $\times$ .

La operación binaria de reunión permite ampliar los datos de una relación con datos de otra relación, a través de un atributo en común que deben poseer las relaciones, que obtendrá una mayor información.

Sea la relación A ( $x_1, x_2, x_3$ ) y B ( $x_3, x_4, x_5$ ), la expresión será:

$$\overset{\wedge}{A} \sqsupseteq^J_{x_3=x_3} B$$

y la reunión dará como resultado una relación con:

- Cabecera formada por  $x_1, x_2, x_3, x_4, x_5$ , en la que aparece una sola vez la columna en común.
- Cuerpo formado por tuplas, en las que el valor de  $x_3$  en A sea igual al valor de  $x_3$  en B.

Ejemplo:

Si se desea obtener los datos de las localidades que incluyan la descripción de la provincia a la que pertenece cada una, la expresión es:

$$\text{Localidades } X \quad \text{Id Provincia} = \text{IdProvincia} \text{ Provincias}$$

El resultado es:

| IdLocalidad | Descripción         | IdProvincia | DescripProv  |
|-------------|---------------------|-------------|--------------|
| 27          | Ciudad Buenos Aires | 4           | Buenos Aires |
| 31          | Córdoba             | 7           | Córdoba      |
| 30          | Villa María         | 7           | Córdoba      |
| 33          | Godoy Cruz          | 2           | Mendoza      |
| 25          | Villa Mercedes      | 8           | San Luis     |
| 20          | Villa Dolores       | 7           | Córdoba      |
| 36          | Mar del Plata       | 4           | Buenos Aires |
| 22          | Cruz del Eje        | 7           | Córdoba      |
| 21          | San José            | 9           | Misiones     |
| 34          | Las Heras           | 2           | Mendoza      |
| 29          | Oberá               | 9           | Misiones     |
| 32          | Tunuyán             | 2           | Mendoza      |
| 37          | La Plata            | 4           | Buenos Aires |

### 2.2.3.1 Reunión externa

En el resultado anterior, no aparece la descripción de la provincia de Entre Ríos porque la reunión expresada es una equirreunión; es decir: en el resultado aparecen las filas que cumplen con la igualdad de los valores del atributo común entre ambas relaciones.

Algo semejante hubiera sucedido si la expresión fuera:

$$\text{Estudiantes } X ] \text{ IdLocalidad} = \text{IdLocalidad} \text{ Localidades}$$

En este caso, no aparecería la fla:

|       |        |           |     |          |      |          |
|-------|--------|-----------|-----|----------|------|----------|
| 22941 | Folmer | Alfonsina | DNI | 31892170 | nulo | Femenino |
|-------|--------|-----------|-----|----------|------|----------|

Aquí, el atributo “IdLocalidad” tiene valor nulo en la fla y el valor nulo no puede ser comparado con ningún valor, es algo desconocido.

La reunión externa se utiliza para resolver este tipo de situaciones y es una ampliación de la operación reunión:

- Cuando se desea que aparezcan en el resultado las flas en las que el valor de la clave foránea o ajena tiene valor desconocido o nulo.
- Cuando alguna fla en la relación que posee la clave primaria no se referencia por ninguna fla de la otra relación.
- Ambas a la vez.

#### 2.2.3.2 Reunión externa derecha

Si quisieramos que la provincia de Entre Ríos apareciera en la reunión entre “Localidades” y “Provincias”, la expresión

Localidades 1X1  $_{\text{IdProvincia}=\text{IdProvincia}}$  Provincias

debe ser reemplazada por:

Localidades XE  $_{\text{IdProvincia}=\text{IdProvincia}}$  Provincias

Como la provincia de Entre Ríos no se referencia por ninguna localidad, con esta operación sí aparecerá, pero los demás datos de la tupla serán desconocidos. La relación resultado será:

| IdLocalidad | Descripción         | IdProvincia | DescripProv  |
|-------------|---------------------|-------------|--------------|
| 27          | Ciudad Buenos Aires | 4           | Buenos Aires |
| 31          | Córdoba             | 7           | Córdoba      |
| 30          | Villa María         | 7           | Córdoba      |
| 33          | Godoy Cruz          | 2           | Mendoza      |
| 25          | Villa Mercedes      | 8           | San Luis     |
| 20          | Villa Dolores       | 7           | Córdoba      |
| 36          | Mar del Plata       | 4           | Buenos Aires |
| 22          | Cruz del Eje        | 7           | Córdoba      |
| 21          | San José            | 9           | Misiones     |
| 34          | Las Heras           | 2           | Mendoza      |
| 29          | Oberá               | 9           | Misiones     |
| 32          | Tunuyán             | 2           | Mendoza      |
| 37          | La Plata            | 4           | Buenos Aires |
| Nulo        | Nulo                | 5           | Entre Ríos   |

### 2.2.3.3 Reunión externa izquierda

La situación para exemplificar es la de “Estudiantes” con “Localidades”, porque hay una estudiante que no posee localidad asignada y, por eso, el atributo “IdLocalidad” posee valor nulo.

Si quisieramos obtener un listado completo de alumnos, posean o no localidad asignada, la expresión sería:

Estudiantes  $\Sigma \Delta X$   $_{\text{IdLocalidad}=\text{IdLocalidad}}$  Localidades

En el resultado aparece la alumna con localidad desconocida.

| Legajo | Apellido | Nombres           | TipoDoc | NroDoc   | IdLocalidad | Sexo      | Descripción         | IdProvincia |
|--------|----------|-------------------|---------|----------|-------------|-----------|---------------------|-------------|
| 11904  | González | Ana Carolina      | DNI     | 35900342 | 31          | Femenino  | Córdoba             | 7           |
| 13789  | López    | Hugo Sebastián    | DNI     | 29762007 | 27          | Masculino | Ciudad Buenos Aires | 4           |
| 25784  | Acuña    | Juan Martín       | DNI     | 33457821 | 30          | Masculino | Villa María         | 7           |
| 23892  | Salinas  | Rubén             | DNI     | 32890128 | 31          | Masculino | Córdoba             | 7           |
| 19098  | Romero   | Maria Amparo      | DNI     | 28089727 | 27          | Femenino  | Ciudad Buenos Aires | 4           |
| 22941  | Folmer   | Alfonsina         | DNI     | 31892170 | Nulo        | Femenino  | Nulo                | nulo        |
| 24561  | Moreno   | Santiago Rubén    | DNI     | 30892109 | 31          | Masculino | Córdoba             | 7           |
| 13490  | Funes    | Elizabeth         | DNI     | 32782340 | 33          | Femenino  | Godoy Cruz          | 2           |
| 25801  | Sosa     | Marcelo Alejandro | DNI     | 29872301 | 27          | Masculino | Ciudad Buenos Aires | 4           |
| 12802  | López    | Analía            | DNI     | 27001287 | 31          | Femenino  | Córdoba             | 7           |

### 2.2.3.4 Reunión externa completa

Se continúa con la reunión entre “Estudiantes” y “Localidades” para mostrar qué sucede si la expresión es una reunión externa completa.

Con la expresión siguiente se logrará mostrar las tuplas de la relación “Estudiantes”—que no poseen localidad conocida— y las localidades que no se referencian por la relación “Estudiantes”.

Estudiantes  $\Sigma \Delta L T$   $_{\text{IdLocalidad}=\text{IdLocalidad}}$  Localidades

El resultado será:

| Legajo | Apellido | Nombres           | TipoDoc | NroDoc   | IdLocalidad | Sexo      | Descripción         | IdProvincia |
|--------|----------|-------------------|---------|----------|-------------|-----------|---------------------|-------------|
| 11904  | González | Ana Carolina      | DNI     | 35900342 | 31          | Femenino  | Córdoba             | 7           |
| 13789  | López    | Hugo Sebastián    | DNI     | 29762007 | 27          | Masculino | Ciudad Buenos Aires | 4           |
| 25784  | Acuña    | Juan Martín       | DNI     | 33457821 | 30          | Masculino | Villa María         | 7           |
| 23892  | Salinas  | Rubén             | DNI     | 32890128 | 31          | Masculino | Córdoba             | 7           |
| 19098  | Romero   | María Amparo      | DNI     | 28089727 | 27          | Femenino  | Ciudad Buenos Aires | 4           |
| 22941  | Folmer   | Alfonsina         | DNI     | 31892170 | Nulo        | Femenino  | Nulo                | nulo        |
| 24561  | Moreno   | Santiago Rubén    | DNI     | 30892109 | 31          | Masculino | Córdoba             | 7           |
| 13490  | Funes    | Elizabeth         | DNI     | 32782340 | 33          | Femenino  | Godoy Cruz          | 2           |
| 25801  | Sosa     | Marcelo Alejandro | DNI     | 29872301 | 27          | Masculino | Ciudad Buenos Aires | 4           |
| 12802  | López    | Analia            | DNI     | 27001287 | 31          | Femenino  | Córdoba             | 7           |
| Nulo   | Nulo     | Nulo              | Nulo    | Nulo     | 25          | Nulo      | Villa Mercedes      | 8           |
| Nulo   | Nulo     | Nulo              | Nulo    | Nulo     | 20          | Nulo      | Villa Dolores       | 7           |
| Nulo   | Nulo     | Nulo              | Nulo    | Nulo     | 36          | Nulo      | Mar del Plata       | 4           |
| Nulo   | Nulo     | Nulo              | Nulo    | Nulo     | 22          | Nulo      | Cruz del Eje        | 7           |
| Nulo   | Nulo     | Nulo              | Nulo    | Nulo     | 21          | Nulo      | San José            | 9           |
| Nulo   | Nulo     | Nulo              | Nulo    | Nulo     | 34          | Nulo      | Las Heras           | 2           |
| Nulo   | Nulo     | Nulo              | Nulo    | Nulo     | 29          | Nulo      | Oberá               | 9           |
| Nulo   | Nulo     | Nulo              | Nulo    | Nulo     | 32          | Nulo      | Tunuyán             | 2           |
| Nulo   | Nulo     | Nulo              | Nulo    | Nulo     | 37          | Nulo      | La Plata            | 4           |

## 2.2.4 División

Ésta es una operación binaria que permite obtener los datos de los estudiantes que aprobaron todas las materias o empleados que trabajaron en todas las sucursales de una empresa.

La condición es que la cabecera de la relación del divisor esté incluida en la cabecera del dividendo.

El símbolo es + y la expresión se escribe así:

$$A + B$$

La estructura de las relaciones tendría que ser A (u, v, w) y B (w), donde se observará que la cabecera de B se incluirá en la cabecera de A.

La relación resultante quedaría con:

- Cabecera formada por u y v, en la que no aparece el atributo común.
- Cuerpo formado por todas las tuplas de A, en las que los valores de los atributos u y v se relacionan con todos los valores de w en B.

Para utilizar las relaciones de la relación “Estudiantes”, se utilizarán muchas filas más en algunas relaciones para visualizar el efecto en una relación resultado. Por ello, se planteará, a continuación, un ejemplo genérico que buscará el entendimiento de la operación.

Ejemplo:

Sean las relaciones A y B con las estructuras mencionadas anteriormente:

| A  |    |    | B  |
|----|----|----|----|
| u  | v  | w  | W  |
| u2 | v4 | w1 | w1 |
| u1 | v5 | w5 | w3 |
| u1 | v7 | w4 | w4 |
| u2 | v1 | w4 | w5 |
| u2 | v1 | w5 |    |
| u2 | v4 | w3 |    |
| u1 | v2 | w3 |    |
| u2 | v3 | w5 |    |
| u3 | v2 | w1 |    |
| u2 | v4 | w5 |    |
| u1 | v4 | w1 |    |

El resultado de A + B será:

| u  | v  |
|----|----|
| u2 | v4 |

Se observa que los valores de los atributos u, v **G** A se relacionan en B con todos los valores del atributo w. Esto es, cada par de valores (ui, vj) del resultado de la relación anterior está acompañado en la relación A + B por todos los valores del atributo w de la tabla B. Por cada par (ui, vj), hay 4 filas donde se tienen los 4 valores de w.

Además, la columna w ya no forma parte del resultado.

## 2.2.5 Renombrar

La operación renombrar no está dentro de las mencionadas ya que es una operación auxiliar, pero necesaria en determinadas situaciones, como en las que se aplican expresiones complejas o se necesita cumplir con las exigencias de otra operación del Álgebra relacional.

Esta operación se simboliza con **p** (rho) y permite renombrar una relación, unos atributos o los dos a la vez.

Sea la relación A con cabecera formada por los atributos u, v y w, se analizarán las distintas posibilidades que brinda la operación renombrar:

**p<sub>s</sub>(A)**: renombra a la relación A como S.

**P(a,b,c)(A)**: renombra a los atributos de la relación A como a, b, c.

**Ps(a,b,c)(A)**: renombra a la relación A como S y a sus atributos como a, b, c.

## 2.2.6 Funciones

A continuación, se mencionará un conjunto de funciones que proporcionarán la posibilidad de aplicar cálculos con valores de las relaciones.

Las funciones disponibles son:

- SUM(Atributo): suma de los valores numéricos contenidos en el atributo.
- AVG(Atributo): promedio de valores numéricos del atributo.
- MIN(Atributo): menor valor asumido por el atributo, no necesariamente numérico.
- MAX(Atributo): valor máximo asumido por el atributo, no necesariamente numérico.
- COUNT(Atributo): contar la ocurrencia de valores en una columna de datos de cualquier tipo.

Estas funciones se escribirán junto al símbolo  $\wedge$  (F gótica), de la siguiente manera:

 función(w) (A)

Y w **G** A es el atributo sobre el que se aplica la función.

Ejemplo:

Para obtener la cantidad de alumnos de la universidad, la expresión será:

\$COUNT(Legajo) (Estudiantes)

que dará como resultado una relación de una única tupla y una columna; es decir, un valor que, en este caso, será 10.

En cuanto a la cabecera de la relación, se puede decir que no asume un nombre determinado; por ello, se asumirá que, en la cabecera, aparecerá un nombre FUNCIÓN\_Atributo.

Por lo antedicho, la relación resultado será:

|              |
|--------------|
| COUNT_Legajo |
| 10           |

Ejemplo:

Si se quisiera calcular la cantidad de exámenes y la nota promedio en la universidad, se escribirá:

 COUNT(Legajo),AVG(Nota) (Exámenes)

Y el resultado será:

| COUNT_Legajo | AVG_Nota |
|--------------|----------|
| 13           | 5.23     |

## 2.2.7 Agrupación

A las funciones mencionadas, si se les aplica lo que se denomina agrupación, se las proveerá de más potencial.

Para explicar qué es agrupar, se tomará la relación “Exámenes”.

La expresión indica que deben agruparse las tuplas por legajo y, por cada grupo de flas, aplicar la función AVG sobre los valores del atributo “Nota” para el cálculo de promedio.

Como ya se calculó el promedio de todos los exámenes y se explicó que devolvía una tupla, si se quisiera calcular el promedio en “exámenes por materia”, se aplicará la agrupación de la siguiente manera:

$\text{Legajo}^T \text{AVG}(\text{Nota}) \text{ (Exámenes)}$

| IdMateria | Legajo | Turno | Año  | Fecha    | Nota | LegDoc |  |
|-----------|--------|-------|------|----------|------|--------|--|
| 207       | 11904  | 2     | 2010 | 08/02/10 | 2    | 20871  |  |
| 306       | 12802  | 1     | 2010 | 05/02/10 | 10   | 22654  |  |
| 459       | 13789  | 10    | 2009 | 22/12/09 | 8    | 21823  |  |
| 333       | 13789  | 2     | 2010 | 10/02/10 | 7    | 21987  |  |
| 290       | 19098  | 8     | 2009 | 27/11/09 | 5    | 21722  |  |
| 306       | 19098  | 1     | 2010 | 05/02/10 | 4    | 22654  |  |
| 333       | 19098  | 2     | 2010 | 10/02/10 | 7    | 21987  |  |
| 207       | 19098  | 2     | 2010 | 08/02/10 | 3    | 20871  |  |
| 333       | 24561  | 10    | 2009 | 21/12/09 | 8    | 21987  |  |
| 306       | 25801  | 9     | 2009 | 07/12/09 | 2    | 22654  |  |
| 333       | 25801  | 1     | 2010 | 03/02/10 | 4    | 21987  |  |
| 306       | 25801  | 2     | 2010 | 12/02/10 | 2    | 22654  |  |
| 290       | 25801  | 3     | 2010 | 24/02/10 | 6    | 21722  |  |

Grupo 1-Legajo=11904-AVG= 2  
 Grupo 2-Legajo=12802-AVG= 10  
 Grupo 3-Legajo=13789-AVG= 7.5  
 Grupo 4-Legajo=19098-AVG= 4,75  
 Grupo 5-Legajo=24561-AVG= 8  
 Grupo 6-Legajo=25801-AVG= 3.5

La relación resultado tendrá:

- Cabecera formada por dos columnas: en la primera, por el atributo de la agrupación Legajo y, en la segunda, por el promedio AVG aplicado sobre la columna “Nota”.
- Cuerpo formado por seis tuplas, una fla por grupo, que corresponden a los distintos legajos de la relación “Exámenes”.

| Legajo | AVG_Nota |
|--------|----------|
| 11904  | 2        |
| 12802  | 10       |
| 13789  | 7.5      |
| 19098  | 4.75     |
| 24561  | 8        |
| 25801  | 3.5      |

## 2.2.8 Relaciones temporales

En expresiones complejas, suelen utilizarse las relaciones temporales para que se mantengan o recuerden los resultados intermedios de las operaciones del Álgebra relacional. Esto significa que también se hace en la práctica con SQL, para evitar, de esta manera, las consultas de muy difícil lectura y seguimiento.

Para expresar la asignación del resultado de una operación a una relación temporal, se escribirá:

temp (E)

donde E es una expresión válida del Álgebra compuesta por el número de operaciones que se desee.

Ejemplo:

Si al resultado del cálculo de promedios que se hizo al agrupar, se deseara agregar los datos del alumno, para poder identificarlo, se grafcaría de la siguiente manera:

1. temp1 ( Legajo T AVG(Nota) (Exámenes))

| Legajo | AVG_Nota |
|--------|----------|
| 11904  | 2        |
| 12802  | 10       |
| 13789  | 7.5      |
| 19098  | 4.75     |
| 24561  | 8        |
| 25801  | 3.5      |

Se aplicó agrupación y la función AVG.

2.  $\text{temp2} \leftarrow (\text{temp1} \times \text{I}_{\text{Legajo}=\text{Legajo}} (\text{Estudiantes}))$

| Legajo | AVG_Nota | Apellido | Nombres           | TipoDoc | NroDoc   | IdLocalidad | Sexo      |
|--------|----------|----------|-------------------|---------|----------|-------------|-----------|
| 11904  | 2        | González | Ana Carolina      | DNI     | 35900342 | 31          | Femenino  |
| 12802  | 10       | López    | Analía            | DNI     | 27001287 | 31          | Femenino  |
| 13789  | 7.5      | López    | Hugo Sebastián    | DNI     | 29762007 | 27          | Masculino |
| 19098  | 4.75     | Romero   | María Amparo      | DNI     | 28089727 | 27          | Femenino  |
| 24561  | 8        | Moreno   | Santiago Rubén    | DNI     | 30892109 | 31          | Masculino |
| 25801  | 3.5      | Sosa     | Marcelo Alejandro | DNI     | 29872301 | 27          | Masculino |

Se aplicó reunión entre la relación temporal temp1 y la relación “Estudiantes” por el atributo “Legajo”, para ampliar con los datos de los estudiantes.

3.  $\text{temp3} \leftarrow (\text{TI}_{\text{Legajo}, \text{Apellido}, \text{Nombres}, \text{AVG_Nota}} (\text{temp2}))$

Y el resultado final, asignado a temp3, es la relación:

| Legajo | Apellido | Nombres           | AVG_Nota |
|--------|----------|-------------------|----------|
| 11904  | González | Ana Carolina      | 2        |
| 12802  | López    | Analía            | 10       |
| 13789  | López    | Hugo Sebastián    | 7.5      |
| 19098  | Romero   | María Amparo      | 4.75     |
| 24561  | Moreno   | Santiago Rubén    | 8        |
| 25801  | Sosa     | Marcelo Alejandro | 3.5      |

Se aplicó proyección sobre la relación temporal temp2.

La operación anterior, escrita con tres relaciones temporales, se podría escribir, también, anidando las expresiones, de manera sensiblemente más compleja, pero con el mismo resultado. Entonces, se la podría escribir como sigue:

$\pi_{\text{Legajo}, \text{Apellido}, \text{Nombres}, \text{AVG_Nota}} (\text{Legajo T}_{\text{AVG(Nota)}} (\text{Exámenes}) \times_{\text{Legajo}=\text{Legajo}} (\text{Estudiantes}))$

### 2.2.8.1 Ejemplos combinando operaciones

En este punto, se plantearán algunos ejercicios y resoluciones propuestas con las operaciones analizadas, con el agregado de funciones, de agrupación y de relaciones temporales.

Todos los ejemplos que se basan en las relaciones que se vienen utilizando, correspondientes a los datos de la universidad, tienen una nota aclaratoria que intenta clarificar lo enunciado y, también, la búsqueda de los datos.

A continuación, se recordarán las cabeceras de las relaciones para facilitar el seguimiento de las soluciones propuestas.

| Relación Estudiantes |          |         |         |        |             |      |
|----------------------|----------|---------|---------|--------|-------------|------|
| Legajo               | Apellido | Nombres | TipoDoc | NroDoc | IdLocalidad | Sexo |

| Relación Localidades |             |             |
|----------------------|-------------|-------------|
| IdLocalidad          | Descripción | IdProvincia |

| Relación Materias |             |         |     |
|-------------------|-------------|---------|-----|
| IdMateria         | Descripción | Carrera | Año |

| Relación Provincias |             |
|---------------------|-------------|
| IdProvincia         | DescripProv |

| Relación Exámenes |        |       |     |       |      |        |
|-------------------|--------|-------|-----|-------|------|--------|
| IdMateria         | Legajo | Turno | Año | Fecha | Nota | LegDoc |

| Relación Docentes |          |         |         |        |             |      |
|-------------------|----------|---------|---------|--------|-------------|------|
| Legajo            | Apellido | Nombres | TipoDoc | NroDoc | IdLocalidad | Sexo |

| Relación NoDocentes |          |         |         |        |             |      |
|---------------------|----------|---------|---------|--------|-------------|------|
| Legajo              | Apellido | Nombres | TipoDoc | NroDoc | IdLocalidad | Sexo |

### Ejemplo:

Obtener legajo, apellido y nombres de los estudiantes que aprobaron alguna asignatura en el turno 9, del año 2009.

Nota: son los alumnos que tienen nota igual o superior a 4 (cuatro) en la relación “Exámenes”, pero con Turno 9 y Año 2009.

```
tempApr - o Nota>4 ATurno=9 AAño=2009 (Exámenes)
tempReu <- (tempApr 1 X 1 Legajo=Legajo Estudiantes)
tempPro - <c Legajo,Apellido,Nombres (tempReu)
```

### Ejemplo:

Obtener apellido y nombres de los estudiantes que nunca se presentaron a rendir exámenes.

Nota: son los alumnos que se encuentran en la relación “Estudiantes”, pero que no están en la relación “Exámenes”. Esto implica una diferencia de relaciones, pero antes se las hará compatibles.

```

tempLegExa ← πLegajo (Exámenes)
tempLegEst ← πLegajo (Estudiantes)
tempDif ← tempLegEst – tempLegExa
tempReu ← tempDif ⋙ Legajo=Legajo Estudiantes
tempFin ← πLegajo,Apellido,Nombres (tempReu)

```

Ejemplo:

Obtener la cantidad de alumnos que aprobaron cada materia, junto con la descripción de la materia.

Nota: la solución es contar los alumnos después de filtrar los aprobados en “Exámenes”.

```

tempApr ← oNota>4 (Exámenes)
tempCon ← IdMateria TCOUNT(IdMateria) (tempApr)
tempReu ← tempCon CXI IIdMateria=IdMateria Materias
tempFin ← TI Count_ IdMateria,Descripción (tempReu)

```

## 2.2.9 Cálculo relacional

En el Álgebra relacional —como se habrá observado—, las expresiones son operaciones anidadas que indican lo que se desea obtener y cómo conseguirlo; incluso, especificando un determinado orden para obtener lo que se necesita sin perder datos. Por todo esto, se dice que el Álgebra relacional es un lenguaje procedural para expresar consultas.

El Cálculo relacional, por el contrario, es un lenguaje no procedural, en el que se indica —sin un orden de escritura— qué se desea obtener, pero no cómo obtenerlo.

Si bien existe esta diferencia, con ambos se pueden escribir, en el modelo relacional, el mismo tipo de consultas.

Según el tipo de variable que se use para el Cálculo relacional, se determina el tipo de cálculo:

- Cálculo relacional de tuplas.
- Cálculo relacional de dominios.

### 2.2.9.1 Cálculo relacional de tuplas

Para obtener todos los datos de los estudiantes masculinos de la universidad, en el Álgebra relacional, se escribirá la expresión:

$\circ \text{Sexo}=\text{Masculino}$  (Estudiantes)



El Álgebra relacional es un lenguaje procedural para expresar consultas. En cambio, el Cálculo relacional es un lenguaje no procedural, en el que se indica —sin un orden de escritura— qué se desea obtener, pero no cómo obtenerlo.

Mientras que en el Cálculo relacional de tuplas, la expresión para la misma consulta es:

$$\{e|Estudiantes(e) \text{ A } e.\text{Sexo} = \text{'Masculino'}$$

Esto es, obtener que la tupla  $e$  de la relación “Estudiantes” cumpla la condición de que el atributo “Sexo” tenga el valor “Masculino”.

Cuando se escribe  $e.\text{Sexo}$ , se dice que el atributo “Sexo” está cualificado para mayor claridad, lo que es imprescindible para consultas complejas con datos de múltiples relaciones.

Por ejemplo, para escribir una reunión del Álgebra relacional, en la que se necesitan las descripciones de las localidades de la provincia de Córdoba, la expresión será:

$$\pi_{\text{Descripción}(\text{Localidades } u \wedge_j \text{ IdProvincia}=\text{IdProvincia} \wedge \text{DescripProv}=\text{'Córdoba'}(\text{Provincias}))}$$

Esa expresión se puede escribir, en el Cálculo relacional de tuplas, de la siguiente manera:

$$\left\{ \begin{array}{l|l} \text{I.Descripción} & \text{Localidades (l) A Provincias (p)} \\ \hline & \text{l.IdProvincia = 1} \\ & \text{p.Id Provincia A p.DescripProv = 'Córdoba'} \end{array} \right\}_J$$

Se observarán las partes de la expresión:

- Inicia nombrando solo la descripción de la Localidad, lo que en Álgebra se denomina proyección.
- Luego se definen las relaciones con las que se trabaja.
- Sigue con la igualdad para la reunión de las relaciones “Localidades” y “Provincias”.
- Finaliza con las condiciones denominadas selecciones en Álgebra, unidas por el conector lógico **A**, para indicar que deben cumplirse todas (AND). También pueden conectarse con **v** (OR).

#### 2.2.9.2 Cálculo relacional de dominios

A diferencia del cálculo relacional de tuplas, aquí las variables se refieren a los dominios de atributos y no a las tuplas.

Para establecer las coincidencias y las diferencias con el Álgebra relacional y el cálculo relacional de tuplas, se volverá a la consulta: “Obtener todos los datos de los estudiantes masculinos de la universidad”, que ya había sido resuelta:

- En Álgebra relacional la expresión era:  $a_{\text{Sexo}=\text{'Masculino'}}(\text{Estudiantes})$
- En Cálculo relacional de tuplas:  $\{e|\text{Estudiantes}(e) \text{ A } e.\text{Sexo} = \text{'Masculino'}$

En cálculo relacional de dominios, se trabaja con una variable de dominio por atributo de cada relación que participa, que indica a qué relación pertenece y, luego, se utilizarán teniendo en cuenta el orden en que se las menciona.

Por ello, se resolverá esa consulta en cálculo relacional de dominios:

$$\{\text{abcdefg}|(3a)(3b)(3c)(3d)(3e)(3f)(3g) (\text{Estudiantes}(\text{abcdefg}) \text{ A } g = \text{'Masculino'})\}$$

Se verán las distintas partes de la expresión:

- Primero se escriben las variables de dominio que representan a los atributos solicitados. Como en esta consulta no se pide alguno en particular, sino que se necesitan todos los datos de los estudiantes, se mencionan siete variables, una para cada atributo de la relación "Estudiantes" (abcdefg).
- Despues de la barra (), se indica que la tupla formada por abcdefg debe pertenecer a la relación "Estudiantes", y que el atributo g tiene valor "masculino".

Lo mismo sucede con la consulta con una reunión de relaciones, por ejemplo, Obtener las descripciones de las localidades de la provincia de Córdoba.

En Álgebra se había resuelto como:

$\pi_{\text{Descripción} (\text{Localidades} \ 1> \sim \text{J} \ \text{IdProvincia} = \text{IdProvincia} (\circ \ \text{DescripProv} = \text{'Córdoba'} (\text{Provincias})))}$

En cálculo relacional de tuplas, la solución planteada fue con la expresión:

$$\left[ \begin{array}{c|c} \text{I} & \text{Localidades} (\text{I}) \ A \ \text{Provincias} (\text{p}) \ A \ \text{I.IdProvincias} = 1 \\ \hline \text{I.Descripción} & \text{p.Id Provincia} \ A \ \text{p.DescripProv} = \text{'Córdoba'} \\ \hline \text{J} & \end{array} \right]$$

Ahora, en cálculo relacional de dominios, la expresión será:

$\{ab|(3a)(3b)(3c)(3d)(3e) (\text{Localidades}(abc) \ A \ \text{Provincias}(de) \ Ac = dAe = \text{'Córdoba'})\}$

Donde b es el atributo solicitado y corresponde a la descripción de la localidad en la relación "Localidades".

Aparecen las dos relaciones que se reunirán y, luego, las condiciones que se cumplirán. En este caso, incluye a la condición de reunión  $c=d$ , en la que el atributo e contiene la palabra Córdoba.

## 2.3 Normalización

### 2.3.1 Fundamentos de la normalización

Un experimentado diseñador de base de datos relacionales —sin que importe la herramienta que utilice para sus propósitos— quizá conozca el porqué de la normalización. Sin embargo, si el mismo diseñador se sintiera con una gran experiencia en el diseño de bases de datos y no conociera el término “normalización” —o si conociera el término, pero no su propósito—, entonces estaría en un serio problema, pues dejaría de lado el concepto primigenio que fundamenta la tecnología de los motores de base de datos relacionales. Dicho de otro modo: no se podría considerar un experto en el diseño de bases de datos relacionales, si omitiera este concepto. A continuación, se describirán los conceptos que —tal vez— ayudarán a esclarecer el porqué de la normalización, cuya importancia se encuentra en el mismo nivel que el correcto manejo del lenguaje de consultas estructurado (SQL).

En este capítulo solo se hará referencia al concepto asociado con las bases de datos relacionales y se omitirá su aspecto más amplio, es decir, el que se refiere a las bases de datos en todas sus formas.

¿Por qué son tan importantes la normalización y el SQL? Porque se requerirá el manejo del SQL para interactuar con una base de datos relacional; entonces, es imprescindible comprender qué es una base de datos relacional. La normalización de datos se refiere, precisamente, a este concepto, que significa modelar una base de datos relacional y saber cómo se genera una estructura relacional de datos válida. Por esta razón, el concepto de estructura de datos no debería ocasionar ningún problema en su compresión, aunque, para que esta estructura sea relacional y válida, se requiere la compresión de algunas normas o principios teóricos que la fundamentan.



La normalización de datos se refiere a modelar una base de datos relacional y saber cómo se genera una estructura relacional de datos válida.

También se descubrirá —después de leer este capítulo y de haber realizado una intensa práctica— que, en algunos casos, el diseño se alcanza en forma espontánea; es decir que, de forma intuitiva, se logran las estructuras que, sin duda, ya poseen un estado relacional válido, sin que ello represente una dificultad o un proceso adicional en el diseño de la base de datos.

Se podría agregar que, el lenguaje de consultas estructurado SQL, se comporta correctamente con una base de datos relacional normalizada en tercera forma normal (3FN). Esto es para que el motor de la base de datos funcione correctamente y para que tenga una respuesta apropiada. Una estructura no normalizada se puede procesar, también, por el motor de base de datos y acceder a ella a través de SQL, aunque existe la posibilidad de que se encuentren algunas dificultades y/o anomalías, distorsiones como redundancia de datos, valores no reales en sumatorias, etc. lo que contribuirá a que se piense que la herramienta no es útil.

En resumen, SQL es un lenguaje de consultas estructurado que opera sobre bases de datos relacionales y la normalización —con sus formas normales— es el fundamento teórico-práctico que se requiere para que el diseñador de una base de datos logre una estructura relacional apropiada, que funcionará correctamente con el motor de la base de datos, a la que se accede a través del SQL.

### 2.3.2 A qué se refiere la normalización

Con normalización nos referimos:

- A cómo se estructuran los datos representados por el concepto de atributo en un conjunto de relaciones que los contienen.
- A que los atributos no posean estructura interna.
- A la dependencia que se forma entre los atributos dentro de una tupla.
  - Que esta relación no se diluya en el acto de transformación.
  - Se mantenga persistente.
- A cómo se establecen relaciones entre las entidades, a través de las estructuras de atributos (estado relacional – dependencia funcional).
- A cómo se conforman estas estructuras de atributos.
  - Para localizar un registro dentro de una relación, con una clave primaria.
  - Para poder relacionar una relación con otra, a través de estas estructuras de atributos, con claves primarias y claves foráneas.

- Que los componentes de la clave primaria siempre contengan valores ciertos.
- A la cantidad de atributos que componen la clave primaria, observando que esta cantidad sea mínima.
- A la eliminación de la redundancia de datos, para que la transformación logre eliminar la repetición de datos, se sustituirán éstos, por valores de referencia que establecerán una relación con nuevas relaciones.
- A que todas las referencias entre relaciones, a través de sus atributos, estén representadas. Que no se establezcan valores referenciales sin que existan las entidades que contienen las descripciones de estas referencias.
- A la creación a una estructura lógica que sea fácil de comprender y de mantener.
- A facilitar el control de la manipulación de datos para que no se violen las restricciones de integridad, en cuanto a:
  - Inserción de flas.
  - Modificación de datos.
  - Borrado de flas.

### 2.3.3 A qué no se refiere la normalización

- Al comportamiento de los atributos no clave.
- Al tipo de los atributos.
  - Referido a su contenido.
    - \_ Texto
    - \_ Número
    - \_ Fecha
    - \_ Otros
- A la longitud de los atributos, ya que no tiene ninguna importancia el tamaño o dimensión de un atributo.
- A la nomenclatura de atributos y relaciones. No importa cómo se nombran las relaciones y sus atributos.
- A la ubicación de los atributos dentro de la cabecera de la relación. Representa lo mismo que un atributo esté en el primer lugar ordinal dentro de la cabecera de la relación o en cualquier otro orden.
- A la cantidad de relaciones, ya que puede tener una estructura de tres relaciones o de 300.
- A la cantidad de atributos en una relación y a la calidad de los datos que se cargan dentro de la entidad, más específicamente dentro de los atributos, ya que no tienen un valor intrínseco dentro de la normalización.
- A la forma de manipulación de los datos, a los que se podrá acceder desde cualquier medio habilitado por el motor de la base de datos. Esto no le compete a la normalización.
- A la compresión de los datos. Si alguno de los datos representa realmente una “caja negra”, a la normalización no le interesa, siempre y cuando esta información no pueda ser descompuesta en varios atributos distintos.



Normalizar una base de datos significa transformar un conjunto de datos que tienen una cierta complejidad en su entendimiento y que su distribución en el modelo provoca problemas de lógica en las acciones de manipulación de datos, tanto en las consultas como en la inserción y modificación de ellos, en una estructura de datos que posea un diseño claro, donde estos datos guarden coherencia y no pierdan su estado de asociación.

### 2.3.4 Normalización aspecto formal

Dado un conjunto de atributos y un conjunto de dependencias funcionales existentes entre los atributos que conforman un esquema de relación, se trata de transformar este esquema en un conjunto de n esquemas de relación, que satisfagan las siguientes premisas:

- Misma cantidad de información.
- No alterar la dependencia funcional.
- Reducir la redundancia de datos a su mínima expresión.
- Reducir al mínimo los problemas de lógica en la administración de los datos.

### 2.3.5 Concepto de normalizar

Normalizar una base de datos significa transformar un conjunto de datos que tienen una cierta complejidad en su entendimiento y que su distribución en el modelo provoca problemas de lógica en las acciones de manipulación de datos, tanto en las consultas como en la inserción y modificación de ellos, en una estructura de datos que posea un diseño claro, donde estos datos guarden coherencia y no pierdan su estado de asociación. Estos datos tienen una distribución balanceada, lo que permite la observación de los mismos en forma clara, no redundante, y que, por encima de esto, su lógica sea comprensible, fácil de entender y de mantener. Esta estructura resultante se denomina base de datos normalizada.

La transformación, desde el punto de vista de las entidades, significa que se operarán diversas acciones, entre las que se pueden citar el agrupamiento de atributos y su descomposición, en la que los datos compuestos se expresarán como una colección de datos simples.

Luego de estas transformaciones, quedarán conformadas las nuevas relaciones que cumplirán con determinadas restricciones, de acuerdo con el nivel de desarrollo alcanzado en la normalización. Las entidades están compuestas por una colección de dominios que, por lo general, califican el concepto y establecen la semántica de la entidad a la que pertenecen, como, por ejemplo, formato, tipo, estilo, ubicación, definición, etc. Estas relaciones contendrán la totalidad de los atributos originales, ni más ni menos de los que existían en la estructura no normalizada. Entonces, las relaciones se conformarán en una estructura de datos normalizada. Esta transformación se fundamenta por una serie de normas o reglas que establecen cómo se deben presentar las relaciones y sus atributos para que compongan una estructura normalizada. Estas normas o reglas se denominan formas normales.

Este pasaje se puede realizar de diversas formas, según la comprensión que realice el involucrado en la tarea. Esto quiere decir que el acto de transformación de un conjunto de relaciones no normalizadas a una estructura normalizada de relaciones, se puede realizar con cualquier método. Concretamente, no existe una técnica estándar para poder efectuar esta acción. El que no exista una técnica de normalización no quiere decir que no se pueda crear una que lo lleve a realizar esta acción con éxito. Se debe aclarar que la expresión antes vertida no está relacionada con la acción de realizar las proyecciones de una relación que no satisface una forma normal con las relaciones que sí tienen un formato más deseable y sí satisfacen una determinada forma normal, sino a que el diseñador experimentado podrá establecer una técnica de elaboración del modelo que no requiera tener que generar relaciones no normalizadas, para luego pasarlas a normalizadas en una acción de proyección, sino que

podrá, quizá, como una opción entre tantas, implantar en forma directa la relación que ya satisface la forma normal deseada para ese modelo.

Si bien el pasaje de un conjunto de relación no normalizadas a un conjunto de relaciones se puede realizar de cualquier forma, el resultado debe ser el mismo, independientemente del método que se haya utilizado.

La bibliografía en la materia —consultada para la elaboración de este capítulo— difiere con este autor en la manera de abordar el concepto “normalización”. Sin embargo, la metodología aquí utilizada se considera indispensable para la comprensión total del término.

Si n diseñadores:

- parten del mismo origen de datos y los interpretan como la misma colección de datos que les permite observar la estructura inicial, por ejemplo, una planilla que contiene el conjunto de datos en una estructura no normalizada,
- poseen el mismo nivel de conocimiento acerca del tema y cuentan con la misma documentación explicativa y
- con el conocimiento apropiado sobre cómo realizar la transformación hasta llegar a una estructura normalizada de relaciones, es decir que todos saben realizar el pasaje a un estado normalizado,

pueden realizar la transformación con cualquier método de razonamiento o con cualquier procedimiento lógico, pero todos siempre deben llegar al mismo resultado.

### 2.3.6 Objetivos de la normalización

- Reducir la redundancia de datos y, por lo tanto, las inconsistencias.
- Facilitar el mantenimiento de los datos y de los programas.
- Evitar anomalías en las operaciones de manipulación de datos.
- Reducir el impacto de los cambios en los datos.

## 2.4 Origen de los datos

### 2.4.1 Los datos

Se refiere a la obtención de los datos, recurso del que se parte para modelar y lograr, como resultado, una estructura de relaciones normalizadas.

Los datos pueden proceder de algún documento, de la información suministrada por un especialista en el tema (experto calificado o informante clave) o de su propia observación.

Cuando el origen de los datos proviene de un documento, ya se tiene gran parte de la tarea realizada y deberá hacer apuntes sobre algunas sugerencias.

Cuando la información proviene de un especialista, se tendrá, también, gran parte de la tarea resuelta; es importante documentar, en detalle, la información que vaya suministrando el especialista. En todos los casos, la interpretación —si existiera— debe ser acotada y validada con él y es recomendable que no se viole esa frontera, puesto que el acto de interpretar (explicar acciones, dichos o sucesos que pueden ser

entendidos de diferentes modos) no significa lo mismo que entender (Tener las ideas claras de las cosas o saber con perfección algo). Esto puede acarrear serias discrepancias en el resultado obtenido respecto de la realidad que debe refejar el modelo. Si se tuvieran dudas acerca de algún tema determinado, se evitará la interpretación. En estos casos, es indispensable el contacto con un especialista que suministrará la información necesaria que despejará todas las dudas.

Pero cuando el origen de datos proviene de la observación, y no se posee un documento o un especialista calificado que informe acerca del circuito de datos, es conveniente que se establezca un límite de entendimiento e interpretación del problema sobre el que se proyectará. Es claro que, cuando es uno el que observa una problemática, tiende a realizar una macrovisión del problema y de sus límites, desde los conocimientos que posee de la dificultad que desea resolver. Si el conocimiento del problema tratado es amplio, la normalización comprenderá una vasta cantidad de entidades y de atributos. Ahora, si el conocimiento es mínimo, la normalización tenderá a ser pequeña y endeble.

Esta contraposición pone de manifiesto que, cuando se realiza el diseño de una base de datos, siempre es conveniente poseer documentación que avale el diseño o la información suministrada por un especialista en el tema. Lo que queda claro es que fuera cual fuese el problema que se transformará en una base de datos normalizada, se deberá aprender del problema para entenderlo con total claridad.

Se recuerda que existe un dominio del conocimiento existente sobre el tema por tratar y esto es la totalidad de conocimiento sobre el tema, pero seguramente esta totalidad de conocimiento no representa el origen de los datos con los que se debe trabajar. Se trabajará con una porción acotada, que representa el problema en sí mismo. Entonces, se podrá determinar como el origen de los datos el dominio del problema. Éste está, sin dudas, contenido en el dominio del conocimiento general del tema que se tratará, pero esta acotado exclusivamente al problema.



El ambiente —desde el punto de vista de la informática— es el conjunto de procedimientos y acciones que se producen en el tratamiento de los datos y que fundamentan la estructura en la que éstos actúan.

#### 2.4.2 Problemática asociada al origen de datos

El origen de los datos no tiene ningún sentido si no se establece en el ambiente en el que aquellos interactúan. El ambiente, quizás más claramente expresado —desde el punto de vista de la Informática— como entorno, es el conjunto de procedimientos y acciones que se producen en el tratamiento de los datos y que fundamentan la estructura en la que éstos actúan. Este fundamento o conocimiento es la semántica, que se define como el conjunto de reglas que proporcionan el significado de funcionalidad de un proceso y que indican cómo se deben presentar los datos (o cómo deben ser presentados) en la estructura normalizada, para que sea funcional con respecto al propósito de la solución del problema.

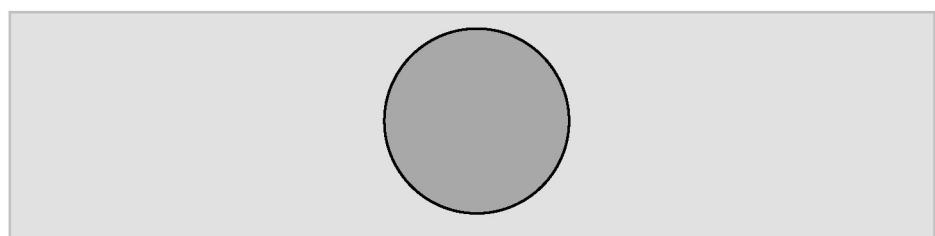


Fig. 2.1 Dominio del conocimiento total sobre el tema del problema.

En la Fig. 2.1, el recuadro externo simboliza el dominio de todo el conocimiento disponible sobre el tema que se normalizará y el círculo interno el conocimiento concreto sobre el que tratará la normalización. Lo que se intenta expresar es que la información que se obtendrá en el origen de datos, nunca alcanzará al total de la información disponible acerca del tema, sino que solo será una porción bien definida y acotada.

Es conveniente que se tenga una idea muy clara sobre el dominio del problema y cuál es su límite, pues esto permitirá que se alcance un resultado definido y limitado que se manejará con facilidad.

#### 2.4.3 Los resultados

Normalizar, ¿qué significa? En los párrafos precedentes, se expresó que los diseñadores siempre deben llegar, bajo ciertas condiciones, a un mismo resultado.

En su origen, el conjunto de datos es el mismo. Por esta razón, en el resultado obtenido luego de realizar la normalización, no debería faltar ningún dato y, tampoco, obtener otros adicionales.

El mismo resultado se logra cuando se tiene el mismo conjunto de datos, que realiza una correcta comprensión de la problemática, de la que se establecerá una semántica. Dentro de ésta, se prestará especial atención a dos de los elementos —que forman parte del vocabulario específico de la problemática de esta conceptualización—: los datos y los procesos de datos. De la observación de los datos, junto con su comportamiento funcional, se establecerán las entidades que tendrán un conjunto de características que las calificarán y determinarán los atributos de la entidad. Los procesos internos son las acciones que se desprenden del análisis de la problemática, que darán un estado de asociación, actividad y dependencia funcional entre los atributos establecidos en cada entidad.

Luego de realizar este análisis y una vez constituidos los componentes que actuarán en esta problemática, se diseñará un modelo de datos que se encuentre normalizado y que la represente de la manera más fácil posible. Este modelo no tiene mayores posibilidades de que se represente de otra forma más que la establecida, puesto que no existe una diversidad de estructuras o de relaciones para el mismo concepto y el conjunto de estructuras básicas está limitado a un número muy pequeño de éstas. Solo, en casos muy puntuales, existen estructuras compatibles entre sí, con la característica de que son levemente distintas, pero siempre una de ellas es más deseable que la otra en cuanto a las premisas del modelo relacional. Entonces, si la representación fuese distinta, se estaría ante el modelado de una problemática distinta de la entendida.



El término “semántica” se refiere a los aspectos del significado, sentido o interpretación del significado de un determinado elemento, símbolo, palabra, expresión o representación formal.

### 2.5 Las formas normales

#### 2.5.1 Las normas

Está claro que las seis formas normales no expresan las cosas simplemente. Es por esto que se expresarán estos conceptos, que tienen categoría de normas o de reglas, de otra manera. Para hacer una correcta interpretación de cada forma normal, se realizará un análisis detallado de la semántica que expresa los términos que utiliza y su porqué.

### 2.5.2 Dominio (extensión conceptual)

La menor unidad semántica de información es el valor de un dato indivisible: un escalar; por ejemplo, el código de cliente. Al ser este valor indivisible es de característica atómica; es decir, el dato no posee una estructura interna de composición, que no se debe interpretar como la falta de semántica del contenido, sino como la imposibilidad de dividir el dato que compone el dominio.

Otra característica del dominio es la homogeneidad de sus valores, todos del mismo tipo. Por ejemplo, el dominio "códigos de proveedores" es el conjunto de todos los posibles números de proveedores.

Los atributos se definen sobre los dominios subyacentes. Cada atributo se define sobre un solo dominio existente.

En su semántica, el dominio restringe su utilización, puesto que su definición no solo se refiere a las características de contenido como un valor de tipo cierto (ej. numérico), sino que va más allá y establece un tipo descriptivo propio. Por ello, dos atributos con contenido de tipo numérico son comparables si poseen la misma semántica de dominio. Para explicarlo de forma comparativa: si un atributo contiene el código de cliente y otro, el código de barrio, ambos pueden ser numéricos, pero no se pueden comparar, pues pertenecen a distintos dominios.

¿En dónde actúa y para qué? El motor de la base de datos solo permitirá, ejerciendo un control, que los atributos de las relaciones se carguen con valores permitidos por la semántica de su dominio. Ya se verá, más adelante, cómo se define esto en el nivel de diseño de estructura.

### 2.5.3 Fundamento teórico de las normas

#### 2.5.3.1 Dependencia Funcional

La Dependencia Funcional o DF es una limitación al conjunto de relaciones aceptadas, según las formas normales. Además, establece una relación fuerte y persistente entre dos atributos, al punto que el cambio de valor de uno determina en su valor al otro.

Las restricciones de la relación, en un momento dado, deben poder verificar el diseño de la relación.

Las tuplas deben, en todo momento, convalidar el diseño de la relación y sus restricciones.

La DF permite la formulación de algunas restricciones del diseño que se construye en la base de datos.

La DF establece en forma forzada que los valores contenidos en un conjunto de atributos definen únicamente el valor de otro conjunto de atributos.

Definición:

Dada una relación R, el atributo Y de R depende funcionalmente del atributo X de R ( $R.X$  determina funcionalmente a  $R.Y$ ), denotado como:

$$R.X \rightarrow R.Y$$

Significa que un solo valor Y en R está asociado a cada valor X en R. Donde X, Y pueden ser compuestos.

Ejemplo:

Sea el atributo X clave primaria en R, el atributo Y en R debe depender funcionalmente de forma forzada de X, por definición de clave primaria.

$$R.X \rightarrow R.Y$$

Ahora, ejemplificando con una relación R con una estructura del tipo factura (reducida), los atributos no clave de R —que representan el conjunto Y cliente, fecha, total— dependen cada uno de ellos de la clave primaria en R —representada por R.R#— , entonces:

$$\left. \begin{array}{l} R.R\# \rightarrow R.\text{cliente} \\ R.R\# \rightarrow R.\text{fecha} \\ R.R\# \rightarrow R.\text{total} \end{array} \right\} R.R\# \rightarrow R.(\text{cliente}, \text{fecha}, \text{total})$$

Por definición de DF y partiendo de la premisa de que X no es obligadamente clave primaria, al presentarse un mismo valor de X en diferentes tuplas dentro de R deben tener asociado el mismo valor de Y.

Otra definición:

Dada una relación R y el atributo Y de R tiene dependencia funcional del atributo X de R.

Significa que, cuando en dos tuplas de R, el valor X sea el mismo, entonces, debe corresponderles forzadamente el mismo valor de Y:

$$R.R\# \rightarrow R.(\text{cliente}, \text{fecha}, \text{total})$$

En la siguiente planilla se muestra un ejemplo esperado, según la definición antes expuesta.

| R.R#   |         | R.(\text{cliente}, \text{fecha}, \text{total}) |        |
|--------|---------|--|--------|
| Número | Cliente | Fecha  | Total  |
| 1      | Miranda | 01/01/2000                                     | 100,20 |
| 2      | López   | 25/02/2009                                     | 23,10  |
| 1      | Miranda | 01/01/2000                                     | 100,20 |

### 2.5.3.2 Dependencia Funcional Completa:

Dada una relación R, el atributo Y de R depende funcionalmente, “de forma completa”, del atributo X de R.

Depende funcionalmente de X y no depende funcionalmente de ningún subconjunto de X.

Si consideramos la relación:

$RA.(R\#,A\#) \rightarrow RA.(cliente, fecha, total, nombre\ articulo, precio)$

se observa que esta relación tiene dos conjuntos de valores que dependen cada uno de una parte de la clave primaria. El cliente, fecha y total dependen de la parte de la clave primaria  $R\#$ , mientras que nombre artículo y precio dependen de parte de la clave  $A\#$ . Por lo tanto, en esta relación expuesta no se cumple la dependencia funcional completa.

Si se descompone la relación RA en dos relaciones como se muestra a continuación:

$R.R\# \rightarrow R.(cliente, fecha, total)$

$A.A\# \rightarrow A.(nombre\ articulo, precio)$

Las relaciones R y A ahora poseen dependencia funcional completa, pues, para el caso de R, todos los atributos dependen por completo del atributo X de R. También se hace extensiva esta condición para la relación A.

### 2.5.4 Enumeración de las normas

Las formas normales son más de tres pero, para poder crear, entender y utilizar una base de datos relacional, que es nuestro objetivo, es suficiente con concebir una estructura de datos en tercera forma normal (3FN).

A continuación, se definen las formas normales:

#### Primera Forma Normal (1FN)

- Una relación se encuentra en primera forma normal si, y solo si, todos los dominios simples subyacentes de los atributos contienen valores atómicos y monovalentes.

#### Segunda Forma Normal (2FN)

- Una relación se encuentra en segunda forma normal si, y solo si, se encuentra en 1FN y, además, todos los atributos no clave dependen por completo de la clave primaria.

#### Tercera Forma Normal (3FN)

- Una relación se encuentra en tercera forma normal si, y solo si se encuentra en 2FN; y si ningún subconjunto de atributos no claves tiene dependencia funcional entre sí.

Atributos no primarios: se refiere a atributos no clave y que no pertenecen a ninguna combinación de claves candidatas posibles.

#### Forma Normal de Boyce-Codd (FNBC)

- Una relación está en FNBC si, y solo si, todo determinante es una clave candidata. Determinante es el atributo del que depende, funcionalmente y de manera completa, otro atributo o conjunto de éstos.
- Una relación está en FNBC si primero está en tercera forma normal y luego, si, y solo si, las únicas dependencias funcionales triviales se encuentran dadas entre la clave primaria y uno o varios atributos.

#### Cuarta Forma Normal (4FN)

- Una relación R está en cuarta forma normal, si primero está en forma normal Boyce-Codd y, además, todas las dependencias multivaluadas en esta relación R son “dependencias funcionales”.
- El concepto de dependencia multivaluada, pero que no son dependencias funcionales, es lo que impide que una relación se encuentre en cuarta forma normal.

#### Quinta Forma Normal (5FN)

- Una relación R está en quinta forma normal si, y solo si, toda dependencia de reunión en R es una consecuencia de las claves candidatas en R.
- Una relación R está en 5FN si, y solo si, toda dependencia de reunión está condicionada por las claves candidatas de R.

#### 2.5.5 Interpretación y aplicación de las formas normales

Se inicia la interpretación con un conjunto de datos que conforman una estructura clásica de un sistema de ventas. Estos datos están conformados por tres (3) facturas de una empresa cualquiera, que solicita crear una base de datos de ventas. Estos datos se muestran en una planilla que, de ahora en más, se denominará relación “Origen de datos” y que representa la totalidad de los datos que hay en las facturas de esta empresa, tal como muestra la siguiente relación.

| Relación Origen de los datos |                     |                       |                    |                    |                     |                     |                       |                              |                       |                     |
|------------------------------|---------------------|-----------------------|--------------------|--------------------|---------------------|---------------------|-----------------------|------------------------------|-----------------------|---------------------|
| Sucursal y número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Código del artículo | Nombre del artículo | Cantidad del artículo | Precio unitario del artículo | Subtotal del artículo | Total de la factura |
| 01-500                       | 1/01/06             | E                     | 01                 | ALVAREZ            | 01                  | LÁPIZ               | 3                     | 1.25                         | 3.75                  | 48.20               |
| 01-500                       | 1/01/06             | E                     | 01                 | ALVAREZ            | 02                  | GOMA                | 6                     | 0.75                         | 4.50                  | 48.20               |
| 01-500                       | 1/01/06             | E                     | 01                 | ALVAREZ            | 10                  | HOJAS               | 8                     | 5.00                         | 40.00                 | 48.20               |
| 01-501                       | 2/01/06             | CC                    | 107                | CASTRO             | 08                  | COMPÁS              | 4                     | 4.00                         | 16.00                 | 16.00               |
| 02-500                       | 3/01/06             | E                     | 110                | LIZ                | 20                  | REGLA               | 2                     | 2.45                         | 4.90                  | 14.90               |
| 02-500                       | 3/01/06             | E                     | 110                | LIZ                | 10                  | HOJAS               | 2                     | 5.00                         | 10.00                 | 14.90               |

### 2.5.5.1 Interpretación de la Primera Forma Normal (1FN)

Se recordará que la 1FN se definió de la siguiente manera: una relación se encuentra en Primera Forma Normal si, y solo si, todos los dominios simples subyacentes de los atributos contienen valores atómicos y monovalentes.

Si observa el origen de datos de la relación “Origen de datos”, las columnas que contienen una composición de datos o combinación de datos serán, en este caso, de nuestro interés.

La primera columna, “Sucursal y Número de factura”, contiene una combinación de datos: “Sucursal” y “Número de factura”.

Esta columna puede dividirse en dos columnas con valores atómicos: por un lado, “Sucursal” y, por otro, “Número de factura”.

Entonces, al aplicar una parte de la 1FN —la referida a atributos atómicos—, resulta en un cambio de la relación “Origen de datos”, que queda de la forma siguiente:

| Relación Origen de los datos |                   |                     |                       |                    |                    |                     |                     |                       |                              |                       |                     |
|------------------------------|-------------------|---------------------|-----------------------|--------------------|--------------------|---------------------|---------------------|-----------------------|------------------------------|-----------------------|---------------------|
| Sucursal                     | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Código del artículo | Nombre del artículo | Cantidad del artículo | Precio unitario del artículo | Subtotal del artículo | Total de la factura |
| 01                           | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 01                  | LÁPIZ               | 3                     | 1.25                         | 3.75                  | 48.20               |
| 01                           | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 02                  | GOMA                | 6                     | 0.75                         | 4.50                  | 48.20               |
| 01                           | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 10                  | HOJAS               | 8                     | 5.00                         | 40.00                 | 48.20               |
| 01                           | 501               | 2/01/06             | CC                    | 107                | CASTRO             | 08                  | COMPÁS              | 4                     | 4.00                         | 16.00                 | 16.00               |
| 02                           | 500               | 3/01/06             | E                     | 110                | LIZ                | 20                  | REGLA               | 2                     | 2.45                         | 4.90                  | 14.90               |
| 02                           | 500               | 3/01/06             | E                     | 110                | LIZ                | 10                  | HOJAS               | 2                     | 5.00                         | 10.00                 | 14.90               |



Cuando se define 1FN y se refiere a atributo monovalente, se quiere expresar que, dentro de una estructura normalizada, los valores no pueden repetirse y que deben expresarse una sola vez por ocurrencia. Las tuplas que tienen repetidas varias veces la misma información se reducirán a una única instancia. En otras palabras, se eliminarán los grupos repetitivos.

Cuando se define 1FN y se refiere a atributo monovalente, se quiere expresar que, dentro de una estructura normalizada, los valores no pueden repetirse y que deben expresarse una sola vez por ocurrencia. Las tuplas que tienen repetidas varias veces la misma información se reducirán a una única instancia. En otras palabras, se eliminarán los grupos repetitivos.

En este caso, una factura posee un cliente, una fecha de emisión, una sucursal, un número de factura, un total y una forma de pago, los valores de estos atributos se repiten varias veces para la misma factura.

Veamos la siguiente relación, en la que cada factura está sombreada por un contraste diferente. Estos datos deben presentarse una sola vez en la estructura normalizada en 1FN. Entonces, sucursal, número de factura, fecha, forma de pago, identificación del cliente, nombre del cliente y total de factura deben figurar una sola vez.

| Relación Origen de los datos |                   |                     |                       |                    |                    |                     |                     |                       |                              |                       |                     |
|------------------------------|-------------------|---------------------|-----------------------|--------------------|--------------------|---------------------|---------------------|-----------------------|------------------------------|-----------------------|---------------------|
| Sucursal                     | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Código del artículo | Nombre del artículo | Cantidad del artículo | Precio unitario del artículo | Subtotal del artículo | Total de la factura |
| 01                           | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 01                  | LAPIZ               | 3                     | 1.25                         | 3.75                  | 48.20               |
| 01                           | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 02                  | GOMA                | 6                     | 0.75                         | 4.50                  | 48.20               |
| 01                           | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 10                  | HOJAS               | 8                     | 5.00                         | 40.00                 | 48.20               |
| 01                           | 501               | 2/01/06             | CC                    | 107                | CASTRO             | 08                  | COMPAS              | 4                     | 4.00                         | 16.00                 | 16.00               |
| 02                           | 500               | 3/01/06             | E                     | 110                | LIZ                | 20                  | REGLA               | 2                     | 2.45                         | 4.90                  | 14.90               |
| 02                           | 500               | 3/01/06             | E                     | 110                | LIZ                | 10                  | HOJAS               | 2                     | 5.00                         | 10.00                 | 14.90               |

Si se reducen las ocurrencias de los datos que se repiten, se perderá el detalle de la factura; entonces, para que se realice la eliminación de la repetición y para que no se pierda el detalle de la factura, compuesto por los atributos restantes —los que no se repiten—, no hay otro camino que descomponer los datos de la relación original en dos tablas: una que contenga los datos que se repiten y otra, con los datos que no se repiten, tal como lo ilustran las siguientes tablas:

| Tabla A  |                   |                     |                       |                    |                    |                     |
|----------|-------------------|---------------------|-----------------------|--------------------|--------------------|---------------------|
| Sucursal | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Total de la factura |
| 01       | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 48.20               |
| 01       | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 48.20               |
| 01       | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 48.20               |
| 01       | 501               | 2/01/06             | CC                    | 107                | CASTRO             | 16.00               |
| 02       | 500               | 3/01/06             | E                     | 110                | LIZ                | 14.90               |
| 02       | 500               | 3/01/06             | E                     | 110                | LIZ                | 14.90               |

| Tabla B             |                     |                       |                              |                       |
|---------------------|---------------------|-----------------------|------------------------------|-----------------------|
| Código del artículo | Nombre del artículo | Cantidad del artículo | Precio unitario del artículo | Subtotal del artículo |
| 01                  | LAPIZ               | 3                     | 1.25                         | 3.75                  |
| 02                  | GOMA                | 6                     | 0.75                         | 4.50                  |
| 10                  | HOJAS               | 8                     | 5.00                         | 40.00                 |
| 08                  | COMPAS              | 4                     | 4.00                         | 16.00                 |
| 20                  | REGLA               | 2                     | 2.45                         | 4.90                  |
| 10                  | HOJAS               | 2                     | 5.00                         | 10.00                 |

Ahora se tienen dos tablas separadas: la tabla "A" tiene los atributos que contienen los datos repetidos, y la tabla "B", los atributos que no contienen los datos repetidos por grupo (entiéndase por grupo a una factura).

En la tabla "A", es evidente que los datos se repiten conformando un grupo en el que quedan más de una tupla o fila con los mismos datos. En este caso, hay que reducir las tuplas duplicadas y dejar solo una por grupo.

Hasta ahora se ha realizado la separación de los atributos y datos en dos grupos: uno que contiene los datos que se repiten y el otro, con los datos que no se repiten (tabla "B"). Entonces, en la tabla "A" se eliminarán las ocurrencias duplicadas de cada grupo, pero si esto se realiza sin establecer de antemano un mecanismo de referencias entre las tablas, se perderá la relación entre éstas que, por ahora, es evidente en forma visual. En definitiva, no podrá establecer qué artículos corresponden a qué facturas y se tendrán los detalles sin saber a qué factura corresponden.

Por esta razón, se debería dejar solo una fila de cada grupo en la siguiente tabla:

| Sucursal | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Total de la factura |
|----------|-------------------|---------------------|-----------------------|--------------------|--------------------|---------------------|
| 01       | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 48.20               |
| 01       | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 48.20               |
| 01       | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 48.20               |
| 01       | 501               | 2/01/06             | CC                    | 107                | CASTRO             | 16.00               |
| 02       | 500               | 3/01/06             | E                     | 110                | LIZ                | 14.90               |
| 02       | 500               | 3/01/06             | E                     | 110                | LIZ                | 14.90               |

quedando así lo que muestra la tabla de la izquierda:

| Sucursal | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Total de la factura |
|----------|-------------------|---------------------|-----------------------|--------------------|--------------------|---------------------|
| 01       | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 48.20               |
| 01       | 501               | 2/01/06             | CC                    | 107                | CASTRO             | 16.00               |
| 02       | 500               | 3/01/06             | E                     | 110                | LIZ                | 14.90               |

| Código del artículo | Nombre del artículo | Cantidad del artículo | Precio unitario del artículo | Subtotal del artículo |
|---------------------|---------------------|-----------------------|------------------------------|-----------------------|
| 01                  | LAPIZ               | 3                     | 1.25                         | 3.75                  |
| 02                  | GOMA                | 6                     | 0.75                         | 4.50                  |
| 10                  | HOJAS               | 8                     | 5.00                         | 40.00                 |
| 08                  | COMPAS              | 4                     | 4.00                         | 16.00                 |
| 20                  | REGLA               | 2                     | 2.45                         | 4.90                  |
| 10                  | HOJAS               | 2                     | 5.00                         | 10.00                 |

De esta forma, las dos tablas no contienen valores repetidos, pero surge —como ya se planteará con otro problema— que los datos que antes estaban integrados en

una sola relación, y que tenían correspondencia entre sí, ahora están disociados. Los datos se encuentran en dos relaciones del margen inferior de la página anterior y no se puede establecer su dependencia funcional, sin saber qué detalle corresponde a cada factura; por ello, en la relación inferior derecho, se colorearon los grupos que pertenecen a la misma factura para individualizarlos. En otras palabras, no se sabe qué artículo de detalle corresponde a qué factura, salvo por su color de fondo y por su orden.

Entonces, surge una nueva necesidad que es establecer una referencia que indique qué artículos de los detalles corresponden a qué factura. También, es necesario establecer el nombre para cada relación, para poder identificarlas claramente en la nueva estructura resultante:

- A la relación en la que se produce la eliminación de los grupos repetitivos, se la denominará “Factura”.
- A la relación en la que quedaron los atributos que no conforman el grupo repetitivo, se la denominará “Detalle de factura”.

De esta manera, se establece, como identificador único (PK) en la relación “Factura”, la combinación de las columnas “Sucursal” y “Número de factura”.

| Relación Factura |                   |                     |                       |                    |                    |                     |
|------------------|-------------------|---------------------|-----------------------|--------------------|--------------------|---------------------|
| Sucursal         | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Total de la factura |
| 01               | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 48.20               |
| 01               | 501               | 2/01/06             | CC                    | 107                | CASTRO             | 16.00               |
| 02               | 500               | 3/01/06             | E                     | 110                | LIZ                | 14.90               |

| Relación Detalle de factura |                     |                       |                              |                       |
|-----------------------------|---------------------|-----------------------|------------------------------|-----------------------|
| Código del artículo         | Nombre del artículo | Cantidad del artículo | Precio unitario del artículo | Subtotal del artículo |
| 01                          | LAPIZ               | 3                     | 1.25                         | 3.75                  |
| 02                          | GOMA                | 6                     | 0.75                         | 4.50                  |
| 10                          | HOJAS               | 8                     | 5.00                         | 40.00                 |
| 08                          | COMPÁS              | 4                     | 4.00                         | 16.00                 |
| 20                          | REGLA               | 2                     | 2.45                         | 4.90                  |
| 10                          | HOJAS               | 2                     | 5.00                         | 10.00                 |

| Relación Factura |          |                   |                     |                       |                    |                    |
|------------------|----------|-------------------|---------------------|-----------------------|--------------------|--------------------|
| Pk               | Sucursal | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente |
|                  | 01       | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            |
|                  | 01       | 501               | 2/01/06             | CC                    | 107                | CASTRO             |
|                  | 02       | 500               | 3/01/06             | E                     | 110                | LIZ                |

Estas dos columnas conforman la clave primaria de la tabla “Factura”.

### ¿Por qué la combinación de ambos?

La columna “Número de factura” (relación “Factura”) tiene valores que se repiten en dos facturas distintas (valor 500), pero se puede observar que provienen de dos sucursales diferentes. Por esta razón, el número de factura, para este modelo de datos, es insuficiente como identificador o clave primaria. Para que se logre una localización inequívoca de cada registro, se incorporará otro atributo complementario que ayudará a tal efecto; en este caso, el atributo “Sucursal”.

Este identificador inequívoco, como se definió en los párrafos precedentes, se denomina clave primaria y se representa con la sigla PK (Primary Key).

La relación que se establece entre el conjunto clave primaria y el conjunto formado por los atributos no claves es única. Cada componente de la clave primaria se relaciona únicamente con un único conjunto de valores de atributos no clave.

La forma que se tiene para relacionar las relaciones “Detalle de factura” y “Factura”, es que cada registro del detalle de factura pueda indicar, con información propia del registro, con qué registro se asocia dentro de la relación “Factura”. Esto se logrará si se incorporan, dentro de la relación “Detalle de factura”, los componentes de PK de la relación “Factura”. Entonces, se agregarán las columnas que componen la PK de la relación “Factura” en la relación “Detalle de Factura”:

| Relación Factura |                   |                     |                       |                    |                    |                     |
|------------------|-------------------|---------------------|-----------------------|--------------------|--------------------|---------------------|
| Sucursal         | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Total de la factura |
| 01               | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 48.20               |
| 01               | 501               | 2/01/06             | CC                    | 107                | CASTRO             | 16.00               |
| 02               | 500               | 3/01/06             | E                     | 110                | LIZ                | 14.90               |

| Relación Detalle de Factura |                   |                     |                     |                       |                              |                       |
|-----------------------------|-------------------|---------------------|---------------------|-----------------------|------------------------------|-----------------------|
| Sucursal                    | Número de factura | Código del artículo | Nombre del artículo | Cantidad del artículo | Precio unitario del artículo | Subtotal del artículo |
| 01                          | 500               | 01                  | LAPIZ               | 3                     | 1.25                         | 3.75                  |
| 01                          | 500               | 02                  | GOMA                | 6                     | 0.75                         | 4.50                  |
| 01                          | 500               | 10                  | HOJAS               | 8                     | 5.00                         | 40.00                 |
| 01                          | 501               | 08                  | COMPAS              | 4                     | 4.00                         | 16.00                 |
| 02                          | 500               | 20                  | REGLA               | 2                     | 2.45                         | 4.90                  |
| 02                          | 500               | 10                  | HOJAS               | 2                     | 5.00                         | 10.00                 |

Columnas agregadas en la tabla “Detalle de Factura”, para establecer la relación con la tabla “Factura”.

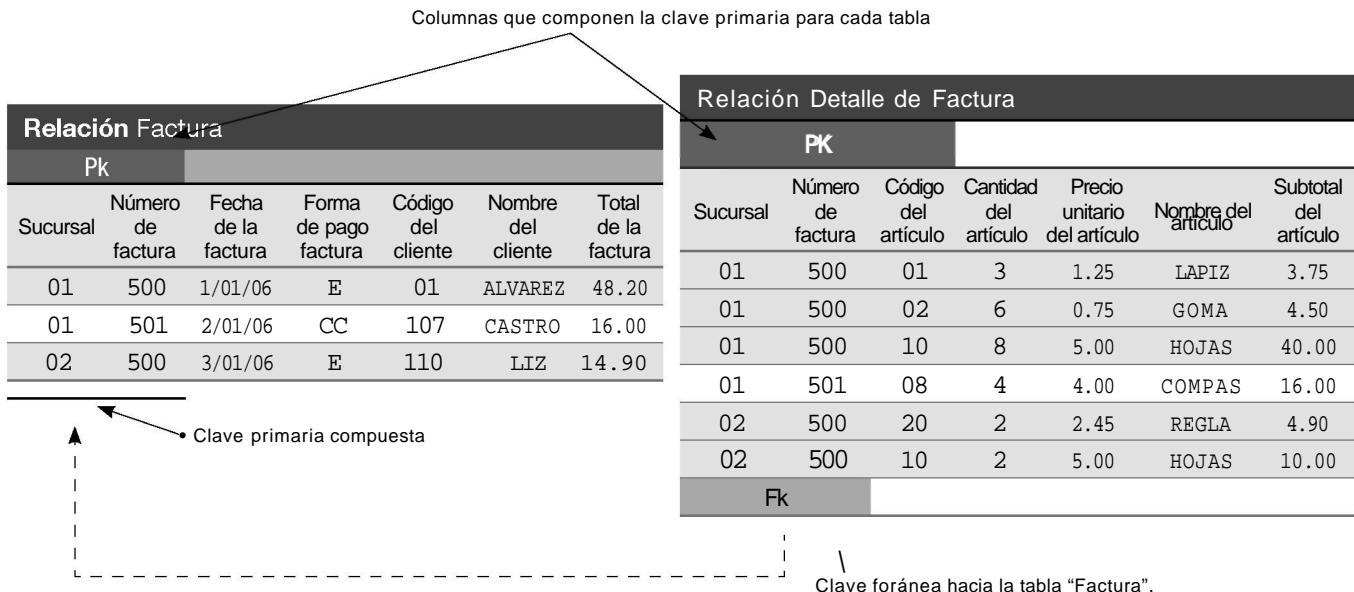
Dentro de la relación “Detalle de Factura”, es visible que hay datos duplicados luego de arrastrar la clave primaria desde la relación “Factura”, en este caso, “Sucursal” y “Número de factura”.

La cuestión es que, cuando se construye una estructura relacional en un base de datos relacional, la única forma de establecer la relación entre las relaciones es arrastrando la PK de una relación hacia la otra. Si bien hay redundancia de datos, ésta es una redundancia controlada y necesaria para establecer el estado de relaciones y de dependencias entre los datos, que no se debe perder en la creación del nuevo modelo.

En la relación “Detalle de factura”, la PK, que se trajo desde la relación “Factura”, no es suficiente para identificar las filas de la relación “Detalle de factura” en forma inequívoca.

¿Por qué? Porque se repite la combinación de las columnas “Sucursal” y “Número de factura”.

Ahora, si se agregara algún componente nativo de la relación, quizás se logre una combinación única. La correcta sería “Código de artículo”, entonces, quedaría (relación “Detalle de Factura”):



Esta nueva combinación de atributos sí identifica a cada fila de la relación.

La clave primaria se simbolizará con PK, que será el identificador inequívoco de la tupla dentro de la relación, que se podría componer con uno o más atributos, la que será mínima, ya que en una relación se podrían establecer varias combinaciones de clave primaria; pero solo será válida y conveniente la que se forme con la menor cantidad de atributos que mantenga la condición de unicidad.

La clave foránea se simbolizará con FK (Foreign Key) y se recordará que es el identificador de la relación que tendrá una relación determinada para establecer un vínculo con otra relación. Estas relaciones, que integrarán la nueva estructura, se vincularán entre sí de forma fácil, sin perder la dependencia funcional establecida en la estructura antes de ser normalizada, y cuya lógica sea fácil de entender y mantener.

Lo que interesa en la normalización son las estructuras de datos. Por esta razón, se verá, a continuación, cómo han quedado las estructuras de datos en su conversión a 1FN.

Es conveniente declarar las claves primarias con la sigla PK a la izquierda de los atributos que la componen y se los identificará con una sola expresión, “PK”. En el caso de que la clave primaria sea compuesta, se adoptarán las medidas para que se interpreten correctamente cuáles son los atributos integrantes.

Para la clave foránea, se utilizará la sigla FK y se la ubicará a la derecha de los atributos que intervienen. En caso de que la FK sea compuesta, se manejará el mismo método que para la PK.

**Lo correcto, en ambas situaciones, para clave compuesta o clave simple.**

|    |     |    |     |
|----|-----|----|-----|
| Pk | Suc | Pk | Nro |
|    | Nro |    |     |

**Lo incorrecto**

|    |     |
|----|-----|
| Pk | Suc |
| Pk | Nro |

El modelo en estudio, transformado a 1FN, se representa como sigue (Modelo 1FN):

| Modelo 1FN: Factura |                       | Detalle de Factura |                              |
|---------------------|-----------------------|--------------------|------------------------------|
| Pk                  | Sucursal              | Sucursal           | Fk                           |
|                     | Número de factura     |                    | Número de factura            |
|                     | Fecha de la factura   |                    | Código del artículo          |
|                     | Forma de pago factura |                    | Nombre del artículo          |
|                     | Código del cliente    |                    | Cantidad del artículo        |
|                     | Nombre del cliente    |                    | Precio unitario del artículo |
|                     | Total de la factura   |                    | Subtotal del artículo        |

Cada una representa una relación del modelo de datos que se está diseñando y que contienen la misma cantidad de datos que el modelo original, y mantienen la dependencia entre los componentes igual que en el primer diseño.

#### 2.5.5.2 Interpretación de la Segunda Forma Normal (2FN)

Una relación se encuentra en segunda forma normal si, y solo si, se encuentra en 1FN y si todos los atributos no clave dependen por completo de la clave.

Que refiera a que primero se encuentre en Primera Forma Normal deja sentado explícitamente que nunca se puede llegar a Segunda Forma Normal sin antes cumplir con la Primera.

Cuando se refiere a que los atributos no clave dependen por completo de la clave primaria y no a una parte de ella, se debe entender como atributos no clave a todos los atributos que no componen la clave primaria. Ninguno de ellos dependerá de una parte de la clave primaria.

Si un atributo no clave hace referencia una parte de la clave primaria, se interpretará que existe un subconjunto de la PK que satisface una dependencia funcional, y por ello no se cumplimentará la 2FN.

Al hacer referencia a una parte de la clave primaria, se interpretará que no existe un subconjunto de la PK que satisface una dependencia funcional con un atributo no clave

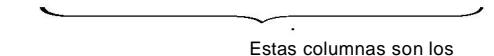
de la relación. Cuando la explicación habla de algún atributo no clave, no se refiere a cualquiera, sino a uno determinado que mantiene esa DF con otro perteneciente a la clave primaria. Esta DF podría también ser compuesta en cualquiera de los extremos.

Si existe un atributo “A”, componente de la clave primaria, que determina el contenido del atributo “B”, que no pertenece a la clave primaria, entonces no se cumple la Segunda Forma Normal.

Es necesaria una clave primaria compuesta, conformada por varios atributos, para que se pueda verificar esta situación. En el caso de la clave primaria simple, no hay razones para analizar la Segunda Forma Normal, pues se cumple por defecto.

Si en las dos relaciones que hasta ahora se tienen como resultado, podemos detectar la situación de que un atributo no clave depende de uno o varios integrantes de la clave primaria y no depende de toda la clave primaria, entonces encontramos un caso de Segunda Forma Normal.

Estas son las dos relaciones:

| Relación Factura  |                   |                     |                       |                    |                    | Relación Detalle de Factura  |          |                   |                     |                       |                              |                     |                       |
|---|-------------------|---------------------|-----------------------|--------------------|--------------------|--|----------|-------------------|---------------------|-----------------------|------------------------------|---------------------|-----------------------|
| Pk  |                   |                     |                       |                    |                    | PK   |          |                   |                     |                       |                              |                     |                       |
| Sucursal  | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Total de la factura  | Sucursal | Número de factura | Código del artículo | Cantidad del artículo | Precio unitario del artículo | Nombre del artículo | Subtotal del artículo |
| 01  | 500               | 1/01/06             | E                     | 01                 | ALVAREZ            | 48.20  | 01       | 500               | 01                  | 3                     | 1.25                         | LAPIZ               | 3.75                  |
| 01  | 501               | 2/01/06             | CC                    | 107                | CASTRO             | 16.00  | 01       | 500               | 02                  | 6                     | 0.75                         | GOMA                | 4.50                  |
| 02  | 500               | 3/01/06             | E                     | 110                | LIZ                | 14.90  | 01       | 500               | 10                  | 8                     | 5.00                         | HOJAS               | 40.00                 |
|  |                   |                     |                       |                    |                    |  |          |                   |                     |                       |                              |                     |                       |

Estas columnas son los atributos no clave.

En la relación “Factura”, se debería encontrar que algún atributo no clave depende de una parte de la clave primaria (PK).

Por ejemplo, la fecha ¿podría depender solamente de la “Sucursal” o solo del “Número de factura”? Claramente, no, la fecha es un atributo que depende de la PK completa. Para la primera factura, cuya sucursal es 01 y número de factura 500, la fecha se asocia en el momento en que se generó la factura; ésta está representada por ambos componentes de la PK, por lo tanto, la fecha depende por completo de la PK, y no de una parte de ella. Para los demás componentes de esta relación, se podría llegar a la misma conclusión.

¿Por qué actúan en forma conjunta los dos atributos de la clave primaria y se deben interpretar como una unidad?

No es lo mismo la factura 500 de la sucursal 01 que la factura 500 de cualquier otra sucursal. Cada sucursal tendrá —sin dudas— la factura 500, que corresponderá cada una a un momento particular y con distintos valores asignados. Por esta razón, para distinguir entre tantas facturas número 500 es imprescindible encontrar un elemento que las caracterice y que permita individualizarla, este es el atributo “Sucursal”.

Se analizará la siguiente relación:

| Relación Detalle de Factura |                   |                     |                       |                              |                     |                       |
|-----------------------------|-------------------|---------------------|-----------------------|------------------------------|---------------------|-----------------------|
| PK                          |                   |                     |                       |                              |                     |                       |
| Sucursal                    | Número de factura | Código del artículo | Cantidad del artículo | Precio unitario del artículo | Nombre del artículo | Subtotal del artículo |
| 01                          | 500               | 01                  | 3                     | 1.25                         | LAPIZ               | 3.75                  |
| 01                          | 500               | 02                  | 6                     | 0.75                         | GOMA                | 4.50                  |
| 01                          | 500               | 10                  | 8                     | 5.00                         | HOJAS               | 40.00                 |
| 01                          | 501               | 08                  | 4                     | 4.00                         | COMPAS              | 16.00                 |
| 02                          | 500               | 20                  | 2                     | 2.45                         | REGLA               | 4.90                  |
| 02                          | 500               | 10                  | 2                     | 5.00                         | HOJAS               | 10.00                 |

Fk

↑

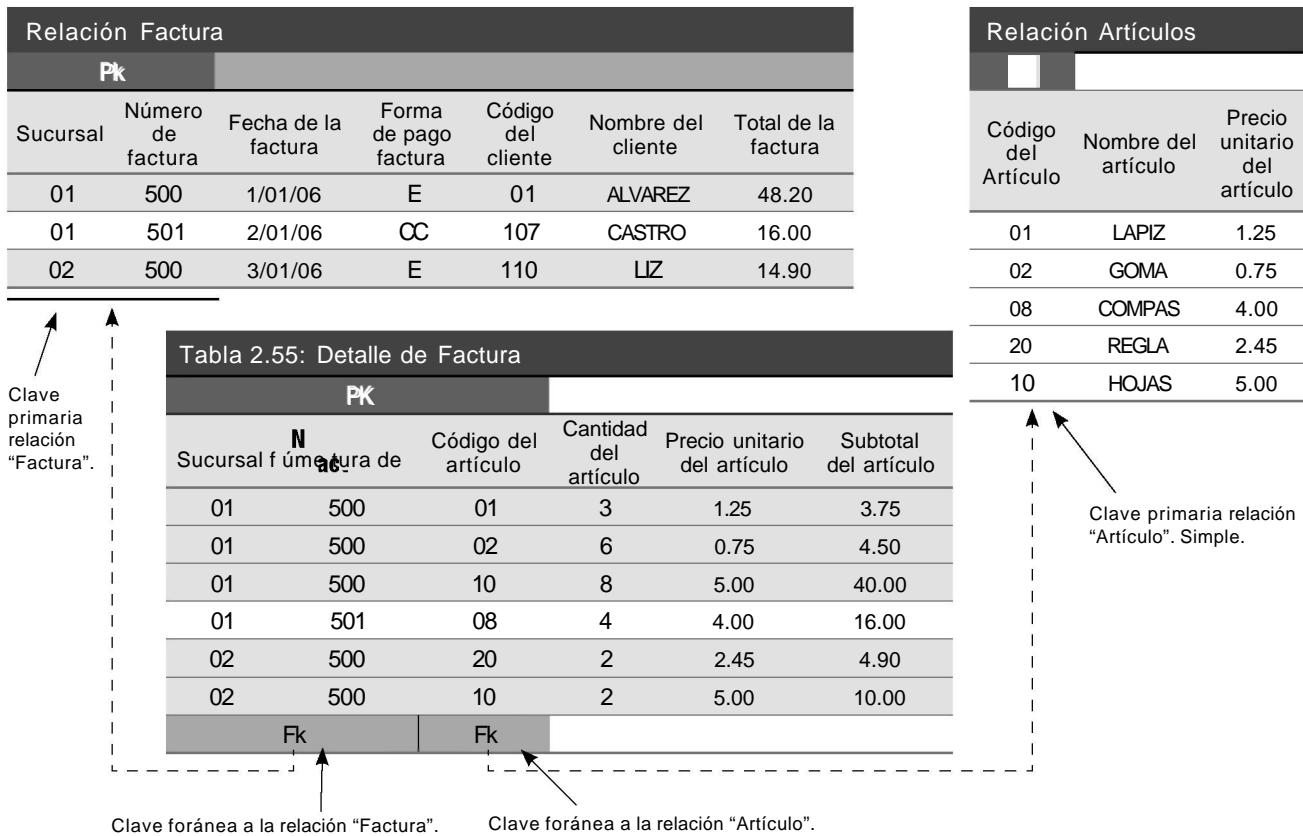
Subconjunto de la clave primaria que determina los atributos no primarios dentro de la relación.

Columnas no primarias que están asociadas al código del artículo, que integran la clave primaria.

Aquí, sí existe una relación entre atributos no clave y un componente de la clave primaria. Este caso, es la relación directa que existe entre la columna “Nombre del artículo” y “Precio unitario del artículo”, con el atributo “Código de artículo”, componente de la clave primaria. Si cambia el código del artículo debe cambiar, necesariamente, la descripción del artículo y el precio.

#### 2.5.5.2.1 ¿Cómo solucionar esta situación?

Se descompondrá la relación “Detalle de Factura” en dos relaciones: una que se mantendrá con el nombre “Detalle de Factura” y otra que se denominará “Artículos”. Se extraerá de la relación “Detalle de Factura” las columnas correspondientes al nombre del artículo y al precio, que se encontrarán en la relación “Artículos”. Además, se establecerá un vínculo entre las dos relaciones, tal como se detallará a continuación:



Ahora se tiene una nueva relación que se denominará "Artículos" y que contendrá el identificador del artículo, su nombre y su precio.

¿Se tienen de nuevo los datos duplicados? Sí, en efecto, el código de artículo se encuentra en las relaciones "Detalle de Factura" y "Artículos", ésta es la única forma de establecer una relación referencial, que permite mantener la misma dependencia funcional que existía en el modelo de la relación "Factura" anterior.

¿Pero por qué el precio en ambas relaciones? La respuesta es sencilla pero contundente a la hora de la calidad de la información que reside en el análisis de los datos. En "Artículos", la columna "Precio" indica el precio actual de cada artículo que, en el pasado, pudo haber sido otro y que, seguramente, variará en el futuro. En cambio, la columna "Precio" de la relación "Detalle de Factura" contiene el precio del artículo en el momento en que se realizó la venta.

Sería correcto interpretar que determinados atributos poseen diferentes valores asociados en el tiempo y que no permanecen constantes; entonces, de ninguna manera, se referenciará el precio en la relación "Artículos" como constante. En este caso, se toma el último precio de la relación "Artículos" y se lo transfiere a la relación "Detalle de factura", en el momento en que se realiza la registración.

El modelo en estudio transformado a 2FN se representa de la siguiente manera (Mod. 2FN):

## Mod. 2FN

| Factura |                       | Detalle de Factura |                              | Artículos |                     |
|---------|-----------------------|--------------------|------------------------------|-----------|---------------------|
| Pk      | Sucursal              | Pk                 | Sucursal                     | Pk        | Código del artículo |
|         | Número de factura     |                    | Número de factura            |           | Nombre del artículo |
|         | Fecha de la factura   |                    | Código del artículo          |           | Precio unitario     |
|         | Forma de pago factura |                    | Cantidad del artículo        |           |                     |
|         | Código del cliente    |                    | Precio unitario del artículo |           |                     |
|         | Nombre del cliente    |                    | Subtotal del artículo        |           |                     |
|         | Total de la factura   |                    |                              |           |                     |

## 2.5.5.3 Interpretación de la Tercera Forma Normal (3FN)

Una relación se encuentra en Tercera Forma Normal si, y solo si, se encuentra en 2FN, y si los atributos no clave dependen de forma no transitiva de la clave primaria.

Que primero se encuentre en "Segunda Forma Normal" deja sentado, explícitamente, que nunca se llegará a "Tercera Forma Normal" sin antes cumplir con la segunda.

Estas son las relaciones con su contenido y cómo han quedado:

| Relación Factura |          |                   |                     |                       |                    |                    | Relación Artículos  |                     |                     |                              |
|------------------|----------|-------------------|---------------------|-----------------------|--------------------|--------------------|---------------------|---------------------|---------------------|------------------------------|
| Pk               | Sucursal | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Total de la factura | Código del Artículo | Nombre del artículo | Precio unitario del artículo |
| 01               | 500      | 1/01/06           | E                   | 01                    | ALVAREZ            |                    | 48.20               | 01                  | LAPIZ               | 1.25                         |
| 01               | 501      | 2/01/06           | OC                  | 107                   | CASTRO             |                    | 16.00               | 02                  | GOMA                | 0.75                         |
| 02               | 500      | 3/01/06           | E                   | 110                   | LIZ                |                    | 14.90               | 08                  | COMPAS              | 4.00                         |

| Relación Detalle de Factura |                   |                     |                       |                              |                       |  |
|-----------------------------|-------------------|---------------------|-----------------------|------------------------------|-----------------------|--|
| PK                          |                   |                     | FK                    |                              |                       |  |
| Sucursal                    | Número de factura | Código del artículo | Cantidad del artículo | Precio unitario del artículo | Subtotal del artículo |  |
| 01                          | 500               | 01                  | 3                     | 1.25                         | 3.75                  |  |
| 01                          | 500               | 02                  | 6                     | 0.75                         | 4.50                  |  |
| 01                          | 500               | 10                  | 8                     | 5.00                         | 40.00                 |  |
| 01                          | 501               | 08                  | 4                     | 4.00                         | 16.00                 |  |
| 02                          | 500               | 20                  | 2                     | 2.45                         | 4.90                  |  |
| 02                          | 500               | 10                  | 2                     | 5.00                         | 10.00                 |  |
|                             |                   | FK                  | FK                    |                              |                       |  |

En la definición, se dice que ningún subconjunto tendrá dependencia transitiva; esto se refiere a que el subconjunto estará compuesto por atributos que no pertenecen a la clave primaria. Esta es la clave del entendimiento de la Tercera Forma Normal.

Entonces, se deberá encontrar, dentro de la relación, un subconjunto de atributos con dependencia transitiva, que ninguno de ellos pertenezca a la clave primaria y que, al cambiar el valor de un atributo, necesariamente cambiarán su valor otros atributos también.

| Relación Factura |          |                   |                     |                       |                    |                    |                     |
|------------------|----------|-------------------|---------------------|-----------------------|--------------------|--------------------|---------------------|
| Pk               | Sucursal | Número de factura | Fecha de la factura | Forma de pago factura | Código del cliente | Nombre del cliente | Total de la factura |
| 01               | 500      | 1/01/06           | E                   | 01                    | ALVAREZ            | 48.20              |                     |
| 01               | 501      | 2/01/06           | CC                  | 107                   | CASTRO             | 16.00              |                     |
| 02               | 500      | 3/01/06           | E                   | 110                   | LIZ                | 14.90              |                     |



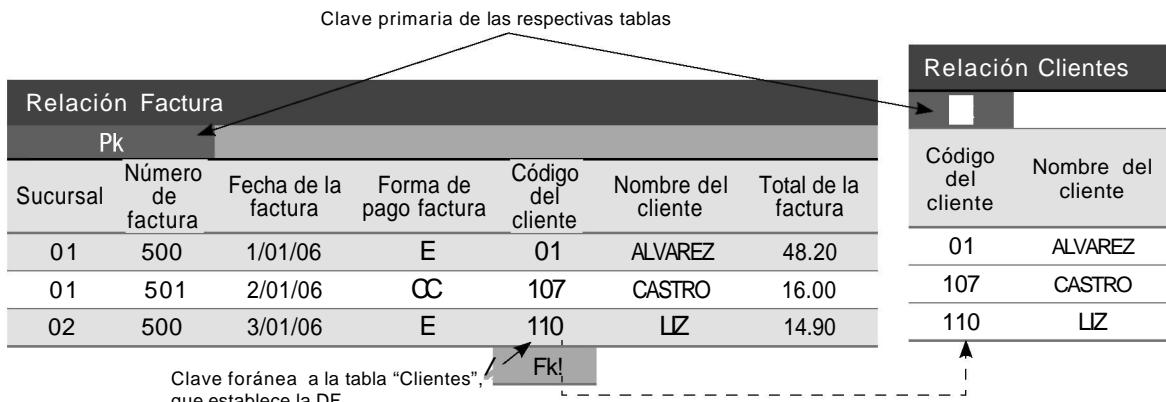
Ningún subconjunto tendrá dependencia transitiva; esto significa que el subconjunto estará compuesto por atributos que no pertenecen a la clave primaria.

Las columnas, resaltadas con el reborde, son dependientes entre sí y, luego, dependerán de la PK.

Esta situación se expone en la relación "Factura", de la relación anterior, en el subconjunto formado por los atributos "Código del cliente" y "Nombre del cliente". Sin duda, hay una dependencia funcional entre ambos atributos; si cambia el identificador del cliente, necesariamente cambiará su nombre. Esta dependencia funcional, que se acaba de detectar, es la que no permite que la relación alcance la Tercera Forma Normal.

¿Cómo se soluciona? Se aplica la misma técnica que se utilizó con anterioridad: la descomposición por proyección.

Se aparta el atributo que está ligado con una dependencia funcional al otro; en este caso, "Nombre del cliente" de la relación "Factura". Para ello, se creará una nueva relación denominada "Clientes" en la que alojará el "Nombre del cliente". Siempre se mantendrá la dependencia funcional, lo que obligará, también, a introducir, en esta nueva relación, el identificador del cliente, "Código del cliente", que quedará como lo muestra la siguiente relación:



Clave foránea a la tabla "Clientes", que establece la DF.

Al quitar de la relación “Factura” el atributo “Nombre del cliente”, se perdió la dependencia transitiva que impedía llegar a 3FN. Se creó una nueva relación “Clientes” en la que se alojan los datos del cliente y en la que se establece una clave primaria que mantiene una relación de asociación con la relación “Factura”.

El modelo en estudio transformado a 3FN, se representa en el formato (Mod. 3FN):

Mod. 3FN

| Factura |                       | Detalle de Factura |                              |
|---------|-----------------------|--------------------|------------------------------|
| Pk      | Sucursal              | Pk                 | Sucursal                     |
|         | Número de factura     |                    | Número de factura            |
|         | Fecha de la factura   |                    | Código del artículo          |
|         | Forma de pago factura |                    | Fk                           |
|         | Código del cliente    |                    | Cantidad del artículo        |
|         | Total de la factura   |                    | Precio unitario del artículo |
|         |                       |                    | Subtotal del artículo        |

| Artículo |                     | Clientes |                    |
|----------|---------------------|----------|--------------------|
| Pk       | Código del artículo | Pk       | Código del cliente |
|          | Nombre del artículo |          | Nombre del cliente |
|          | Precio unitario     |          |                    |

Para explicar las Formas Normales Boyce-Codd —Cuarta Forma Normal y Quinta Forma Normal—, no se puede utilizar el modelo que se venía desarrollando (modelo de facturación), pues éste no presenta las anomalías que son tratadas en las restantes reglas. En otras palabras, este modelo ya está normalizado.

#### 2.5.5.4 Forma Normal Boyce-Codd (FNBC)

- Una relación está en FNBC si, y solo si, todo determinante es una clave candidata. Determinante es un atributo, o conjunto de éstos, del que depende funcionalmente otro atributo completamente.
- Una relación está en FNBC si primero está en Tercera Forma Normal y si las únicas dependencias funcionales triviales se encuentran dadas entre la clave primaria y un atributo.

Lo que impide la FNBC es la dependencia de atributos no clave pero de condición primarios (atributos que son parte de una clave candidata) y un componente de la clave primaria. En otras palabras: no puede haber ninguna dependencia funcional en la que el determinante no sea una clave candidata.

Esta condición establece problemas de actualización y de mantenimiento de datos.

Para analizar esta forma normal, se hará sobre una relación con las siguientes características:

1. En la que exista más de una clave candidata.
2. Que estas claves candidatas tengan la condición de ser compuestas.
3. En la que un atributo componente de una clave candidata es determinante de otro atributo de tipo primario, que compone otra clave candidata.

Se denomina atributos primarios a los atributos que integran cualquier clave candidata de la relación.

Se analizará una relación que represente la correspondencia que existe entre los alumnos de una carrera y las materias en las que está inscripto, con las siguientes características:

Distintas posibilidades de conformar la clave primaria de la relación:



Los atributos primarios son los atributos que integran cualquier clave candidata de la relación.

| Relación Materias por alumno |   |                   |                   |
|------------------------------|---|-------------------|-------------------|
| Legajo del alumno            | Número de Documento de Identidad del alumno | Código de materia | Fecha inscripción |
| 01                           | 10000111                                    | 12                | 01/08/2009        |
| 01                           | 10000111                                    | 08                | 01/08/2009        |
| 02                           | 11222333                                    | 04                | 03/08/2009        |
| 02                           | 11222333                                    | 05                | 03/08/2009        |
| 11                           | 12444666                                    | 04                | 02/08/2009        |
| 11                           | 12444666                                    | 12                | 02/08/2009        |
| 14                           | 45789456                                    | 33                | 01/08/2009        |

Observe que esta entidad posee dos claves candidatas. Se demuestra, con ambos ejemplos, cómo puede quedar conformada la clave primaria.

| Materias por alumno                         |                   |                   |                   |
|---|-------------------|-------------------|-------------------|
| Número de Documento de Identidad del alumno | Legajo del alumno | Código de materia | Fecha inscripción |
| 10000111                                    | 01                | 12                | 01/08/2009        |
| 10000111                                    | 01                | 08                | 01/08/2009        |
| 11222333                                    | 02                | 04                | 03/08/2009        |
| 11222333                                    | 02                | 05                | 03/08/2009        |
| 12444666                                    | 11                | 04                | 02/08/2009        |
| 12444666                                    | 11                | 12                | 02/08/2009        |
| 45789456                                    | 14                | 33                | 01/08/2009        |

Observe que existen dos claves candidatas.

En los dos primeros atributos (“Legajo del alumno”, “Número De Documento De Identidad del alumno”) existe una dependencia funcional no trivial o no común entre ellos. Pueden alternar en su capacidad de pertenecer a la clave primaria; en otras palabras, ambas forman parte de claves candidatas, pero —entiéndase bien— que

“formar parte” no es lo mismo que serlo. Ambos atributos (“Número de Documento de Identidad del alumno” y “Legajo del alumno”) son determinantes, ya que cada uno de ellos ejerce una dependencia directa y no trivial sobre el otro; al cambiar uno de ellos, necesariamente cambia el otro.

Esta dependencia funcional (fuerte) entre dos atributos de una relación, con las siguientes características:

- Ambos son parte de una clave candidata.
- Ambos son determinantes entre sí.
- Cuando uno de ellos pertenece a la PK, el otro es un atributo no clave.

Estas son las condiciones que se deben eliminar para que la relación alcance FNBC, que no existan dos atributos en la relación con estas características.

Para lograr que esta relación alcance FNBC, se debe descomponer la relación en dos, tal como lo muestra la relación “Alumnos”:

| Relación Alumnos  |   | Materias_por_alumno |                   |                   |
|-------------------|---|---------------------|-------------------|-------------------|
| PK                |   | PK                  |                   |                   |
| Legajo del alumno | Número de Documento de Identidad del alumno | Legajo del alumno   | Código de materia | Fecha inscripción |
| 01                | 10000111                                    | 01                  | 12                | 01/08/2009        |
| 02                | 11222333                                    | 01                  | 08                | 01/08/2009        |
| 11                | 12444666                                    | 02                  | 04                | 03/08/2009        |
| 14                | 45789456                                    | 02                  | 05                | 03/08/2009        |
| ▲                 |   | 11                  | 04                | 02/08/2009        |
| ↓                 |   | 11                  | 12                | 02/08/2009        |
| ↓                 |   | 14                  | 33                | 01/08/2009        |
| ↓                 |   | FK                  |                   |                   |

Ahora, la relación “Materias\_por\_alumno”, de la relación “Alumnos”, no tiene más la dependencia funcional trivial que ya se explicó y, por lo tanto, ahora sí está en FNBC.

#### 2.5.5.5 Interpretación de la Cuarta Forma Normal (4FN)

Una relación “R” está en Cuarta Forma Normal si primero está en Forma Normal Boyce-Codd y, además, todas las dependencias multivaluadas en esta relación R son dependencias funcionales.

El concepto de dependencia multivaluada —pero que no es dependencia funcional— es el que impide que una relación se encuentre en Cuarta Forma Normal.

Esta situación se representa por una relación que contiene las siguientes situaciones:

1. La relación tiene que estar compuesta por lo menos por tres atributos; por ejemplo: “AlumnosPorCarreraPorProfesor” (relación “Alumnos”).
2. Se debe observar que para cada atributo de la relación hay una evidente repetición de valores, que hace pensar en redundancia de datos.

3. Dos de los atributos son independientes entre sí, como se manifiesta en la relación de estudio. Los atributos "Alumno" y "Profesor" no están relacionados entre sí.
4. Se debe poder observar una dependencia multivaluada entre el tercer atributo y los dos restantes de la terna. Evidentemente, existe una dependencia entre el "Alumno" y la "Carrera", también entre el "Profesor" y la "Carrera".
5. No existe dependencia funcional entre los valores de los atributos. No hay una relación uno a uno entre ninguno de los atributos.
6. Todos los atributos de la relación son parte de la clave primaria.
7. Que se encuentre en FNBC.
8. Lo expuesto en el punto 2 tiene, como consecuencia, serias anomalías de actualización.

Se observará la siguiente relación no normalizada para poder entender la situación a la que se hace referencia.

Esta relación representa el vínculo existente entre carreras y profesores que dictan clases en la misma universidad y los alumnos inscriptos en ésta (ver relación "AlumnosPorCarreraPorProfesor").

| Relación AlumnosPorCarreraPorProfesor |                 |          |
|---------------------------------------|-----------------|----------|
| Carrera                               | Alumno          | Profesor |
| Lic. en Informática                   | Sánchez José    | Dardo    |
| Lic. en Informática                   | Sánchez José    | Miño     |
| Lic. en Informática                   | Sánchez José    | Abru     |
| Lic. en Informática                   | Salas Miguel    | Dardo    |
| Lic. en Informática                   | Salas Miguel    | Miño     |
| Lic. en Informática                   | Salas Miguel    | Abru     |
| Lic. en Informática                   | Fernández Erika | Dardo    |
| Lic. en Informática                   | Fernández Erika | Miño     |
| Lic. en Informática                   | Fernández Erika | Abru     |
| Ing. en Sistemas                      | Sánchez José    | Quinto   |
| Ing. en Sistemas                      | Salas Miguel    | Quinto   |
| Ing. en Sistemas                      | Fernández Erika | Quinto   |

Es notorio que la relación "AlumnosPorCarreraPorProfesor" tiene gran cantidad de redundancia. Si se quisiera agregar un nuevo alumno en la carrera Licenciado en Informática, se tendrían que agregar tres nuevos registros a la relación, uno para cada profesor.

En forma intuitiva, se puede realizar una división de la relación en dos, que presenta la información de una forma más conveniente (relación "AlumnosPorCarrera").

| Relación AlumnosPorCarrera |                 | ProfesorPorCarrera  |          |
|----------------------------|-----------------|---------------------|----------|
| PK                         |                 | PK                  |          |
| Carrera                    | Alumno          | Carrera             | Profesor |
| Lic. en Informática        | Sánchez José    | Lic. en Informática | Dardo    |
| Lic. en Informática        | Salas Miguel    | Lic. en Informática | Miño     |
| Lic. en Informática        | Fernández Erika | Lic. en Informática | Abru     |
| Ing. en Sistemas           | Sánchez José    | Ing. en Sistemas    | Quinto   |
| Ing. en Sistemas           | Salas Miguel    |                     |          |
| Ing. en Sistemas           | Fernández Erika |                     |          |

En estas dos relaciones, proyectadas en la relación anterior, todos sus atributos son parte de la clave primaria, de la misma manera que en la relación de origen, y también están en FNBC.

Como se verá, no se puede establecer entre ambas relaciones una relación sobre la base de dependencias funcionales, pues no existe. Pero sí, si se hace en función de este nuevo concepto, que se menciona en esta forma normal, denominado dependencia multivaluada.

Dentro de la relación “AlumnosPorCarreraPorProfesor”, existen dos dependencias multivaluadas: una establecida por la relación “carreraprofesor” y, la otra, dada por “carreraalumno”.

Observamos que, en la primera dependencia multivaluada (“carreraprofesor”), no existe una única tupla que represente la dependencia funcional “carrera à profesor”, pero sí existe un grupo de ellas acotado y limitado que define con claridad un conjunto.

Si se extiende la expresión para una carrera determinada y un alumno en particular, se puede establecer con claridad el conjunto de profesores, para la relación “AlumnosPorCarreraPorProfesor”, que depende exclusivamente del valor de la carrera.

Ambas dependencias multivaluadas se explican con la misma concepción.

#### Dependencia multivaluada:

Dada un relación “R” con atributos “X, Y, Z” (que pueden ser compuestos), la dependencia multivaluada

$$R.X \xrightarrow{\Delta} R.Y$$

sucede en “R” si y solo si un conjunto de valores de Y pertinente a un par dado de valores de X y Z, en “R”, depende solo del valor de X y es independiente del valor de Z.

Dada la relación “R(X,Y,Z)”, la dependencia multivaluada

$R.X \xrightarrow{\Delta} R.Y$  se cumple si, y solo si,  $R.X \xrightarrow{\Delta} R.Z$  también se cumple.

Otra forma de notación

$$R.X \xrightarrow{\Delta} R.Y | R.Z$$



No se puede establecer entre ambas relaciones una relación sobre la base de dependencias funcionales porque no existe. Pero sí, si se hace en función de este nuevo concepto denominado dependencia multivaluada.

Una DF es un DMV en el que los valores dependientes del determinante siempre son un único valor. El caso inverso no es válido.

#### 2.5.5.6 Interpretación de la Quinta Forma Normal (5FN)

Una relación "R" está en Quinta Forma Normal si, y solo si, toda dependencia de reunión en R es una consecuencia de las claves candidatas en R.

Hasta este momento, para lograr una forma normal determinada, en cualquiera de las anteriores hasta 4FN, se partía de una relación que no estaba normalizada (no alcanzaba la forma normal deseada) y se la sustituía por dos relaciones que resultaban de la descomposición o de las proyecciones de la primitiva. Estas relaciones resultantes son las que logran la forma normal buscada.

Esta mecánica conduce con éxito desde la 1FN hasta la 4FN. También, este método garantizaba que no se perdiera información para reconstruir la relación primera sin problemas.

Existen casos particulares, en donde las formas normales expresadas anteriormente implican los conceptos de DF y DMV (de 1FN a 4FN); éstos no alcanzan a satisfacer la premisa de no pérdida de información en la transformación, para la Quinta Forma Normal.

A raíz de esto, entra en juego un nuevo concepto denominado Dependencia de Reunión, que implica la descomposición de una relación en n-proyecciones ( $n \geq 3$ ), para lograr la obtención de la forma normal deseada. Este caso se presenta, únicamente, para la Quinta Forma Normal.

Esta situación se presenta bajo las siguientes características:

1. La cantidad de atributos de la entidad es tres o superior a tres.
2. Todos los atributos de la relación componen la clave primaria.
3. No debe haber dependencia funcional entre ninguno de sus atributos y tampoco dependencias multivaluadas no triviales.
4. La entidad se encuentra en 4FN.

#### Defnición de Dependencia de Reunión (DR):

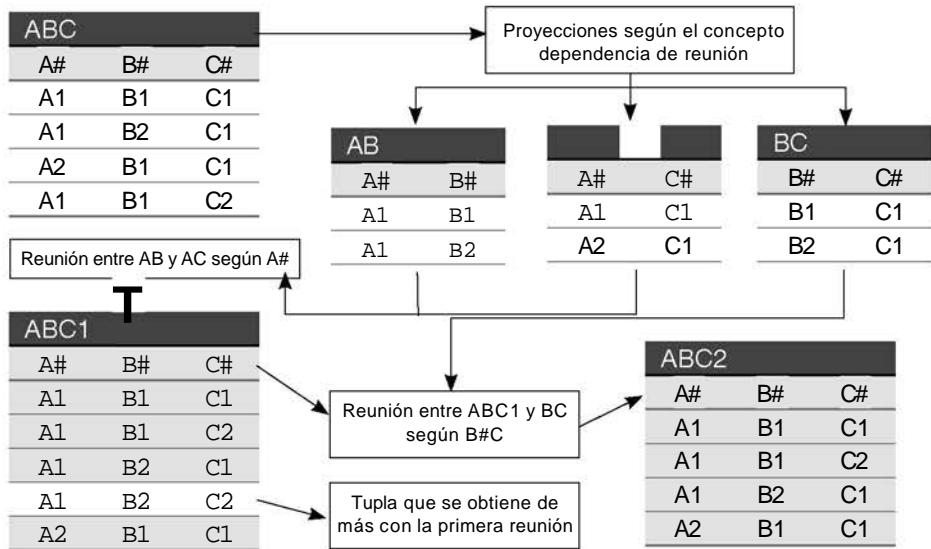
Una relación "R" satisface la dependencia de reunión

$*(A, B, \dots, Z)$

si, y solo si, R se puede construir con la reunión de sus proyecciones A, B, ..., Z siendo A, B, ..., Z subconjuntos de atributos de R.

#### Caso teórico:

Sea una relación cuyos atributos son A, B y C, que se denominará, de ahora en más, ABC, si se descompone con la concepción de DR (dependencia de reunión), se tienen las proyecciones AB, AC y BC en las relaciones (AB, AC y BC).



Si ahora se operara la reunión de estas relaciones, se encontraría el siguiente resultado (relación "ABC1") se opera la reunión según A# entre AB y AC. Por esta acción, se obtiene la relación "ABC1" con una tupla adicional, no encontrada en la relación primitiva.

Para completar el paso de reunión, se realiza la última operación entre ABC1 y BC según B# C#, que da como resultado la eliminación de la tupla que se obtuvo de más en la anterior reunión. Queda una relación "ABC2" (relación "ABC2") de igual contenido que la original ABC.

## 2.6 Las estructuras

### 2.6.1 El modelo y sus estructuras

La información que se observa dentro de la mecánica funcional de un problema que se solucionará, se debe representar por una combinación de estructuras que satisfagan como solución al problema. Esta información que está sustentada por un modelo diseñado con la función de solucionar la problemática observada, conforma la estructura de la base de datos. Esta base de datos, con sus estructuras internas, será pensada para que cumpla con las formas normales. Esto último asegura que los datos mantendrán su estado relacional, su dependencia y que no habrá pérdida de información respecto de lo observado en el problema.

El modelo, en sí mismo, tiene como soporte una combinación de estructuras que representan la realidad. Estas estructuras, que pueden ser utilizadas en una combinación vasta y diversa, terminan representando la información de una forma sencilla, cuya comprensión no debiera causar problemas para su interpretación y uno de sus principales objetivos es la fácil manipulación de los datos, evitando problemas de lógica.

Estas estructuras, claramente limitadas por el funcionamiento del modelo relacional, están acotadas a los formatos que detallaremos a continuación.

Es comprensible que las estructuras expuestas a consideración puedan sufrir variaciones acordes con las distintas problemáticas. Lo que se pretende es establecer el formato tipo de estas estructuras para poder enmarcar estos modelos en distintos tipos diferenciados.

El modelo, en sí mismo, tiene como soporte una combinación de estructuras que representan la realidad. Estas estructuras terminan representando la información de una forma sencilla para lograr la fácil manipulación de los datos, evitando problemas de lógica.

Importante: Las estructuras que a continuación se ponen a consideración, están compuestas por relaciones que están ligadas por una fuerte dependencia relacional, en cada caso, hay un conjunto de relaciones que conforman la estructura tratada, y otras, que forman parte del diseño, y están con el propósito de poder observar el contexto para no perder el sentido.

Las relaciones que conforman la estructura tratada están ligadas entre sí con una línea continua, y las que forman parte del entorno (no conforman la estructura tratada) están relacionadas con una línea intermitente.

Estos modelos están representados en formato de relaciones con contenido de información para poder comprender con mayor facilidad sus características.

Relaciones de referencia:

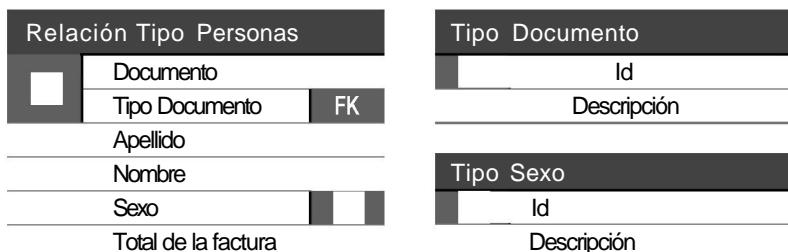
- No posee relaciones asociadas que heredan su clave primaria como componente de la clave primaria de aquellas.
- Sí poseen estado relacional con otras relaciones que establecen referencia con ésta, mediante claves foráneas.
- Ejemplos:
  - Relaciones de tipo: tipo de Documento, tipo de estado civil, tipo de vehículo, etc.
  - Relaciones descriptivas: con mayor cantidad de atributos que las de tipos.



Las relaciones “Tipo Documento” y “Tipo De Sexo”, son en este caso relaciones de referencia, pues se accede a ellas desde la relación “Personas” a través de una clave foránea. La relación “Personas” necesita de la presencia de estas relaciones para completar la representación que el modelo sugiere de la realidad.

Desde otro punto de vista la relación “Personas” también podría ser una relación de referencia hacia otras relaciones.

A continuación, la representación del modelo de la relación “TipoDocumento”:



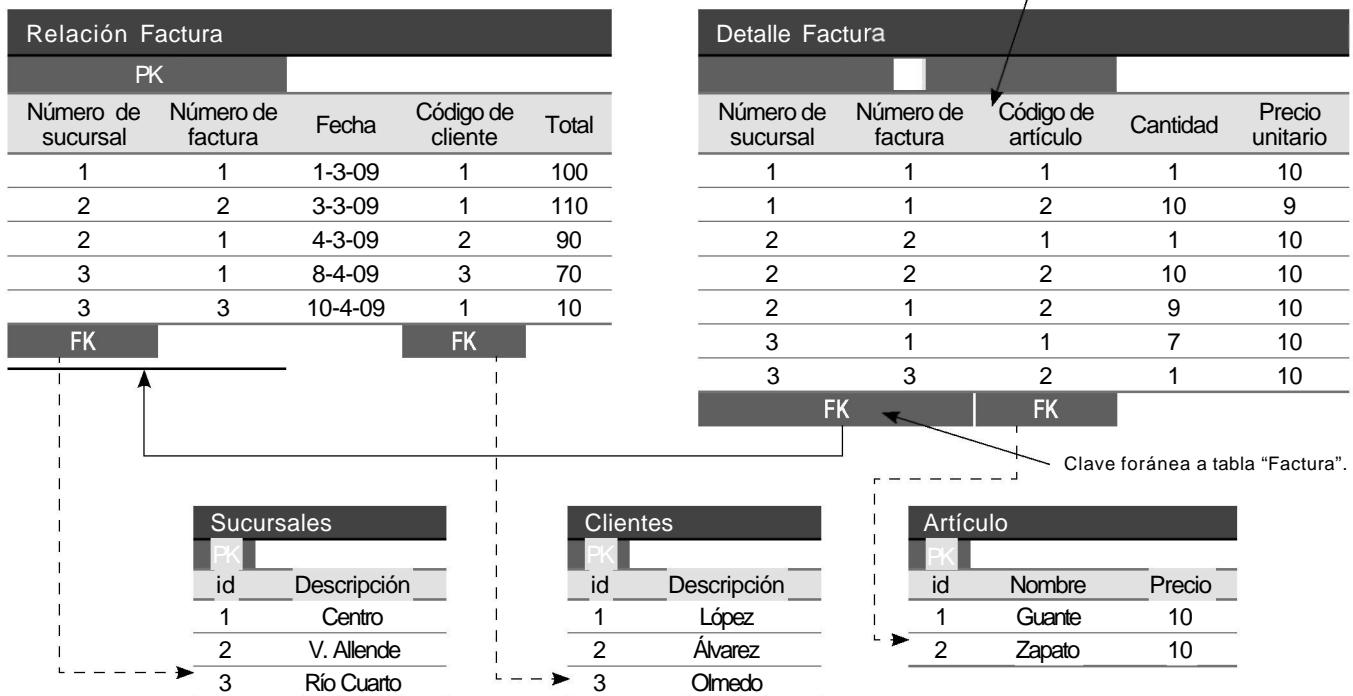
### Relaciones que conforman estructura Padre-Hijo:

Relación denominada “Padre” que posee una relación asociada denominada “Hijo”, que hereda la clave primaria del padre como componente parcial de su propia clave primaria.

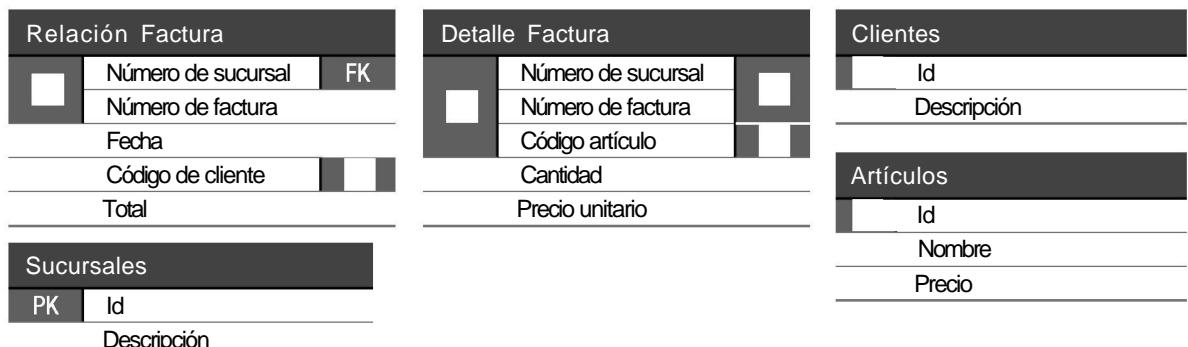
Ejemplo:

- Estructura Factura – Detalle de Factura.
- Personas – Teléfonos varios.
- Personas – Domicilios varios.

Clave primaria conformada con la herencia de la clave primaria de la tabla “Factura” más un componente (código del artículo) típico de la tabla.



A continuación la representación del modelo, correspondiente a la relación “Factura”:

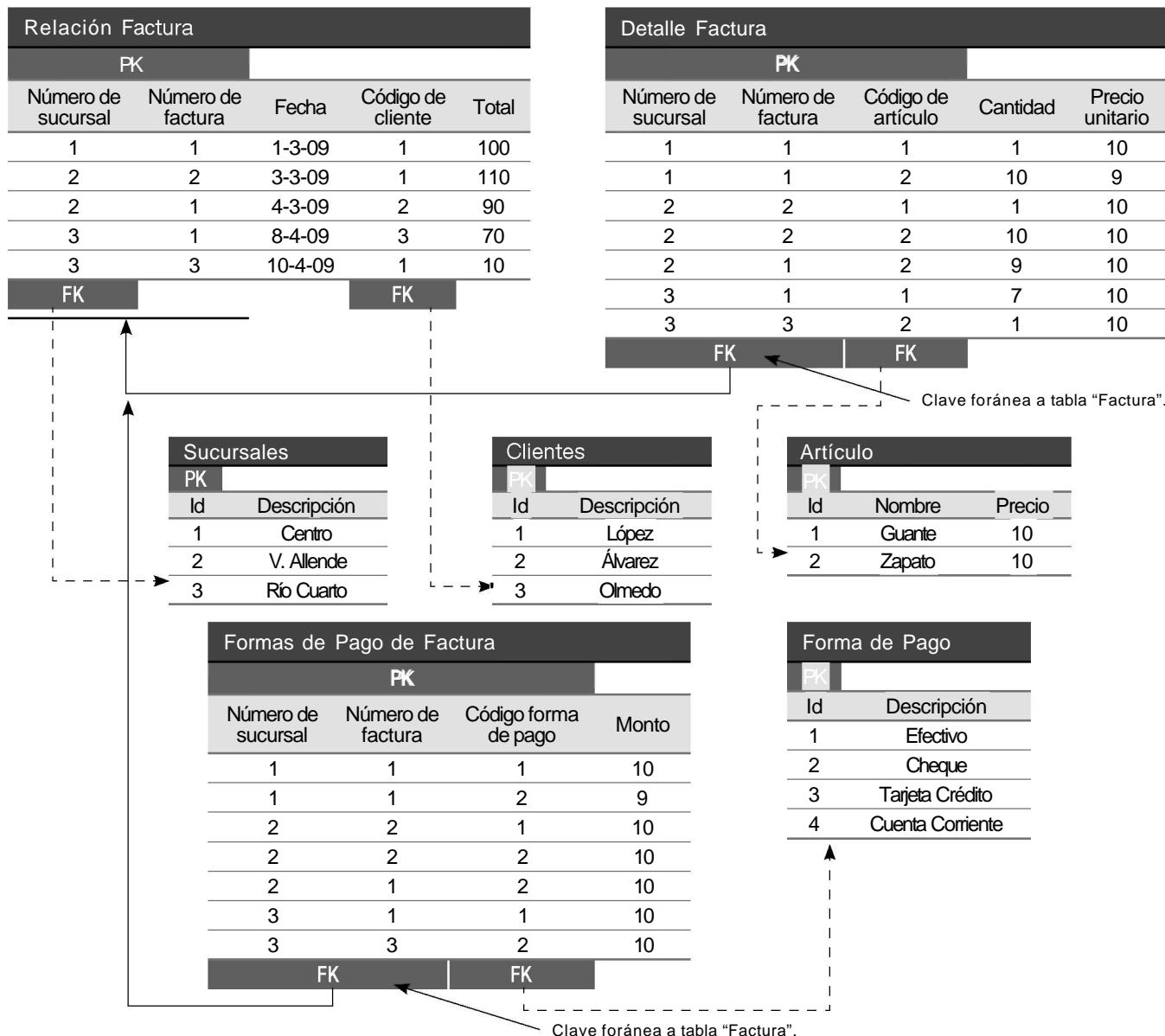


### Relaciones que conforman estructura Padre-Múltiples-Hijo:

Relación denominada “Padre” que posee varias relaciones asociadas denominadas “Hijos”, que heredan la clave primaria del padre como componente parcial de su propia clave primaria.

Ejemplo:

Estructura Factura – Detalle de Factura – Formas de Pago de Factura



A continuación, la representación del modelo correspondiente a la relación “Factura”:

| Relación Factura         |                      | Detalle Factura |                    | Sucursales    |             |
|--------------------------|----------------------|-----------------|--------------------|---------------|-------------|
|                          | Número de sucursal   |                 | Número de sucursal |               | PK Id       |
|                          | Número de factura    |                 | Número de factura  |               | Descripción |
| PK                       | Fecha                | Código artículo |                    |               |             |
|                          | Código de cliente    |                 | Cantidad           |               |             |
|                          | Total                |                 | Precio unitario    |               |             |
| Forma de Pago de Factura |                      | Artículos       |                    | Forma de Pago |             |
|                          | Número de sucursal   |                 | PK Id              |               |             |
|                          | Número de factura    |                 | Nombre             |               |             |
|                          | Código forma de pago |                 | Precio             |               |             |
|                          |                      | Monto           |                    |               |             |
|                          |                      |                 | Descripción        |               |             |

Relación con autoreferencia:

- Son relaciones que poseen un estado de asociación en sí mismas.
- En estas relaciones se encuentra una, o más referencias, a la misma relación.
- Esta clave foránea se denomina autoreferencia o clave foránea recursiva.

Ejemplo:

- Relación “Empleados” en donde existe una asociación al Jefe de éstos, que es también un empleado.
- Relación “Personas” en donde existe una asociación al cónyuge que es otra persona de la misma relación.

| Relación Empleados  |                |          |         |      |                    |                         |
|---------------------|----------------|----------|---------|------|--------------------|-------------------------|
| PK                  |                |          |         |      |                    |                         |
| Número de Documento | Tipo Documento | Apellido | Nombres | Sexo | Documento del Jefe | Tipo Documento del Jefe |
| 1111                | 1              | López    | José    | 1    | null               | null                    |
| 2222                | 2              | Muñoz    | Miguel  | 1    | 1111               | 1                       |
| 3333                | 1              | Juárez   | Antonio | 1    | 2222               | 2                       |
| 4444                | 1              | Romero   | Oscar   | 1    | 2222               | 2                       |
| 5555                | 3              | Oliva    | Beatriz | 2    | 2222               | 2                       |
|                     |                | FK       |         | FK   |                    |                         |

Clave foránea a la misma tabla, también llamado “autoreferencia”.

**Tipo Documento**

| PK | Id | Descripción                    |
|----|----|--------------------------------|
|    | 1  | Documento Nacional Identidad   |
|    | 2  | Libreta Enrolamiento           |
|    | 3  | Libreta Cívica                 |
|    | 4  | Cédula de Identidad Provincial |
|    | 5  | Pasaporte                      |

**Tipo Sexo**

| Id | Descripción |
|----|-------------|
| 2  | Femenino    |
| 1  | Masculino   |

A continuación, la representación del modelo correspondiente a la relación “Empleados”:

| Relación Empleados      |    | Tipo Sexo   |  |
|-------------------------|----|-------------|--|
| Número de Documento     |    | Id          |  |
| Tipo de Documento       |    | Descripción |  |
| Apellido                |    |             |  |
| Nombres                 |    |             |  |
| Sexo                    | FK |             |  |
| Documento de Jefe       |    | Id          |  |
| Tipo Documento del Jefe | FK | Descripción |  |

Relaciones que conforman estructura de componentes:

- Se requieren dos relaciones. Una, la que contiene todos los elementos simples y compuestos (en este caso, se denomina “Elementos”). La otra es la que relaciona un elemento compuesto, con algunos elementos simples o compuestos de la relación “Elementos”. A esta última se la denomina “Componentes”.
- Un elemento simple no posee estructura de componentes.
- Un elemento simple puede ser referenciado por varios elementos compuestos.
- La clave primaria de la relación “Componentes” está construida con dos veces la clave primaria de la relación “Elementos”. La primera vez cita al elemento compuesto de la relación “Elementos”, la segunda hace referencia al elemento componente del primero.
- La relación “Componentes” podrá tener atributos adicionales, si fuera necesario, y referencia a otras relaciones (FK), si así se requiere para el modelo.

Ejemplo:

Definición de “Artículo” compuestos por varios elementos.

| Relación Artículo |             | Componente |                           |          |
|-------------------|-------------|------------|---------------------------|----------|
| Id                | Descripción | PK         | Código Artículo compuesto | Cantidad |
| 1                 | Ruedas      |            | 5                         | 5        |
| 2                 | Carrocería  |            | 5                         | 1        |
| 3                 | Frenos      |            | 5                         | 4        |
| 4                 | Luces       |            | 5                         | 4        |
| 5                 | Automóvil   |            | 6                         | 2        |
| 6                 | Motocicleta |            | 6                         | 2        |
|                   |             | FK         | FK                        |          |

Clave primaria compuesta por ambas columnas.

Referencia al elemento componente del producto (elemento) compuesto.

Referencia al producto (elemento) compuesto, formado por varios elementos simples o compuestos.

A continuación, la representación del modelo correspondiente a la relación “Artículo”:

| Relación Artículo |    | Componentes               |                            |
|-------------------|----|---------------------------|----------------------------|
| PK                | Id | Código artículo compuesto | Código artículo componente |
| Descripción       |    | Cantidad                  |                            |

Relaciones con múltiples asociaciones:

- Cuando dos relaciones se relacionan entre sí con múltiples asociaciones, es inevitable la participación de una tercera que realice la tarea de establecer las múltiples relaciones.
- Las relaciones que se relacionan entre sí tienen el mismo estatus en la estructura. Una tercera, que se puede denominar relación “Relación”, es la que establece la forma en que ambas se relacionan y el volumen de asociación entre las dos. Sin esta tercera relación no se podría tener la estructura deseada.
- Siempre la relación se produce a través de la PK de ambas relaciones.
- También esta relación puede poseer atributos típicos y necesarios.

Ejemplo:

Relaciones de empleados, de computadoras, de asociación de las computadoras que puede utilizar cada empleado.



A continuación, la representación del modelo correspondiente a la relación “Empleados”:

| Relación Empleados |                     | Tipo Sexo |             |
|--------------------|---------------------|-----------|-------------|
|                    | Número de Documento |           | Id          |
|                    | Tipo de Documento   | PK        | Descripción |
|                    | Apellido            |           |             |
|                    | Nombres             |           |             |
|                    | Sexo                | FK        |             |

| Computadoras x Empleados |                     | Tipo Documento |             |
|--------------------------|---------------------|----------------|-------------|
| PK                       | Número de Documento |                | Id          |
|                          | Tipo de Documento   | PK             | Descripción |
|                          | Id Computadora      | FK             |             |

| Computadoras |    |
|--------------|----|
|              | Id |

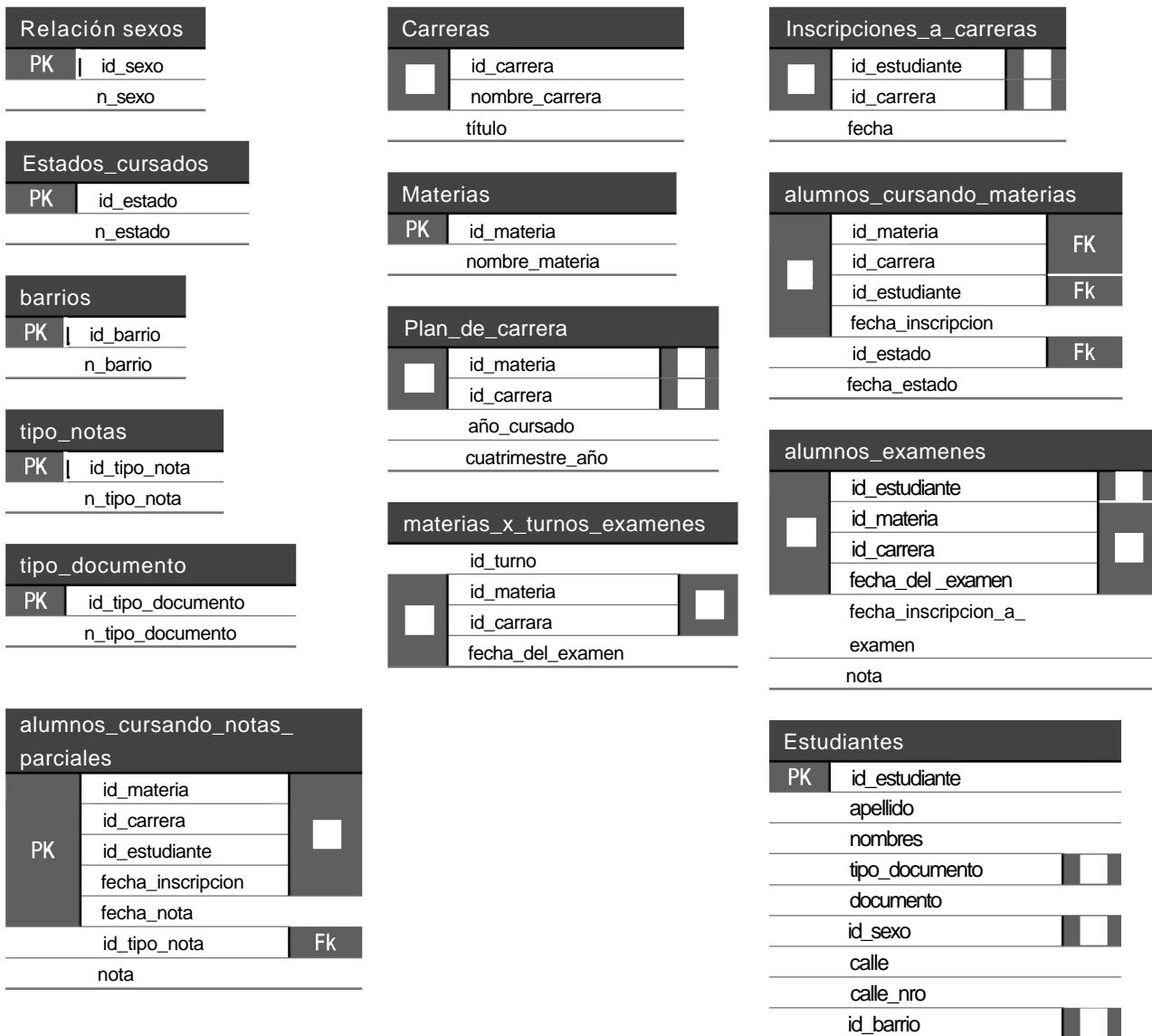
## 2.7 Un caso de estudio

Modelo de Datos para Estudio: “Administración Académica de Institución Educativa”

Descripción del modelo y sus características funcionales.

- El sistema almacena información del alumno: apellido, nombres, tipo de Documento, número de Documento, sexo, domicilio —calle , número de la casa y barrio—. Para esta entidad se utilizará una clave primaria artificial.
- Las carreras que se pueden cursar: nombre de la carrera y título.
- Las materias: nombre de la materia.
- Cada carrera tiene un plan de materias, que consta de la materia, el año de cursada y el cuatrimestre de cursada.
- Los alumnos se pueden inscribir en más de una carrera; se requiere saber la fecha en la que se realizó este trámite.
- Los alumnos se inscriben a cursar materias de la carrera en la que están inscriptos; se requiere la fecha de inscripción, un estado en el que se encuentra la materia (ej. inscripto, cursando, regular, libre, promocionado, etc.) y una fecha de ese estado. No se requiere información histórica del cambio de estados; solo se requiere el último.
- En el transcurso de la cursada de materia, el alumno debe rendir exámenes parciales; se requiere documentar esta información con el tipo de parcial que rindió (tipo\_nota) la fecha en que lo hizo y la nota que obtuvo.
- Para que los alumnos puedan rendir exámenes, se habilitan materias en turnos determinados, que se identifican por un número y que se incrementan en forma secuencial a partir de uno. Se requiere saber la materia que se habilita para examen, carrera a la que pertenece y fecha en la que se tomará el examen.
- Luego de cursar las materias y quedar en condición de examen, los alumnos se pueden inscribir para rendir se requiere saber la materia en la que se inscribió a examen, fecha del examen, fecha en la que se inscribió al examen y la nota que obtuvo.

Solución propuesta:



## 2.8 Resumen

El modelo de datos relacional es el modelo con mayor difusión y uso en los distintos tipos de organizaciones. Si bien este modelo sufrió grandes cambios y modificaciones con el correr de los años, expresa conceptos que no han perdido vigencia y que constituyen un pilar de conocimientos.

Además, en sus inicios, este modelo se respaldaba en la teoría de conjuntos cuyo aporte brindaba la resolución del inconveniente con las grandes bases de datos compartidas.

Posteriormente, se trata el álgebra relacional cuyo objetivo principal de enseñanza es facilitar el aprendizaje de la escritura de sentencias SQL, dado por la manipulación de datos vinculada a la relación con operaciones. El resultado de aplicar una operación de este tipo es otra relación, lo cual habilita la combinación de operaciones y su anidamiento ya que la entrada de una operación podría ser el resultado creado por otra anterior. Se incluyen las operaciones de conjuntos, que son binarias y abarcan dos relaciones de entrada, y las operaciones relacionales, que tienen más potencial aun cuando se combinan con operaciones de conjunto o entre ellas mismas.

También se brindan los fundamentos de la normalización referidos a modelar una base de datos relacional y saber cómo organizar una estructura relacional de datos válida. Además, se profundiza sobre el origen de los datos, como un recurso que permite obtener datos para modelar una estructura de relaciones normalizadas, las formas normales y las estructuras, limitadas por el funcionamiento del modelo relacional.

Para finalizar se presenta un caso de estudio que se utilizará en otros capítulos de la obra.

## 2.9 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) solo está disponible para docentes.

### 2.9.1 Mapa conceptual del capítulo

### 2.9.2 Autoevaluación

### 2.9.3 Presentaciones\*



# 3

## Contenido

|   |     |
|---|-----|
| 3.1 Introducción.....   | 104 |
| 3.2 Algo de historia del lenguaje SQL.....                                | 104 |
| 3.3 Características del lenguaje SQL.....                                 | 105 |
| 3.4 El lenguaje SQL y su sublenguaje de definición de datos<br>o DDL..... | 106 |
| 3.5 Sentencias del sublenguaje DML.....                                   | 112 |
| 3.6 Sentencias del sublenguaje DML. INSERT, UPDATE, DELETE ...            | 123 |
| 3.7 Sentencias del sublenguaje TCL de control<br>de transacciones.....    | 125 |
| 3.8 Procesamiento de consultas.....                                       | 127 |
| 3.9 Resumen.....  | 129 |
| 3.10 Contenido de la página Web de apoyo.....                             | 129 |

## Objetivos

---

- Conocer el origen del lenguaje SQL.
- Entender la estructura de las sentencias del lenguaje SQL.
- Enunciar el propósito de cada sentencia SQL.
- Conocer las funciones disponibles para ser usadas en las sentencias SQL.
- Entender el procesamiento de las sentencia y de las transacciones.
- Obtener consejos de optimización de consultas SQL.
- Ver ejemplos prácticos de sentencias SQL.

### 3.1 Introducción

En este capítulo se estudiará el Lenguaje Estructurado de Consultas (Structured Query Language o SQL), cuyas sentencias se agrupan en sublenguajes. Para que el estudiante aprenda su sintaxis —que se explica en numerosos sitios de la Web— se realizarán algunas sugerencias de estilo para escribir las sentencias SELECT —también denominadas “consultas”— y otras indicaciones útiles para aprovechar todo su poder; de esta manera, se facilitará la obtención de información que, normalmente, se almacena en una o varias tablas de una base de datos y, además, para cubrir las tareas de una aplicación comercial, científica, de un sitio Web, en la que se insertarán, borrarán o modificarán los datos ya almacenados en los motores relacionales que soportan la mayoría de las aplicaciones.

Es importante destacar el rol de SQL como lenguaje especializado en la comunicación con la base de datos, que funciona dentro o embebido en otros lenguajes de uso general que construyen las demás funcionalidades de una aplicación de negocios, como los numerosos lenguajes procedimentales conocidos como COBOL, C, C++, orientados a objetos como Java, PHP y Python. A estos lenguajes se los podría caracterizar como de propósitos múltiples, ya que con ellos se construirán pantallas, menús, botones, campos listas, mientras que para acceder a la base de datos para obtener las flas haciendo consultas o inserciones, borrados y actualizaciones, se requerirán las correspondientes sentencias SQL. También hay que mencionar que las bases de datos proveen otros lenguajes —procedimentales o basados en procedimientos que se ejecutan en el servidor de base de datos, como PL/SQL de Oracle, SQL PL de DB2, T-SQL de MS SQL Server— que se tratarán en otro capítulo.

### 3.2 Algo de historia del lenguaje SQL



Raymond Boyce (1947-1974). Científico informático, conocido por desarrollar la forma normal de Boyce-Codd (o FNBC) utilizada en la normalización de bases de datos.

En su libro *El modelo Relacional para la administración de Bases de Datos* en el capítulo de “Introducción al Modelo Relacional”, E.J. Codd da pistas sobre el origen del lenguaje y lo atribuye a un grupo de investigación de IBM a finales del año 1972. En esta publicación, el padre del modelo relacional afirma que es inconsistente en diversas formas con las máquinas abstractas del modelo relacional V1 (versión 1) y V2 (versión 2) y que algunas características no están implementadas directamente y otras no están implementadas correctamente y que, cada una de las inconsistencias, pueden reducir su utilidad. En esta publicación, Codd afirma que el lenguaje QUEL, por Query Language o Lenguaje de Consulta —inventado por Held y Stonebraker de la Universidad de California, Berkeley— era el más completo pero, pese a esto, el SQL —desarrollado en IBM por Andrew Richardson, Donald C. Messerly y Raymond F. Boyce— ganó preeminencia porque se adoptó como estándar ANSI y, a partir de ello, lo soportan todos los motores de base de datos.

En 1970, luego de que el Dr. E. F. Codd introdujera el concepto del modelo relacional, IBM desarrolló este lenguaje. Se buscó que fuera fácil de aprender y potente en sus posibilidades expresivas. Se basa en la estructura idiomática del inglés para permitir, en teoría, que un usuario no experto escriba sentencias SQL fluida y eficazmente. Aunque esta ventaja se consiguió solo parcialmente, el SQL es fácil de aprender y, con la ayuda de los programas con asistentes, el técnico no especializado puede lograr resultados muy interesantes.

En 1979, casi diez años después de la presentación del modelo relacional de Codd, aparece la primera base de datos comercialmente disponible con SQL como único lenguaje para acceder a los datos; luego, las otras bases de datos empezaron a incorporar esta característica. En 1986, se convirtió en un estándar de la American National Standard Institute (ANSI). En la actualidad, existen siete versiones (SQL86, SQL89, SQL92 o SQL2, SQL99 o SQL 3, SQL2003, SQL2006, SQL2008) a las que les agregaron funcionalidades que adoptaron todos los servidores que quisieron cumplir con el último estándar de la industria.

Los avances en motores de bases de datos, en paralelo con los del lenguaje, son múltiples: el descubrimiento de Internet como una forma de conectarse a los motores de bases de datos fue uno de los transformadores de la WWW, que de un rol presentadora de documentos estáticos o folletería electrónica, pasó a soportar aplicaciones corporativas que fueron la base para la explosión del e-commerce como soporte de una nueva economía y se transformó en un ejemplo concreto de la formación de la sociedad del conocimiento. En este movimiento, hay que mencionar a MySQL como un protagonista de mucho menor tamaño que los motores tradicionales, pero con las prestaciones suficientes para publicar un sitio Web y hacer funcionar los sitios de todo tamaño que aparecieron en esos tiempos. La estrategia de esta empresa fue focalizarse en ofrecer lo que las grandes empresas no hicieron, una implementación en pequeño de las funcionalidades básicas de SQL.



Para conocer un poco más acerca de la historia del lenguaje SQL visite:  
<http://qoo.gl/pmUvh>

### 3.3 Características del lenguaje SQL

El lenguaje SQL se considera, por un lado, un lenguaje diseñado específicamente para la comunicación entre usuarios y, por otro con la base de datos para realizar todas las tareas requeridas para resolver los requerimientos como obtener información almacenada, realizar cálculos, modificar lo existente y agregar nuevas filas que contengan información de clientes, productos, transacciones (como ventas, compras, pedidos, reservas, asistencia, ausencias, llegadas tardes, etc.), puesto que la mayoría de las aplicaciones actuales usan bases de datos relacionales y, por ende, SQL.

La característica más destacada del lenguaje SQL es que es no procedimental, es decir, no se indica en sus sentencias cómo realizar la tarea sino que se limita a describir el resultado buscado y queda todo el trabajo de resolver lo solicitado al servidor de bases de datos que, de acuerdo con su optimizador y con los metadatos (se los denomina así porque son datos acerca de los datos) del diccionario, cumplirá con la sentencia SQL provista.

Por lo antedicho, SQL no tiene sentencias de control de flujo desde su origen aunque, recientemente, se han incluido como partes opcionales aceptadas del estándar de SQL, ISO/IEC 9075-5: 1996.

Es común que estas sentencias de control de flujo —conocidas como módulos persistentes almacenados— se consideren como extensiones procedimentales del lenguaje. Los proveedores de bases de datos tienen estas extensiones (por ejemplo: PL/SQL de Oracle, T-SQL de SQL Server de Microsoft, SQL-PL de DB2 de IBM y PSQL de PostgreSQL). En general, estas extensiones complementan a SQL para construir procedimientos almacenados, paquetes y disparadores, que se explicarán —como ya se anticipó— en el siguiente capítulo.



La característica más destacada del lenguaje SQL es que es no procedimental, es decir, no se indica en sus sentencias cómo realizar la tarea sino que se limita a describir el resultado buscado.

### 3.4 El lenguaje SQL y su sublenguaje de definición de datos o DDL

Las sentencias SQL se agrupan en sublenguajes de acuerdo con sus propósitos: primero se revisarán las sentencias del lenguaje de definición de datos o DDL (Data Definition Language), que permitirán crear, borrar o modificar objetos dentro de la base de datos. Por ejemplo, se puede utilizar CREATE, DROP, ALTER y RENAME para crear, eliminar, modificar su estructura y cambiarle el nombre a los objetos de datos de una base de datos como en el caso de las tablas, las vistas comunes, las vistas materializadas, los índices, las funciones, los procedimientos y los paquetes que pertenecen a un esquema lógico. También se aplican estas acciones sobre los usuarios, los roles, las estructuras de almacenamiento como una base de datos propiamente dicha, un espacio de tablas (Tablespace), los segmentos de RollBack (o RollBack Segments) y numerosas estructuras propias de un motor de base de datos.

## S

TRUNCATE es una sentencia que permite borrar todo el contenido de una tabla, sin eliminar la estructura y sin generar transacciones.

TRUNCATE es una sentencia que permite borrar todo el contenido de una tabla, sin eliminar la estructura y sin generar transacciones. Por esta razón, se la nombró aparte.

En DCL se encuentran, también, las sentencias GRANT y REVOKE, que permiten que se otorguen privilegios sobre objetos y sobre acciones o de sistema, a los usuarios y a los roles de la base de datos.

Además, COMMENT es una sentencia de DDL que permite guardar comentarios sobre tablas y columnas.

Las sentencias AUDIT y NOAUDIT —presente en algunos motores— que activa o desactiva el registro de todos los cambios hechos sobre los objetos por sesiones, también denominada auditoría automática.

A continuación, se verán algunos ejemplos en los que se utilizarán las tablas normalizadas en los capítulos anteriores.

Para el modelo de datos del sistema de ventas, se defnieron cuatro tablas: "Artículos", "Clientes", "Facturas" y "Detalle de Factura". Se analizarán las sentencias para su creación. A la tabla "Clientes", se le agregará una columna "olvidada" en el momento de creación o agregada por nuevos requerimientos para exemplifcar la sentencia ALTER. Además, se creará una vista que mostrará las facturas del cliente "A01". Una vez creadas las tablas y la vista, se creará, además, un usuario para luego darle privilegios de lectura al usuario sobre las tablas. Por último, se agregarán los comentarios sobre las tablas y una columna.

Nótese que las sentencias SQL empiezan siempre con un verbo que indica la acción; luego, el tipo de objeto sigue solo en la creación y en la modifcación y, después, el nombre del objeto que se tratará. Se continuará —si es necesario— con otra información que completará la tarea. Si bien SQL no es sensible a las mayúsculas y a las minúsculas, es bueno que se adopte un protocolo de escritura que facilite la lectura del código. En este libro, se adoptará el uso de las mayúsculas para las palabras reservadas y de las minúsculas para los nombres de los objetos.

En SQL, los nombres de los objetos tienen una longitud máxima (depende de cada motor), son sensibles a las mayúsculas y a las minúsculas y no soportan espacios intermedios en los nombres; normalmente, se utilizan el carácter guión bajo o ‘\_’ para unir las palabras y, de esta manera, evitar los espacios. En Oracle, los nombres,

que pueden tener hasta 30 caracteres, deben empezar con una letra y no con número o signo. Los signos permitidos son: '\$', '\_' y '#'.

Los nombres de los objetos se identificarán con su esquema y con su nombre. Para ello, se utilizará la notación "esquema.nombre". Esta regla se puede resolver, también, con el uso de sinónimos públicos que referenciarán directamente a los objetos. Originalmente, en el modelo de datos se incluyeron todos los acentos de las palabras, pero en la creación no se los utilizó para facilitar el trabajo de escritura de las sentencias SQL.

Las sentencias SQL se pueden extender por varias líneas y, por ejemplo en Oracle, finalizan con un punto y coma " ; " y tienen como separador interno o semántico el espacio ' ' o la coma ','.

Ejemplos de creación de objetos del modelo de datos de ventas:

```
CREATE TABLE articulo  (codigo_del_articulo VARCHAR2(8),
nombre_de_articulo VARCHAR2(30) NOT NULL,
precio_unitario NUMBER(8,3));

CREATE TABLE clientes  (codigo_del_cliente VARCHAR2(8),
nombre _del _cliente VARCHAR2(30) NOT NULL);

CREATE TABLE facturas  (sucursal           NUMBER(2),
número_de_factura    NUMBER(2),
fecha_de_factura     TIMESTAMP,
forma_de_pago_factura VARCHAR2(3),
codigo_del_cliente   VARCHAR2(8),
total_de_la_factura  NUMBER(14,2));

CREATE TABLE detalle_de_factura (sucursal           NUMBER(2),
numero_de_factura    NUMBER(2),
codigo_del_articulo  VARCHAR2(8),
cantidad_del_artículo NUMBER(8,3) NOT NULL,
precio_unitario_del_articulo NUMBER(8,3),
subtotal_del_articulo NUMBER(9,3));
```

En las columnas en las que se prohíbe, por regla de negocios, que tenga valores *null*, se ubican las restricciones de la columna con las palabras NOT NULL.

A modo de ejemplo, se usan los tipos de datos de Oracle, donde NUMBER es un tipo de dato que soporta hasta 38 dígitos, en los que se incluyen los decimales y sus signos. Para los caracteres alfanuméricos de longitud variable, se utiliza VARCHAR2, que soporta hasta 4096 caracteres. Para la fecha, se emplea *Timestamp* —que es el tipo

de datos que se ha definido como estándar— que soporta seis posiciones para fracciones de segundo y define la zona horaria, lo que aumenta las prestaciones para el registro de tiempos.

#### Ejemplos de modificación de objetos de la base:

Una vez creada la tabla “Clientes”, resultó necesario separar los nombres y los apellidos con la sentencia ALTER TABLE. Esto es, agregar una columna.

```
ALTER TABLE cliente ADD apellido_del_cliente VARCHAR2(30));
```

Otra alternativa —si aún no se tiene flas ni restricciones referenciales sobre la tabla que se modificará— sería eliminarla y volverla a crear con todas las columnas o restricciones, según los últimos requerimientos.

```
DROP TABLE clientes;
CREATE TABLE clientes  (codigo_del_cliente VARCHAR2(8),
nombre _del _cliente VARCHAR2(30),
apellido _del _cliente VARCHAR2(30));
```

Se notará que, en el momento de la creación de las tablas, no se incluyó nada acerca de las claves primarias y las foráneas; estas restricciones (*constraints*) se agregarán con ALTER.

La sentencia ALTER TABLE tiene, como modificadores, las acciones ADD, MODIFY y DROP, que pueden agregar, modificar y eliminar columnas. En algunas implementaciones de lenguajes se permite el uso, dentro de ALTER, de RENAME, que renombra las columnas de una tabla, pero su explicación detallada excede el propósito del libro.

Se agregarán, con ALTER, las restricciones correspondientes a las claves primarias y a las foráneas de las tablas anteriores; luego, se ilustrará, con un ejemplo, cómo se puede hacer, siempre bajo el supuesto de que no tienen flas y que las tablas referenciadas en la clave foránea ya existen.

```
ALTER TABLE clientes
```

```
ADD CONSTRAINT cliente_pk PRIMARY KEY (codigo_del_cliente));
```

Después de crear la tabla “Artículos” se define su clave primaria.

```
ALTER TABLE articulo
```

```
ADD CONSTRAINT articulos_pk PRIMARY KEY (codigo_del_articulo));
```

Para crear la clave primaria en el momento de crear la tabla, la sintaxis tendría estas dos variantes:

- Variante uno, como restricción de columna (no se puede usar cuando la clave es múltiple).

```
CREATE TABLE articulo (codigo_del_articulo VARCHAR2(8) PRIMARY KEY,  
nombre_de_articulo VARCHAR2(30),  
precio_unitario NUMBER(8,3));
```

- Variante dos, como restricción de tabla (es la única que se puede usar cuando la clave es múltiple).

```
CREATE TABLE articulo (codigo_del_articulo VARCHAR2(8),  
nombre_de_articulo VARCHAR2(30),  
precio_unitario NUMBER(8,3),  
CONSTRAINT articulos_pk PRIMARY KEY (codigo_del_articulo));
```

En este último ejemplo, se usó la cláusula CONSTRAINT para que la restricción tuviera un nombre “articulos\_pk”, que es muy útil cuando una aplicación trata de cargar una fila repetida y el servidor se lo impide. Éste le enviará un mensaje diciendo que esta restricción, “articulos\_pk”, ha sido atacada, lo que permitirá entender que se ha intentado cargar una fila con estas columnas con *null* o con un valor de clave repetido.

Ahora se agregarán las claves foráneas y se advertirá que la tabla referenciada debe existir. En una implementación real, también se considerará que si las tablas ya existen desde hace tiempo y tienen filas que no cumplen con las relaciones que se introducirán, se le indicará que ignore los incumplimientos previos, pero no se lo verá en la sintaxis porque es un tema avanzado en el estudio de SQL, que excede el alcance de este libro.

```
ALTER TABLE factura  
ADD CONSTRAINT cliente_fk FOREIGN KEY (codigo_del_cliente)  
REFERENCES clientes(codigo_del_cliente);
```

```
ALTER TABLE detalle_de_factura  
ADD CONSTRAINT articulo_fk FOREIGN KEY (codigo_del_artículo)  
REFERENCES artículo(codigo_del_artículo);
```

```
ALTER TABLE detalle_de_factura  
ADD CONSTRAINT factura_fk FOREIGN KEY (numero_de_la_factura)  
REFERENCES factura(numero_de_la_factura);
```

Para completar las restricciones que permite el lenguaje SQL, se deberá también nombrar el de validación de contenido o CHECK, que impide ingresar un valor en una columna que no cumpla la condición definida en la restricción. Se aplicará un requerimiento que indicará que los precios de los artículos no serán negativos, entonces:

```
ALTER TABLE artículo
ADD CONSTRAINT precio_ck CHECK (precio_unitario_del_articulo > 0);
```

Para exemplificar otras sentencias DDL, se ha omitido erróneamente una ‘s’ en el nombre de la tabla “Artículos”, para corregirlo se utilizará RENAME. Será tarea del motor actualizar todos objetos que hayan sido creados con referencias a este objeto para que se mantenga esta relación.

```
RENAME TABLE artículo TO artículos;
```

Ahora es el momento de crear otros objetos de datos: las vistas y los índices.

Aunque aún no se ha visto la sentencia SELECT, se aplicará una sentencia muy simple que, sin analizar los detalles, traerá todas las facturas que tengan en el código de cliente un valor ‘A01’.

```
CREATE VIEW v_factura_clienteA01 AS
SELECT numero_de_factura, fecha_de_factura, monto_de_factura
FROM facturas
WHERE codigo_de_cliente = A01'
```

Así, se habrá creado una vista que se usará de la misma forma que una tabla; por eso, se recomienda que en el nombre se identifique claramente que se trata de una vista. Por ejemplo, si se utiliza un prefijo como ‘v\_...’, como se mostró en el ejemplo anterior.

Para los índices la sintaxis es:

```
CREATE INDEX factura_fecha_idx ON factura(fecha_de_la_factura);
```

La consecuencia de esta sentencia es que se ha creado un índice —implementado como un segmento de datos— similar a una tabla en la que se guarda el identificador único de las que tienen cada fecha distinta; esto facilitará el acceso a las páginas y a las filas donde estén las fechas buscadas, de acuerdo con el funcionamiento del motor de la base. Habitualmente, estos índices tienen la característica de ser árboles B, lo que garantiza el acceso rápido a las filas que cumplen el criterio de búsqueda que use la columna “fecha\_de\_la\_factura”. En otras variantes de sintaxis, se puede indicar que no se permitan las repeticiones (“CREATE UNIQUE INDEX ...”), o que arme una estructura de índice que no se base en árboles, sino en una estructura en mapas de bit (“CREATE BITMAPPED INDEX ...”) con características dirigidas a la indexación de columnas con baja selectividad, o con pocos valores distintos, lo que permite que se resuelvan rápidamente las consultas basadas en las columnas con pocos valores diferentes y gran cantidad de filas cada uno.

La creación de roles, a los que se puede definir como un conjunto de privilegios que se asignan siempre en bloque, se presentará luego de la creación de usuarios.

```
CREATE USER jgomez IDENTIFIED BY lacontraseña;
```

El rol de operador de consultas “op\_consultas” se crea de la siguiente manera:

```
CREATE ROLE op_consultas;
```

Los roles se crean para facilitar la administración de privilegios de muchos objetos a grupos de usuarios que necesitan los mismos accesos porque realizan trabajos similares.

En este ejemplo, se le darán todos los privilegios al rol “op\_consulta” y, luego, se le asignará al usuario “jgomez”.

En Oracle, se puede definir un rol con contraseña para que sea activado después de que el usuario que lo posee se conecte, cuando tenga que realizar una tarea que requiera los privilegios del rol y solicite activarlo. A la vez, se le solicita una contraseña específica para que esa activación pueda ser posible.

En algunas bases de datos, con la mera creación del usuario alcanza para poder conectarse inmediatamente; en Oracle, es preciso que se le dé los permisos de conexión “create session”, con el siguiente grupo de sentencias DDL: GRANT y REVOKE. Para esto, se aplicará al usuario un privilegio de acción o de sistema.

```
GRANT CREATE_SESSION TO jgomez;
```

Otro privilegio de acción o de sistema sería darle al usuario la posibilidad de consultar cualquier tabla en la base de datos, que se aplicará a las actuales y a las que no se han creado aún.

```
GRANT SELECT ANY TABLE TO jgomez;
```

Ahora, se le otorgará al rol todos los privilegios y, luego, se los asignaremos todos juntos al usuario con un solo GRANT.

Con los privilegios de objetos, que son SELECT, ALTER, DROP, DELETE, REFERENCE, INDEX y el que abarca a todos, ALL, la sintaxis indicará sobre qué objetos se asignarán los privilegios al rol:

```
GRANT SELECT ON articulos TO op_consulta;
```

Si quisieramos darle los otros privilegios solamente a jgomez, se hace directamente:

```
GRANT DELETE ON articulos TO jgomez;
```

Para evitar escribir todas las sentencias, que podría ejecutar el usuario, se puede abbreviar con:

```
GRANT ALL ON articulo TO jgomez;
```

Luego, le otorgaremos el rol de op\_consultas al usuario jgomez, de esta manera:

```
GRANT op_consultas TO jgomez;
```

Si por alguna razón se quisiera quitar estos privilegios, se usará la sentencia REVOKE como sigue:



En Oracle, se puede definir un rol con contraseña para que sea activado después de que el usuario que lo posee se conecte, cuando tenga que realizar una tarea que requiera los privilegios del rol y solicite activarlo.

```
REVOKE CREATE_SESSION FROM jgomez;
REVOKE op_consulta FROM jgomez; (aquí le quitamos el rol al usuario)
REVOKE ALL FROM jgomez;
```

Se finalizará el tratamiento de DDL con un ejemplo de la creación de comentarios sobre una tabla y sobre una columna.

COMMENT ON articulos ('Contendrá todos la información de los artículos que se venden en este negocio');

COMMENT ON articulos.nombre\_del\_articulo ('Almacena el nombre completo de cada artículo')

### 3.5 Sentencias del sublenguaje DML



El sublenguaje DML abarca a **SELECT**, **INSERT**, **DELETE** y **UPDATE** que sirven para realizar consultas, insertar filas, para borrarlas, y para cambiarle el contenido de las columnas en las filas existentes y para que cumplan con las condiciones de la cláusula **WHERE**.

El sublenguaje DML (*Data Manipulation Language*) contiene sentencias para la administración de los datos. Este sublenguaje abarca a **SELECT** —que se tratará inmediatamente— **INSERT**, **DELETE** y **UPDATE** que sirven —respectivamente— para realizar consultas, insertar filas, para borrarlas, y para cambiarle el contenido de las columnas en las filas existentes y que cumplan con las condiciones de la cláusula **WHERE**, como se verá en los siguientes ejemplos.

**SELECT** es la sentencia que permite la obtención de los valores almacenados en las columnas de las filas recuperadas como resultado del acceso a las tablas o a las vistas de la base de datos. A esta sentencia también se la denomina “Consulta”, porque es, justamente, lo que hace el motor: busca las filas de las tablas incluidas en su cláusula **FROM** y realiza las operaciones definidas en el Álgebra relacional para seleccionar aquellas filas que cumplen las condiciones expresadas.

Esta sentencia se estructura en distintas cláusulas, con el funcionamiento que se describirá a continuación:

```
SELECT <lista de columnas, valores constantes, funciones, subconsultas, etc.>
FROM <lista de tablas, vistas, subconsultas>
WHERE <condicion> [AND, OR, NOT] <condicion>
GROUP BY <lista de columnas>
HAVING <condicion> [AND, OR, NOT] <condicion>
ORDER BY <lista de columnas, numero de orden o alias de columna> [DESC, ASC];
```

Las cláusulas obligatorias son **SELECT** y **FROM**, en Oracle; en otras bases como MySQL, solo es obligatoria la cláusula **SELECT**.

En esta cláusula, se puede realizar la operación Proyección del Álgebra relacional cuando se enumeran solo algunas columnas de la tabla que se consultará. Para evitar esta operación, se incluirán todas las columnas, una por una, o se usará el operador

asterisco (' \* ') que indicará que es necesario la recuperación de todas las columnas de la o las tabla/s del FROM.

En la primera cláusula, se pueden combinar las columnas de las tablas, las vistas o las subconsultas de la cláusula FROM, funciones como las que veremos en los párrafos siguientes y subconsultas. Estas subconsultas devuelven un solo valor, por lo que se las denomina "escalares". También se agregan valores literales que se repetirán por cada fila mostrada. Por ejemplo: la sentencia muestra datos constantes, una columna con el nombre de los empleados, luego el sueldo y, en la columna aumento, el cálculo de sueldo con un 20% de aumento. Es para destacar que en los dos últimos se adoptó un "Alias de Columna" para titular las columnas con un nombre distinto al de las columnas de la tabla.

```
SELECT 'El nombre es:', ename, salario salarioAnterior, salario*1.20 aumento,
FROM empleados;
```

Resulta en:

|                 | ename  | salarioAnterior | aumento |
|-----------------|--------|-----------------|---------|
| 'El nombre es:' | Perez  | 1100            | 1200    |
| 'El nombre es:' | Lopez  | 900             | 1080    |
| 'El nombre es:' | Suarez | 1100            | 1320    |
| 'El nombre es:' | Gianni | 700             | 840     |
| 'El nombre es:' | Vinci  | 1500            | 1800    |



Las funciones de fila simple retornan un solo valor por cada fila de una tabla o de una vista consultada.

### 3.5.1 Funciones de fila simple

En el ejemplo anterior, se vio el uso de una función de fila simple, el operador multiplicación '\*\*'. En este caso, '\*\*' es la función multiplicación que, en otro contexto, significa "todas las columnas de las tablas que figuran en el FROM".

Las funciones de fila simple retornan un solo valor por cada fila de una tabla o de una vista consultada. Estas funciones pueden usarse en la lista de columnas del SELECT, en las cláusulas WHERE y HAVING, que forman parte de las condiciones y demás.

### 3.5.2 Funciones numéricas

La mayoría de las funciones numéricas, que reciben y devuelven valores numéricos, retornan valores NUMBER con exactitud en 38 dígitos decimales.

Las más simples son los operadores matemáticos:

'\*\*' por, '/' dividido, '+' más, '-' menos

Las funciones como COS, COSH, EXP, LN, LOG, SIN, SINH, SQRT, TAN y TANH tienen una exactitud de 36 de dígitos decimales y las funciones ACOS, ASIN, ATAN una exactitud de 30 dígitos decimales. A modo de ejemplo, algunas funciones del SQL de Oracle 10g son:

ABS (Valor absoluto)  
ACOS (Arc coseno)  
ASIN (Arc seno)  
ATAN (Arc tangente)  
CEIL (Valor tope)  
COS (Coseno)  
COSH (Coseno hiperbólico)  
EXP (Exponente)  
FLOOR (Valor mínimo)  
LN (Logaritmo neperiano)  
LOG (Logaritmo decimal)  
MOD (Resto de la división)  
POWER (Elevar a una potencia)  
ROUND (Number) (Redondeo numérico)  
SIGN (Signo)  
SIN (Seno)  
SINH (Seno hiperbólico)  
SQRT (Raíz cuadrada)  
TAN (Tangente)  
TANH (Tangente hiperbólico)  
TRUNC (Number) (Truncar numérico)



Para una lista completa de funciones de fia simple en Oracle visite:  
<http://tahiti.oracle.com>

Las funciones de carácter son aquellas que reciben argumentos numéricos, de fecha o de caracteres y devuelven valores en formato carácter. Éstos son:

CHR (devuelve un carácter de acuerdo con un argumento numérico, normalmente el valor ASCII)  
CONCAT (Concatena valores)  
INITCAP (Pone mayúscula a las primeras letras de cada palabra)  
LOWER (Transforma en minúsculas)  
LPAD (Completa a la izquierda con un carácter hasta una cantidad de caracteres)  
LTRIM (Recorta una cantidad de caracteres a la izquierda)  
REPLACE (Reemplaza un carácter por otro)  
RPAD (Completa a la derecha con un carácter hasta una cantidad de caracteres)  
RTRIM (Recorta una cantidad de caracteres a la derecha)  
SUBSTR (Toma una subcadena de una cantidad de caracteres desde una posición determinada)  
UPPER (Transforma en mayúsculas)

Las funciones de fecha y hora son:

ADD\_MONTHS (Suma meses a una fecha)  
CURRENT\_DATE (La fecha del sistema)  
EXTRACT (*Datetime*) (Extrae fecha u hora)  
LAST\_DAY (Último día del mes de la fecha argumento)  
MONTHS\_BETWEEN (Cantidad de meses entre dos fechas argumento)  
ROUND (*date*) (Redondeo de fechas, a mes o a año)  
SYSDATE (Fecha del sistema)  
TO\_CHAR (*Datetime*) (Traducir a caracteres una fecha)  
TRUNC (*Date*) (Truncar una fecha)

### 3.5.3 Funciones generales de comparación

Las funciones generales de comparación determinan los valores mayores o menores de un conjunto de datos:

GREATEST (Mayor)  
LEAST (Menor)

### 3.5.4 Funciones de conversión

Convierten un valor de un tipo de datos a otro tipo de datos:

TO\_CHAR (*Character*)  
TO\_CHAR (*Datetime*)  
TO\_CHAR (*Number*)  
TO\_CLOB  
TO\_DATE  
TO\_LOB  
TO\_MULTI\_BYTE  
TO\_NUMBER

### 3.5.5 Funciones de grupo

Las funciones de grupo devuelven una fila con el resultado de sumar, contar, etc. de todas las filas resultantes de la consulta. En la cláusula WHERE de la consulta, no se pueden usar las funciones de grupo para filtrar filas porque el resultado de una función de grupo se conoce luego de reunir todas las filas. Es decir, luego de ejecutar el WHERE. Por eso, las funciones de grupo se usan en la cláusula HAVING.

Las funciones de grupo son usadas comúnmente en combinación con la cláusula GROUP BY, con la que se divide a las filas seleccionadas porque cumplen con las condiciones de filtro del WHERE, en grupos basados en los distintos valores de las columnas incluidas en el GROUP BY.

Dada la tabla de estudiantes:

| Tabla Estudiantes |          |         |
|-------------------|----------|---------|
| Legajo            | Apellido | Carrera |
| 7898              | Messi    | SIS     |
| 6677              | García   | SIS     |
| 6644              | Bremen   | SOF     |
| 3566              | Pérez    | ABO     |
| 5522              | Palermo  | SOF     |
| 2123              | López    | ABO     |
| 5221              | Vianni   | ABO     |
| 4512              | Suárez   | SOF     |

“GROUP BY columna” separa las filas del conjunto, en grupos de filas que tienen el mismo valor en la columna.

| Tabla Estudiantes agrupada por carrera |          |         |
|--|----------|---------|
| Legajo                                 | Apellido | Carrera |
| 7898                                   | Messi    | SIS     |
| 6677                                   | García   | SIS     |
| 6644                                   | Bremen   | SOF     |
| 5522                                   | Palermo  | SOF     |
| 4512                                   | Suárez   | SOF     |
| 3566                                   | Perez    | ABO     |
| 2123                                   | López    | ABO     |
| 5221                                   | Vianni   | ABO     |

La consulta:

```
SELECT carrera, count(*)
```

```
FROM estudiantes
```

```
GROUP BY carrera;
```

Nos da como resultado:

Carrera COUNT(\*)

|     |   |
|-----|---|
| ABO | 3 |
| SIS | 2 |
| SOF | 3 |

En el ejemplo, se aplicó la función COUNT(\*), que cuenta las filas seleccionadas y filtradas por la consulta y muestra el resultado del conteo.

Si se usa una función de grupo sin la cláusula GROUP BY, éstas se aplicarán a todo el conjunto de filas seleccionadas. Se usan las funciones de grupo también en la cláusula HAVING, para filtrar los grupos que no cumplen con las condiciones. WHERE filtra grupos antes de que se armen los grupos y HAVING filtra filas una vez que los grupos están armados.

Todas las funciones de grupo ignoran los valores nulls; esto es, si en la columna "Apellido" se va a contar con COUNT(Apellido) y tiene en todas las filas valores null, esta función devolverá 0 (cero); si solo en algunas tuviera nulls, contará todas las filas con valores distintos de null.

El asterisco en COUNT(\*) le indica a la función que cuente todas las filas de la o las tablas que están en el FROM. Según la definición de una tabla relacional, ésta no puede tener una fila con todos valores null o fila nula.

Es posible anidar funciones de grupo, dada la tabla "Notas" (se muestran solo algunas columnas):

| Tabla Notas |            |      |
|-------------|------------|------|
| Notas       |            |      |
| Legajo      | Id_Materia | Nota |
| 7898        | 1          | 7    |
| 6677        | 1          | 7    |
| 6644        | 2          | 8    |
| 2123        | 2          | 9    |
| 5522        | 3          | 10   |
| 2123        | 3          | 10   |
| 5221        | 5          | 2    |
| 4512        | 5          | 4    |

Para calcular el promedio de la nota más alta de cada materia, se puede hacer:

```
SELECT AVG(MAX(notas))
FROM examenes
GROUP BY id_materia;
AVG(MAX(notas))
```

7,5

Esta consulta evalúa el agrupamiento interno (MAX(notas)) trayendo las notas más altas de cada materia, las agrupa nuevamente y, luego, les calcula el promedio.

Tomó la nota máxima de cada material (7, 9, 10, 4) y las promedió (30/4)= 7,5.

Algunas de las funciones de grupo más usadas son:

AVG (Average o promedio)  
COUNT (Conteo)  
MAX (Máximo)  
MIN (Mínimo)  
STDDEV (Desviación estándar)  
SUM (Suma)  
VARIANCE (Varianza)

### 3.5.6 La cláusula FROM

En la cláusula FROM, se enumeran las tablas, las vistas y las subconsultas que se consultarán para buscar las columnas que se enumeran en el SELECT. Cuando hay más de una referencia a tablas, se dice que la consulta es MULTITABLA. En este caso, se está frente a la aplicación de la operación reunión (JOIN) del Álgebra relacional y, si no se escriben las condiciones de reunión en el WHERE, el optimizador de consultas realizará un producto cartesiano que relacionará todas las filas de las tablas reunidas, lo que generará resultados falsos: si se tuvieran cinco alumnos y tres exámenes rendidos, la reunión sin “condiciones de reunión”, daría quince filas ( $5 \times 3$  filas). Esto produce un gran volumen de filas reunidas que, habitualmente, no se utilizarán.

Lo expresado en el párrafo precedente es en general, pero se observa que, para algunos casos, esta operación —producto cartesiano— será necesaria, por ejemplo, en el caso en que se requiera relacionar todos los jugadores de un torneo de tenis para armar los partidos posibles, teniendo en cuenta que se eliminarán los pares en los que se repiten los mismos jugadores. ¿Cómo sería esto? Supongamos que hay catorce empleados y se quiere tener previstos todos los partidos posibles (los imposibles serían, por ejemplo, que un empleado juegue contra sí mismo), para esto se puede hacer el producto cartesiano de la tabla “Empleados” consigo misma con distinto alias de tablas, para identificar a qué rol corresponde.

En el primer intento, dejará un producto cartesiano de la tabla “Empleados” consigo misma. Esta forma de reunión se denomina, también, Auto Join.

```
SELECT l.ename Local, 'juega con ', v.ename Visita  
FROM empleados l, empleados v
```

Si esta tabla tiene 14 filas, da un resultado de 196 filas (ver en la Fig. 3.1, en la que se muestran, a modo ilustrativo, solo las primeras).

En la siguiente consulta, se resolverá el requerimiento que armará los partidos en el campeonato interno de la empresa (ver Fig. 3.2).

```
SELECT l.ename Local, 'juega con ', v.ename Visita  
FROM empleados l, empleados v  
WHERE l.ename <> v.ename
```

Si bien la intención no es adelantarse al próximo apartado, se observa que hay una cláusula WHERE con una condición que evita que se una al mismo empleado como local y como visitante.

Es importante destacar que se han usado alias de tablas para identificar las columnas “ename”, que como vienen de la misma tabla, pero repetida en el FROM, se las ha calificado con la letra que se escribe, en esta cláusula, a continuación de la tabla. Esta forma de nombrar las columnas de una tabla, en una posición determinada, se denomina alias de tabla y, como recomendación, se usó la letra inicial del rol o del significado que tiene cada repetición de la tabla en el SELECT; para este caso, se adoptó la “l” para Local y la “v” para Visita.

|   | Local  | juega con | Visita |
|---|--------|-----------|--------|
| ▶ | SMITH  | juega con | SMITH  |
|   | ALLEN  | juega con | SMITH  |
|   | WARD   | juega con | SMITH  |
|   | JONES  | juega con | SMITH  |
|   | MARTIN | juega con | SMITH  |
|   | BLAKE  | juega con | SMITH  |
|   | CLARK  | juega con | SMITH  |
|   | SCOTT  | juega con | SMITH  |
|   | KING   | juega con | SMITH  |
|   | ADAMS  | juega con | SMITH  |
|   | JAMES  | juega con | SMITH  |
|   | FORD   | juega con | SMITH  |
|   | MILLER | juega con | SMITH  |
|   | TURNER | juega con | SMITH  |
|   | SMITH  | juega con | ALLEN  |
|   | ALLEN  | juega con | ALLEN  |
|   | WARD   | juega con | ALLEN  |
|   | JONES  | juega con | ALLEN  |
|   | MARTIN | juega con | ALLEN  |
|   | BLAKE  | juega con | ALLEN  |
|   | CLARK  | juega con | ALLEN  |
|   | SCOTT  | juega con | ALLEN  |
|   | KING   | juega con | ALLEN  |
|   | ADAMS  | juega con | ALLEN  |
|   | JAMES  | juega con | ALLEN  |
|   | FORD   | juega con | ALLEN  |
|   | MILLER | juega con | ALLEN  |

Fig. 3.1 Producto cartesiano: se observa que es imposible que SMITH juegue contra SMITH, sin embargo, la operación lo permite.

| Local  | juega con | Visita |
|--------|-----------|--------|
| ALLEN  | juega con | SMITH  |
| WARD   | juega con | SMITH  |
| JONES  | juega con | SMITH  |
| MARTIN | juega con | SMITH  |
| BLAKE  | juega con | SMITH  |
| CLARK  | juega con | SMITH  |
| SCOTT  | juega con | SMITH  |
| KING   | juega con | SMITH  |
| ADAMS  | juega con | SMITH  |
| JAMES  | juega con | SMITH  |
| FORD   | juega con | SMITH  |
| MILLER | juega con | SMITH  |
| TURNER | juega con | SMITH  |
| SMITH  | juega con | ALLEN  |
| WARD   | juega con | ALLEN  |
| JONES  | juega con | ALLEN  |
| MARTIN | juega con | ALLEN  |
| BLAKE  | juega con | ALLEN  |
| CLARK  | juega con | ALLEN  |
| SCOTT  | juega con | ALLEN  |
| KING   | juega con | ALLEN  |
| ADAMS  | juega con | ALLEN  |
| JAMES  | juega con | ALLEN  |
| FORD   | juega con | ALLEN  |
| MILLER | juega con | ALLEN  |
| TURNER | juega con | ALLEN  |

182 rows fetched in 0,0174s (0,2117s)

Fig. 3.2 Reunión sin producto cartesiano. Se agregó una restricción en el WHERE y se armó el conjunto de partidos posibles

Nótese que el resultado de 182 filas es porque se excluyeron los partidos entre un empleado contra él mismo; es decir: aquellos partidos imposibles que totalizan 14 partidos.

Entonces, cuando en el FROM se mencionan más de una tabla y se desea evitar el producto cartesiano, se debe escribir una condición de reunión en el WHERE generalmente, en estas consultas, se reúnen las tablas a través de sus claves foráneas y claves primarias, que concretan la reunión entre tablas relacionadas. Pero esto no es excluyente, también se pueden escribir otras condiciones de reunión con columnas del mismo dominio de valores; es decir, comparables.

Para reunir las tablas, existen dos alternativas de sintaxis: la primera y más antigua, es nombrarlas en el FROM y, luego, se evitirá el producto cartesiano con condiciones de reunión en el WHERE. En la segunda, se escribe la palabra JOIN entre las tablas y, después, en el mismo FROM, se escribirá la condición dentro del operador ON. De esta manera, se obtendrá el mismo resultado que la Fig. 3.2:

```
SELECT l.ename Local, ' juega con ', v.ename Visita
FROM empleados l JOIN empleados v ON (l.ename <> v.ename)
```

Esto se considera como estándar desde SQL3. Se adjunta la ayuda de MySQL sobre la sintaxis de esta forma de escribir las reuniones de tablas.

```
table_reference [INNER | CROSS] JOIN table_factor [join_condition]
| table_reference STRAIGHT_JOIN table_factor
| table_reference STRAIGHT_JOIN table_factor ON condition
| table_reference LEFT [OUTER] JOIN table_reference join_condition
| table_reference NATURAL [LEFT [OUTER]] JOIN table_factor
| table_reference RIGHT [OUTER] JOIN table_reference join_condition
| table_reference NATURAL [RIGHT [OUTER]] JOIN table_factor

join_condition:
  ON conditional_expr
  | USING (column_list)
```

En la sintaxis mostrada aparecen operadores que no serán tratados en este libro.

### 3.5.7 La cláusula WHERE

En esta cláusula, se escriben las condiciones de filtro que permiten elegir aquellas filas que se quieren mostrar. La condición es una construcción que tiene tres partes en general y, como excepción, cuando se usa el operador de comparación EXISTS o NOT EXISTS, dos.

Una condición posee como elementos una columna, un operador de comparación y un valor constante, otra columna, una variable o una subconsulta que devuelva uno o varios valores, que se aceptan solamente si el operador es IN, NOT IN, EXISTS y NOT EXISTS, que son los que tratan con lista de valores.

Ejemplos de condiciones: “Una fila es elegida si...

WHERE sal > 1000 ... es verdad que el salario de la fila es mayor a 1000”.

WHERE ename = ‘SMITH’ ...es verdad que el apellido de la fila es ‘SMITH’ (en mayúsculas)”.

WHERE hiredate < ‘01/03/1998’ ... es verdad que la fecha de contratación es anterior al primero de marzo de 1998”.

o negarlas con NOT,

“Una fila es elegida si ...

WHERE NOT sal > 1000 ... NO es verdad que el salario de la fila es mayor a 1000”.

También se pueden combinar las condiciones, lo que permite que se las evalúe, simultáneamente, con los coordinadores AND y OR. De esta manera, se obtendrá que:

“Una fila es elegida si...



En la cláusula WHERE, se escriben las condiciones de filtro que permiten elegir aquellas filas que se quieren mostrar.

WHERE (sal > 1000) AND (ename = 'SMITH') ... es verdad que el salario de la fila es mayor a 1000 y que tiene un valor en apellido igual a 'SMITH' (en mayúsculas).

WHERE hiredate < '01/03/1998' OR (ename = 'SMITH') ... es verdad que la fecha de contratación es anterior al primero de marzo de ese año o si el valor de su columna ename es igual a 'SMITH' (en mayúsculas)".

No hay límites en la cantidad de condiciones a definir en la cláusula WHERE. Las múltiples combinaciones posibles de éstas, permiten obtener las filas requeridas para responder cualquier requerimiento de datos de tablas correctamente normalizadas.

Los operadores de comparación son:

= igual

< menor que

> mayor que

<> distinto que

= ANY/ALL compara con todos los valores de una lista, =ANY es equivalente a IN

< ALL es menor que todos los valores de una lista o subconsulta.

> ALL es mayor que todos los valores de una lista o subconsulta.

<= ALL permite detectar al menor de todos los valores de una lista o subconsulta.

>= ALL permite detectar al mayor de todos los valores de una lista o subconsulta.

< ANY es menor que algunos de los valores de una lista o subconsulta.

> ANY es mayor que algunos de los valores de una lista o subconsulta.

IN es igual que al menos uno de los valores de una lista o subconsulta.

NOT IN no es igual que al menos uno de los valores de una lista o subconsulta.

BETWEEN límite inferior AND límite superior está dentro de un rango inclusivo.

NOT BETWEEN límite inferior AND límite superior está fuera de un rango inclusivo.

LIKE patrón como % (comodines de múltiples valores y múltiple cantidad) y \_ idem pero solo de una posición.

NOT LIKE no cumple con un patrón como % (comodines de múltiples valores y múltiple cantidad) y \_ idem pero solo de una posición.

EXISTS test de existencia en una subconsulta.

NOT EXISTS test de no existencia en una subconsulta.

### 3.5.8 La cláusula GROUP BY

Como se vio en la explicación de las funciones de grupo, esta cláusula define sobre qué valores de columna o columnas se basará el optimizador para agrupar las filas por sus distintos valores.

### 3.5.9 La cláusula HAVING

Una vez definidos los grupos, se puede escribir en HAVING una condición que usa las funciones de grupo para poder filtrar o seleccionar y mostrar solo aquellos grupos que cumplen con las condiciones presentes en HAVING.

Se la suele comparar con la cláusula WHERE, porque aplica condiciones, pero que quede claro que WHERE filtra filas; HAVING filtra grupos armados en GROUP BY.

### 3.5.10 La cláusula ORDER BY

La última cláusula de la sentencia SELECT es la que permite dar el orden deseado a las filas encontradas que cumplen la condición, junto con los valores de las funciones y otros cálculos o formateos realizados con esta sentencia. Es posible combinar varias columnas para ordenar y el orden de aparición de estas columnas en la cláusula ORDER BY no es trivial, comenzará por la primera columna que aparece luego de la palabra BY.

Existen dos indicadores del sentido en el que se debe ordenar, DESC de descendente o ASC de ascendente. Si no se indica ninguno, se toma por defecto ASC.

Así, si se usa DESC como indicador del sentido del ordenamiento, mostrará de mayor a menor y, si es ASC, de menor a mayor. Esto es, en cada columna.

## 3.6 Sentencias del sublenguaje DML. INSERT, UPDATE, DELETE

### 3.6.1 Sentencia INSERT

La sentencia INSERT del grupo de sentencias del Data Manipulation Language permite insertar filas en una tabla e ingresar columna a columna los valores de cada una de ellas:

```
INSERT INTO empleados (empno, ename, job, mgr, hiredate, sal, comm, deptno)
VALUES (8000, 'Alves, Pedro', 'Programa', 7411, 3500, null, 20)
```

La lista de columnas se puede obviar, pero por claridad se sugiere no dejar de consignarla.

Otra forma para insertar filas en una tabla —y más de una por ejecución— es la que incluye un select dentro del insert. A esta sentencia subordinada, habitualmente se la denomina subconsultas:

```
INSERT INTO empleados_historico (empno, ename, job, mgr, hiredate, sal,
comm, deptno, fecha_carga);
SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno, current_date
FROM empleados
WHERE hiredate < '01/01/2000';
```

En este caso, se parte del supuesto de que estamos insertando, en la tabla de empleados históricos, los empleados contratados antes del 1 de enero de 2000, y se agrega la fecha actual en la columna "fecha\_carga" de la mencionada tabla. En



La sentencia INSERT del grupo de sentencias del DML permite insertar filas en una tabla e ingresar columna a columna los valores de cada una de ellas.

esta operación, se insertarán más de una fila en la tabla objetivo, que es una de las aplicaciones de las subconsultas y se verá más en detalle en las siguientes lecturas.

### 3.6.2 Sentencia UPDATE

La sentencia para cambiar el valor de una o más columnas en una tabla es UPDATE —que tiene la siguiente sintaxis en Oracle:

```
UPDATE tbl_name
  SET col_name1 =expr1 [, col_name2=expr2 ...]
    [WHERE where_condition];
```

Por ejemplo, si se desea aumentar el sueldo de todos los empleados la sentencia es:

```
UPDATE empleados
  SET sal = sal * 1.2;
```

Esto provoca un aumento de sueldo de un 20% a todos los integrantes de la empresa; si se deseara que se le aumente a unos pocos, se utilizará la cláusula WHERE, que tiene las mismas características que la sentencia SELECT. Si, por ejemplo, a la empresa le interesa solo aumentarle el sueldo a los que ganan entre 1000 y 1200, la sintaxis sería la siguiente:

```
UPDATE empleados
  SET sal = sal * 1.2
 WHERE sal BETWEEN 1000 AND 1200;
```

En esta sentencia UPDATE, se pueden usar subconsultas como argumento del SET y como parte del WHERE, como en cualquier sentencia SELECT.

```
UPDATE empleados
  SET sal = (SELECT AVG(sal) FROM empleados)
 WHERE sal BETWEEN 1000 AND 1200;
```

En la sentencia anterior, se ha modificado el sueldo de los que ganan entre 1000 y 1200 por el valor del sueldo promedio de todos los empleados.

### 3.6.3 Sentencia MERGE

La sentencia MERGE es una combinación de INSERT y UPDATE y, con la sintaxis que contiene las dos sentencias, al ejecutarse, busca la fila; si la encuentra la modifica de acuerdo con la sintaxis del UPDATE. Si no existe ninguna fila que cumpla con el

 La sentencia MERGE es una combinación de INSERT y UPDATE y, con la sintaxis que contiene las dos sentencias, al ejecutarse, busca la fila; si la encuentra la modifica de acuerdo con la sintaxis del UPDATE.

criterio del WHERE, la inserta con la sintaxis del INSERT y si una fila cumple con una condición, la puede borrar.

Ejemplo de la sintaxis en Oracle 10g:

```
MERGE INTO bonus b
USING (
    SELECT empno, sal, deptno
    FROM empleados
    WHERE deptno =20) e
ON (b.empno = e.empno)
WHEN MATCHED THEN
    UPDATE SET b.sal = e.sal * 0.1
    DELETE WHERE (e.sal < 40000)
WHEN NOT MATCHED THEN
    INSERT (b.ename, b.sal)
    VALUES (e.name, e.sal * 0.05)
    WHERE (e.sal > 40000);
```

### 3.6.4 Sentencia DELETE

Para borrar filas de una tabla, la sentencia DELETE es muy simple: alcanza con completar el nombre de la tabla y, opcionalmente, se puede incorporar la cláusula WHERE, con el funcionamiento que se vio en la sentencia SELECT.

```
DELETE emp1
WHERE sal > 5000;
```

Sin la cláusula WHERE y su condición, borraría todas las filas de la tabla; en cambio, con esa cláusula, en la columna "sal", borraría aquellas filas con valores mayores a 5000.

## 3.7 Sentencias del sublenguaje TCL de control de transacciones

Todos los cambios de datos en una base de datos se pueden hacer a través de una transacción. Estas transacciones se confirmarán con COMMIT o se desharán con ROLLBACK. Estas dos sentencias cierran la transacción. El motor de la base de datos, si la transacción se confirma, asegurará que se encuentren los datos modificados. Si una transacción se deshace, el sistema asegurará que no quede truncado ningún dato modificado sin corregir, como si nunca se hubiera realizado el cambio que se borró.

En el siguiente cuadro se observa una transacción:

Cuadro 3.1

```
UPDATE cust_accounts  
SET balance = balance - 1500  
WHERE  
account_no = '70-490930.1';  
COMMIT;  
UPDATE cust_accounts  
SET balance = balance + 1500  
WHERE  
account_no = '70-909249.1';  
COMMIT;
```

Si este código se ejecuta y la base tiene un problema luego del primer COMMIT, las cuentas quedarán desbalanceadas, ya que se han debitado \$1500 de la primera cuenta.

En cambio, si se dejara solamente el último commit, estas sentencias se completarían o anularían conjuntamente, lo que impediría que quedaran desbalanceadas.

La sentencia COMMIT confirma y guarda los cambios de la transacción en curso y libera los recursos bloqueados por cualquier actualización hecha con la transacción actual.

Cuadro 3.2

```
COMMIT [WORK];
```

Si ejecutamos:

Cuadro 3.3

```
DELETE FROM t_pedidos WHERE cod_pedido = 15;  
COMMIT;
```

Borra un registro y se guardan los cambios.

ROLLBACK es la sentencia que permite deshacer la última transacción. Cuando se ejecuta, vuelve atrás las transacciones en todas las tablas hechas desde el último COMMIT. Su sintaxis es muy simple:

```
ROLLBACK;
```

En una transacción pueden definirse puntos de salvaguarda o SAVEPOINTS, para poder realizar ROLLBACKs parciales, deshaciendo solamente los cambios realizados a partir de estos puntos.

```
-----
-----
SAVEPOINT inicio;
---
INSERT...
UPDATE ...
DELETE ...
...
ROLLBACKTO SAVEPOINT inicio;
```

Y deshace los cambios realizados a partir del SAVEPOINT inicio. Es necesario mencionar que si se escribiese un COMMIT entre SAVEPOINT y ROLLBACK, el punto de salvaguarda desaparecerá al confirmarse el cambio.

### 3.8 Procesamiento de consultas

Las aplicaciones de software funcionan de acuerdo con los requerimientos de negocio y, normalmente, se prueban con volúmenes de datos iniciales, que no dan una idea del crecimiento potencial de las cantidades de filas que se agregarán con el funcionamiento continuo de la operativa normal del negocio. Esto llevará a que, en un razonable período de tiempo, las aplicaciones comiencen a cumplir su tarea más lentamente, a medida que las tablas en las que se basa incrementen la cantidad de filas dentro de las tablas afectadas.

Esta lógica evolución necesita que se entienda cómo funciona el mecanismo de ejecución de consultas para, de esta manera, mejorar la performance del sistema en general. Por esta razón, se describirán, brevemente, los componentes del procesamiento de consultas y las formas de resolver el plan de ejecución.

Existen diferentes métodos para determinar qué recursos usará el ejecutor para resolver una sentencia SQL y, así, obtener el resultado esperado. Con esta información, se puede decidir qué alternativa realizará el trabajo en el menor tiempo posible.

El ejecutor de sentencias SQL utiliza un proceso denominado plan de consulta, que se encarga de elegir el método de resolución para encontrar los datos requeridos por la sentencia. Este optimizador tiene dos estrategias o modos de funcionar:

El primero, orientado a la sintaxis con la que está escrito, tiene un ranking con las reglas de acceso a los datos. Este modo se denomina Optimizador Basado en Reglas (RBO, Rules Based Optimizer) y, como se afirmó al comienzo de este párrafo, utiliza la forma en la que está escrita la sentencia y, con la información del diccionario sobre estructuras de datos, determina el plan de ejecución para responderle. Esta estrategia se desarrolló en el origen de las primeras bases de datos y como es estático en el modo de resolver el plan de ejecución, en muchos casos se reemplaza por otro que se apoya en las estadísticas de uso de los objetos y que se describe a continuación.

En el modo basado en costos o CBO (Cost Based Optimizer), el optimizador inspecciona cada sentencia e identifica los caminos posibles de acceso a los datos y establece los costos basado en la cantidad de lecturas lógicas que se realizarán para



El RBO utiliza la forma en la que está escrita la sentencia y, con la información del diccionario sobre estructuras de datos, determina el plan de ejecución para responderle.

ejecutar la sentencia. Este modo emplea las estadísticas que se almacenan sobre los objetos afectados por la sentencia SQL para, de esta manera, determinar el plan de ejecución que demande menos recursos.

El modo del optimizador, que se aplicará en las bases de datos Oracle, se fija por parámetros generales o de la base, en el nivel de la sesión, y, también, para cada sentencia en particular.

### 3.8.1 Plan de consultas

Para cada sentencia, el optimizador prepara un árbol de operaciones denominado plan de ejecución que define el orden y los métodos de operaciones que el servidor seguirá para resolverla.

En un motor Oracle existen numerosas herramientas que permiten la elección de un determinado plan de ejecución y la *performance* asociada. A continuación, se enumerarán los principales recursos:

En el paquete STATPACK, un conjunto de programas obtiene la información de las estadísticas almacenadas sobre los objetos de la base de datos.

El comando SQL, EXPLAIN PLAN genera, en una sesión de base de datos, el plan que se ejecutará con una sentencia SQL dada.

El utilitario TKPROF toma la salida de una sesión que está generando archivos de pistas de auditoría o TRACE y crea un archivo legible con la información que facilitará el estudio de las características del plan de ejecución.

También es posible contar con AUTOTRACE —una opción de SQL\*Plus— que elabora un plan de ejecución y las estadísticas relativas a las operaciones que se realizarán para cumplir con la sentencia.

En una base de datos Oracle con el cliente SQL\*Plus se puede usar el comando EXPLAIN PLAN que, en lugar de ejecutar una sentencia SQL, crea el plan de consultas en una tabla especial llamada PLAN\_TABLE, que contendrá las operaciones que realizará al ejecutarse la sentencia. Por ejemplo:

```
SQL> EXPLAIN PLAN FOR
2      SELECT apellido FROM estudiantes;
```

Se crea el plan y la consulta como se ilustra a continuación:

```
SQL> SELECT *
2      FROM TABLE(dbms_xplan.display);
```

Este método de acceso a la PLAN\_TABLE, vía el paquete provisto DBMS\_XPLAN, es más efectivo ya que resume la información útil del resultado de EXPLAIN PLAN.

### 3.8.2 Optimización de consultas

Con la información de los recursos utilizados por las distintas posibilidades de ejecución de una consulta determinada, será posible entender y elegir la manera de resolver la sentencia que mejor se adapte a las circunstancias requeridas y, eventualmente, se podría decidir si se tomarán medidas complementarias, como la creación de índices que optimicen los tiempos de acceso a la información requerida.

Como hay diferentes modos de ejecutar una sentencia SQL —por ejemplo: alterando el orden de acceso a una tabla o a un índice—, el desarrollador puede influir en la manera en la que el optimizador considere los distintos factores en un proceso en particular. Por ejemplo: incluir en la sentencia los HINTS, que son indicaciones específicas que el optimizador utilizará en la realización de determinadas tareas mediante la utilización de un índice especial o mediante la alteración del orden de acceso a las tablas de consulta.

El tratamiento detallado de las distintas técnicas excede el alcance de este libro. Por esta razón, se recomienda al lector que amplíe estos conceptos en la documentación oficial de los principales proveedores de bases de datos y en las fuentes de información técnica especializada en la materia.

## 3.9 Resumen

Resumiendo, el lenguaje SQL es especializado en comunicarse con los servidores de Bases de Datos relacionales y que puede combinarse con lenguajes de uso general, como JAVA, Cobol, C, Fortran, etc., para construir aplicaciones. Recordemos, también, que SQL es declarativo y no procedimental, porque el escritor de sentencias SQL solamente describe el resultado deseado y no indica cómo resolver el trabajo para obtener ese resultado. Es relativamente sencillo de aprender y le permite, al usuario no especializado en programación, obtener resultados satisfactorios.

### 3.10 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) solo está disponible para docentes.

#### 3.10.1 Mapa conceptual del capítulo

#### 3.10.2 Autoevaluación

#### 3.10.3 Presentaciones\*



# 4

## Lenguaje procedural como extensión de SQL

### Contenido

|  |     |
|--|-----|
| 4.1 Introducción.....  | 132 |
| 4.2 Vista general de PL/SQL.....                               | 132 |
| 4.3 Uso del PL/SQL para acceder a la base de datos Oracle..... | 132 |
| 4.4 Programas con PL/SQL.....                                  | 133 |
| 4.5 Componentes de un bloque PL/SQL.....                       | 134 |
| 4.6 Cursorés.....  | 135 |
| 4.7 Errores.....   | 136 |
| 4.8 El desarrollo de un bloque PL/SQL.....                     | 136 |
| 4.9 Interacción con la base de datos Oracle.....               | 138 |
| 4.10 El tratamiento de transacciones en PL/SQL.....            | 139 |
| 4.11 Sentencias de control de flujo.....                       | 141 |
| 4.12 Manejo de cursorés.....                                   | 144 |
| 4.13 Manejo de errores.....                                    | 148 |
| 4.14 Construyendo procedimientos y funciones con PL/SQL.....   | 151 |
| 4.15 Resumen.....  | 159 |
| 4.16 Contenido de la página Web de apoyo.....                  | 159 |

### Objetivos

- Conocer los antecedentes y el protocolo de estandarización de la extensión procedural de SQL.
- Conocer detalles de un lenguaje procedural de SQL, PL/SQL de Oracle.
- Conocer las estructuras de programación que se pueden escribir con PL/SQL.
- Entender cómo se complementan las sentencias SQL con sentencias de control y estructuras de datos especiales.
- Entender el manejo de errores de PL/SQL.
- Ver ejemplos prácticos de las construcciones posibles con PL/SQL.

## 4.1 Introducción

Este capítulo de SQL/PSM del estándar ISO/IEC 9075, con PSM (por sus siglas en inglés, Persistent Stored Modules o Módulos persistentes almacenados), se ocupa de la extensión procedimental de SQL, que incluye el control de flujo, el manejo de condiciones, las sentencias de señalización de condiciones, los cursor y las variables locales y la forma de asignar valores o expresiones a variables y parámetros. Además, SQL/PSM formaliza la declaración y el mantenimiento de las rutinas persistentes escritas en lenguaje de bases de datos, como los procedimientos almacenados. Esta parte del estándar es completamente opcional, ya que las distintas proveedoras de bases de datos tienen sus propios lenguajes de extensión procedural de SQL: Transactional-SQL o T-SQL de Microsoft, para su SQLServer; Procedural Language SQL o PL/SQL, de Oracle; Procedural SQL o PSQL, para Postgresql. Este capítulo se centrará en PL/SQL.

Para escribir aplicaciones con acceso a bases de datos con los lenguajes de uso general como Cobol, C o ForTran, sin usar las extensiones procedimentales de SQL, se escriben sentencias de este lenguaje (conocido como Embedded SQL o SQL Empotrado). Así, al programa en lenguaje anftrión con sentencias SQL empotradas, se lo precompila convirtiendo a las sentencias SQL en llamadas a librerías especiales provistas con el software de Base de Datos, y finaliza el proceso con el compilador común del lenguaje anftrión. Con las extensiones procedimentales de SQL, se escriben funciones, procedimientos y paquetes que son llamados desde cualquier programa que lo necesite, aumentando la reutilización de código almacenado en la base.



La extensión procedimental de SQL tiene ventajas sobre el SQL embebido ya que permite fijar la lógica y las reglas del negocio de las aplicaciones y almacenarlas en un solo lugar, dentro de la base de datos.

## 4.2 Vista general de PL/SQL

La extensión procedimental de SQL tiene ventajas sobre el SQL embebido que otros lenguajes de programación ofrecen desde hace mucho tiempo, ya que permite fijar la lógica y las reglas del negocio de las aplicaciones y almacenarlas en un solo lugar, dentro de la base de datos. En general, los lenguajes de extensión son simples y directos y poseen las construcciones comunes de los lenguajes de uso general. PL/SQL es así y agrega cosas que le permiten el tratamiento de los datos, la modularización de los bloques de código en funciones, los procedimientos, los triggers y los paquetes, además de un manejo de errores muy robusto. Las sentencias, que se comunican internamente con la base de datos, se almacenan en la base de datos Oracle.

Observaremos, en esta vista general, los detalles básicos de las ventajas del uso de PL/SQL y sus construcciones básicas.

## 4.3 Uso del PL/SQL para acceder a la base de datos Oracle

Muchas aplicaciones que usan la arquitectura Cliente/Servidor tienen una cuestión en común: la dificultad de mantener las reglas de negocio de una aplicación.

Cuando las reglas de negocio están descentralizadas a lo largo de la aplicación, los desarrolladores deben realizar cambios en toda esta extensión e implementar pruebas

de sistemas para determinar si los cambios han sido suficientes. Sin embargo, en situaciones críticas en tiempo, estas pruebas integrales no pueden llevarse adelante, ya que aumentan el riesgo de salidas inesperadas de sistemas. La estrategia de diseño apunta a la centralización de la lógica en la aplicación, que permita una administración más sencilla de los cambios. Esta capa intermedia de la lógica de negocio en la aplicación puede ser diseñada con PL/SQL y con estos beneficios:

- PL/SQL se administra centralmente dentro de la base de datos Oracle. Se manejan el código fuente y los privilegios de ejecución con la misma sintaxis que se usó para manejar otros objetos de base de datos.
- PL/SQL se comunica en forma directa o nativa con otros objetos de la base de datos Oracle.
- PL/SQL es fácil de leer y tiene muchas características para modularizar el manejo de código y de los errores.

## 4.4 Programas con PL/SQL

Hay varias formas de construir programas con PL/SQL, desde los diferentes tipos de módulo disponibles a los componentes de un bloque PL/SQL hasta las construcciones lógicas que manejan el flujo de los procesos.

Se verán los componentes del lenguaje y algunos puntos importantes sobre cada una de sus áreas.

### 4.4.1 Modularidad

Este lenguaje permite al desarrollador crear módulos de programas para mejorar la reusabilidad del software y para esconder la complejidad de la ejecución de una operación específica detrás de un nombre de procedimiento o función. Por ejemplo, en un proceso complejo, como el que requiere agregar un registro de un nuevo estudiante, que afecta a varias tablas relacionadas de distintas aplicaciones asociadas: la asistencia, la gestión académica y la contable, es posible crear un procedimiento que, a su vez, use varias construcciones PL/SQL almacenadas separadas o en un paquete para que, en un solo paso de carga de información, se cubra la complejidad que un alta de estudiante implica, y que estos distintos programas se usen en conjunto y también por separado en otros procesos.

### 4.4.2 Procedimientos, funciones, triggers y paquetes

Los módulos de PL/SQL se dividen en cuatro categorías: procedimientos, funciones, triggers y paquetes, cuyas definiciones siguen a continuación:

**Procedimientos:** son un conjunto de sentencias que aceptan y retornan cero o más variables, que se denominarán parámetros.

**Funciones:** son un conjunto de sentencias que aceptan parámetros y que su tarea principal es calcular un valor y devolverlo para finalizar su trabajo; no pueden realizar transacciones mientras son ejecutadas.



Este lenguaje permite al desarrollador crear módulos de programas para mejorar la reusabilidad del software y para esconder la complejidad de la ejecución de una operación específica detrás de un nombre de procedimiento o función.

Triggers: también se conocen por su traducción como disparadores. Sin embargo, en este libro, se mantendrá la denominación de origen, que se podría definir como “un conjunto de sentencias asociadas a una tabla de la base y también a los eventos de sistema”, como, por ejemplo, la conexión de un usuario o el inicio del apagado o del encendido de la base.

Paquetes: son una colección de procedimientos y funciones que tienen dos partes: una especificación de los procedimientos, funciones que están disponibles o que se ponen como públicas (además, pueden tener estructuras de datos definidas por el usuario) y la otra parte del paquete —denominada cuerpo del paquete—, que contiene el código de los procedimientos y de las funciones especificadas.

## 4.5 Componentes de un bloque PL/SQL

Existen tres componentes de cualquier bloque PL/SQL de los presentados anteriormente: la sección de declaración de variables, la sección ejecutable y el manejo de excepciones. La sección de declaración contiene la identificación de todas las variables que se usarán en el bloque de código. Una variable puede ser de un tipo de datos disponible en las base de datos Oracle, como así también de algún tipo exclusivo de PL/SQL.

La sección ejecutable de un bloque comienza con la palabra clave BEGIN y finaliza con la palabra END, o con la palabra EXCEPTION, que es el inicio del último componente encargado de atender los errores que ocurrieron en la sección ejecutable y de definir cómo se debe manejar. Esta sección es opcional en PL/SQL.

Existen dos tipos de bloques de código: los denominados bloques con nombre y los anónimos. Se verán dos ejemplos: el primero de un código de PL/SQL con nombre, más precisamente, una función denominada convertir\_moneda:

```

FUNCTION convertir_moneda
(cantidad           IN NUMBER(12,3),
moneda_original   IN VARCHAR2,
moneda_destino    IN VARCHAR2) RETURNS NUMBER(12,3)
IS
/* ahora comienza la declaración */
conversion         IN NUMBER(12,3);
datos_mal          EXCEPTION;
BEGIN
/*inicio de la sección ejecutable de un bloque*/
IF cantidad > 0 THEN
...
ELSE

```

```

    . . .;
END IF;
EXCEPTION          /*inicio de la sección de manejo de excepciones */
    WHEN datos_mal THEN
    . . .;
END;

```

La otra clase de bloques se conoce como bloque sin nombre o anónimo. En esta clase de bloques, la sección de declaración se inicia, explícitamente, con la palabra DECLARE y contiene las mismas secciones que los bloques con nombre.

```

DECLARE           /*inicio de sección de declaración de bloque*/
conversion        IN NUMBER(12,3);
datos_mal         EXCEPTION;
BEGIN             /*inicio de la sección ejecutable de un bloque*/
    IF cantidad > 0 THEN
    . . .;
    ELSE
    . . .;
    END IF;
EXCEPTION          /*inicio de la sección de manejo de excepciones */
    WHEN datos_mal THEN
    . . .;
END;

```

#### 4.5.1 Las construcciones lógicas y de control de flujo

Están disponibles las estructuras lógicas como loops o bucles FOR, WHILE y las sentencias IF-THEN-ELSE, la asignación de valores y de expresiones. Existen otras construcciones lógicas que incluyen tablas y registros propios de PL/SQL, diferentes de las tablas y de las flas de la base de datos. Todos estos ítems permiten al lenguaje soportar reglas de negocios y, también, proveer datos a las aplicaciones que lo soliciten.



#### 4.6 Cursos

Una de las fortalezas de PL/SQL es la habilidad de manejar cursos, que son los controladores de áreas de memoria que almacenan los resultados de una sentencia SQL. Se utilizan para realizar operaciones con cada fla resultante de una sentencia SELECT, ya que, en el entorno SQL, el resultado de una sentencia SELECT siempre es tratado como un conjunto de flas y no a cada una de ellas como

Los cursos son los controladores de áreas de memoria que almacenan los resultados de una sentencia SQL

una individualidad. Para el manejo de estas flas en un cursor, es necesario que se utilicen construcciones repetitivas con lógica incluida, como los loop y, también, las operaciones de manipulación de los cursores. Esto se profundizará más adelante con ejemplos concretos.



Los errores se denominan excepciones en PL/SQL y se analizan implícitamente en cualquier lugar del código ejecutable en el que aparecen.

## 4.7 Errores

Los errores se denominan excepciones en PL/SQL y se analizan implícitamente en cualquier lugar del código ejecutable en el que aparecen. Si en algún momento ocurriera un error en el bloque de código, la correspondiente excepción al error se levantaría con la acción RAISE. En este punto, la ejecución en el código se detendrá y el control se transferirá al manejador de excepciones dentro de la sección de correspondiente, que se podrá escribir al final de la sección ejecutable. Esta sección, como es opcional en el bloque, puede que no se haya escrito, lo que hará que el error se envíe al entorno que convocó a este bloque, y allí se atenderá o se ignorará, según cómo se haya previsto la condición de error en este lenguaje anftrión.

## 4.8 El desarrollo de un bloque PL/SQL

En esta sección, se verá cómo se declaran, se asignan valores y se usan las variables, dentro de los bloques que se han visto anteriormente.

### 4.8.1 Los tipos de datos de la base de datos

Hay varios tipos de datos que se utilizan en PL/SQL y que corresponden a los tipos de datos que se usan en la base de datos. Estos son:

NUMBER(tamaño[, precisión]): se usa para almacenar cualquier número.

CHAR(tamaño), VARCHAR2(tamaño): se usan para almacenar una cadena de texto alfanumérico. El tipo CHAR ocupa todo el tamaño y agrega, al contenido significativo, los blancos hasta que cubra el tamaño de la variable. VARCHAR2 almacena solo el contenido significativo, lo que ahorra espacio.

DATE: se usa para almacenar fechas, horas, minutos y segundos.

LONG: almacena grandes bloques de texto, hasta 2 GigaBytes de largo.

LONG RAW: almacena grandes bloques de datos en formato binario, como puede ser sonido digital e imágenes. Actualmente, Oracle recomienda que, en su lugar, se use el tipo de datos CLOB.

ROWID: se usa para almacenar el formato especial de ROWIDs en la base de datos. El ROWID es un identificador único para una fila dentro de una base de datos, que no puede ser modificado por el usuario y es de uso exclusivo de Oracle.

BLOB, CLOB, NCLOB, BFILE: son tipos de datos para objetos de gran tamaño que, respectivamente, están en formato binario, en formato carácter, con soporte de caracteres nacionales y en archivos con información en formato binario y que, por su tamaño, no se almacenan en tablas, sino en archivos administrados por el motor.

#### 4.8.2 Tipos de datos que son propios del lenguaje PL/SQL

Existen otros tipos de datos no diseñados para almacenar datos en una tabla, sino para manipulación de datos dentro de los bloques PL/SQL. Se clasifican de la siguiente manera:

DEC, DECIMAL, REAL, DOUBLE\_PRECISION, INTEGER, INT, SMALLINT, NATURAL, POSITIVE, NUMERIC: son un subconjunto del tipo NUMBER que se usan para declaración de variables.

BINARY\_INTEGER, PLS\_INTEGER: estos tipos de datos almacenan enteros y las variables definidas así se deben convertir explícitamente a un formato de base de datos para que se almacenen en una columna.

BOOLEAN: almacena un valor TRUE, FALSE o NULL.

TABLE, RECORD: los tipos de datos TABLE pueden almacenar algo similar a un arreglo, mientras un RECORD puede almacenar variables con tipos de datos compuestos.

En general, las variables que almacenarán datos para luego insertarlos en columnas de tablas de bases de datos, se definen con el mismo tipo de datos que las columnas destino y, para no tener que buscar qué tipo de dato tiene cada columna que se afectará, cuando se definen las variables se puede usar el modificador %TYPE en la declaración de la variable; por ejemplo:

```
DECLARE  
v_legajo          estudiantes.legajo%TYPE;
```

De esta manera, la variable v\_legajo tendrá el tipo de datos declarado en la tabla “Estudiantes para legajo”. Esto ahorra esfuerzo y disminuye la probabilidad de errores en el tiempo de ejecución, con una pequeña sobrecarga en el momento de la compilación.

Existe algo similar con el modificador %ROWTYPE, que permite al desarrollador crear un tipo de dato compuesto con todas las columnas de la tabla referenciada: por ejemplo, con la tabla “Estudiantes”:

```
DECLARE  
r_estudiante      estudiantes%ROWTYPE;
```

Con el ejemplo anterior, se habrá creado un RECORD con las columnas y con sus respectivos tipos de datos idénticos a las columnas de la tabla “Estudiantes”.

En PL/SQL, también se deben declarar las constantes, según se ve en el ejemplo:

```
DECLARE  
pi           CONSTANT NUMBER(15,14) := 3,14159265358;
```

En el ejemplo, además de declarar como constante al identificador pi, se ha inicializado su valor con el símbolo de asignación ‘:=’ y el valor exacto, que ya no podrá modificarse durante la ejecución del programa. En el caso de las variables, se pueden usar varios modos de asignar variables, además de ‘:=’, una función, cuya cláusula RETURN, le asignará el resultado a la variable, como parámetro de salida de un procedimiento y con la cláusula INTO en el SELECT, como se exemplificará más adelante.

## 4.9 Interacción con la base de datos Oracle

Se verá cómo se usan las sentencias SELECT, INSERT, UPDATE y DELETE en los bloques de programación, cómo se utilizan los cursores implícitos y cómo se aplica el procesamiento de transacciones en PL/SQL.

Para la interacción de los bloques de programación en PL/SQL, no es necesaria ninguna interfaz, como ODBC, con lo que se gana en simpleza y en *performance*.

La sentencia SELECT agrega una cláusula INTO para que pueda definirse a qué variable asignar cada columna de la lista del SELECT, como se verá en el siguiente ejemplo de código. Cabe mencionar que se ha usado un RECORD a imagen y semejanza de la tabla “Estudiantes”, para almacenar una fila que tenga como legajo el número 7547. Este conjunto de valores se puede referenciar completamente o por sus partes, con la notación “record.columna”, de la misma manera que en el INSERT Se asigna un legajo cualquiera a la variable del registro, pero cambiándole el contenido original por el valor 7544 y, luego, se lo usará para elegir qué fila se borrará en la última sentencia DELETE.

```

DECLARE
    r_estudiante      estudiantes%ROWTYPE;
    v_apellido        VARCHAR2(30) := 'Perez';
    v_nombre          VARCHAR2(30) := 'Juan';
BEGIN
    SELECT *
    INTO r_estudiante
    FROM estudiantes
    WHERE legajo = 7547;

    UPDATE estudiantes
    SET nombre = nombre||', '||v_nombre
    WHERE legajo = r_estudiante.legajo;

    INSERT INTO estudiantes (legajo, apellido, nombre, tipo_documento, documento,
                           id_sexo, calle, calle_nro, id_barrio)
    VALUES (r_estudiante.legajo, r_estudiante.apellido, r_estudiante.nombre, r_estudiante.

```

```
tipo_documento, r_estudiante.documento, r_estudiante.id_sexo, r_estudiante.calle,
r_estudiante.calle_nro, r_estudiante.id_barrio);

r_estudiante.legajo := 7544;

DELETE FROM estudiantes
WHERE legajo = r_estudiante.legajo;
END;
```

## 4.10 El tratamiento de transacciones en PL/SQL

Las mismas opciones del proceso de transacciones de SQL están disponibles en PL/SQL. El inicio de la transacción lo marca la primera sentencia de modificación de datos y deben ser explícitamente definidos los puntos de confirmación COMMIT o de corrección ROLLBACK y los eventuales puntos de salvaguarda SAVEPOINT, para definir apropiadamente la transacción.

En un programa, la primera sentencia SQL inicia la transacción y, cuando ésta finaliza, la próxima comienza otra automáticamente. Por esta razón, cada sentencia SQL es parte de una transacción. El uso de COMMIT y ROLLBACK asegura que los cambios hechos con sentencias SQL se confirmen o se deshagan en conjuntos consistentes simultáneamente. Con SAVEPOINT se pone el nombre y se marcan las posiciones de las distintas sentencias de una transacción, para poder deshacer sus partes en el caso en que fuera necesario por alguna regla de negocio.

Pero, ¿qué sucede cuando se realizan transacciones desde distintas sesiones de usuarios en forma concurrente? En esta situación, el motor de base de datos debe garantizar que no haya interferencias o anomalías con los cambios que se están llevando a cabo y, además, que las sesiones que están haciendo lecturas de las tablas modificadas no puedan leer datos que todavía no han sido confirmados. Esta anomalía se denomina lecturas sucias.

Para tener una primera idea de los niveles de aislamiento, es preciso indicar que se aplica el principio de que los lectores no bloquean a los escritores, y viceversa.

Para esto, el estándar SQL define niveles de aislamiento que previenen ésta y otras anomalías, como la modificación de datos leídos y la grabación destructiva de cambios que no estén correctamente aislados. Para una mejor compresión, podríamos describir estas situaciones de la siguiente manera: dadas dos transacciones T1 y T2 tendrían un conflicto de escritura-lectura, si la segunda leyera un cambio sin confirmación de la primera (denominada, en el párrafo anterior, lectura sucia o, también, lectura desfasada). Si T2 modifica el valor de una fila mientras T1 ha leído el valor anterior a este cambio, se produce una anomalía de lecto-escritura conocida como lectura no repetible. Por último, se produce un conflicto de escritura-escritura si T2 modifica un valor que ha sido alterado por T1 en una transacción que no ha finalizado aún. Por esta razón, a esta anomalía se la denomina actualización perdida, ya que no se llega a confirmar el cambio de T1.



En un programa, la primera sentencia SQL inicia la transacción y, cuando ésta finaliza, la próxima comienza otra automáticamente.

El estado por defecto de todas las transacciones en Oracle garantiza la consistencia de lectura en el nivel de Sentencia, que es el que asegura que en una consulta verá solamente los cambios confirmados por otras sesiones antes de que la sentencia SELECT inicie su procesamiento, más los cambios que se hubieran realizado antes en la misma transacción.

Se podrá usar la sentencia SET TRANSACTION para establecer una transacción solo lectura (*read-only transaction*), que provee consistencia de lectura en el nivel de sentencia, de tal modo que se garantiza que una consulta leerá solamente los cambios confirmados antes que la sentencia SELECT, dentro de la transacción actual, haya iniciado. Esta sentencia no tiene argumentos y debe ser la primera sentencia SQL en la transacción de solo lectura, como se expresa en el ejemplo siguiente:

```
SET TRANSACTION READ ONLY;
```

Además de esta sentencia, es posible determinar cerramientos o bloqueos de las tablas que se tratarán en una transacción, asignando diferentes niveles de aislamiento con la sentencia LOCK TABLE y la indicación de los modos disponibles según el siguiente ejemplo:

```
LOCK TABLE mitabla IN <modo> MODE [NOWAIT];
```

Los distintos modos se detallan a continuación:

- EXCLUSIVE permite consultas en la tabla, pero prohíbe cualquier otra actividad sobre ella.
- SHARE permite acceso concurrente a la tabla afectada, pero prohíbe cualquier actualización sobre ella.
- ROW SHARE permite acceso concurrente a la tabla afectada, pero impide que otro usuario establezca un LOCK EXCLUSIVE sobre ella.
- ROW EXCLUSIVE es un bloqueo similar al anterior pero, además, impide que otra sesión fije un modo SHARE sobre el objeto. Es el modo que se fija automáticamente cuando se realiza un UPDATE, INSERT o DELETE sobre una tabla.
- SHARE ROW EXCLUSIVE bloquea toda la tabla y permite leer las filas, pero impide las actualizaciones y que otra sesión establezca bloqueos del tipo SHARE.
- SHARE UPDATE —sinónimo de ROW SHARE— ha sido incluido para mantener la compatibilidad con versiones anteriores del motor Oracle.

Es posible que algunos de estos modos de bloqueos se establezcan en la misma tabla simultáneamente, como los SHARE y otros. Al igual que los EXCLUSIVOS, se pueden aplicar solamente una vez en una tabla.

Finalmente, el parámetro NOWAIT indica que el motor debe retornar el control de la sesión que intenta hacer alguna operación o establecer un LOCK sobre una tabla que ya posee un bloqueo de otra sesión. Si no se incluye este parámetro, la sesión esperará hasta que se hayan liberado los bloqueos de la otra sesión.

## 4.11 Sentencias de control de flujo

Su característica procedural obliga a PL/SQL a tener sentencias de control de flujo, que ya hemos mencionado anteriormente, y dos categorías de sentencias, como las expresiones condicionales y el loop. A continuación, se verán algunos detalles de estas sentencias para controlar la ejecución de un bloque PL/SQL.

Una condición en un programa equivale a la idea de tomar una decisión y, detrás de esta condición, subyace el concepto de valores booleanos de verdadero o falso. Estos valores son el resultado de la ejecución de un tipo de sentencias, denominadas operaciones de comparación. Algunas comparaciones pueden ser como las siguientes:

- $3 + 5 = 8$
- Mi mano tiene 16 dedos.
- $4 = 10$
- Hoy es jueves.

Estas operaciones de comparación se pueden evaluar por su validez, ya que su valor puede resultar verdadero o falso. En el primer caso, es cierto que  $3+5$  es igual a 8, por lo que esta comparación devolverá el valor TRUE. La segunda no es cierta, ya que mi mano tiene 5 dedos y devolvería un valor FALSE. La tercera también devolverá un valor FALSE porque 4 no es igual a 10. Por último, la afirmación "Hoy es jueves", si se realiza una vez por día durante toda una semana, devolverá seis veces FALSE y una sola TRUE.

El mecanismo de proceso de sentencias condicionales permite estructurar sentencias que se ejecutarán, o no, de acuerdo con los resultados de la evaluación de las condiciones. La sintaxis para las sentencias condicionales es "IF (SI) la comparación es verdadera THEN (ENTONCES) haga lo siguiente". En PL/SQL, se ofrecen dos agregados:

- ELSE (SI NO), que permite decir "SI NO es verdadero, haga lo que sigue a esta cláusula ELSE".
- ELSIF, que permite evaluar por distintas condiciones definidas con cada una de sus líneas, que se analizan ordenadas y, si ninguna de ellas es verdadera, sale por el ELSE.

Se agrega una cláusula al IF-THEN: el ELSIF, que permite evaluar por distintas condiciones definidas con cada una de sus líneas, que se analizan ordenadas y, si ninguna de ellas es verdadera, sale por el ELSE.

Es importante aclarar, en este punto, que las comparaciones se harán entre datos del mismo tipo, y que la comparación con el valor null se realizará con la expresión IS null o por la negativa IS NOT null.

Se analizarán los ejemplos de sintaxis de las comparaciones:



Una condición en un programa equivale a tomar una decisión y además, subyace el concepto de valores booleanos de verdadero o falso.



El mecanismo de proceso de sentencias condicionales permite estructurar cuales se ejecutarán o no, de acuerdo con los resultados de la evaluación de las condiciones.

```
DECLARE
```

```
    lado_a  number(2) := 20;
    lado_b  number(2) := 30;
    lado_c      number(2) := 40;
```

```

        resultado      number(2) := 0;
BEGIN
    carga_de_datos(parametro_externo, lado_a,lado_b,lado_c);
    IF lado_a > lado_b THEN
        Calcular _ perimetro (lado_a, lado_b,lado_c, resultado);
    ELSIF lado_b > lado_c THEN
        Calcular _ perimetro2 (lado_a, lado_b,lado_c, resultado);
    ELSE
        Resultado=350;
    END IF;
END;

```

El ejemplo crea las variables y las inicializa. Cuando inicia el bloque, hay un procedimiento que recibe una variable ya definida en el entorno de ejecución de llamada que cambiará el valor de las tres variables locales. La sentencia condicional evaluará las variables lado\_a, lado\_b, lado\_c y, según su valor, calculará el perímetro y lo asignará a la variable resultado, inicializada en 0. En caso de que lado\_a sea mayor a lado\_b, se calculará el perímetro con el procedimiento y, si lado\_a es menor a lado\_b, pero lado\_b es mayor a lado\_c, se calculará con otro procedimiento diferente y, si las dos condiciones anteriores devuelven valores FALSE, el bloque asignará al resultado un valor fijo de 350.

#### 4.11.1 Usando loops

Otra situación que surge en programación es cuando es necesario ejecutar un conjunto de sentencias repetidamente. Estas repeticiones se pueden controlar de dos maneras: la primera, es repetir el código una determinada cantidad de veces y, la segunda, es repetir el código según se cumpla o no una condición, en función de la sentencia de bucle que se aplique y cómo se escriba la condición lógica. Hay tres tipos de repetidores o bucles, también denominados loops por la sentencia asociada, por ejemplo:

- LOOP-EXIT o bucle simple: repite las sentencias hasta que procesa la sentencia de salida EXIT.
- WHILE-LOOP: entra y se repite según se cumpla la condición que se incluye en su inicio.
- FOR-LOOP: repite la cantidad de veces indicada en el bucle.

Se analizarán, a continuación, ejemplos de estas sentencias:

La sentencia LOOP-EXIT está diseñada para determinar si la condición dentro del loop tiene el valor para terminar y procesar la sentencia EXIT, que se debe escribir dentro del grupo de sentencias del loop. Se escribirá esta sentencia para evitar que el ciclo se repita indefinidamente, y se asegurará que la condición que permitirá ejecutarlo también se cumpla cuando es deseado que el loop termine. En el ejemplo que sigue, si lado\_b nunca alcanza el valor 25, no se detendrá el loop, habrá que cancelar desde afuera la ejecución del bloque.



La sentencia LOOP-EXIT está diseñada para determinar si la condición dentro del loop tiene el valor para terminar y procesar la sentencia EXIT, que se debe escribir dentro del grupo de sentencias del loop.

```

DECLARE
    lado_a  number(2) := 20;
    lado_b  number(2) := 30;
    lado_c      number(2) := 40;
    resultado   number(2) := 0;
BEGIN
    LOOP
        Preguntar_datos(parametro_externo, lado_a,lado_b,lado_c);
        Calcular _ perimetro (lado_a, lado_b,lado_c, resultado);
        IF  lado_b =25 THEN
            EXIT;
        END IF;
        END LOOP;
    END;

```

Existe una variante en la sentencia EXIT que es la cláusula WHEN a continuación del EXIT en el que se evalúa la condición y se evita que se construya el IF-THEN, para decidir cuándo se ejecuta la salida. Un ejemplo:

```

DECLARE
    lado_a  number(2) := 20;
    lado_b  number(2) := 30;
    lado_c      number(2) := 40;
    resultado   number(2) := 0;
BEGIN
    LOOP
        Preguntar_datos(parametro_externo, lado_a,lado_b,lado_c);
        Calcular _ perimetro (lado_a, lado_b,lado_c, resultado);
        EXIT WHEN lado_b =25;
    END LOOP;
END;

```

La sentencia WHILE-LOOP difiere del loop básico en que evalúa, primero, la condición para salir del bucle cuando devuelve FALSE. Tiene como ventaja que lo procesa siempre y que la evaluación no se puede eludir. Nótese que se debe invertir el sentido de la comparación, porque se desea que, cuando sea distinto a 25, se ejecute el conjunto de sentencias necesarias. Por ejemplo:

```

DECLARE
    lado_a  number(2) := 20;
    lado_b  number(2) := 30;
    lado_c      number(2) := 40;
    resultado   number(2) := 0;
BEGIN
    WHILE lado_b <>25;
    LOOP
        Preguntar_datos(parametro_externo, lado_a,lado_b,lado_c);
        Calcular _ perimetro (lado_a, lado_b,lado_c, resultado);
    END LOOP;
END;

```

La sentencia FOR-LOOP difiere del loop básico en que define, exactamente, la cantidad de veces que el código se ejecuta antes que se salga del mismo, ya que crea un contador y un rango entre el cual se valorará el mismo. Puede definirse que este contador incremente su valor o que lo vaya disminuyendo hasta encontrar el valor inferior del rango. Vemos el ejemplo:

```

DECLARE
    lado_a  number(2) := 20;
    lado_b  number(2) := 30;
    lado_c      number(2) := 40;
    resultado   number(2) := 0;
    conta       number(2) := 0;
BEGIN
    FOR conta IN 1 .. 25 LOOP
        Preguntar_datos(parametro_externo, lado_a,lado_b,lado_c);
        Calcular _ perimetro (lado_a, lado_b,lado_c, resultado);
    END LOOP;
END;

```

En esta sentencia, la variable conta inicializada en 0, cuando se procesa el FOR-LOOP, se lo valora en 1 y, por cada vuelta del LOOP, se lo incrementa de 1 en 1 hasta llegar al valor 25; en la próxima vuelta, cuando se lo valora en 26, se interrumpe la ejecución y continúa a partir de la sentencia END LOOP.

## 4.12 Manejo de cursos

Ya se definió qué es un cursor; ahora se verá que existen dos tipos, definidos explícitamente o implícitamente, que pueden tener parámetros y cómo se combinan

cursos y sentencias de control de *loops* y la sentencia que integra ambos elementos llamados LOOP FOR de cursos.

Se recordará que los cursos son una dirección de memoria en la que se almacenan los resultados de una sentencia SQL. Se utilizan, frecuentemente, en PL/SQL para que procese el conjunto de filas que devuelve un SELECT.

Los cursos explícitos tienen un nombre definido por el desarrollador y agregan un control mayor sobre la ejecución de la sentencia. Se verá, a continuación, una declaración de un cursor explícito:

```
DECLARE
    CURSOR cursor_estudiante      IS
        SELECT * FROM estudiantes;
BEGIN
    ...
END;
```

Cada vez que se ejecuta una sentencia SQL, se crea un cursor implícito. El desarrollador puede necesitar controlar la operación de una sentencia, aprovechando la definición que tiene PL/SQL asociada con estas direcciones de memoria. Estos atributos son: %notfound, que se valúa en TRUE cuando la sentencia SELECT no trae filas; el caso inverso, valúa en TRUE a %found y, cuando trae filas, la cantidad se asigna al atributo %rowcount; por último, define en TRUE al %isopen. A continuación, se verá un ejemplo de un atributo de un cursor implícito que corresponde al procesamiento del UPDATE: aquí, el desarrollador puede preguntar si el resultado de la búsqueda de las filas que se actualizarán —con el atributo %notfound (en este caso, como es implícito, se le antepone el prefijo SQL)— fue positivo o negativo. Si no hubiera filas con el legajo 7547 se insertará, con ese legajo, el apellido Pérez y el documento 16883022.

```
DECLARE
    mi_estudiante    estudiantes.legajo%TYPE := 7547;
    mi_apellido       estudiantes.apellido%TYPE := 'Perez';
    mi_documento     estudiantes.documento%TYPE := 16883022;
BEGIN
    UPDATE estudiante
    SET documento = mi_documento
    WHERE legajo = mi_estudiante;

    IF SQL%NOTFOUND THEN
        INSERT INTO estudiantes (legajo, apellido, documento)
        VALUES (mi_estudiante, mi_apellido, mi_documento);
    END IF;
END;
```

Para operar adecuadamente un cursor se necesitan tres pasos: abrir, obtener un valor y cerrarlo. Se usará una tabla denominada "Empleados" que cuenta con el número de empleado, el apellido y el sueldo.

```

DECLARE
    mayor_porc_aumento    constant number (10,5) :=1.2;
    medio_porc_aumento   constant number (10,5) := 1.1;
    menor_porc_aumento   constant number (10,5) :=1.05;
    TYPE t_emp IS RECORD (
        t_sueldo empleado.sueldo%TYPE,
        t_nro_emp     empleado.nroempleado%TYPE);
        r_empleado      t_emp;
CURSOR c_empleado IS
    SELECT nroempleado, sueldo
    FROM empleados;

BEGIN
    OPEN c_empleado;
    LOOP
        FETCH c_empleado INTO r_empleado;
        EXIT WHEN c_empleado%NOTFOUND;
        IF r_empleado. t_nro_emp = 688
            OR r_empleado. t_nro_emp = 700 THEN
            UPDATE empleados
            SET sueldo = r_empleado.t_sueldo*mayor_porc_aumento
            WHERE nroempleado = r_empleado.t_nro_emp;
        ELSIF r_empleado.t_nro_emp = 777
            OR r_empleado.t_nro_emp = 788 THEN
            UPDATE empleados
            SET sueldo = r_empleado.t_sueldo*medio_porc_aumento
            WHERE nroempleado = r_empleado. t_nro_emp;
        ELSE
            UPDATE empleados
            SET sueldo = r_empleado.t_sueldo*menor_porc_aumento
            WHERE nroempleado = r_empleado. t_nro_emp;
        END IF;
    END LOOP;
    CLOSE c_empleado;
END;

```

En el ejemplo se ha realizado un acceso a la base buscando todos los empleados. Con las filas en memoria, en la dirección del cursor, el proceso abre el cursor, que consiste en ejecutar el SELECT asociado y se ubica en su primer registro. Con FETCH toma los valores de éste y se los asigna al RECORD r\_empleado. A medida que se vaya repitiendo la ejecución de este FETCH, se moverá al siguiente registro del cursor, correspondiente a la siguiente fila recogida por el SELECT, y los datos se cambian en r\_empleado. El loop tiene la condición de finalización o salida, que será cuando no encuentre filas usando el atributo del cursor definido (c\_empleados%NOTFOUND), que será porque el SELECT no trajo ninguna fila o ya se ha procesado la última. Es importante destacar que el uso de una definición de tipo de dato diseñado por el usuario, en la línea en la que TYPE indica que se deben precisar un tipo de dato compuesto, un vector con dos campos —t\_nro\_emp y t\_sueldo—, con la misma definición de las columnas relacionadas en la tabla “Empleados”.

En el momento de abrir un cursor, se pueden definir parámetros para que, al ejecutar el SELECT asociado, se determinen las condiciones por las que, este SELECT, filtrará las filas haciendo más chico el conjunto de filas para, de esta manera, mejorar la performance del proceso.

El lenguaje PL/SQL combina dos construcciones en el denominado LOOP FOR para cursos que abrevia todos los pasos: de apertura, de recorrido y de cierre. A continuación, se verá un bloque que hace lo mismo pero, al usar el LOOP FOR para cursos, con menos líneas de código.

```

DECLARE
    mayor_porc_aumento    constant number (10,5) :=1.2;
    medio_porc_aumento   constant number (10,5) := 1.1;
    menor_porc_aumento   constant number (10,5) :=1.05;
    TYPE t_emp IS RECORD (
        t_sueldo empleado.sueldo%TYPE,
        t_nro_emp      empleado.nroempleado%TYPE);
    r_empleado          t_emp;
    CURSOR c_empleado IS
        SELECT nroempleado, sueldo
        FROM empleados;
BEGIN
    FOR r_empleado IN c_empleado LOOP
        IF r_empleado. t_nro_emp = 688
            OR r_empleado. t_nro_emp = 700 THEN
            UPDATE empleados
            SET sueldo = r_empleado.t_sueldo*mayor_porc_aumento
            WHERE nroempleado = r_empleado.t_nro_emp;
        ELSIF r_empleado.t_nro_emp = 777
            OR r_empleado.t_nro_emp = 788 THEN
    
```

```

        UPDATE empleados
        SET sueldo = r_empleado.t_sueldo*medio_porc_aumento
        WHERE nroempleado = r_empleado. t_nro_emp;

    ELSE
        UPDATE empleados
        SET sueldo = r_empleado.t_sueldo*menor_porc_aumento
        WHERE nroempleado = r_empleado. t_nro_emp;
    END IF;
END LOOP;
END;

```

No es necesario escribir la apertura, asignar los valores al registro, la condición de cierre y el cierre propiamente dicho del cursor, ya que todo se procesa internamente por la sentencia FOR LOOP asociada con un cursor.

#### 4.13 Manejo de errores



Los tres tipos de excepciones son: las predefinidas, las definidas por el usuario y las internas.

En esta sección, se verán los tres tipos básicos de errores, las excepciones comunes y cómo codificar la sección de excepción. Esta posibilidad de manejo de errores es una de las mejores contribuciones a la robustez de las aplicaciones construidas en Oracle. Los errores necesitan ser tratados explícitamente, sin necesidad de usar IF para detectar la situación anómala. En PL/SQL, se usan los manejadores de excepciones que identifican el problema y le asignan una porción de código para resolver la situación o, al menos, para dejar registro de lo sucedido y para poder deshacer la transacción en curso.

Los tres tipos de excepciones son: las predefinidas, las definidas por el usuario y las internas. El manejo de excepciones ofrece ventajas en simplicidad y en flexibilidad. Las excepciones predefinidas brindan al desarrollador la posibilidad de revisar los problemas predefinidos; las excepciones definidas por el usuario permiten determinar situaciones que se tratarán como excepción y tratarlas de la misma manera. Las excepciones internas se desarrollarán en los siguientes ejemplos.

Las excepciones predefinidas se usan en conjunto con las situaciones predefinidas que pueden ocurrir en la base; por ejemplo, cuando una sentencia SELECT no devuelve filas —NO\_DATA\_FOUND—, se ejecuta automáticamente y no es necesario que se ejecute la sentencia RAISE. Cuando este tipo de excepción ocurre, si fuera imprescindible realizar algún cambio, se escribirá el código necesario en la sección de excepciones, en el manejador definido para esta excepción.

Las excepciones definidas por el usuario se pueden usar para aplicar situaciones previstas en las reglas de negocios, para que se traten de la misma manera que las otras excepciones. Éstas últimas no se consideran errores y se podrían escribir bloques que traten estas situaciones de manera consistente en toda la aplicación. Para utilizar estas definiciones, se cumplirán los siguientes pasos: la declaración de la excepción con un nombre con el que se la invocará durante el proceso cuando

esta situación se presente. Luego, se necesitará una prueba de aparición de la excepción, con un código específico que preguntará si las condiciones indican la aparición de esta situación y, además, ejecutará el comando RAISE para que la ejecución se dirija a la próxima etapa. El manejo de la excepción, en la sección EXCEPTION, con la cláusula WHEN y el nombre de la excepción con el que se creó y el bloque de código o la llamada a un procedimiento construido para que siempre se resuelva de la misma manera.

A continuación, un bloque con los dos tipos de excepciones:

```
DECLARE
    mi_estudiante      estudiantes.legajo%TYPE := 7547;
    r_estudiante        estudiantes%ROWTYPE;
    documento_null      EXCEPTION; /*declara la Excepción del usuario
*/
BEGIN
    SELECT *
    INTO r_estudiante
    FROM estudiantes
    WHERE legajo = mi_estudiante;
    IF r_estudiante.documento IS NULL THEN
        RAISE documento_null; /*Levanta la Excepción declarada*/
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN /*Excepción predefinida*/
        DBMS_OUTPUT.PUT_LINE('No hay flas');
    WHEN documento_null THEN /*Excepción definida por el usuario */
        DBMS_OUTPUT.PUT_LINE('La columna Sueldo es nula para el
estudiante: '||r.estudiante.documento);
END;
```

Cabe aclarar que se ha usado un procedimiento provisto por la base denominado PUT\_LINE, que se almacena en el paquete de la base DBMS\_OUTPUT, que muestra este mensaje en la consola de usuario o en el área de mensajes de los programas que llamarían a este bloque de datos.

Las excepciones internas asocian un nombre de excepción con un error de la base de datos. El usuario o desarrollador puede extender la lista de excepciones asociadas con errores de la base de datos con el uso de la palabra reservada pragma, seguido por exception\_init, que es una directiva al compilador que permite asociar un numero de error, entre el -20 000 y -21 000, que es el rango asignado internamente en la base de datos a los errores del usuario. En el siguiente uso, se observa cómo se utiliza esta sentencia, basado en el código anterior. Cabe destacar que, cuando se recurre a exception\_init, se logra que el motor devuelva el código asignado a la excepción al entorno anftrión de este bloque de programación.

```

DECLARE
    mi_estudiante      estudiantes.legajo%TYPE := 7547;
    r_estudiante       estudiantes%ROWTYPE;
    documento_null     EXCEPTION; /*declara la Excepción del usuario
*/
    PRAGMA EXCEPTION_INIT(documento_null, -20001) /*asigna el numero
de error a
la Excepción del usuario */

```

#### 4.13.1 Excepciones comunes

Existen numerosas excepciones que se pueden usar y que permiten manejar los errores o las situaciones inesperadas en los programas; algunas de las definidas son:

**too\_many\_rows:** cuando un SELECT, ubicado donde se espera solo una fila, devuelve más de una.

**no\_data\_found:** cuando un SELECT, un UPDATE o DELETE no encuentran filas en la búsqueda.

**invalid\_cursor:** ocurre cuando se quiere cerrar un cursor que no ha sido abierto.

**cursor\_already\_open:** ocurre cuando se intenta abrir un cursor que ya está abierto.

**dup\_val\_on\_index:** ocurre cuando se quiere insertar una fila con valor de clave primaria ya existente.

**zero\_divide:** ocurre cuando se intenta dividir un número por cero.

Las dos primeras son las excepciones más utilizadas; en el ejemplo anterior, se vio el empleo de este tipo de excepciones. Es importante recalcar que no hay que definirlas ni "levantarlas" con RAISE.

#### 4.13.2 Codificando en la sección de excepciones

Las excepciones aparecen y se deben tratar con código ejecutable que permita registrar y, si es posible, resolver la situación planteada. En la sección de excepciones, se codificará y se separará en párrafos que se ejecutarán cuando una excepción se detecte. Estos párrafos comienzan con la palabra WHEN y, a continuación, se coloca el nombre de la excepción. Cabe destacar que se pueden tratar excepciones que no tengan un párrafo WHEN asociado, esto se logra si se usa el nombre genérico de excepción, OTHERS. Por lo tanto, esta sería una sección de excepciones muy común:

```

EXCEPTION
    WHEN NO_DATA_FOUND THEN /*Excepción predefinida*/
        DBMS_OUTPUT.PUT_LINE('No hay filas');
    WHEN OTHERS THEN/*Código ejecutable ante cualquier otro tipo de
Excepción */
        DBMS_OUTPUT.PUT_LINE('Se ha presentado una excepcion');
END;

```

Cabe destacar que una vez ejecutada la sección dedicada a una excepción, el control del programa sale del bloque que lo contiene, esto quiere decir que, si se ingresa a una excepción, no se puede procesar el resto de las excepciones.

## 4.14 Construyendo procedimientos y funciones con PL/SQL

Se han visto hasta aquí, los elementos del lenguaje y las estructuras de programas o bloques de programación, que se denominan anónimos porque no tienen un nombre asignado; resta conocer cómo se construyen los procedimientos, las funciones, los paquetes y los *triggers*.

Estas construcciones —denominadas, genéricamente, programas almacenados— tienen, como los bloques anónimos, una sección de declaración, codificación y manejo de errores. Dos diferencias significativas con los bloques anónimos son: los programas almacenados tienen nombres que los bloques anónimos no poseen, esto permite que se los pueda invocar en cualquier línea de ejecución, simplemente indicando el nombre y los parámetros con que deben ingresarse. La segunda diferencia ha sido nombrada, los bloques anónimos no pueden usar parámetros y los programas almacenados sí lo permiten, lo que les da mayor flexibilidad.

A su vez, existen diferencias entre los procedimientos y las funciones que se tratarán a continuación. Un procedimiento puede aceptar ninguno, uno o más valores como parámetros, pero una función siempre debe devolver uno y nada más que un solo valor.

Antes de ver algunos ejemplos, unas palabras acerca de los aspectos de seguridad. Para crear los procedimientos y las funciones almacenadas, se requiere que el usuario o desarrollador tenga el permiso de creación, que se asigna como sigue:

```
GRANT CREATE PROCEDURE TO miusuario;
```

Y una vez desarrollado el código, cuando se lo crea, se define el esquema en el que se agrupa junto con otros objetos del mismo usuario. Para que otros usuarios puedan ejecutarlo, el dueño del procedimiento o función almacenada les debe dar el permiso de ejecución.

```
GRANT EXECUTE ON miprocedimiento TO miusuario;
```

Los bloques anónimos tienen la desventaja de que, una vez ejecutados, la base de datos no registra nada acerca de éstos; en cambio, cuando se usan los procedimientos almacenados, la base guarda la compilación y el código *parseado* —código verificado y validado por un analizador sintáctico—, lo que define una mayor rapidez en el procesamiento.

Un ejemplo de construcción de un procedimiento almacenado:

```
CREATE OR REPLACE PROCEDURE cambios_estudiantes
(p_legajo      IN NUMBER,
p_apellido     IN VARCHAR2,
```



Los bloques anónimos tienen la desventaja de que, una vez ejecutados, la base de datos no guarda registros en cambio, cuando se usan los procedimientos almacenados, guarda la compilación y el código parseado, para mayor rapidez en el procesamiento.

```

    p_nombre          IN VARCHAR2) AS
BEGIN
    UPDATE estudiantes
    SET nombre = p_nombre, apellido = p_apellido
    WHERE legajo = p_legajo;
EXCEPTION
    WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE('No existe el empleado con el legajo'|| p_legajo)
END;

```



El uso de parámetros permite una mayor flexibilidad y se declara junto con el nombre del programa o función.

El uso de parámetros es una de las características principales de estos programas almacenados, ya que permiten una mayor flexibilidad y se declaran junto con el nombre del programa o función. Esto los hace parte del nombre completo, por lo que luego se deben valorizar al ejecutarse desde otro bloque o desde la línea de comandos. A continuación, un ejemplo de cómo se debe ejecutar este procedimiento:

```

BEGIN
cambios_estudiantes(7547, Alvarez', 'Juan');
END;

```

El resultado es que al empleado con legajo 7547 se le habrá cambiado el nombre y el apellido por los valores que se ingresaron en la llamada del procedimiento cambios\_estudiantes. Cabe destacar que, en este caso, se usan solamente los parámetros de entrada, que podrán modificarse en otra versión para mostrar un ejemplo de parámetros de salida. A continuación:

```

CREATE OR REPLACE PROCEDURE cambios_estudiantes2
(p_legajo          IN NUMBER,
p_apellido         IN VARCHAR2,
p_nombre           IN VARCHAR2,
p_resultado        OUT NUMBER) AS
BEGIN
    UPDATE estudiantes
    SET nombre = p_nombre, apellido = p_apellido
    WHERE legajo = p_legajo;
    p_resultado := 1;
EXCEPTION
    WHEN no_data_found THEN

```

```

p_resultado := 0;
DBMS_OUTPUT.PUT_LINE('No existe el empleado con el legajo '|| p_legajo)
WHEN others THEN
p_resultado := -1;
END;

```

Este nuevo parámetro cambia el modo de ejecutarlo; ahora, el bloque que lo convocó debe recibir el resultado de la ejecución que serán tres valores: el 1 significa que el proceso ha sido exitoso, el 0 que el proceso fracasó porque no existía el empleado con el legajo ingresado y con -1 también fracasó, pero por otro tipo de problema. Se verá el código que llamará al procedimiento.

```

DECLARE
    s_resultado NUMBER := 0;
BEGIN
cambios_estudiantes2(7547, 'Alvarez', 'Juan', s_resultado);
    IF s_resultado = 1 THEN
        DBMS_OUTPUT.PUT_LINE('actualizado');
    ELSIF s_resultado = -1 THEN
        DBMS_OUTPUT.PUT_LINE('no existe el estudiante');
    ELSE
        DBMS_OUTPUT.PUT_LINE('problemas');
    END IF;
END;

```

Se finalizará este capítulo mostrando una función almacenada y la forma en que se puede ejecutar. Se preverá el ingreso de las dos primeras letras de las monedas y, luego, se buscará en la tabla "Cotizaciones", el registro que tiene previsto las combinaciones de monedas y el coeficiente que se aplicará en la conversión. Si todo resultara exitoso, se devolverá el valor de conversión del cambio entre monedas. Si se ingresara una combinación no prevista, la función saldrá por una excepción y retornará un valor de 0 de conversión y un mensaje.

```

CREATE OR REPLACE FUNCTION convertir_moneda
(cantidad           IN NUMBER(12,3),
moneda_original   IN VARCHAR2,
moneda_destino    IN VARCHAR2) RETURNS NUMBER(12,3)
IS
    conversion          NUMBER(12,3) := 0;
    coeficiente         NUMBER(6,3);
    datos_mal          EXCEPTION;

```

```

BEGIN                                /*inicio de sección ejecutable de bloque*/
    IF moneda_destino IS NULL OR moneda_original IS NULL THEN
        RAISE datos_mal;
    END IF;
    SELECT coeficiente_conversión
    INTO coeficiente
    FROM cotizaciones
    WHERE origen = moneda_original
    AND destino = moneda_destino;
    conversion = cantidad * coeficiente;
    RETURN conversion;
EXCEPTION                         /*inicio de la sección de manejo de excepciones */
    WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE('Combinación de monedas no prevista');
        RETURN conversion;
    WHEN datos_mal THEN
        DBMS_OUTPUT.PUT_LINE('No ingresa datos de moneda');
        RETURN conversion;
END;

```

A continuación, se ilustrará cómo se ejecuta esta función, a la que se le pedirá la conversión de 100 pesos a euros. El resultado sería la cantidad de euros que se comprarían con 100 pesos.

```

DECLARE
    resultado NUMBER(12,3) := 0;
BEGIN
    resultado := convertir_moneda(100,'PE', 'EU');
END;

```

Otra forma de usar la función dentro de un SELECT sobre una tabla de Oracle, que tiene una sola fila y que se adopta como comodín para superar la restricción de obligatoriedad de la cláusula FROM y de una tabla en ella es la siguiente:

```

DECLARE
    resultado NUMBER(12,3) := 0;
BEGIN
    SELECT convertir_moneda(100,'PE', 'EU')
    INTO resultado
    FROM dual;
END;

```

#### 4.14.1 Creando paquetes con PL/SQL

En las bases de datos Oracle, existe una construcción denominada paquete —en inglés, package— que ayuda a los desarrolladores a agrupar todos los bloques PL/SQL que pueden estar relacionados dentro de una aplicación o por las características de su funcionamiento. Además de que este agrupamiento permite tener ordenado todos los programas en uso, otorga ventajas en los tiempos de ejecución porque, cuando es convocado por primera vez algún componente del paquete, todo el paquete se sube a la memoria; incluso, puede mantenerse permanentemente allí, para mejorar la performance de los procesos.

En los paquetes se pueden almacenar programas, tipos definidos por el usuario, excepciones, variables y hasta definición de cursos y tablas temporarias. Responde a las características del concepto de encapsulamiento de la orientación a objetos y, para convocar a un programa, se usa la notación de esta modalidad; es decir, para llamar una construcción cualquiera que esté dentro de un paquete se utiliza la sintaxis nombre\_paquete.miprocedimiento.

Los paquetes tienen dos partes: la especificación y el cuerpo. Esta estructura proviene de lenguajes como ADA y C. Se podría afirmar que la especificación es una unidad de código PL/SQL que nombra los procedimientos, las funciones, las excepciones, los tipos definidos por el usuario, las variables y otras construcciones disponibles para uso público en ese paquete. Se podría pensar en esta declaración como un índice de contenido del paquete, sin sus detalles de construcción. El cuerpo del paquete contiene el código de todos los procedimientos y las funciones que se han nombrado en la especificación. Si la especificación tiene un nombre de procedimiento o de función y el código de este programa no se incluyera en el cuerpo, la compilación del cuerpo fallará con un error hasta que se escriba el código de este programa presente en la especificación. Dada esta condición, también se podría decir que puede haber construcciones dentro del cuerpo que no están declaradas en la especificación, y que a estas construcciones —que se denominan privadas— las utilizan otros programas que existen dentro del cuerpo del paquete. A continuación, veremos un ejemplo de un paquete que agrupe al código visto hasta ahora.

Inicio de la declaración del paquete útiles\_varios:

```
CREATE OR REPLACE PACKAGE utiles_varios IS
FUNCTION convertir_moneda
(cantidad          IN NUMBER(12,3),
moneda_original  IN VARCHAR2,
moneda_destino   IN VARCHAR2) RETURNS NUMBER(12,3);
/*----- s e p a r a c i o n d e p r o g r a m a s -----*/
PROCEDURE cambios_estudiantes2(p_legajo      IN NUMBER,
                                p_apellido     IN
VARCHAR2,
                                p_nombre       IN
VARCHAR2,
                                p_resultado    OUT NUMBER);
END PACKAGE utiles_varios;
/
```



En los paquetes se pueden almacenar programas, tipos definidos por el usuario, excepciones, variables, definición de cursos y tablas temporarias.



Los paquetes tienen dos partes: la especificación y el cuerpo. La especificación es una unidad de código PL/SQL que nombra los procedimientos, las funciones, las excepciones, los tipos definidos por el usuario, las variables y otras construcciones disponibles para uso público en ese paquete.

Inicio del cuerpo del paquete:

```

CREATE OR REPLACE PACKAGE BODY utiles_moneda IS
CREATE OR REPLACE FUNCTION convertir_moneda
(cantidad           IN NUMBER(12,3),
moneda_original   IN VARCHAR2,
moneda_destino    IN VARCHAR2) RETURNS NUMBER(12,3)
IS
    conversion          NUMBER(12,3) := 0;
    coeficiente        NUMBER(6,3);
    datos_mal          EXCEPTION;
BEGIN
    /*inicio de la sección ejecutable de un bloque*/
    IF moneda_destino IS NULLOR moneda_original IS NULLTHEN
        RAISE datos_mal;
    END IF;
    SELECT coeficiente_conversión
    INTO coeficiente
    FROM cotizaciones
    WHERE origen = moneda_original
    AND destino = moneda_destino;
    conversion = cantidad * coeficiente;
    RETURN conversion;
EXCEPTION
    /*inicio de la sección de manejo de excepciones */
    WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE('Combinación de monedas no prevista');
        RETURN conversion;
    WHEN datos_mal THEN
        DBMS_OUTPUT.PUT_LINE('No ingresa datos de moneda');
        RETURN conversion;
END FUNCTION convertir_moneda;
/*- - - -----separacion     de programas----- - - */
PROCEDURE cambios_estudiantes2(p_legajo           IN NUMBER,
                                p_apellido         IN VARCHAR2,
                                p_nombre          IN VARCHAR2,
                                p_resultado       OUT NUMBER) AS
BEGIN
    UPDATE estudiantes
    SET nombre = p_nombre, apellido = p_apellido

```

```
WHERE legajo = p_legajo;
p_resultado := 1;
EXCEPTION
  WHEN no_data_found THEN
    p_resultado := 0;
    DBMS_OUTPUT.PUT_LINE('No existe el empleado con el legajo '|| p_legajo)
  WHEN others THEN
    p_resultado := -1;
END PROCEDURE cambios_estudiantes2;

END PACKAGE útiles_varios;
/
```

De esta manera, se describió la construcción de programación de PL/SQL que permite trabajar mejor las unidades de programación individuales, agrupándolas, simplificando su administración y dándoles mejor performance.

El último punto de este capítulo trata sobre las unidades de programación PL/SQL que se asocian a las tablas o a los eventos de sistema y tienen la característica de que no se ejecutan por el motor, sin necesidad de que se las convoque explícitamente por la aplicación o por un usuario determinado. Se ejecutan si el evento para el que se definen acontece.

Su utilización se justifica por la dependencia que existe entre un objeto principal y otro derivado dentro de la aplicación y, cuando en aquél existe un cambio, el secundario, lo debe refejar y se requiere que, implícitamente, se ejecute el código necesario para mantener la integridad de la información. Por ejemplo, esta dependencia marca que la aplicación que se encarga de la gestión de envíos de los productos comercializados no puede procesar un registro hasta que la aplicación de cobranza no marque este pedido como pagado, y esto no será posible hasta que la aplicación de cobranzas no haya enviado la factura al cliente. Cada una de estas aplicaciones puede ir actualizando el valor de la columna "Estado" en una tabla; sin embargo, si es necesario registrar la cantidad de días que llevará a la factura transformarse en un remito, se requerirá un procesamiento especial. Si se decide que habrá una tabla de historia con el pedido, el estatus inicial y el final, junto con la fecha de cambio, un trigger o disparador de base de datos será muy útil para procesar este trabajo.

#### 4.14.2 Creando triggers con PL/SQL

Los triggers o disparadores de base de datos se almacenan como objetos compilados en la base de datos y el código fuente de su cuerpo se puede consultar en el diccionario de la base de datos que lo almacena.

Un trigger se ejecuta automáticamente cuando cierto tipo de sentencias se ejecutan. Si existe un trigger para la sentencia UPDATE sobre una tabla determinada, cuando se actualice una de sus filas, se ejecutará, sin necesidad de convocarlo por su nombre, en forma implícita.



Los triggers se almacenan como objetos compilados y el código fuente de su cuerpo se puede consultar en el diccionario de la base de datos que lo almacena.

El tipo básico de *trigger* de base de datos es el que se encuentra en el nivel de la sentencia. Éste se ejecutará cada vez que una sentencia que lo active sea ejecutada sobre la tabla a la que el *trigger* está asociado. Como ejemplo, se puede definir que se quiere monitorear la actividad de borrado de la tabla “Pagos de estudiantes” de tal forma que, cuando se borren cuotas de un estudiante, se guardará en otra tabla la información de la fecha de borrado y la identificación del usuario que realizó la operación. A continuación, se verá el código que realiza esta operación:

```
CREATE OR REPLACE TRIGGER bd_pagos_estudiantes
BEFORE delete ON pagos_estudiantes
BEGIN
    INSERT INTO auditoría_pagos_estud (usuario, fechaHoraCambio,
                                         motivo)
    VALUES (user, to_char(sysdate, 'dd/mm/yy hh:mi:ss'), 'borrado de fila de
            pagos_estudiantes');
END;
```

El nombre del *trigger*, elegido por el autor, tiene un prefijo ‘BD\_’, que indica que es antes del borrado (*before delete*) porque, al ejecutarse automáticamente, un error en su ejecución genera un mensaje que trae el nombre del *trigger* que ha fallado. La práctica común de nombres ayuda al mantenimiento de la aplicación. Este *trigger* se ejecutará cuando se ejecute un *DELETE* sobre la tabla “Pagos\_estudiantes” y guardará en la tabla de “Auditoría de pagos de estudiantes” la acción de borrado, fecha y autor de la acción.

El otro tipo de *trigger* permitirá tener la información de cada fila borrada en la acción anterior; se trata de los *triggers* a nivel de la fila. Para esto, la sintaxis de la creación permite la definición del valor de las columnas antes de cada cambio y el valor después del cambio. A continuación, se verá el ejemplo de un *trigger* después de la actualización:

```
CREATE OR REPLACE TRIGGER bd_pagos_estudiantes
AFTER update ON pagos_estudiantes
REFERENCING OLD AS old NEW AS new
FOR EACH ROW
BEGIN
    INSERT INTO h_pagos_estudiantes (usuario, fechaHoraCambio,
                                      fechaAnterior, importeAnterior, fechaPosterior, importePosterior)
    VALUES (user, to_char(sysdate, 'dd/mm/yy hh:mi:ss'), :OLD.fecha, :OLD.importe,
            :NEW.fecha, :NEW.importe);
END;
```

En el código anterior, se observa que la segunda línea especifica el momento de ejecución, que es después de ejecutada la sentencia UPDATE sobre la tabla "pagos\_estudiantes"; luego se determina el prefijo con el que se definirán los valores anteriores y posteriores a la acción de las columnas y su contenido. En este caso diferenciará el anterior del resultante por la operación UPDATE. A continuación, la línea que establecerá que este trigger se ejecutará una vez por cada fila actualizada. En este tipo de triggers, se debe considerar detenidamente la cantidad de proceso que se define, porque puede ser ejecutado miles de veces en una acción muy extendida por las filas de la tabla original.

Se pueden escribir doce triggers diferentes sobre una tabla, un grupo antes, otro después; uno en el nivel de la sentencia para cada acción DML (INSERT, UPDATE y DELETE). Es decir, seis diferentes y, luego, el mismo cálculo para los que se encuentran en el nivel de la fila, hacen los seis restantes. Esta cantidad hace nuevamente llamar la atención que un trigger puede disparar otros triggers, imaginemos, definida sobre la tabla "h\_pagos\_estudiantes", lo que constituiría un efecto llamado cascada de triggers. En sí esto no es un problema, pero debe ser diseñado adecuadamente, ya que podría darse que en esta cascada se afecte a la tabla que inicialmente disparó el evento.

Se finaliza, así, el estudio del lenguaje de extensión procedimental de SQL en las bases de datos Oracle, denominado PL/SQL, luego de mostrar las construcciones básicas, las anónimas y con nombre de procedimientos, las funciones, los paquetes y los triggers. Se anima al lector a que practique la creación de todos los objetos para que se asegure el entendimiento de los temas expuestos.

## 4.15 Resumen

Luego de lo expuesto en este capítulo podemos afirmar que el lenguaje procedimental como extensión de SQL engloba el control de flujo, el manejo de condiciones, las sentencias de señalización de condiciones, los cursor y las variables locales y la forma de asignar valores a variables y parámetros.

Por otro lado, este lenguaje formaliza el mantenimiento de las rutinas persistentes escritas en lenguaje de bases de datos, como los procedimientos registrados. Además, su rol principal consiste en fijar la lógica y las reglas del negocio de las aplicaciones y almacenarlas en un único lugar dentro de la base.

## 4.16 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) solo está disponible para docentes.

### 4.16.1 Mapa conceptual del capítulo

### 4.16.2 Autoevaluación

### 4.16.3 Presentaciones\*



# 5

## Bases de datos multidimensionales y tecnologías OLAP

### Contenido

|   |     |
|---|-----|
| 5.1 Introducción.....   | 162 |
| 5.2 Bases de datos multidimensionales.....                    | 162 |
| 5.3 Tecnologías OLAP.....                                     | 169 |
| 5.4 Integración entre bases de datos y herramientas OLAP..... | 173 |
| 5.5 Arquitecturas OLAP y OLTP.....                            | 174 |
| 5.6 OLAP: multidimensional contra relacional.....             | 175 |
| 5.7 Evaluación de servidores y herramientas OLAP.....         | 182 |
| 5.8 Desarrollo de aplicaciones OLAP.....                      | 184 |
| 5.9 Áreas de aplicación de las tecnologías OLAP.....          | 185 |
| 5.10 Ventajas y desventajas de OLAP.....                      | 186 |
| 5.11 Resumen.....   | 187 |
| 5.12 Contenido de la página Web de apoyo.....                 | 188 |

### Objetivos

- Introducir en el uso de las bases de datos multidimensionales.
- Aplicar tecnologías OLAP sobre bases de datos multidimensionales o relacionales.
- Diferenciar los modelos transaccionales (OLTP) y los modelos relacionales (OLAP).

## 5.1 Introducción

En este capítulo y en los subsiguientes, se hará hincapié en la gestión de los datos, pero a diferencia de los apartados precedentes —en los que se analizaba la información desde su procesamiento transaccional—, aquí se buscarán los parámetros que facilitarán la obtención de pronósticos de los entes que suelen interactuar con una organización.

Este proceso —denominado inteligencia del negocio (o, en inglés, Business Intelligence)— utiliza los datos internos y externos de una organización para —como se afirmó en el párrafo anterior— analizar los posibles escenarios y, de esta manera, proyectar futuras situaciones.

Las organizaciones —para garantizar su subsistencia— necesitan ciertos parámetros de eficiencia que les permitan, en primer lugar, la maximización de su inversión y, en segundo lugar, su subsistencia en el tiempo mediante una correcta planificación de su futuro.

Por esta razón, establecen dos modelos que se ejecutan en forma simultánea y que permiten, por un lado, el procesamiento de la operación diaria de la organización (On-line Transaccional Processing u OLTP) y, por otro, el procesamiento analítico de las situaciones pasadas que faciliten la planificación de su futuro (On-line Analytical Processing u OLAP).

El modelo OLTP que, como se dijo, se ocupa del trabajo diario de la empresa, analiza —entre otras cuestiones— las ventas y las compras, los pagos a los proveedores, etc. En cambio, el modelo OLAP que, como se mencionó, considera lo ya acontecido, utiliza un sistema unificado de análisis que resume e integra o distribuye la información por las diferentes áreas de una organización.

En una economía globalizada, las empresas necesitan el establecimiento de ciertos parámetros de calidad que contribuyan con la diferenciación de sus productos en el mercado y les permita, además, su expansión hacia nuevos escenarios. Por esta razón, es fundamental —si se quiere buscar nuevas oportunidades de mercados y nichos específicos— el análisis exhaustivo de la información.

## 5.2 Bases de datos multidimensionales

### 5.2.1 Evolución de las bases de datos

El Dr. Codd, creador del modelo relacional —Relational Data Base Management Systems (RDBMS) o bases de datos relacionales—, posicionó su descubrimiento como la solución adecuada para el almacenamiento y la manipulación de datos. Sin embargo, se dio cuenta de que los RDBMS, si bien servían para almacenar la información, no satisfacían plenamente las necesidades de los negocios, porque el análisis de los datos requería de movimientos electrónicos y de procedimientos muy complejos. Para solucionar las limitantes de este modelo, Codd diseñó el denominado modelo multidimensional.

El modelo multidimensional (MMD) utiliza —a diferencia de los RDBMS, que estaban limitados a dos dimensiones— bases de datos más grandes que permiten al mercado decidir el futuro de sus negocios de manera rápida y eficiente.

  
La inteligencia del negocio utiliza los datos internos y externos de una organización para analizar los posibles escenarios y, de esta manera, proyectar futuras situaciones.

  
El modelo OLTP se ocupa del trabajo diario de la empresa, analiza las ventas y las compras, los pagos a los proveedores, etc. El modelo OLAP utiliza un sistema unificado de análisis que resume, integra y, también, distribuye la información por las diferentes áreas de una organización.

  
El modelo multidimensional utiliza bases de datos más grandes que permiten al mercado decidir el futuro de sus negocios de manera rápida y eficiente.

En la década del setenta, Ken Iverson creó el APL, A Program Language, que fue la primera herramienta de aplicaciones OLAP. Esta herramienta —implementada por IBM— tenía, gracias a la utilización de datos multidimensionales, aplicaciones empresariales con funcionalidades muy similares a las de los productos OLAP de la actualidad. Sin embargo, por su alto grado de complejidad —aun para programadores expertos— no logró un alto grado de adhesión.

En los años noventa, algunas empresas de software investigaron —sobre la base de las reglas establecidas por Codd— la implementación de un método (o arquitectura) que unificara la tecnología multidimensional con las bases de datos relacionales. De esta manera, se llegó a que las herramientas de apoyo que se utilizaban para la toma de decisiones tuvieran una nueva definición de requerimientos que debían incluir el uso de datos relativos y la tecnología multidimensional.

En los últimos años, la demanda del mercado se orientó hacia el uso de aplicaciones multidimensionales de mayor potencia que ampliaran las posibilidades a la hora de trabajar con bases de datos multidimensionales. Para suplir las falencias de las bases de datos, nacieron las herramientas OLAP relacionales. Éstas, además de incluir la visión multinivel, en algunos casos, utilizan hojas de cálculo y almacenan los datos en un RDBMS.

En la actualidad, ciertas herramientas OLAP permiten, para su procesamiento, que se almacenen en las computadoras personales pequeños cubos generados a partir de grandes bases de datos. Como esto ha resultado de mucha utilidad, las empresas distribuidoras ofrecen la herramienta de consultas relacionales y las herramientas de análisis multidimensionales.

Las bases de datos multidimensionales (MDB), al ofrecer una visión multidimensional de los datos en un concepto de dimensión, permiten la observación de cada uno de los patrones de interés que se evaluarán como una dimensión determinada. De esta manera, no se necesita del concepto de relaciones para la integración y unificación de datos y, además, se evitan las filas y columnas.

La Fig. 5.1, que se muestra a continuación, describe cómo se distribuyen los patrones de interés sin la utilización de filas o de columnas.

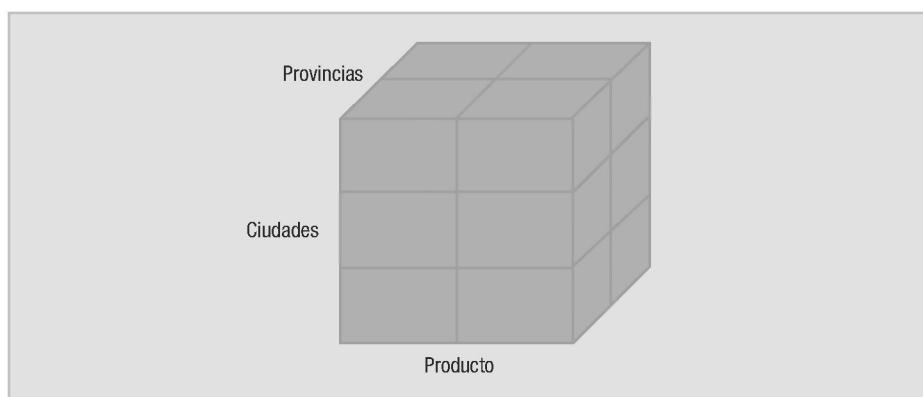


Fig. 5.1 Cubo dimensional.

En los últimos años, la demanda del mercado se orientó hacia el uso de aplicaciones multidimensionales. Para suplir las falencias de las bases de datos, nacieron las herramientas OLAP relacionales que, además de incluir la visión multinivel, utilizan hojas de cálculo y almacenan los datos en un RDBMS.

Kenneth Iverson (17 de diciembre de 1920-19 de octubre de 2004). Científico informático y matemático, se destacó por el desarrollo del lenguaje de programación de APL.

En la figura precedente, la intersección entre las dimensiones genera un valor almacenado que sirve para la conjunción de los dos parámetros representados por cada una de las dimensiones. Entonces, si una dimensión fuese "Productos" y, otra, "Clientes", en la intersección se almacenaría la cantidad de cada producto adquirida por cada cliente.

La base de datos multidimensional contribuye a que el usuario obtenga una visión analítica de la información. Esto le facilitará su procesamiento ya que no deberá recurrir a preguntas complejas y verá con mayor claridad las relaciones entre las diferentes tablas.

Por otra parte, las mayores dudas en la implementación de estas bases de datos surgen en la administración, su almacenamiento y en su velocidad de respuesta, temas que se tratarán en este capítulo.



En las bases de datos multidimensionales, la información se almacena en forma dimensional y no relacional. Las dimensiones determinan la estructura de la información almacenada y definen caminos de consolidación.

### 5.2.2 Concepto

En las bases de datos multidimensionales, la información se almacena en forma dimensional y no relacional. Las dimensiones determinan la estructura de la información almacenada y definen adicionalmente caminos de consolidación.

La información que se reúne se muestra como variables que, a la vez, se determinan por una o más dimensiones. De esta manera, y a partir de la intersección de esas dimensiones, se almacena el valor.

En la Fig. 5.2, se ilustra lo antedicho con la representación de un cubo multidimensional:

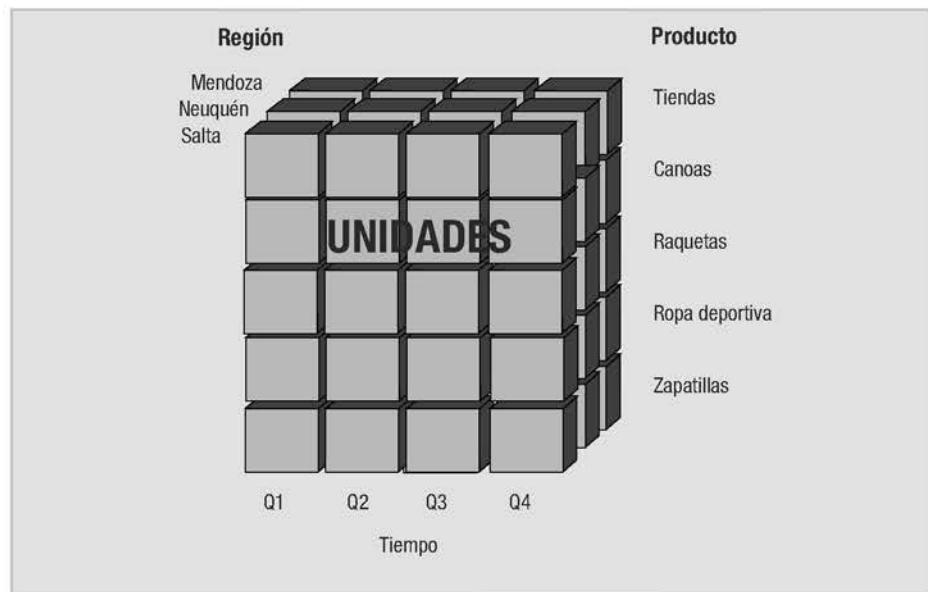


Fig. 5.2 Cubo multidimensional.

Como se observa en la figura precedente, el cubo multidimensional posee tres dimensiones: tiempo, productos y regiones. La información de una variable se puede analizar dentro del cubo que se forma en la intersección de sus dimensiones. La información que se obtiene en estas bases es muy útil para las empresas, dado que pueden determinar su desempeño.

La figura a representa la división del cubo en un plano vertical. En este caso, un gerente de área puede medir el desempeño de la región que le corresponde.

La figura b, por ejemplo, sería muy útil para un gerente de producto puesto que podría analizar en el plano horizontal el desempeño del bien o servicio que ofrece.

En la figura c, se comparan dos años diferentes. Esta visión temporal permite medir, por ejemplo, las ventas de dos períodos consecutivos.

En la figura d, se observa lo ocurrido en un periodo de tiempo, una región y un producto determinado, el comportamiento del mismo.



La información de una variable se puede analizar dentro del cubo que se forma en la intersección de sus dimensiones. La información que se obtiene en estas bases es muy útil para las empresas, dado que pueden determinar su desempeño.

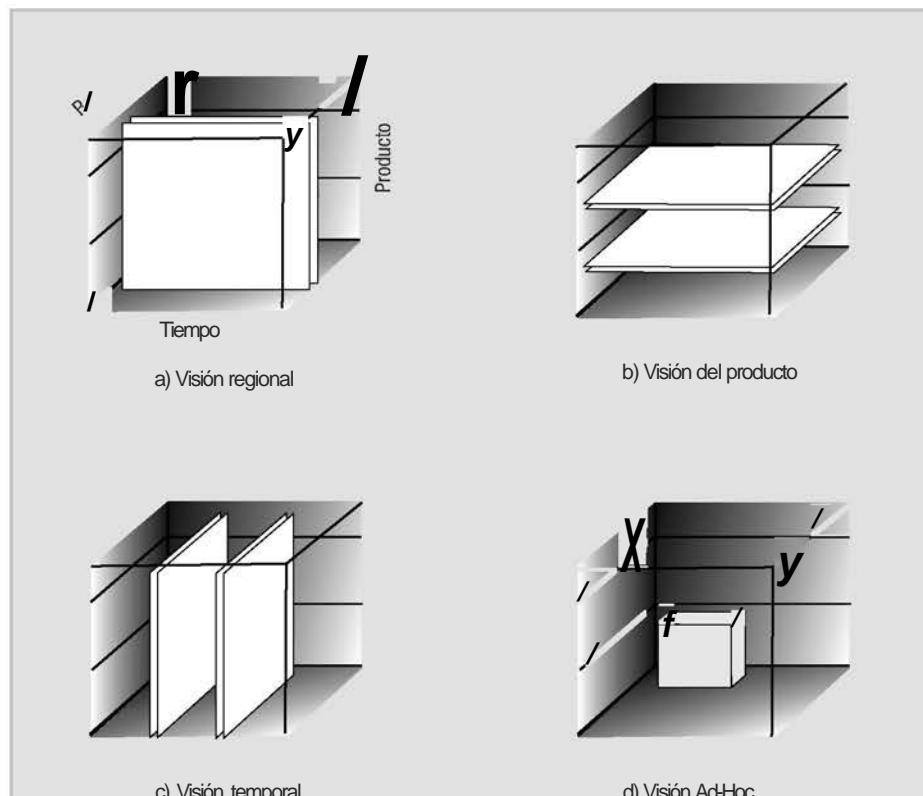


Fig. 5.3 Vistas multidimensionales.

Estas dimensiones se pueden estructurar jerárquicamente para construir caminos de consolidación que analicen la información —desde lo más general hasta lo más específico—mediante un mecanismo denominado drill-down.

### 5.2.3 Estructura de almacenamiento

Cuantas más dimensiones posea una base de datos multidimensional, mayor será el número de datos o celdas. Por esta razón, es necesario que se consideren todas las celdas que se generarán al establecer el producto cartesiano de las intersecciones que se formarán entre todas las dimensiones de la base.

En el ejemplo anterior, que tenía tres dimensiones, si se le agregara una cuarta, ampliaría notablemente la estructura de almacenamiento, puesto que la cantidad de celdas que se deberían añadir en la estructura sería igual al tamaño de la nueva dimensión por el tamaño de las tres dimensiones que ya existían.

Se podría dar el caso en que no todas las sucursales estarían habilitadas para vender la misma cantidad de productos y que las más pequeñas solo ofrecieran el 20%. En este caso, si tuviera las mismas dimensiones que en el ejemplo citado, es decir, regiones, producto y tiempo, quedaría el 80% de las celdas vacías para las dos dimensiones restantes.

Esta situación es muy común en la práctica, puesto que la gran parte de las bases de datos tienen el 95% de las celdas vacías; esto significa que los valores son nulos. En la literatura de la materia, se denominan poblados dispersos (*sparsely populated*) o dispersión de datos.

Este fenómeno de explosión es muy común en las bases de datos multidimensionales. Esta situación puede causar errores en muchas de las aplicaciones; por esta razón, es necesario que los que trabajen con este tipo de aplicaciones o con bases de datos multidimensionales tomen ciertas precauciones que prevean esta posibilidad y su posible solución.

Entonces, es necesario que se consideren e identifiquen las causas de explosión de las bases de datos:

Los datos dispersos: si bien una mala redistribución o eliminación de datos dispersos provoca que se ocupe una mayor proporción del disco duro, la diferencia entre un buen o mal almacenamiento de datos dispersos es uno de los factores que no contribuye en gran medida a la dispersión de datos dimensional.

- Almacenamiento de la base de datos multidimensional: la tecnología utilizada no produce dispersión de datos.
- Errores en el software: La explosión de los datos no tiene nada que ver con errores en el software o con bases de datos corruptas.

Aunque en muchas bases de datos multidimensionales y en algunos productos OLAP existe la posibilidad de comprimir datos, esta característica no soluciona su explosión y tampoco la dispersión multidimensional. La manera más sencilla de almacenamiento de datos dispersos es la utilización de celdas cuyos datos contengan alguna forma preordenada o de indexación. Sin embargo, esto trae aparejado sus propios inconvenientes puesto que se podría dar el caso en que el índice y las claves ocuparan más espacio que los propios datos. Entonces, lo que se debe considerar es cuál de las dos situaciones acarrea un problema mayor: si la dispersión natural de datos en las dimensionales o las medidas que se toman para evitarla.

En general, los grandes productos dividen los datos en pequeños grupos, denominados objetos multidimensionales. En algunos casos, ciertos fabricantes de bases de datos multidimensionales presentan los datos al usuario en forma de hipercubo. De esta manera, los datos en la aplicación aparecen como una sencilla estructura multidimensional. En otros, los fabricantes hacen explícitamente una aproximación denominada multicubo, en la que la base de datos multidimensional se estructura en un gran número de objetos separados normalmente con diferentes dimensiones y que unidos constituyen la base de datos en su totalidad.

A continuación, se analizará el funcionamiento de cada una de estas técnicas de almacenamiento.

### 5.2.4 Dispersión de datos

Para evitar la dispersión de datos y agruparlos, los diseñadores de productos multidimensionales se valen de diferentes estrategias entre las que se destacan las ya mencionadas hipercubos y multicubos. Estas dos opciones no son visuales; esto significa que el usuario no las percibe porque pertenecen a la capa interna de almacenamiento que utiliza el motor de la base de datos para almacenarlos. Esto condiciona el modo en que se procesarán los datos y los cálculos que se realizarán.

#### 5.2.4.1 Hipercubos

Algunos productos del mercado ofrecen un único cubo como estructura de almacenamiento de datos con modelos más sofisticados de compresión de datos dispersos. Este tipo de hipercubos permite que se introduzcan los valores de los datos a través de la combinación de dimensiones. Cabe destacar que todas las partes del espacio de datos (cubo) poseen la misma dimensión.

Esta estructura de datos —denominada hipercubo— no limita el número de dimensiones a un determinado valor y tampoco considera que éstas sean del mismo tamaño. El hipercubo se utiliza de forma específica para la identificación de estructuras con más de tres dimensiones. De todas maneras, no se refiere a un formato de almacenamiento de datos y se aplica en bases de datos multidimensionales y también en relacionales.

El trabajo con este tipo de estructura se realiza con un único cubo de varias dimensiones. Esta estructura pasa a ser estática que, en principio, produce un nivel de almacenamiento mayor y mejora la posibilidad de que se generen dispersiones de datos con valores nulos.

Los que se inclinan por este tipo de estructura resaltan la simplicidad que posee para el usuario final y su velocidad de acceso a la información ya que ésta se encuentra almacenada en forma contigua.



La manera más sencilla de almacenamiento de datos dispersos es la utilización de celdas cuyos datos contengan alguna forma preordenada o de indexación.



El hipercubo se utiliza de forma específica para la identificación de estructuras con más de tres dimensiones.

Arbor, con Essbase, y Cognos, con Powerplay, son dos de las compañías que utilizan esta estructura en sus aplicaciones.

Por último, es relevante comentar que hay una variante de esta estructura de hipercubo denominado hipercubo bordeado (fringed hypercube), que es más denso y con un pequeño número de dimensiones a las que se le pueden añadir, además, dimensiones de análisis.



La estructura de los multicubos divide el universo en diferentes cubos de menor tamaño e intenta dinamizarla mediante punteros. Así, disminuye el espacio de almacenamiento y la posibilidad de dispersión. Esta situación permite que los cubos no deban replicar el tamaño de una de las dimensiones a las restantes.

#### 5.2.4.2 Multicubos

Los productos que utilizan esta estructura dividen la aplicación de la base de datos en un conjunto de estructuras multidimensionales. Cada una de ellas se compone de un subconjunto de las dimensiones de la de la base de datos. Si bien los diferentes productos del mercado le otorgan diferentes denominaciones a estas pequeñas estructuras (variables, universos, estructuras, cubos, etc.), todas se referen al mismo tipo de subestructura.

Esta estructura divide el universo en diferentes cubos de menor tamaño e intenta dinamizarla mediante estos cubos con punteros. De esta manera, disminuye el espacio utilizado para el almacenamiento, ya que, al reducir el tamaño de los cubos que conforman el universo de datos, también lo hace la posibilidad de dispersión. Esta situación permite que los cubos no deban replicar el tamaño de una de las dimensiones a las restantes.

Sin embargo, existe la posibilidad de que los valores nulos no sean muy altos y, entonces, este tipo de estructura necesitaría más espacio de almacenamiento, puesto que los punteros que unen los diferentes cubos también ocupan espacio.

Existen dos tipos principales de multicubos: el *blocky* o bloque y el *series*. El primero utiliza dimensiones ortogonales lo que permite que no haya dimensiones especiales en este nivel de datos. De esta manera, un bloque se puede constituir por cualquier número de dimensiones definidas y medidas, por ejemplo: la dimensión tiempo. Esto significa que las medidas o variables se tratan como si fueran dimensiones. El segundo tipo trata a cada medida o variable como si fueran series de tiempo, con un conjunto propio de dimensiones.



Existen dos tipos principales de multicubos: el *blocky* y el *series*. El primero utiliza dimensiones ortogonales lo que permite que no haya dimensiones especiales en este nivel de datos. El segundo tipo trata a cada medida o variable como series de tiempo, con un conjunto propio de dimensiones.

#### 5.2.4.3 Elección de la estructura adecuada

En general, los multicubos, gracias a su gran flexibilidad y versatilidad, son los preferidos por los profesionales con experiencia; en cambio, los hipercubos son los elegidos por los usuarios finales porque son más fáciles de comprender y ofrecen una visión de alto nivel.

Los multicubos almacenan de manera más eficiente los datos dispersos y reducen el efecto de la explosión de datos, pero si este no es muy grande, es decir que no supera el 30%, utilizan más espacio de almacenamiento que los hipercubos.

Algunos fabricantes han optado por ofrecer un modelo mixto: permiten que el almacenamiento físico de los datos se realice como si se tratara de un multicubo, pero los cálculos se ejecutan como si se estuviera trabajando con un hipercubo. De esta manera, combinan la simplicidad del hipercubo con la flexibilidad del multicubo.

## 5.3 Tecnologías OLAP

### 5.3.1 Introducción

En el desarrollo de software sobre bases de datos conviven dos modelos: en primer lugar, el OLTP (On-line Transaccional Processing) integrado por la mayoría de los sistemas desarrollados que realizan operaciones destinadas al procesamiento de transacciones en línea. Dentro de estos sistemas encontramos los clásicos, que modelan la operatoria de una organización, y también los de ventas, compras, cuentas corrientes, etc. En segundo lugar, el OLAP (On-line Analytical Processing) que analiza la información mediante parámetros de interés específicos y que se consideran relevantes para la organización y permiten que esta información se integre en los diferentes modelos OLTP.

S

OLAP es un proceso que se utiliza para el análisis de información, para la elaboración de reportes administrativos y consolidaciones, para el análisis de la rentabilidad, para reportes de calidad y para otras aplicaciones que necesitan una visión exhaustiva del negocio.

### 5.3.2 Concepto de OLAP

OLAP representa el procesamiento analítico en línea que permite el análisis de la información sobre determinados patrones de interés. Esta tecnología se aplica a áreas funcionales de la empresa como producción, ventas, análisis de rentabilidad de la comercialización, consolidaciones financieras, presupuestos, pronósticos, planeación de impuestos y contabilidad de costos.

OLAP es un proceso que se utiliza para el análisis de información, para la elaboración de reportes administrativos y consolidaciones, para el análisis de la rentabilidad, para reportes de calidad y para otras aplicaciones que necesitan una visión exhaustiva del negocio. Asimismo, realiza los reportes sumarios que los ejecutivos necesitan para la toma de decisiones, los cálculos complejos, los enfoques de detalles operativos y las consultas no programadas. Además, como se alimenta principalmente de los OLTP, también necesita administrar la base de datos de manera eficiente y proveer un nivel adecuado de seguridad.

### 5.3.3 Características de OLAP

Las tecnologías que se aplican sobre un modelo OLAP ofrecen una visión multidimensional lógica de los datos; esto significa que se basa en la dimensión de interés que representa cada uno de ellos, con independencia del modo en que se almacenen.

Este modelo siempre abarca la consulta interactiva y el análisis de datos. En general, como la interacción se realiza a través de varias pasadas, profundiza en niveles cada vez más detallados o en el ascenso a otros niveles superiores de resumen y acumulación.

Otra de las características fundamentales del OLAP es ofrecer algunas opciones para el modelado analítico, que incluyen un motor de cálculo para la obtención de proporciones, desviaciones respecto a la media, etcétera, que abarcan las mediciones de datos numéricos realizadas a través de muchas mediciones.

Su objetivo es la creación de resúmenes y adiciones —denominadas también consolidaciones—, de jerarquías y, además, en las dimensiones de interés, cuestionar todos los niveles de adición y resumen en cada una de sus intersecciones.



La rapidez de respuesta del modelo OLAP a las consultas permite que el proceso de análisis no se interrumpa y, por lo tanto, que la información no pierda vigencia. Esto se debe al modelado de datos dimensional que se realiza durante el armado del modelo, en el que se efectúan los cálculos previos para la integración de información que lo simplifican. Además, cuenta con un motor de depósito de datos multidimensional con capacidad para almacenar los datos en arreglos.

Como manipula modelos funcionales de pronóstico, análisis de tendencias y datos estadísticos, tiene la capacidad de recuperar y exhibir datos tabulares en dos o tres dimensiones, cuadros y gráficas con un pivoteo fácil de los ejes. Este pivoteo es fundamental para los usuarios, porque necesitan analizar los datos desde diferentes perspectivas. Cada una de ellas conducirá a otra visión de la organización que se evaluará, a la vez, desde otra perspectiva.

La rapidez de respuesta de este modelo a las consultas permite que el proceso de análisis no se interrumpa y, por lo tanto, que la información no pierda vigencia. Esto se debe al modelado de datos dimensional que se realiza durante el armado del modelo OLAP, en el que se efectúan los cálculos previos para la integración de información que lo simplifican. Además, cuenta con un motor de depósito de datos multidimensional con capacidad para almacenar los datos en arreglos (representación lógica de las dimensiones organizacionales).

La bibliografía especializada sostiene que mientras la definición actual de OLAP ha justificado su aceptación, poco se ha recorrido en la definición de un estándar computacional. Los autores proponen el concepto FASMI (Fast Analysis Shared Multidimensional Information) para definir un OLAP.

**Fast (rápido):** porque el sistema tiene tiempos de respuesta de cinco segundos. En las consultas sencillas demora un segundo, en las complejas, puede tardar hasta 30. En situaciones cotidianas, si las respuestas tomaran más tiempo, el usuario perdería la concentración en lo que busca analizar.

**Analysis (análisis):** significa que al usuario se le debe proporcionar la funcionalidad necesaria para la resolución de sus problemas, sin el apoyo de sistemas o de una preprogramación. De esta manera, se podrán realizar consultas no definidas, cálculos de diferencias, variaciones y tendencias, consolidar y llevar a cabo análisis de sensibilidad y de búsqueda de metas.

**Shared (compartida):** se debe acceder a la lectura y a la escritura de forma simultánea, con un esquema de seguridad adecuado que guarde la confidencialidad (probablemente, en el nivel de celda).

**Multidimensional (multidimensional):** fue y será un requisito, debe manejar varias jerarquías y dimensiones.

**Information (información):** son todos los datos y la información derivada, cuando y donde sea necesaria, en su contexto, tanto información suave como dura, interna o externa.

#### 5.3.4 Comparación entre el modelo OLTP y el modelo OLAP

Para comprender el concepto de OLAP es necesario que se entiendan las diferencias entre estos dos tipos de modelos. En el caso de los diseñadores esto es fundamental a la hora de llevar adelante un proyecto de esta naturaleza.

A continuación, se analizarán las diferencias entre ambos modelos de procesamiento, según las características disímiles que los componen.

| Tabla Comparaciones               |   |   |      |
|-----------------------------------|---|---|------|
|                                   | Concepto  | OLTP  | OLAP |
| Orientación o alineación de datos | <ul style="list-style-type: none"> <li>Aplica los conceptos de la normalización de datos.</li> <li>Orientación de los datos por aplicación.</li> <li>Los datos son organizados inherentemente por cada aplicación.</li> <li>Está focalizado en resolver requerimientos de aplicaciones específicas.</li> </ul>  | <ul style="list-style-type: none"> <li>Es un modelo desnormalizado.</li> <li>La orientación de los datos es por patrones de interés o dimensiones.</li> <li>Los datos son organizados por dimensiones definidas en el negocio.</li> <li>Está focalizado en responder los requerimientos de análisis estratégico de la organización.</li> </ul>  |      |
| Integración                       | <ul style="list-style-type: none"> <li>No están integrados.</li> <li>Cada área del negocio tiene su propio modelo.</li> <li>Diferentes sistemas poseen diferentes datos.</li> <li>Diferentes nomenclaturas de datos en los distintos modelos.</li> <li>Diferentes formatos de archivos para cada sistema.</li> <li>Diferentes plataformas de hardware para cada sistema.</li> </ul> | <ul style="list-style-type: none"> <li>Deben estar integrados para poder analizar la información en su conjunto.</li> <li>Toda la información de interés se integra, sin que importe si se origina en modelos transaccionales o en sistemas diferentes.</li> <li>Todos los datos están en un mismo modelo.</li> <li>Se realizan convenciones de nomenclatura con el objeto de estandarizar la información.</li> <li>Utilización de formatos de archivo estándar.</li> <li>Una sola plataforma de hardware.</li> </ul> |      |

| Concepto                                   | OLTP   | OLAP  |
|--|--|---|
| Acceso y manipulación de datos del usuario | <ul style="list-style-type: none"> <li>Los usuarios son los que ingresan, modifican y manipulan los datos en función de las necesidades requeridas en cada sistema.</li> <li>Se ejecutan muchas veces las mismas acciones como ser altas, bajas o modificaciones.</li> </ul>   | <ul style="list-style-type: none"> <li>Los usuarios solo consultan la información, sin realizar inserciones, modificaciones ni borrado de los datos.</li> <li>Las operaciones de consulta no son repetitivas, de manera continua se cambia el tipo de consulta a la base de datos.</li> </ul> |
| Administradores de base de datos           | <ul style="list-style-type: none"> <li>La manipulación de datos se realiza registro a registro.</li> <li>Las transacciones y rutinas de validación se ejecutan en el nivel de registro o transacción.</li> </ul>   | <ul style="list-style-type: none"> <li>La carga y acceso a los datos es masiva.</li> <li>Las validaciones se realizan antes o después de cada carga, nunca en el nivel de registro o transacción.</li> </ul>  |
| Manejo de transacciones                    | <ul style="list-style-type: none"> <li>Se manejan cientos de transacciones por día.</li> <li>Se verifica la integridad y consistencia por transacción.</li> </ul>  | <ul style="list-style-type: none"> <li>Se realiza una sola transacción de carga global del modelo, que contiene gran cantidad de transacciones originadas en el modelo OLTP.</li> <li>Si la carga termina bien, se garantiza la consistencia de todo el conjunto de datos.</li> </ul>         |
| La dimensión tiempo                        | <ul style="list-style-type: none"> <li>Que falta un soporte explícito para la reconstrucción de la historia previa. Esto significa que si los datos se reescriben, es imposible que se recupere su estado anterior; si se mantiene un historial, al aumentar los cambios, será prácticamente imposible reconstruir hasta un punto</li> </ul> | <ul style="list-style-type: none"> <li>Similar a las capas geológicas, la base de datos se puede ver como una serie de capas. Cada una de ellas está compuesta por una foto del modelo OLTP tomada en un intervalo de tiempo determinado.</li> </ul>  |

| Concepto            | OLTP   | OLAP   |
|---------------------|--|--|
| La dimensión tiempo | <p>determinado. Los datos operacionales son altamente volátiles, cambian a medida que opera la organización y sus sistemas computacionales reflejan la operación.</p> <ul style="list-style-type: none"> <li>Puede haber cambios en la base de datos mientras se los consulta debido al alto grado de transaccionalidad de las bases.</li> </ul> | <ul style="list-style-type: none"> <li>Los datos son altamente estables, ya que se insertan en un momento determinado y no se modifican o se borran.</li> <li>No se realizan cambios en la base de datos, solo cargas de actualizaciones.</li> </ul> |

## 5.4 Integración entre bases de datos y herramientas OLAP

Las bases de datos multidimensionales suelen adquirir datos de otras fuentes como bases de datos relacionales, herramientas de escritorio u hojas de cálculo; también los usuarios finales pueden introducir algunos datos. En un ROLAP (Relational OLAP), los datos se almacenan físicamente en un RDBMS, mientras que en un MOLAP (Multidimensional OLAP) están en un MDB en el que se almacenan en diferentes estructuras que se optimizan por un procesamiento multidimensional.

Por último, existen los modelos HOLAP (Hybrid OLAP), en los que se mezclan la arquitectura de bases de datos multidimensionales y de datos relacionales.

Si los datos se almacenan en un MDB, en condiciones normales utilizarán menos espacio del que ocuparían si se encontraran en el sistema originario del que se recogieron los datos. Esto se debe, principalmente, a las claves, las indexaciones y las estructuras dimensionales, que o no se necesitan o se optimizan para ocupar mucho menos espacio. También la dispersión de datos se elimina de una manera más eficaz y éstos se pueden comprimir y resumir.

En general, las aplicaciones OLAP se orientan hacia el uso interactivo para que el usuario obtenga una rápida respuesta a sus interrogantes (en tiempos muy cortos). Si la respuesta se reduce a reunir la información de la base de datos con un formato adecuado para el usuario, estas expectativas se cumplen; en cambio, en los casos en los que sea necesario la realización de grandes cálculos, la respuesta será mucho más lenta. En términos de tiempo consumido, el principal costo es la recuperación de los datos que afectan a los elementos que se recuperarán y no los cálculos aritméticos.

Para obtener una rápida respuesta, las aplicaciones multidimensionales necesitan precalcular algunos de los datos que se utilizarán en el análisis. En este tipo de bases, los datos precalculados se almacenan automáticamente; en cambio, en los ROLAP, es común el uso de tablas de resumen.



En un ROLAP, los datos se almacenan físicamente en un RDBMS, mientras que en un MOLAP están en un MDB en el que se almacenan en diferentes estructuras que se optimizan por un procesamiento multidimensional.



Las aplicaciones multidimensionales necesitan precalcular algunos de los datos que se utilizarán en el análisis. En este tipo de bases, los datos precalculados se almacenan automáticamente; en cambio, en los ROLAP, es común usar tablas de resumen.

El hecho de precalcular toda la información necesaria puede llegar a ser un problema en lugar de una ventaja. El problema se encuentra en las relaciones multidimensionales que existen en todas las aplicaciones OLAP y en el hecho de que la mayoría de los datos introducidos son considerados dispersos. Esto provoca que los resultados precalculados basados en datos multidimensionales dispersos sean bastante más voluminosos de lo que sería deseable.

De acuerdo con lo desarrollado hasta aquí, se puede afirmar que la consulta se basa en tres tipos de datos: a) de entrada, introducidos por el usuario; b) precalculados y c) los cálculos que se realizan en el momento.

Para que no se produzca una explosión de datos se pueden utilizar dos principios: en primer lugar, se debe evitar cualquier objeto multidimensional con más de cinco dimensiones, ya que esto provocará la multiplicación de datos dispersos. En segundo lugar, se puede reducir la dispersión de objetos de datos individuales a través de un buen diseño y gracias a la utilización de una aproximación de multicubos en la que cada objeto posea solo la mínima cantidad de dimensiones necesaria.

Para calcular la cantidad exacta de lo que se debe precalcular dependemos de los siguientes factores: el hardware, la topología y el tamaño de la red, las características del software, el número de usuarios, la complejidad de los cálculos, etcétera.

Normalmente, los datos que se precalculan son:

- Datos lentos de calcular en tiempo de ejecución.
- Datos que se pidan con cierta asiduidad.
- Datos que constituyan la base para el cálculo de otros datos.



En las aplicaciones OLTP, los usuarios crean, actualizan o retienen registros individuales.

## 5.5 Arquitecturas OLAP y OLTP

En las aplicaciones OLTP, los usuarios crean, actualizan o retienen registros individuales. Por esta razón, las bases de datos con este tipo de aplicaciones se optimizan para la actualización de las transacciones.

En cambio, en las aplicaciones OLAP —utilizadas por analistas y gerentes que requieren vistas con un alto nivel de datos como, por ejemplo, las ventas de una línea de productos—, la base de datos se actualiza, generalmente, por bloques de múltiples fuentes y provee, además, poderosas aplicaciones multiusuario de gran poder analítico. En consecuencia, las bases de datos OLAP se optimizan para el análisis.

Si bien las bases de datos relacionales son excelentes para mantener un número reducido de registros de una manera rápida, no tienen el mismo rendimiento para una gran cantidad de registros que se deben resumir en un solo paso. En este caso, su respuesta es lenta y el uso desordenado de recursos son características comunes en la construcción de aplicaciones de soporte de decisiones creadas por encima de la base de datos relacional.

Muchas veces, se intenta resolver problemas con la tecnología relacional cuando, en realidad, son de naturaleza multidimensional. Por ejemplo, en el caso de las búsquedas con SQL en las que se quiere crear resúmenes de ventas de un producto por región y ventas en una región por producto, es probable que esto involucre la lectura de todos los registros de una base de datos de marketing, que tomaría horas. En cambio, con un servidor OLAP, se pueden manejar en segundos.

Las aplicaciones OLAP emplean datos resumidos; las OLTP, manejan datos actuales y atómicos y, en general, no requieren datos históricos. En cambio, casi toda la aplicación OLAP requiere de ellos.

Mientras que las aplicaciones OLTP y sus bases de datos se organizan alrededor de procesos específicos, las OLAP lo hacen por medio de temas y pueden responder a preguntas como la siguiente: "¿Qué productos se están vendiendo bien?".

## 5.6 OLAP: multidimensional contra relacional

Son dos las opciones para el almacenamiento de datos: el depósito multidimensional y el relacional. Desde la perspectiva del usuario, los datos que se compararán son:

- ¿Cuántos datos organizacionales están almacenados y disponibles?
- ¿Es adecuada la capacidad de almacenamiento?
- ¿Es aceptable el desempeño?
- ¿Se justifica el costo?
- ¿La consulta de los mismos es manejable por los usuarios?
- ¿Qué tipos de vínculos se puede asociar?

La siguiente tabla resume los aspectos de selección de uno u otro.



En las aplicaciones OLAP, la base de datos se actualiza, generalmente, por bloques de múltiples fuentes y provee, además, poderosas aplicaciones multiusuario de gran poder analítico.

**Tabla Comparaciones entre la base de datos relacional y la base de datos multidimensional**

|                                    | Base de datos relacional   | Base de datos multidimensional  |
|------------------------------------|--|---|
| Depósito de datos, acceso y visión | <ul style="list-style-type: none"> <li>• Relacional</li> <li>• Tablas de filas y columnas</li> <li>• SQL con ampliaciones</li> <li>• Herramientas API</li> </ul> | <ul style="list-style-type: none"> <li>• Dimensional</li> <li>• Arreglos: Hipercubo/Multicubo</li> <li>• Tecnología de matriz dispersa</li> <li>• Propietario de hoja de cálculo</li> </ul> |

|   |   |  |
|---|---|--|
| Utilización e incorporación                 | <ul style="list-style-type: none"> <li>OLTP</li> <li>Motor RDBMS</li> <li>Profundización a nivel de detalle</li> <li>Desempeño de consultas: rango amplio</li> </ul>  | <ul style="list-style-type: none"> <li>OLAP</li> <li>Motor multidimensional</li> <li>Profundización a nivel de resumen/adición</li> <li>Desempeño de consultas: rápido</li> </ul>                                |
| Tamaño y actualización de la base de datos  | <ul style="list-style-type: none"> <li>Gigabytes a Terabytes</li> <li>El depósito de índices y el retiro de normas incrementan el tamaño</li> <li>Consulta y carga paralelas</li> <li>Actualización durante el año</li> </ul> | <ul style="list-style-type: none"> <li>Gigabytes</li> <li>Compresión y adición de datos dispersos</li> <li>Difícil actualización durante el uso; los cambios pequeños podrían requerir reorganización</li> </ul> |
| Alcance de interacción del usuario          | Transaccional   | Base de datos completa   |
| Datos afectados por interacción del usuario | Registros individuales  | Grupos de registros  |
| Utilización de la máquina                   | Estática  | Dinámica   |
| Prioridades                                 | Alto desempeño<br>Alta disponibilidad   | Alta flexibilidad<br>Alta facilidad de consulta por usuarios   |

En un análisis basado en la tabla precedente, un modelo RDBMS es la mejor opción para el caso en que los datos sean voluminosos (10 GB en adelante) o se considere el crecimiento constante del volumen de datos. De la misma manera, es conveniente si el número de usuarios esperados supera los cincuenta.

También un MDB es una excelente opción si se tienen volúmenes cercanos a los 10 GB, con pocos usuarios concurrentes. En el desarrollo de aplicaciones de funcionalidad superior, que se realizarán en períodos cortos, el MDB —gracias a la integración natural de sus herramientas con la estructura de la representación dimensional de los datos— es una opción viable.

El costo del administrador es una de las características no técnicas que se debe evaluar. En el caso de las organizaciones grandes que, en general, cuentan con un RDBMS utilizado normalmente por la organización, la estrategia adecuada para reducir los costos, sería el pago de licencias adicionales para desarrollar su almacenamiento global en el mismo RDBMS. La implementación de este almacenamiento in-

tegrado se favorece si la organización cuenta con diseñadores y administradores con probada experiencia en la optimización de aplicaciones y en la afnación de bases de datos, conocimiento que es muy difícil de alcanzar en el corto plazo con un MDB.

#### 5.6.1 OLAP multidimensional (MOLAP)

El diseño lógico o el modelo de información son los que conducen el diseño inicial y la actividad de configuración. A continuación, se describen los pasos básicos para su implementación:

- Seleccionar la función de la organización, como análisis de ingresos por ventas y reportes financieros.
- Identificar los valores numéricos; es decir, las mediciones que se almacenarán, tales como ingresos por ventas y clientes.
- Determinar las dimensiones (tiempo, geografía y producto). Por ejemplo, la dimensión tiempo, por mes y por trimestre; la geográfica, por estado o región.
- Definir el modelo lógico y cargar el depósito de datos multidimensionales, ya sea directamente de la fuente de datos o filtrando y ajustando el contenido seleccionado.

Las funciones principales que se ofrecen a los usuarios de la organización comprenden:

- Rápida respuesta a la consulta de cómputos intensiva, tal como escenarios “¿qué pasa si...?”.
- Actualización constante e interactiva de la base de datos multidimensional, con el fin de permitir aplicaciones de pronósticos que permitan proyectar situaciones en el futuro y los presupuestos que se realizarán.
- Explotación de relaciones ricas entre los elementos o valores de las dimensiones para descubrir relaciones insospechadas.
- Un poderoso motor de cálculo y análisis comparativo: posiciones, comparaciones, porcentaje de clase, máximo, mínimo, promedios, promedios móviles, comparaciones entre períodos y otros.
- Cálculos cruzados entre dimensiones, como asignación de costos y eliminaciones dentro de la compañía, o cálculos en el nivel de filas para aplicaciones orientadas a hojas de cálculo, como las declaraciones de pérdidas y ganancias.
- Ampliación de las funciones básicas con funciones definidas por el usuario.
- Potentes funciones estadísticas y financieras.
- Inteligencia de tiempo.
- Pivoteo, tabulación cruzada, profundización, niveles de resumen para una o varias dimensiones y otras funciones poderosas de navegación.

En lo que respecta a la administración en general y a la administración de sistemas en particular, se requiere contar con lo siguiente:



La implementación de este almacenamiento integrado se favorece si la organización cuenta con diseñadores y administradores con probada experiencia en la optimización de aplicaciones y en la afnación de bases de datos, conocimiento que es muy difícil de alcanzar en el corto plazo con un MDB.

- El modelado inicial de datos en los que es clave elegir las dimensiones correctas, para prever cómo se accederá a ellos y se seleccionarán los filtros apropiados para cargarlos.
- Transferencias periódicas y actualizaciones en bloque, debido a que las actualizaciones en incrementos son un reto y casi imposibles mientras la base de datos está en uso.
- Acumulación, resúmenes y precálculo durante el proceso de cálculo.
- Capacitación en una tecnología diferente y uso de nuevas habilidades.
- Escritura de nuevas aplicaciones en lenguaje propietario para ampliar y mejorar los procesos frontales comunes de la base de datos.

Se necesitan varias dimensiones porque no se deben mezclar diferentes informaciones (ciudades con provincias, provincias con regiones, etc.).

En las MDB sin jerarquías, la solución puede ser por dimensiones separadas.

### 5.6.2 OLAP relacional (ROLAP)

Los datos se presentan al usuario en dimensiones, aunque se almacenan en forma relacional (fila, columna). Para ocultar este tipo de depósito, es necesario que se cree una capa semántica compuesta por metadatos (conjunto de datos sobre los datos), que posiciona las dimensiones para las tablas relacionales. Además, y para mejorar los tiempos de respuesta, es necesaria la creación de metadatos adicionales para cualquier resumen o adición. De esta manera, los datos se almacenan en la base relacional para su mantenimiento y administración.

La actividad inicial de diseño y configuración se conduce mediante el diseño técnico de una base de datos, a través de los siguientes pasos:

- Construir el diseño dimensional utilizando técnicas como el esquema Estrella (modelo Star), copo de nieve o constelaciones, que se verá en el apartado 5.6.2.1.
- Incorporar los datos adecuados de adición y resumen.
- Dividir los grandes conjuntos de datos en segmentos más pequeños y manejables para mejorar el desempeño, por ejemplo, dividiendo las unidades de tiempo.
- Agregar índices creativos o de mapa de bits para mejorar el desempeño.
- Crear y almacenar los metadatos. Los metadatos incluyen las definiciones de las dimensiones, su ubicación en las tablas relacionales, las relaciones jerárquicas entre dimensiones, la información de segmentos, las definiciones y descripciones de resúmenes y adiciones, las fórmulas y cálculos, la vigencia de uso y muchas otras cosas.

Desde la perspectiva operacional, los pasos para ejecutar una consulta son los siguientes:

- Construir la herramienta "Cliente" utilizando una visión dimensional u organizacional de los datos.

Los datos se presentan al usuario en dimensiones, aunque se almacenan en forma relacional (fila, columna). Para ocultar este tipo de depósito, es necesario que se cree una capa semántica compuesta por metadatos y, además, otros adicionales para cualquier resumen o adición.



- Consultar el servidor OLAP desde la herramienta “Cliente” y examinar los metadatos en tiempo real.
- Crear declaraciones SELECT de pasos múltiples y/o subconsultas correlacionadas y someterlas a la base de datos relacional.
- Sobre los resultados de la consulta a la base de datos, se deben realizar funciones multidimensionales, como cálculos y fórmulas, traducción de bits para descripciones de la organización, etcétera.
- Devolver los resultados a la herramienta “Cliente” para un mayor procesamiento y exhibición.

Como las herramientas ROLAP mapean una estructura multidimensional en una capa por encima de la estructura de tablas original, proveen la capacidad de generar los cálculos analíticos que limitan las bases relacionales y al usuario de herramientas que permiten la observación de datos bajo un esquema multidimensional. Dado que una base de datos relacional no entiende las estructuras multidimensionales, la capa multidimensional es de vital importancia.

Con el objeto de optimizar el desempeño de un ROLAP, se hace necesaria la creación de una tabla de acumulación que debe pasar a un servidor de cálculos, que se encuentra fuera de la base de datos relacional. Un ROLAP requiere de la creación y mantenimiento de un índice. Para mantener un sistema eficiente, es necesario que todos los renglones se ordenen o indexen, con el objeto de que este índice adquiera un tamaño considerablemente grande, que al no poder “subir” a la memoria, afecta el desempeño de la aplicación.

#### 5.6.2.1 Esquemas de generación de modelos ROLAP

##### Esquema Estrella (Modelo Star)

El esquema Star o Estrella es una técnica de modelado de datos que se utiliza para hacer corresponder un modelo multidimensional sobre una base de datos relacional. Se lo denomina de esta manera porque su forma se parece a una estrella. El modelo Estrella tiene cuatro componentes: hechos, dimensiones, atributos y jerarquías de atributos. Los dos primeros se representan con tablas físicas relacionales en el almacén de datos; de esta forma, la tabla de hechos se vincula con cada dimensión en una relación “uno a muchos”. Ambas tablas se encuentran relacionadas entre sí por medio de claves foráneas (foreign key).

El siguiente gráfico muestra un modelo Estrella para el sector de compras de una organización con tres dimensiones (patrones de interés): tiempo, proveedores y productos.



El modelo Estrella tiene cuatro componentes: hechos, dimensiones, atributos y jerarquías de atributos. Los dos primeros se representan con tablas físicas relacionales en el almacén de datos que se relacionan entre sí a través de claves foráneas. De esta forma, la tabla de hechos se vincula con cada dimensión en una relación “uno a muchos”.

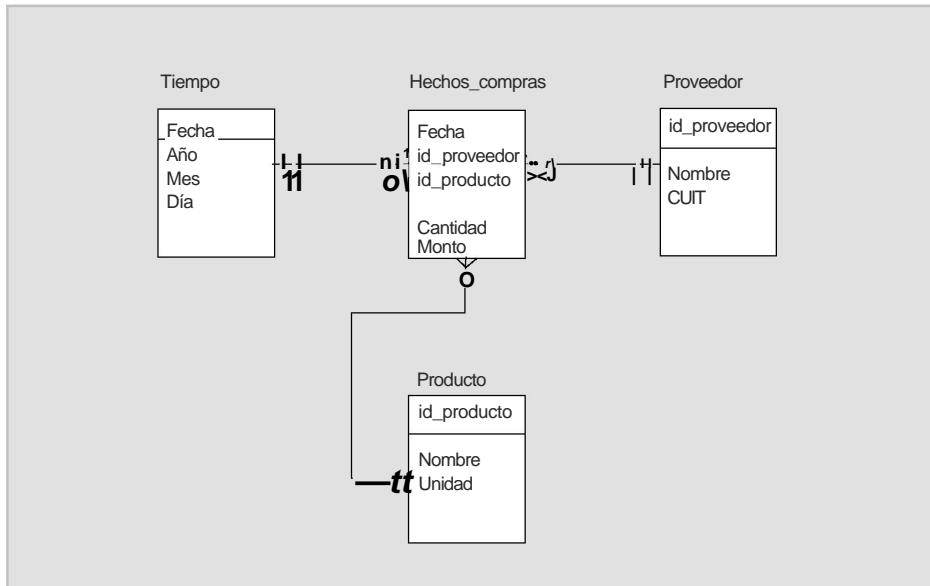


Fig. 5.4 Ejemplo del modelo Star.

Aquí se observa una tabla de hechos que refeja los eventos o hechos registrados y tres tablas de dimensiones: la primera corresponde a la dimensión tiempo; la segunda, a proveedores y la tercera, a productos, en las que se describen los atributos de cada una de ellas con sus atributos asociados.

De esta manera, se puede obtener información de forma independiente por cualquiera de las dimensiones y, asimismo, a través de la intersección de todas las dimensiones, que es el contenido de la tabla de hechos.

Como vemos la información en un modelo estrella, como en cualquier otro aplicado a la construcción de un modelo OLAP, se almacena, pero primero se acumula, registrando solamente cantidades y valores, dado que es ese el nivel de detalle que se requiere en este caso, sin llegar a un nivel de detalles menores, los cuales ya están modelizados en los diferentes modelos OLTP que son los generadores de datos para la construcción del modelo OLAP.

# S

El esquema Copo de nieve es una variación del esquema Estrella tradicional y en cada dimensión se almacenan las jerarquías de atributos o se dividen en otra entidad que les permita la obtención de una mejor performance y la utilización del espacio de almacenamiento.

## Esquema Copo de nieve

En el esquema Copo de nieve, si bien es una variación del esquema Estrella tradicional, en cada dimensión se almacenan las jerarquías de atributos o, simplemente, se dividen en otra entidad que les permita la obtención de una mejor performance y la utilización del espacio de almacenamiento.

En la figura que se muestra a continuación, se modificó la dimensión “producto” y se la dividió en dos entidades para diferenciar la dimensión de sus atributos esenciales de otros que lo redimensionan en otras categorías.

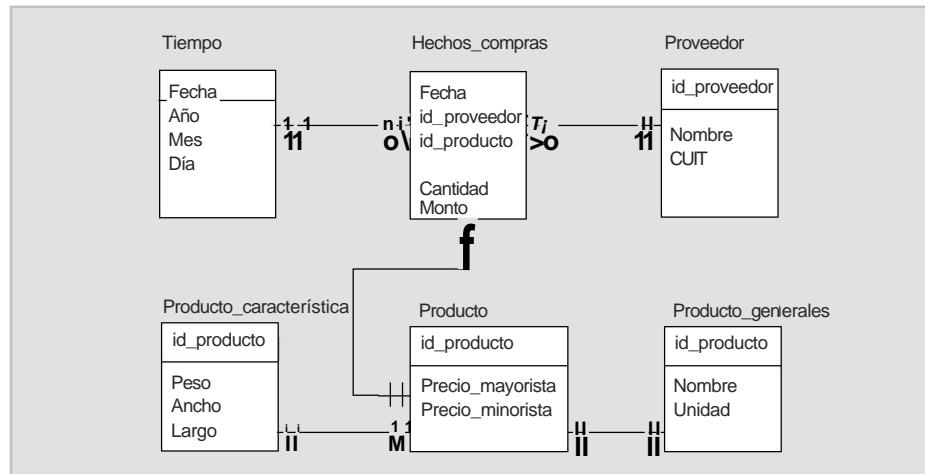


Fig. 5.5 Ejemplo del modelo Copo de nieve.

### Esquema Constelaciones

Este modelo también es una modificación del modelo Estrella. En este caso, algunos de los atributos de las dimensiones se separan para formar una nueva entidad que se puede compartir con otros cubos, tal como se ilustra a continuación:

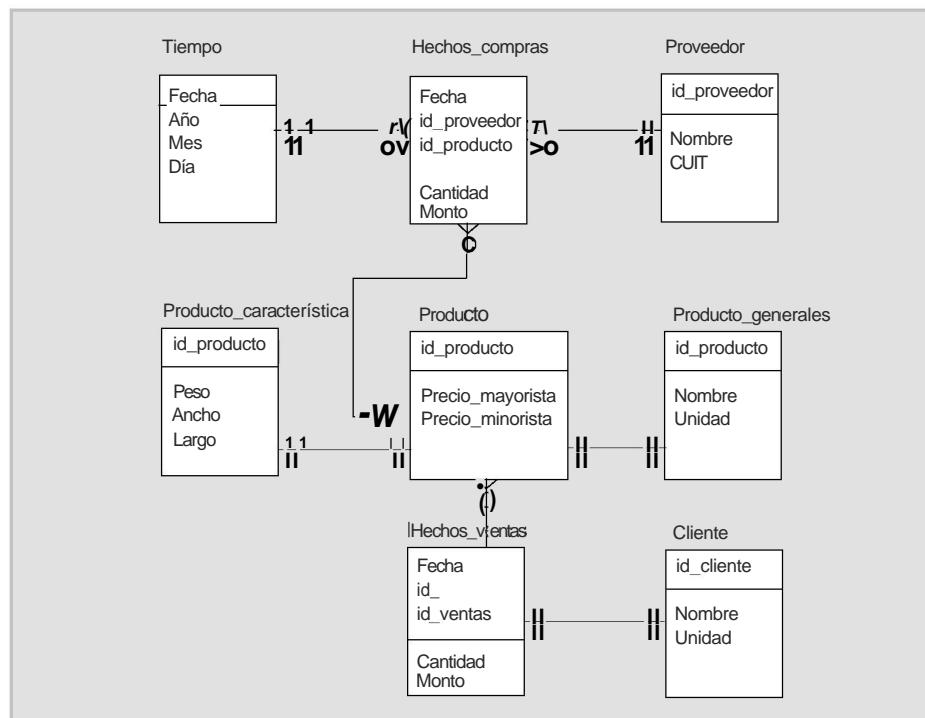


Fig. 5.6 Ejemplo del modelo de Constelaciones.

Este esquema es muy útil ya que, al tener dimensiones que se pueden compartir entre diferentes cubos, utilizará mejor el espacio de almacenamiento y, de esta manera, evitará la redundancia o la explosión de datos.



El modelo OLAP híbrido mantiene los registros de detalle que representan los volúmenes más grandes de datos en bases de datos relacionales para aprovechar su mejor manejo de almacenamiento, mientras que conserva las agregaciones o dimensiones en un almacén multidimensional separado.

### 5.6.3 OLAP híbrido (HOLAP)

Como se mencionó en los párrafos precedentes, existe otro tipo de implementación del Modelo OLAP, que trabaja sobre el concepto de la conjunción de bases de datos multidimensionales y relacionales.

Este modelo mantiene los registros de detalle (tablas de hechos) que representan los volúmenes más grandes de datos en bases de datos relacionales para aprovechar su mejor manejo de almacenamiento, mientras que conserva las agregaciones o dimensiones en un almacén multidimensional separado.

De esta forma, se toma ventaja de la potencia de cada uno de los dos tipos de bases de datos y se optimiza la solución.

## 5.7 Evaluación de servidores y herramientas OLAP

Los servidores y las herramientas OLAP se evalúan mediante cuatro conjuntos de criterios específicos que se analizarán a continuación:

### 5.7.1 Características y funciones

Es importante que la interfaz del usuario y el acceso a los servicios OLAP ofrezcan diversas opciones, que le conferan un mayor poder a las habilidades que ya posee el usuario y al conocimiento incorporado en sus modelos analíticos.

Las opciones potenciales deben incluir lo siguiente:

- Hoja de cálculo.
- Herramientas cliente del propietario.
- Herramientas de otros fabricantes.
- Ambiente 4GL.
- Interfaz con lenguajes estándar.
- Navegadores de cubo para clientes.

### 5.7.2 Motores de servicios OLAP

En cualquiera de las configuraciones de los servicios OLAP, el motor —ya sea de depósito dimensional o relacional— debe satisfacer la capacidad, la viabilidad de cambio de escalas y las características tecnológicas del modelo analítico y la aplicación planeados.

Las características de tecnología requeridas dependen del modelo analítico y del uso contemplado.

### 5.7.3 Administración

Las funciones necesarias de administración para fines de preparación inicial, configuración y operación continua incluyen lo siguiente:

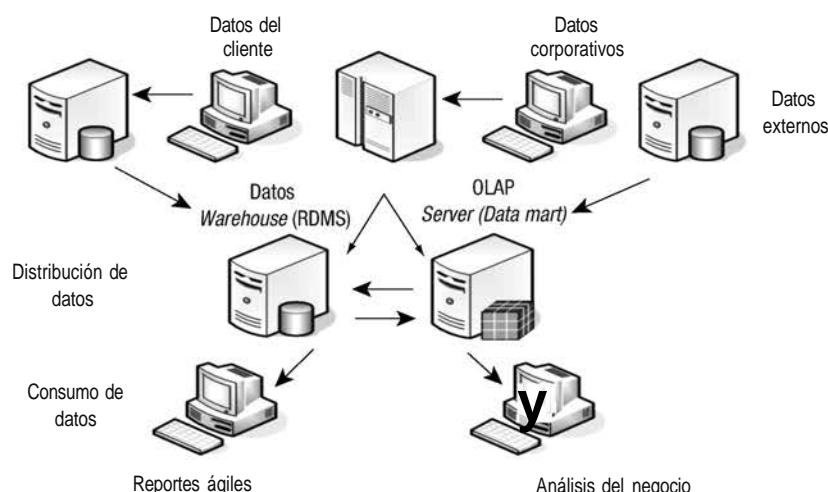
- Definición del modelo analítico dimensional.
- Creación y mantenimiento del depósito de metadatos.
- Control de acceso y privilegios con base en el uso.
- Carga del modelo analítico desde el almacén de datos o Data Warehouse.
- Afnación del desempeño a niveles aceptables para permitir un análisis que no desorganice.
- Reorganización de la base de datos.
- Administración de todas las partes del sistema, incluyendo el Middleware.
- Distribución de los datos al cliente para el análisis adicional y local.

### 5.7.4. Arquitectura global

Desde una perspectiva de arquitectura global, no resulta sencilla la elección entre el depósito de datos multidimensionales o el relacional para el OLAP. La organización necesita proporcionar los criterios para hacer la elección adecuada.

La tendencia de la industria es ofrecer los servicios OLAP con una combinación de un proceso frontal de servidor OLAP, un depósito multidimensional incorporado para datos gruesos y un procesador posterior de depósito relacional. En esta configuración de arquitectura, la información y consultas de acceso frecuente se precalan, se resumen, se agregan y luego, se almacenan en depósitos de datos multidimensionales relativos del almacén de datos. Las consultas complejas o de cómputo intensivo, o los datos complejos basados en cálculos, también se procesan y almacenan.

En la siguiente figura, se observa una arquitectura que se puede construir sobre cualquiera de los modelos o con una combinación de ellos.



La tendencia de la industria es ofrecer los servicios OLAP con una combinación de un proceso frontal de servidor OLAP, un depósito multidimensional incorporado para datos gruesos y un procesador posterior de depósito relacional.

Los datos gruesos son aquellos que son recibidos directamente de los sistemas transaccionales, sin que todavía se hayan relacionado entre sí.

Fig. 5.7 Arquitecturas OLAP corporativas.

## 5.8 Desarrollo de aplicaciones OLAP

El éxito en las organizaciones que implementen aplicaciones OLAP dependerá de sus experiencias y de la metodología que empleen. El usuario es el que desarrolla y mantiene este tipo de aplicaciones que no requieren un soporte amplio del área de sistemas.

En el proceso de toma de decisiones Innovation Decision Process (IDP), la innovación se basa en crear un entorno de herramientas analíticas y de técnicas que faciliten la determinación de herramientas claras acerca de la dinámica del negocio.

IDP es una metodología completa que se enfoca en la integración de tres elementos básicos: el proceso, la tecnología y la cultura de la organización. Su objetivo es mejorar la productividad y la eficiencia de análisis que contribuya con el proceso de toma de decisiones en la organización.

En la siguiente figura, se establecen las etapas de aplicación de las tecnologías OLAP que sustentan un modelo que contribuya con la toma de decisiones dentro de una organización.

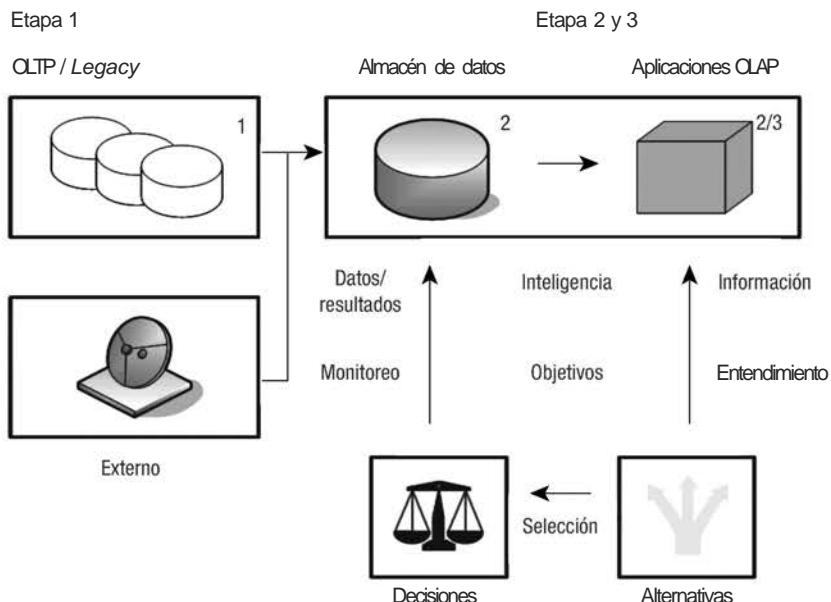


Fig. 5.8 Etapas en la aplicación de tecnologías OLAP.

En la primera etapa, se inicia el proceso de recolección de información a través de sistemas internos OLTP de la organización, sistemas de gestión de ventas, compras, etcétera. y, también, de datos externos como, por ejemplo, las condiciones del mercado o la situación de los competidores.

En la segunda etapa, la información se resume y se integra en un almacén de datos que, en la etapa siguiente, se procesa con herramientas OLAP, que ofrecerán diferentes alternativas para la toma de decisiones.

Tanto el IDP como los mapas de decisión (indicados en el diagrama) contribuyen con la definición de los objetivos y las medidas de desempeño, que abarcan desde el desarrollo de planes y de tendencias claves de decisión con la información requerida para verificar supuestos que permitan los mejores cursos de acción. De este proceso, se obtiene una lista de aplicaciones con sus respectivas prioridades. Gracias a los mapas de decisión se puede precisar el modelo mental del que se valen los ejecutivos para la conceptualización y el control de su negocio.

Las aplicaciones OLAP brindan la inteligencia de negocios requerida para tomar decisiones y, por consiguiente, definen la estructura y el contenido del almacén de datos, el nivel histórico y de detalle necesario de la información.

Finalmente, los OLTP encargados del proceso de colección de datos, con su filtrado e integración, permiten la generación de información valiosa característica del almacén de datos y de los OLAP.



Las aplicaciones OLAP brindan la inteligencia de negocios requerida para tomar decisiones y, por consiguiente, definen la estructura y el contenido del almacén de datos, el nivel histórico y de detalle necesario de la información.

## 5.9 Áreas de aplicación de las tecnologías OLAP

Las aplicaciones OLAP han sido utilizadas por las organizaciones que procesan gran cantidad de datos internos y externos que se necesitan valorar —como en el caso de las empresas de servicios financieros y de marketing.

A continuación, se analizarán los aspectos principales de aplicación de estas tecnologías.

### 5.9.1 Análisis de ventas

Existen ciertas preguntas que se pueden responder con un sistema de análisis de ventas y de marketing:

- ¿Ayuda a comprobar si se consiguen los objetivos propuestos para fines de mes, por ejemplo, por productos y por regiones?
- ¿Permite adecuar la capacidad de producción y de almacenamiento, según la demanda?
- ¿Permite comprobar la aceptación de los nuevos productos en el mercado?
- ¿Ayuda a estudiar la efectividad de campañas publicitarias, comprobando el aumento de las ventas en determinados productos y zonas geográficas?
- ¿Permite el estudio de los descuentos que se aplicarán y, también, si son útiles y adecuados para la organización?
- ¿Permite realizar estudios demográficos de las ventas?
- ¿Comprueba si, de acuerdo con el historial de datos y los planes de los productos, se alcanzan los objetivos para cada producto, período de tiempo y canal de ventas?

Los beneficios de un buen sistema de análisis de marketing y de ventas son predecir y analizar, ya que las oportunidades se visualizan mejor y se producen más ventas.

### 5.9.2 Gestión de informes financieros

En las empresas se necesitan gestionar informes financieros. Sin embargo, los analistas financieros y contables se encuentran, generalmente, con los datos desorganizados. Por esta razón, utilizan el software multidimensional. En las áreas contables, por ejemplo, se comparan diferentes versiones de datos como en el caso de los actuales, los pronosticados y los presupuestados. Aquí los contadores se encuentran con un modelo 4-dimensional. Es usual que en la segmentación del negocio o en un análisis de la línea de producción se sumen otras dimensiones.

#### Informes de gestión

Por lo general, los informes de gestión hacen hincapié en el fujo de dinero y no tanto en la hoja de balance. Como se producen más a menudo que los financieros, se necesita que se generen con rapidez y que soporten cambios en los requerimientos. Esta flexibilidad se consigue con los sistemas basados en OLAP.

#### Análisis de rentabilidad

El análisis de rentabilidad es otra de las aplicaciones de las herramientas OLAP. Las empresas con balances positivos lo necesitan para conocer las causas de los beneficios y las organizaciones con pérdidas también deben utilizar las causas de su mal desempeño.

El análisis de los beneficios es de vital importancia para establecer las actividades promocionales, los precios y los descuentos y, también, poder anticiparse a las presiones del mercado. En general, las empresas toman esta clase de decisiones de manera individual y, en algunos casos, pueden resultar poco efectivas o contraproducentes, si el que tiene el poder de decisión no se encuentra bien informado de los diferentes niveles de rentabilidad de cada producto y de las preferencias de los clientes.



El análisis de los beneficios es de vital importancia para establecer las actividades promocionales, los precios y los descuentos y, también, poder anticiparse a las presiones del mercado.

#### Análisis de calidad

En los sistemas de control de calidad, cuando se controlan las medidas numéricas de determinadas variables como los medios de producción, productos o servicios, tiempo, localizaciones y clientes, se utilizan datos multidimensionales. Si bien la mayoría de estas variables no son financieras, son igual de importantes si se quiere obtener una radiografía acabada de la organización. Para esta clase de análisis, las herramientas OLAP ofrecen la posibilidad de obtener una visión macro que permitirá descubrir las malas tendencias antes de que se produzcan serias irregularidades.

## 5.10 Ventajas y desventajas de OLAP

Como en el resto de las tecnologías informáticas, la tecnología OLAP ofrece algunas ventajas que también traen aparejados ciertos inconvenientes.

Dentro de las ventajas de las herramientas OLAP, se podría afirmar que la herramienta es, en sí misma, una ventaja, porque permite una mayor rapidez y efectividad en el análisis de datos (que van desde datos de ventas hasta del alumbrado de un estadio de un club), que se traduce en una información más depurada en la que se compararán gran cantidad de datos y gráficas y, también, permitirá la extrapolación de relaciones que, a simple vista, serían imposibles de realizar. Con otras herramientas, en cambio, se podrían sacar cuatro o cinco gráficas a ojo de los datos que, aparentemente, resulten más interesantes.

Si el lector quiere buscar alguna tecnología que compita con la OLAP, seguramente no la encontrará. Es más, tecnologías como las bases de datos, hojas de cálculo y sistemas de análisis de datos están en plena evolución hacia la tecnología OLAP. Por esta razón, sus productos se encuentran en proceso de adaptación a las exigencias de una herramienta OLAP, de la misma manera que adecuan sus productos a las tecnologías de redes. Oracle es un claro ejemplo de esta tendencia. Esta empresa siempre fue líder en el sector de la base de datos y, en la actualidad, también en las herramientas OLAP.

Entre las desventajas de esta tecnología —si es que se pueden considerar como tales— se puede mencionar que las herramientas OLAP se deben adquirir como un software que, evidentemente, implican un desembolso de dinero y, también, la necesidad de una base de datos previa —que, normalmente, es ajena al software OLAP— y su posterior actualización.

Este último punto es, quizás, el menos logrado de la tecnología OLAP, ya que sus herramientas necesitan una base de datos ajena al programa para interactuar con ella. Sin embargo, empresas como Oracle han intentado juntar las dos tecnologías y ofrecen en sus bases de datos un soporte para herramientas OLAP.

En cualquier caso, y pese a las desventajas mencionadas que puede traer la adquisición de una herramienta OLAP, las ventajas que conlleva el uso de una herramienta OLAP para el análisis de datos son mucho mayores en número y en importancia.

## 5.11 Resumen

La necesidad de un análisis multidimensional oportuno, como soporte para la toma de decisiones, es cada vez más creciente. Para satisfacer esta necesidad, los departamentos de sistemas se han inclinado por la tecnología de almacenes de datos. Sin embargo, es importante mencionar que existen diversas tecnologías que se utilizan para la consulta, el almacenamiento de datos y el análisis de resultados.

De lo expuesto se desprende que son pocos los almacenes de datos exitosos y que satisfagan las demandas del usuario. Los mercados de datos basados en OLAP —por dentro o por fuera de la arquitectura de almacenes de datos— se han posicionado como una solución muy práctica para el usuario final. Por esta razón, la selección de la arquitectura correcta para el desarrollo de la aplicación es importantísima.

Una manera de contribuir con el proceso de toma de decisiones en una organización es mediante la utilización de IDP y de mapas de decisiones que, de forma simultánea, definen los requerimientos analíticos de la aplicación OLAP.



Las herramientas OLAP permiten una mayor rapidez y efectividad en el análisis de datos, que se traduce en una información más depurada en la que se compararán gran cantidad de datos y gráficas y, también, permitirá la extrapolación de relaciones que, a simple vista, serían imposibles de realizar.



Entre las desventajas de las herramientas OLAP, podemos mencionar que se deben adquirir como un software que, evidentemente, implica un desembolso de dinero y, también, la necesidad de una base de datos previa y su posterior actualización.

Es necesario que el usuario final se involucre en la definición y en el desarrollo de la aplicación, ya que es de vital importancia para que sea exitosa. Esta aplicación OLAP —que depende del volumen y del nivel de detalle requerido— debe definir el contenido del almacén de datos y los accesos a los sistemas transaccionales, y no lo contrario.

## 5.12 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) solo está disponible para docentes.

### 5.12.1 Mapa conceptual del capítulo

### 5.12.2 Autoevaluación

### 5.12.3 Presentaciones\*

# 6

## Almacén de datos

### Contenido

|  |     |
|--|-----|
| 6.1 Introducción.....  | 190 |
| 6.2 Inteligencia de negocio.....   | 191 |
| 6.3 Sistemas de almacén de datos.....  | 193 |
| 6.4 Concepto de almacén de datos.....  | 194 |
| 6.5 Características de un almacén de datos.....                                | 194 |
| 6.6 Arquitectura del almacén de datos.....                                     | 196 |
| 6.7 Funcionalidades y objetivo.....  | 197 |
| 6.8 Almacén de datos y Data Mart .....   | 200 |
| 6.9 La integridad de los datos.....  | 202 |
| 6.10 Costos y valor del almacén de datos.....                                  | 208 |
| 6.11 Impactos de la implementación de un almacén de datos.....                 | 210 |
| 6.12 Estrategia recomendada para la implementación de un almacén de datos..... | 213 |
| 6.13 Resumen.....  | 216 |
| 6.14 Contenido de la página Web de apoyo.....                                  | 216 |

### Objetivos

- Introducir el concepto de la inteligencia de negocios para la construcción de la información utilizada para la toma de decisiones.
- Conocer las características y las funcionalidades del armado de un *Data Warehouse*.
- Conocer las diferentes arquitecturas tecnológicas para su implementación.

## 6.1 Introducción

En la actualidad, la automatización de los procesos, el control centralizado de un gran caudal de datos y el uso de sistemas de información en línea constituyen los principales pilares sobre los que se sostiene el futuro de una organización. Son, en definitiva, parte de su patrimonio y un recurso primario trascendental.

La realidad de los mercados —con su alto nivel de competitividad— ha exigido a las organizaciones el desarrollo de nuevas herramientas de gestión. En consecuencia, se han producido reestructuraciones que modificaron sustancialmente el modo de trabajar que imperaba hace unos años. De organizaciones piramidales, en las que el poder de decisión estaba en el vértice más alto, se pasó a un modelo descentralizado, en el que se eliminaron los estratos y se permitió a los usuarios mayores márgenes de decisión y responsabilidad. De esta manera, se aprovecha más el potencial de la información. No obstante, esta autorización carece de fundamento si la información no tiene la solidez necesaria para sustentar las decisiones.

Pero, ¿cómo se logra que la información llegue a las diferentes áreas de la empresa? La respuesta se encuentra en el concepto de almacén de datos o Data Warehouse (DW), ya que facilita el traslado de la información desde la base hacia la cúspide de la pirámide y, también, entre todos los empleados que conforman la estructura de la organización. De este modo, el almacén de datos convierte los datos operacionales de la empresa en una herramienta competitiva que contribuye con el análisis y la toma de decisiones.

Los datos que los analistas necesitan se pueden agregar al almacén de datos, para que su contenido y estructura se organicen para que satisfagan los requisitos de información interna que se utilizarán para mejorar la gestión.

Toda organización se sustenta en un proceso productivo, cuyo resultado forma la cadena de valor, en la que cada estadio —proceso de negocios— significa una mejora en su desarrollo. Desde esta perspectiva, es indispensable la optimización de cada uno de sus eslabones pero sin que se diluya la totalidad del proceso.

Por esta razón, el manejo eficiente de la información en cada una de las áreas que componen la organización es el fundamento de las futuras decisiones que contribuirán con el desarrollo de acciones apropiadas que, en definitiva, permitirán un mayor control de la producción.

En el almacén de datos, se puede representar cada uno de los eslabones que, sumados, constituyen el total de la cadena de producción. Esto facilita la administración de las diferentes etapas productivas y los datos asociados a ellas.

Es importante aclarar que un almacén de datos no es un producto que se adquiere en el mercado; es un concepto que se debe definir o, dicho de otra manera, la combinación de conceptos y tecnología que modifican la manera en la que se entrega la información a los usuarios de negocios. Como afirmamos en los párrafos anteriores, lo que se quiere es satisfacer, con eficiencia y facilidad de acceso, los requerimientos de información internos de la organización que signifiquen una mejora en la gestión.

En la actualidad, la manera en que se entrega la información adolece de grandes falencias que no proporcionan seguridad a la hora de distribuirla entre los diferentes sectores de la organización, puesto que, además de ciertas inconsistencias, no cumple con un requisito fundamental: la integración necesaria, porque los sistemas



Un almacén de datos no es un producto que se adquiere en el mercado; es la combinación de conceptos y tecnología que modifican la manera en la que se entrega la información a los usuarios de negocios.

operacionales —desde donde parten los reportes impresos, con consultas en el nivel cliente y alguna extracción ocasional de datos para evitar las actividades basadas en el papel— no han demostrado ser muy eficientes en este sentido.

Se puede comparar a un almacén de datos con un depósito en el que se encuentran almacenados todos los datos necesarios para efectuar las funciones de gestión de la organización.

Los sistemas transaccionales son dinámicos ya que, constantemente, actualizan sus datos. Por esta razón, el análisis de la información puede arrojar diferentes resultados en cuestión de minutos. En consecuencia, es indispensable que se fotografíen (snapshots) los datos para su extracción y almacenamiento para, de esta manera, facilitar la división de la información en períodos. Sin embargo, esto trae aparejado un consumo adicional de recursos de cómputo. Entonces, cuando un sistema transaccional realiza un análisis complejo, puede llevar a una degradación del sistema que impactará en la operación del negocio.

En la actualidad, el almacén de datos (o Data Warehouse) es una de las tecnologías más importantes para la obtención de información, ya que lo hace sin sacrificar el rendimiento de las aplicaciones operacionales. Su manera de generar bases tangibles mediante la fusión de datos de múltiples fuentes, permite que éstos se mantengan actualizados sin que cambien la velocidad de los sistemas transaccionales. En muchos casos, los almacenes de datos se diseñan para que contengan un límite de detalle hasta el nivel de transacción. De esta manera, se puede disponer, para su reporte y análisis, de diferentes datos y características. De este modo, el Data Warehouse se convierte en un almacén de datos transaccionales para consultas operativas, con la información necesaria para realizar un análisis estratégico y decisorio.

Existen muchas definiciones para el almacén de datos, la más conocida fue propuesta por Inmon en 1992: "Un DW es una colección de datos orientados a temas, integrados, no volátiles y variantes en el tiempo, organizados para soportar las necesidades de la organización".

En 1993, Susan Osterfeldt publica una definición que sin duda acierta en la clave del almacén de datos: "Yo considero a DW como algo que provee dos beneficios organizacionales reales: integración y acceso de datos. DW elimina una gran cantidad de datos inútiles y no deseados, como también el procesamiento desde el ambiente operacional clásico".

Esta última definición manifiesta el principal beneficio de un almacén de datos: la eliminación de los datos que obstaculizan la labor de análisis de la información y su respectiva entrega del modo más apropiado para facilitar el proceso de gestión.

## 6.2 Inteligencia de negocio

La inteligencia de negocio consiste en la transformación de datos en información y, ésta, en conocimiento, con la intención de mejorar al máximo el proceso de toma de decisiones de la organización.

Desde la perspectiva de la tecnología de la información, la inteligencia de negocio se define como el conjunto de metodologías, herramientas y estructuras de almacenamiento que permiten la reunión, depuración y transformación de los datos en



Un almacén es una colección de datos orientados a temas, integrados, no volátiles y variantes en el tiempo, organizados para soportar las necesidades de la organización.



El principal beneficio del almacén de datos es la eliminación de los datos que obstaculizan la labor de análisis de la información.



La inteligencia de negocio consiste en la transformación de datos en información y, ésta, en conocimiento, para mejorar al máximo el proceso de toma de decisiones de la organización.

una información integrada que se pueda analizar y convertir en conocimiento para la optimización del proceso de toma de decisiones.

Por lo antedicho, se puede afirmar que la inteligencia de negocio es un factor estratégico para la organización que genera una ventaja competitiva, que se utilizará para captar nuevos mercados, clientes o para la producción de nuevos productos o servicios.

En este capítulo, se describirán los almacenes de datos o *Data Warehouse* y los *Data Marts*, que son los principales componentes de orígenes de datos de BI (*Business Intelligence*, Inteligencia de Negocio).

## 6.2.1 Datos, Información y Conocimiento

En general, estos términos se consideran sinónimos; sin embargo, entre ellos existen grandes diferencias: los datos son valores ya conocidos que se encuentran disseminados en diferentes partes; la información, en cambio, es un dato asociado a una relación que, vinculado y acumulado luego de un proceso de análisis, se transforma en conocimiento; éste último está en poder de diferentes agentes, como personas u organizaciones.

### 6.2.1.1 Datos

Los datos son la mínima unidad semántica que se corresponden con los elementos primarios de la información que, en sí mismos, no tienen ningún valor. Para brindar algún tipo de información necesitan que se los vincule con alguna relación. También se pueden tomar como un conjunto discreto de valores absolutos que no pueden aportar nada que contribuya con la toma de decisiones. Por ejemplo: un número, un nombre, etc. Asimismo, un CD, un DVD o un disco rígido pueden almacenar físicamente una colección de datos.

Los datos, en general, provienen de diferentes orígenes: internos, es decir, de la propia organización, o externos, extraídos del contexto, por ejemplo, de la competencia. A la vez, pueden ser objetivos, subjetivos y de tipo cualitativo o cuantitativo.

### 6.2.1.2 Información

Se puede definir a la información como el conjunto de datos procesados o relacionados con un significado específico. Por ejemplo, un número aislado no tiene significación alguna como dato; sin embargo, si se encuentra asociado al teléfono de un ejecutivo, indica que tiene una relación con el concepto teléfono y, por lo tanto, se convierte en información útil para un fin específico.

En consecuencia, si se le añade algún valor, los datos se pueden convertir en información. A continuación, se describe de qué manera un dato se puede transformar en información:

- Contextualizando: se sabe en qué contexto y para qué propósito se generaron.
- Categorizando: se conocen las unidades de medida que ayudan a interpretarlos.
- Calculando: los datos fueron procesados matemática o estadísticamente.

# S

Los datos son la mínima unidad semántica que se corresponden con los elementos primarios de la información que necesitan que se los vincule con alguna relación para brindar información.

# S

La información es el conjunto de datos procesados o relacionados con un significado específico.

- Corrigiendo: eliminando errores o inconsistencias de los datos.
- Condensando: resumiendo los datos de forma más concisa (agregación de datos).

Por lo tanto, la información comunica conocimientos o inteligencia y modifica el modo en que los receptores perciben determinadas situaciones. Por esta razón, impacta en sus juicios de valor y en sus comportamientos.

#### 6.2.1.3 Conocimiento

El conocimiento — fusión de valores, información y experiencia — es el marco conceptual adecuado para la incorporación de nueva información. A medida que se va incorporando más información se generan, a la vez, nuevos conocimientos que contribuirán con la toma de decisiones.

El trabajo cotidiano se plasma en prácticas y rutinas que originan determinadas normas y reglamentaciones acerca de “cómo se hacen las cosas aquí” y que constituyen, junto con los almacenes de datos y documentos, el conocimiento de una organización.

Para que la información se convierta en conocimiento, se deben llevar adelante las siguientes acciones:

- Comparación con otros elementos.
- Predicción de consecuencias.
- Búsqueda de conexiones.
- Conversación con otros portadores de conocimiento.



El conocimiento es el marco conceptual adecuado para la incorporación de nueva información. A medida que se va incorporando más información se generan nuevos conocimientos que contribuirán con la toma de decisiones.

## 6.3 Sistemas de almacén de datos

A partir de la década del noventa, los sistemas de almacén de datos o *Data Warehouse* se convirtieron en el centro de la arquitectura de los Sistemas de Información y surgieron para resolver los problemas que acarreaba la extracción de información sintética desde datos atómicos almacenados en bases de datos de producción.

Como se comentará oportunamente, esta clase de sistemas se utilizan, principalmente, como base de información para la toma de decisiones. El acceso interactivo e inmediato a la información relevante de un área de negocios permite al usuario final resolver de manera rápida y eficiente sobre la base de datos objetivos, proporcionados por las bases de datos —a veces, diversas— de la organización. Cuanto más importante sea la decisión que se tomará y cuanto más crítico sea el factor tiempo, mayor será el beneficio que suministren estos sistemas.



Un sistema de almacén de datos incluye diferentes funcionalidades: integración de bases de datos heterogéneas, ejecución de consultas complejas no predefinidas, agrupamiento y desagrupamiento de datos interactivos, análisis de problemas de dimensiones y control de calidad de datos..

Un sistema de almacén de datos incluye las siguientes funcionalidades:

- Integración de bases de datos heterogéneas (relacionales, documentales, geográficas, archivos, etc.).
- Ejecución de consultas complejas no predefinidas que visualicen el resultado en forma de gráfca y en diferentes niveles de agrupamiento y totalización de datos.



El almacén es una base de datos corporativa cuya característica principal es la integración y el filtrado de información de una o varias fuentes, que luego procesará para su análisis desde diferentes punto de vista y con una gran velocidad de respuesta.



El almacén es una colección de datos históricos e integrados diseñada para soportar el procesamiento informático. Para las organizaciones, es una excelente herramienta para la tomas de decisiones estratégicas.



William "Bill" Harvey Inmon. Científico informático, reconocido por ser el creador del concepto "almacén de datos". Introdujo este término como una variante de tiempo de datos en apoyo a las decisiones de la administración de una empresa.

- Agrupamiento y desagrupamiento de datos en forma interactiva.
- Análisis de problemas en términos de dimensiones. Por ejemplo, permite analizar datos históricos a través de una dimensión tiempo.
- Control de calidad de datos para asegurar la consistencia de la base y la relevancia de los datos que contribuirán con la toma de decisiones.

## 6.4 Concepto de almacén de datos

El almacén de datos o Data Warehouse es una base de datos corporativa cuya característica principal es la integración y el filtrado de información de una o varias fuentes, que luego procesará para su análisis desde diferentes punto de vista y con una gran velocidad de respuesta.

Desde un punto de vista técnico, la creación de un almacén de datos constituye el primer paso para la implementación de una solución completa y sustentable de la inteligencia de negocios.

El almacén de datos es una colección de datos históricos e integrados diseñada para soportar el procesamiento informático. Para las organizaciones, el almacén de datos es una excelente herramienta para la tomas de decisiones estratégicas que no utilizan para la operatoria diaria.

Bill Inmon definió por primera vez el término Data Warehouse, cuya traducción literal es "almacén de datos". Sin embargo, es mucho más que esto y tiene, además, determinadas características que se describen a continuación.

## 6.5 Características de un almacén de datos

Está orientado a sujetos

En primer lugar, un almacén de datos no se orienta a los procesos u operaciones clásicas, como en el caso de los sistemas y diseños transaccionales. Su modelo operacional —orientado a los sujetos mayores de la organización— se diseña alrededor de operaciones y funciones; por ejemplo: en el caso de una empresa comercial, para pagos, ventas, etc. Por esta razón, se enfoca en la modelización de datos y en el diseño de la base de datos.

Es integrado

La integración de datos es, sin duda, el aspecto más importante de un almacén de datos. Esto significa que los datos, cuando se mueven desde el ambiente transaccional u operacional, se integran antes de ingresar en el almacén de datos. Por ejemplo, un diseñador puede representar el sexo con las letras "M" y "F"; otro, con los números "0" y "1" o como "x" e "y"; también se pueden utilizar palabras completas: "masculino" y "femenino". Es indiferente la fuente desde la cual el sexo llegue al almacén de datos; lo que importa es que se guarde de una manera consistente para que los datos se integren para que se puedan explotar en sí mismos y sin inconsistencias en sus valores.

### Es temático

Desde el entorno operacional, solamente se añadirán los datos que se necesitan en el proceso de generación de conocimiento del negocio. Estos datos, distribuidos por temas para facilitar la comprensión de los usuarios finales, se pueden reunir en una tabla de un almacén de datos. Como toda la información se encuentra en un mismo lugar, los requerimientos de información acerca de los clientes se responderán sin complicaciones.

### Es variante en el tiempo

Los datos en el almacén de datos son precisos para un cierto momento, no necesariamente ahora. Por eso, se dice que los datos en el *Data Warehouse* son variantes en el tiempo. La variación en el tiempo de los datos se manifiesta de muchas maneras, contiene datos de un largo horizonte de tiempo. Las aplicaciones transaccionales u operacionales, sin embargo, contienen datos de intervalos de tiempo pequeños, por cuestiones de *performance*. Toda estructura clave en un almacén de datos contiene implícita o explícitamente un componente de tiempo. Esto no necesariamente ocurre en el ambiente transaccional u operacional. Al ser datos absorbidos a efectos estadísticos para la toma de decisiones, los datos de un *Data Warehouse*, una vez almacenados, no pueden ser modificados (no se permiten realizar *updates*).

### Es simple de manejar

En una base de datos transaccional, los *updates*, *inserts* y *deletes* se afectan a los datos operacionales. El almacén de datos, por el contrario, opera los datos de una forma más simple, ya que solo necesita dos operaciones: la carga inicial y el acceso a los datos. En este caso no se necesitan *updates*. Ésta es una de las principales diferencias con los procesos llevados a cabo con un sistema operacional o transaccional: en el diseño de un almacén de datos, no se necesitan controlar las irregularidades que producen los *updates*. Por otra parte, en la normalización y en el diseño físico se pueden tomar ciertas libertades como la optimización del acceso a los datos. Además, esta tecnología es muy simple en cuanto a la recuperación, *backups*, *locks*, integridad, etc.

### No es volátil

El almacén de información de un *Data Warehouse* se puede leer, pero no admite ninguna modificación. En consecuencia, la información es inalterable y sus actualizaciones no la cambian. Solo se incorporan las últimas variables.

### Se definen metadatos

Los metadatos —es decir: los datos sobre datos— permiten que se conozca la procedencia de la información, con qué periodicidad se actualiza, su fiabilidad, su forma de cálculo, etcétera y, además, simplifican y automatizan la obtención de información desde los sistemas transaccionales y operacionales.

Los objetivos que deben cumplir los metadatos son:

- Dar soporte al usuario final: lo ayuda a que acceda al almacén de datos con su propio lenguaje de negocio y le señala con qué información cuenta y su significado.
- Dar soporte de auditoría: permite que los responsables técnicos del almacén de datos gestionen la información histórica, la administración del *Data*

*Warehouse*, la elaboración de programas de extracción de la información, la especificación de las interfaces para la realimentación a los sistemas operacionales de los resultados obtenidos, etcétera.

## 6.6 Arquitectura del almacén de datos

Un sistema de almacenamiento de datos o Data Warehousing consta de tres niveles: el primero conformado por las bases de datos fuentes (de producción e históricos); el segundo, es una base de datos resumida extraída de las bases de producción (Data Warehouse) y, el tercero, constituido por las interfaces orientadas a usuarios que extraen información para la toma de decisiones. Las clásicas son: análisis multidimensional, consultas y reportes y Data Mining. La arquitectura lógica de un sistema de almacenamiento de datos se representa en la siguiente figura.

Un sistema de almacenamiento de datos consta de tres niveles: el primero conformado por las bases de datos fuentes (de producción e históricos); el segundo, es una base de datos resumida extraída de las bases de producción (Data Warehouse) y, el tercero, constituido por las interfaces orientadas a usuarios que extraen información para la toma de decisiones.

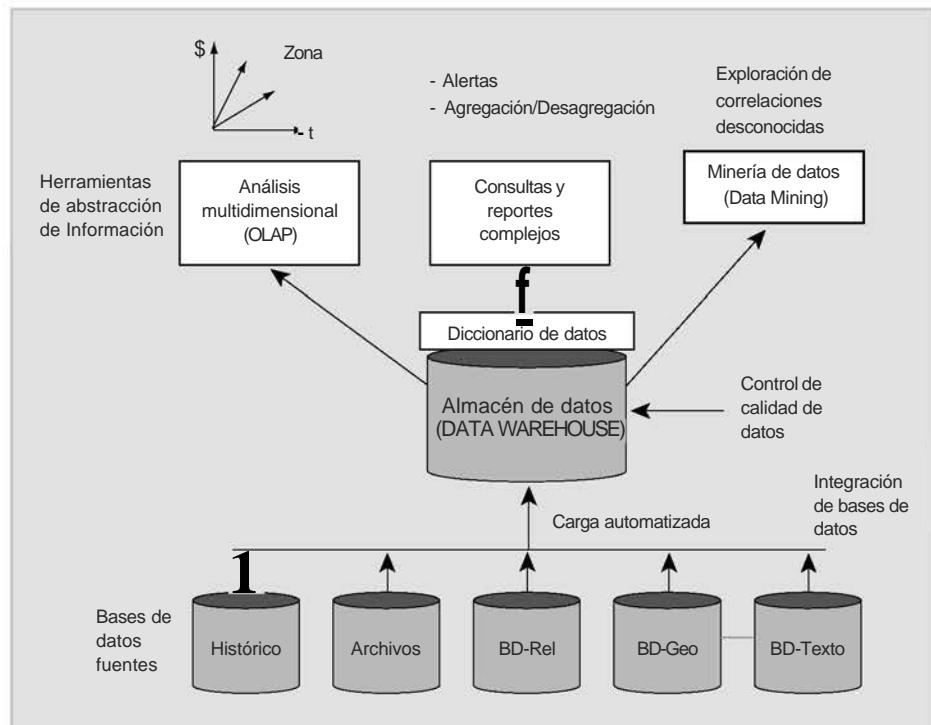


Fig. 6.1 Arquitectura lógica de un sistema de almacén de datos.

A continuación, se detalla cada uno de los niveles que componen la arquitectura de un almacén de datos.

### 6.6.1 Bases de datos fuentes

Éstas son bases de datos de producción que también contienen datos históricos y que, además, se pueden implementar en diferentes tipos de sistemas: BD-Relacionales,

BD-Geográficos, BD-Textos, BD-archivos, etcétera. Tienen en común que en ellos se almacenan ítems de datos atómicos, que revisten cierta importancia como datos de producción, aunque como base para la toma de decisiones resultan demasiado sútiles. Por otra parte, la noción de calidad de los datos en esta clase de bases se fundamenta en la consistencia de estos registros, sin que importe su relevancia dentro del problema.

### 6.6.2 Base de datos con datos resumidos

En esta clase de base de datos se incorporan los datos más destacados que influyen en la toma de decisiones de un área en particular o de la organización en su totalidad. Los datos que almacena un *Data Warehouse* son, principalmente, agrupamientos y totalizaciones de los datos relevantes ubicados en las bases de producción y en los históricos.

En un almacén de datos, el diccionario de datos o *Meta Data* tiene una trascendencia especial ya que describe los datos almacenados con la intención de suministrar su acceso a través de las herramientas de explotación del sistema. Gracias al diccionario de datos, el usuario final puede extraer la información mediante las correspondencias entre los datos del almacén y los conceptos que éstos representan.

### 6.6.3 Interfaces orientadas al usuario

Estas interfaces están compuestas por las herramientas utilizadas para la extracción de la información, entre las que podemos mencionar los análisis multidimensionales OLAP, las técnicas de búsqueda de información oculta (*Data Mining*) —que se tratarán en el capítulo siguiente— y las consultas y reportes planos que se obtengan de un almacén de datos.

## 6.7 Funcionalidades y objetivo

El objetivo de un ambiente del almacén de datos consiste, principalmente, en la conversión de los datos de las aplicaciones del ambiente transaccional (OLTP) en datos integrados de gran calidad. Luego, es necesario que se los almacene en una estructura que facilite el acceso de los usuarios finales en un ambiente destinado a la toma de decisiones (OLAP).

Durante este proceso, la totalidad de los datos se resumen y se incorporan a un almacén de datos. Es decir: se los transfiere —de manera periódica, de acuerdo con el análisis de negocios que se esté tratando— a un almacén de datos.

Las funcionalidades de un almacén de datos se pueden subclasicar en cinco grandes grupos. Cada uno de ellos es responsable de un conjunto de procesos específicos, indispensables para el ambiente de soporte destinado a la toma de decisiones:

- Acceso a Fuentes (*Source*).
- Carga (*Load*).
- Almacenamiento (*Storage*).
- Consultas (*Query*).
- Utilización de Metadatos (*Meta Data*).

Las primeras tres funcionalidades soportan la migración de los datos operacionales al *Warehouse*; la de Consultas, dirige los procesos que sostienen el acceso y el análisis de datos para la toma de decisiones; la de Metadatos, sirve de base para las restantes funcionalidades, puesto que provee los datos que controlan los procesos y las interacciones.



Los factores que recaen directamente sobre el tiempo destinado a las actividades de mapeo, integración y muestreo de datos son: el número de aplicativos fuentes que serán mapeados al almacén de datos, la calidad de los metadatos mantenidos en esas aplicaciones y las reglas de organización que las gobiernan.



La funcionalidad de carga abarca procesos relacionados con la migración de datos. La extracción es el primer paso de la preparación de los datos donde se accede a los datos de los aplicativos. Luego, la depuración consiste en asegurarse que no haya problemas de calidad en los datos. Una de las posibilidades para completar este proceso es realizar una limpieza sistemática de los datos. La conversión es el último paso para la preparación de los datos que se cargarán y necesita reglas de conversión de valores de aplicativos locales a globales e integrados. Cuando finaliza este proceso, se cargan los datos en el almacén de datos.

### 6.7.1 Acceso a fuentes

Esta funcionalidad incluye los procesos que se aplican en las bases de datos fuentes a los datos que se transferirán. Si bien las bases de datos fuentes son las bases de datos operacionales de la organización, en la actualidad se las incluye, cada vez más, a las bases de distribución pública sobre industria, demografía y clientes potenciales. Estos datos llegan, muchas veces, de diferentes fuentes. Por esta razón, es fundamental que se establezca la mejor fuente de datos para, de esta manera, evitar las redundancias innutiles.

Entre un 73 y 80% del tiempo de desarrollo del almacén de datos se destina —durante la fase de análisis y diseño— a los procesos asociados con la función de acceso a fuentes como, por ejemplo, mapeo, integración y muestreo de datos. Lamentablemente, la automatización de estas tareas no es sencilla porque la mayor parte de la información requerida para su desarrollo se encuentra en la mente de los analistas, aunque existen ciertas herramientas que descubren algunos problemas en la calidad de los datos que facilitan la generación de programas para su extracción.

Los factores que impactan directamente sobre el tiempo destinado a estas actividades son: el número de aplicativos fuentes que serán mapeados al almacén de datos, la calidad de los metadatos mantenidos en esas aplicaciones y las reglas de organización que las gobiernan.

### 6.7.2 Carga

La funcionalidad de carga abarca los procesos asociados con la migración de datos —como la extracción, depuración, conversión y carga de datos— desde los aplicativos fuentes hasta las bases del Warehouse.

La extracción es el primer paso de la preparación de los datos y comprende el acceso a los datos de los aplicativos. Para la extracción existen diferentes alternativas que equilibran la performance y las restricciones de tiempo y de almacenamiento.

En el caso de que las aplicaciones fuentes mantengan una base de datos en línea, se puede realizar una consulta que cree directamente los archivos de extracción. Sin embargo, para que no se produzcan inconsistencias es necesario que los datos no se estén actualizando en el mismo momento en que se hace la extracción. Para solucionar este inconveniente, se aconseja la creación de una vista para la extracción de los datos. No obstante, esto trae como consecuencia la necesidad de un espacio de disco adicional para guardar la copia de la base. Como el tiempo es crucial, algunos aplicativos de extracción tienen un ciclo batch, en el que las transacciones fuera de línea se aplican a la base de datos.

Tras la extracción, se debe acceder a los datos para asegurarse de que no haya problemas de calidad. Esta depuración se realiza de diferentes maneras: en el caso de que los errores sean inherentes a los aplicativos fuentes, una posibilidad es la limpieza sistemática de los datos como parte del proceso de conversión. Gran parte de estos errores suceden cuando los aplicativos fuentes tienen una validación mínima de dominio, que contribuye a que aparezcan datos inválidos. Solo hay una solución: correr rutinas pesadas de validación en el nivel de las fuentes. En general, los errores causados por tipeos incorrectos no se detectan con facilidad y es muy complicado resolverlos.

La conversión es el último paso en la preparación de los datos que se cargarán en el Warehouse. Este proceso necesita reglas de conversión de valores de aplicativos locales a globales e integrados. Cuando este proceso finaliza, se cargan los datos en el almacén de datos.

### 6.7.3 Almacenamiento

La funcionalidad de almacenamiento abarca la arquitectura que se necesita para incluir varias vistas en un almacén de datos. Si bien se suele decir que el Warehouse es un único almacén, en realidad, sus datos pueden estar desperdigados en muchas bases que se manejan a través de diferentes DBMS (por sus siglas en inglés, Data Base Management System). Los manejadores que se ajustan a esta tarea son dos: los relacionales (RDBMS) y los multidimensionales (MDDBMS).

En el caso de los MDDBMS, los datos se organizan en un array de n dimensiones. Cada una de ellas representa un aspecto del negocio que se analizará. Las bases de datos multidimensionales no solamente facilitan el acceso a los datos; también contribuyen a que los usuarios los interpreten.

Muchas veces, las diferentes áreas de una organización necesitan sistematizar sus respectivas visiones de los negocios como un array multidimensional que optimice sus requerimientos específicos. Sin embargo, no se aconseja que los requerimientos de todas las áreas sean soportados por la misma base multidimensional. En general, un RDBMS se ajusta más al manejo de una base integrada.

Los datos, que se almacenan en diferentes niveles, se guardan de la siguiente manera: los actuales, en un medio de fácil acceso en línea; los más viejos, en un medio seguro pero más económico; los históricos, en otros medios o eliminados en el caso de que ya no posean un valor decisional.

### 6.7.4 Consultas

El ambiente de consultas —mediante sus herramientas OLAP— permite que el usuario dirija el análisis y la producción de reportes. Nuevas tecnologías prometen soportar la nueva generación de herramientas de análisis: Minería de datos (Data Mining) y simulación de negocios.

Las primeras DM se encargan del análisis de los datos para verificar la existencia de correlaciones inesperadas entre ellos. De entre los objetivos que persiguen estas tecnologías, uno de los más importantes es examinar la efectividad de las reglas de la organización. En cambio, las segundas crean las herramientas necesarias para comprobar el impacto de las transformaciones en el ambiente negocios y establecen, si se considera conveniente, nuevas reglas de organización que realimentarán los aplicativos operacionales.

Establecer cómo se totalizarán los datos, es tarea del arquitecto de un almacén de datos. Esta tarea se puede realizar de diferentes maneras: durante la carga y guardado en el almacén de datos, en la replicación a los Data Marts; o a demanda, por las herramientas de consulta y simulación.

### 6.7.5 Metadatos

El conocimiento de los metadatos es tan esencial como el conocimiento de los datos de un almacén de datos. Incluyen el dominio, las reglas de validación, la derivación y la conversión de los datos extraídos. Además de describir las bases de datos del Warehouse, con las reglas de distribución y control de la migración hacia los Data Marts. Los procesos que monitorean los procesos del Warehouse (como extracción, carga y uso) crean metadatos que se utilizan para determinar cómo se comporta el sistema.

Los datos almacenados en diferentes niveles se guardan de la siguiente manera: los actuales, en un medio de fácil acceso en línea; los más viejos, en un medio seguro, pero más económico y los históricos, en otros medios o eliminados si no poseen valor decisional.

El ambiente de consultas —mediante sus herramientas OLAP— permite que el usuario dirija el análisis y la producción de reportes.

Los metadatos son esenciales para el conocimiento de los datos de un almacén de datos ya que incluyen el dominio, las reglas de validación, la derivación y la conversión de los datos extraídos.



Se denomina *Data Warehouse* a un almacén de datos integrado; *Data Marts*, a las vistas multidimensionales de cada área.



Los *Data Marts* se ajustan mejor a las necesidades que tiene una parte específica de un negocio. Optimizan la distribución de información útil para la toma de decisiones y se enfocan en el manejo de datos resumidos o de muestras.

Los metadatos deben estar disponibles para el análisis que realizan los usuarios. En este caso, los administradores pueden manejar y proveer el acceso a través de los servicios de repositorio.

## 6.8 Almacén de datos y Data Mart

Se denomina *Data Warehouse* a un almacén de datos integrado; *Data Marts*, a las vistas multidimensionales de cada área. La separación que existe entre *Data Warehouse* y los *Data Marts* satélites necesita de una estrategia que coordine la distribución hacia los *Data Marts*. Es importante la implementación de un servidor de replicación que suministre los datos correctos al *Data Mart* en el momento indicado.

Durante muchos años, se denominó almacén de datos a los sistemas de extracción y almacenamiento de datos de diversas fuentes. En la actualidad, se hace una distinción entre grandes y pequeños sistemas de almacenamiento de datos: a los primeros, se los sigue denominando *Data Warehouse*; a los segundos, *Data Marts*. Sin embargo, para hablar del concepto general, se sigue utilizando el término *Data Warehouse*.

Los *Data Marts* se ajustan mejor a las necesidades que tiene una parte específica de un negocio, más que a las de toda una organización. Optimizan la distribución de información útil para la toma de decisiones y se enfocan al manejo de datos resumidos o de muestras, más que a la historia presentada con detalle.

Por otra parte, no es necesario que los administre el departamento de sistemas de una organización, ya que lo puede hacer un grupo específico dentro del área de negocios dedicada a ese tema en particular.

Los *Data Marts* deben su popularidad a que disminuyen de manera significativa los costos asociados a su creación y operación. Por esta razón, muchas organizaciones pueden acceder a él. Con los *Data Marts* se puede llegar a prototipos más rápidamente y obtener sistemas completamente desarrollados e implementados dentro de un período que oscila entre los tres y los seis meses. El problema es que tienen un alcance más limitado que el *Data Warehouse*, ya que se enfocan en un conjunto concreto de necesidades y, por lo mismo, son ideales para trabajar con objetivos puntuales y equipos de trabajo precisos.

A menudo, las pequeñas organizaciones y los departamentos autónomos de una organización prefieren utilizar éstos para construir su propio mecanismo para toma de decisiones. Por esta razón, en muchos departamentos de sistemas existe la tendencia de construir un *Data Warehouse* para un solo tema o un *Data Mart* cada vez que se considere necesario. De esta manera, adquieren cada vez más experiencia y el respaldo de los administradores gracias a los beneficios que genera.

Esta manera de implementar los *Data Marts* permite que las empresas adquieran cada vez más experiencia sobre las fuentes de datos y las necesidades del usuario. Es usual que muchos de los proyectos que se inician como *Data Warehouse* terminen como *Data Marts*. Esto se debe a que las empresas acumulan enormes cantidades de datos históricos que rara vez utilizan. Entonces, para reducir el volumen de información almacenada, optan por la conversión de su *Data Warehouse* en un *Data Mart* más encaminado o por la división del almacén de datos en varios *Data Marts*, que

responden con mayor rapidez, con un acceso más simple. Además, para los usuarios finales es menos complejo.

La duplicación en otro entorno de datos es un término que suele ser mal interpretado e incomprendido. Así, es usado por los fabricantes de DBMS, en el sentido de simple réplica de los datos de un sistema operacional centralizado en sistemas distribuidos.

Cuando se hace referencia al concepto “duplicación en Data Warehouse”, se habla de la creación de Data Marts locales o departamentales que se basan en los subconjuntos de la información contenida en el almacén de datos central o maestro. Por esta razón, se puede definir un Data Mart como “la aplicación de Data Warehouse que se construye rápidamente para soportar una línea de negocio simple”.

Los Data Marts poseen las mismas características que el almacén de datos. Esto es: integración, no volatilidad y orientación temática. Estas características permiten que se utilice, como recurso estratégico, solo una pequeña parte de los contenidos de un almacén de datos.

Se necesitan la segmentación y algunos métodos adicionales de consolidación para la réplica de estos Data Marts.

A continuación, se ilustra una situación posible en la que se podría utilizar una arquitectura descentralizada de Data Mart:

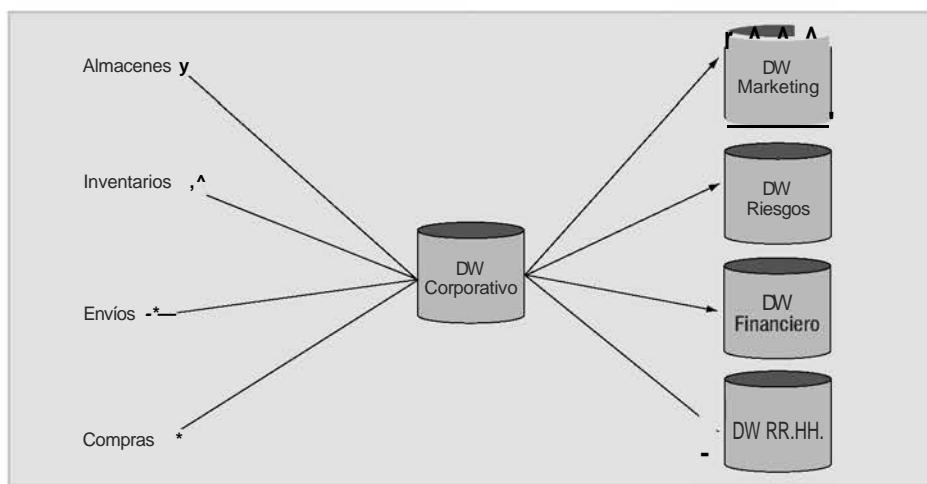


Fig. 6.2 Arquitectura Descentralizada de un Data Mart.

Como se observa en la figura, el departamento de marketing empieza el primer proyecto de Data Warehouse como una solución para su área y, luego, origina el primer Data Mart.

Como la experiencia de este departamento resultó exitosa, otros departamentos —por ejemplo, el de riesgos y el financiero— también crean sus Data Marts propios. Entonces, los diferentes departamentos empiezan a compartir la información. Esta situación es normal porque los Data Marts crecen con el tiempo y su esquema de integración será como se observa en la Fig. 6.3.



Los *Data Marts* poseen las mismas características que el almacén de datos: integración, no volatilidad y orientación temática.

Aquí el esquema de integración de la información de los Data Marts quedó muy desorganizado; sin embargo, lo que falló fue el esquema de integración, no los Data Marts. La manera correcta necesitaría de la coordinación de la información de todos los Data Marts a través de un almacén de datos centralizado.

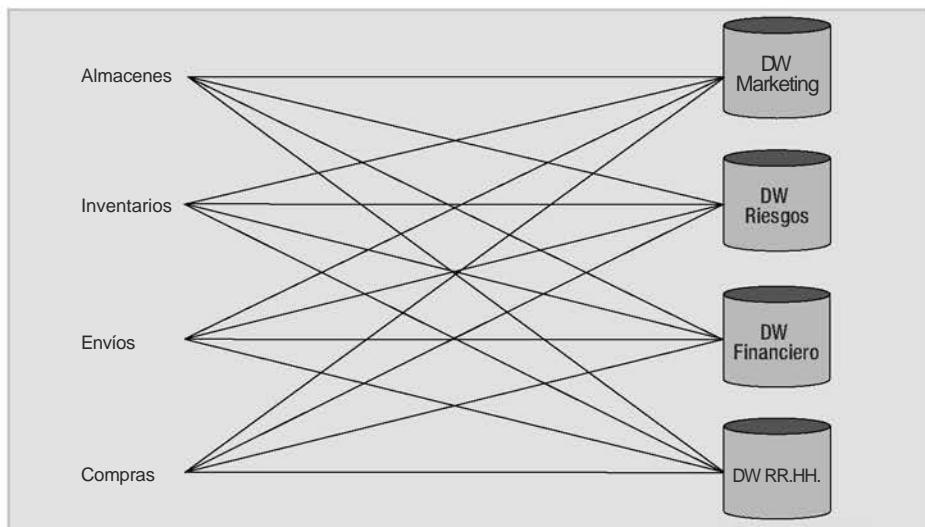


Fig. 6.3 Data Mart centralizado en un Data Warehouse.

De esta manera, el almacén de datos corporativo entrega cargada y depurada la información que necesitan los Data Marts. Así se facilita que la base de conocimientos se amplíe en toda la empresa. La descentralización del trabajo de gestión de los Data Marts, en el almacén de datos corporativo, genera en éstos economías de escala.

Algunas investigaciones recientes sostienen que las implantaciones de Data Warehouse superan en un 50% a las de Data Mart. Sin embargo, es probable que en un futuro cercano la relación sea a la inversa.

Además, estos estudios sostienen que el 5% de los usuarios que cuentan con la tecnología del almacén de datos, están dispuestos a prescindir de ella en el futuro. Es probable, en estos casos, que no se haya realizado un estudio previo de los factores implicados en esta tecnología o que han pasado por una situación inicial y se replantearon su reorganización.

Las causas que provocan los problemas con la información en el almacén de datos se puede deber a uno o a la combinación de los siguientes factores: corrupción o falta de datos, falla de la rutina de migración de datos, inconsistencia de datos no resueltas por el proceso de depuración o construcción inadecuada del proceso de conversión de datos.

## 6.9 La integridad de los datos

De entre los factores que suelen ser la causa de que un proyecto de almacén de datos no funcione, podemos mencionar: la reunión insuficiente de los requerimientos funcionales de un negocio, la falta de personal capacitado o el error de los usuarios al no aceptar el almacén de datos porque la información no es de buena calidad.

Las causas que provocan los problemas con la información en el almacén de datos se pueden deber a uno o la combinación de los factores que se describen a continuación:

- Corrupción de los datos o falta de datos dentro del sistema fuente de origen.
- La rutina de migración de datos falla al mover los datos que son necesarios por los usuarios finales o *Data Warehouse*.
- El proceso de depuración de datos no resuelve las inconsistencias de datos o no verifica correctamente la información seleccionada.
- El proceso de conversión de datos no es construido adecuadamente o contiene reglas de cambio incorrectas y/o el proceso de carga de datos en el *Warehouse* no es construido apropiadamente o falla.

En el próximo apartado, se explicará cómo se debe tratar la integridad de los datos mediante la instauración de un control que los reconozca y modere sus problemas dentro de un almacén de datos.

### 6.9.1 Concepto de Integridad

Se puede definir la integridad como “la información que se adhiere a un estándar estricto de valor y completitud”. Esto significa que los datos dentro del almacén de datos son precisos y que contiene toda la población de datos relevantes.

Como el uso diario de un almacén de datos se apoya exclusivamente en la percepción del usuario, su credibilidad se sostiene por la integridad de sus datos. Para lograr este objetivo, el almacén de datos se debe compaginar para que el usuario y los sistemas tengan la certeza de que los datos están integrados. Cuando se alude al término “compaginar”, se hace referencia a los temas de usuarios que se relacionan con la extracción de datos del *Warehouse*.



La integridad es la información que se adhiere a un estándar estricto de valor y completitud.

### 6.9.2 La perspectiva del usuario final

Como se mencionó más arriba, es fundamental la credibilidad del almacén de datos para que los usuarios no lo cuestionen. En las semanas posteriores a la puesta en producción de un Data Warehouse, los usuarios preguntarán: “¿Puedo confiar en la información del Data Warehouse?”. Incluso, la pregunta del usuario más común que se ha encontrado es: “La información en mi consulta o reporte, ¿es correcta?”. Las respuestas tendrán un impacto inmenso en las percepciones de los usuarios sobre el Data Warehouse.

### 6.9.3 La perspectiva del Sistema de Información

Una vez realizada la carga inicial o una actualización incremental del almacén de datos, es necesario que se verifique si el proceso tuvo éxito o si todavía existen datos en las tablas del Warehouse.

### 6.9.4 Controles de integridad de datos

Los controles de integridad de los datos son los encargados de prevenir el ingreso de datos redundantes, corruptos o sin sentido al almacén de datos, y también de identificar si se cargó toda la información solicitada.

Estos controles, en el flujo de proceso del almacén de datos, garantizan la integridad de los datos.

Se pueden agrupar los controles de integridad de datos en dos categorías: los de prevención y los de detección. Cada uno de ellos trata los datos en diferentes etapas del proceso.

#### Controles de prevención

Los controles de prevención ayudan en la integridad de los datos antes de que se carguen en el almacén de datos. Estos controles se incorporan en los procesos de migración de datos, depuración, conversión y carga, y son los principales medios para prevenir el ingreso de datos redundantes, corruptos o sin sentido al almacén de datos.



Los controles de detección evalúan la precisión y completitud de los datos una vez cargados en el almacén de datos o en cada etapa del proceso. También, son esenciales para detectar la información insuficiente o incorrecta dentro del almacén de datos.

#### Controles de detección

Los controles de detección son aquellos controles que evalúan la precisión y completitud de los datos después de que se cargan en el almacén de datos o en cada etapa del proceso. Estos controles se incorporan en el proceso de reconciliación y son el principal medio para detectar la información insuficiente o incorrecta dentro del almacén de datos.

##### 6.9.4.1 Puntos de control de los datos

En cada etapa de los procesos de migración de datos, depuración, conversión y carga, existe la oportunidad de asegurar la integridad de los datos antes de que ingresen al almacén de datos. Además, el proceso de reconciliación identifica cualquier discrepancia en los datos luego de que han entrado al almacén de datos, o en cada etapa del proceso.

##### 6.9.4.2 Etapas del proceso de datos

A continuación, se analizará cada una de las etapas del proceso, representadas en la siguiente figura:

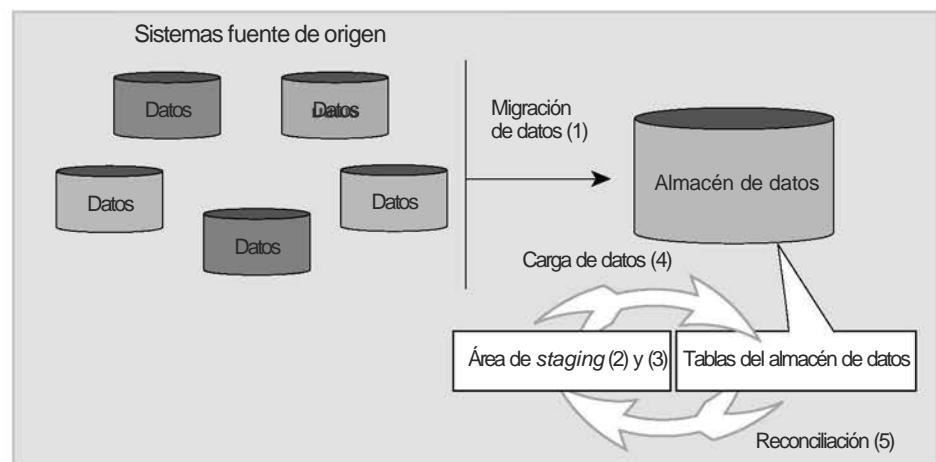


Fig. 6.4 Migración de datos al almacén de datos.

#### 6.9.4.3 Migración de datos (control de prevención)

El propósito de la migración es trasladar los datos desde los sistemas seleccionados de origen hasta el stage del almacén de datos. Solo se moverán los datos solicitados por los usuarios para la emisión de reportes o aquellos que se utilizan durante los procesos de conversión y carga, de esta manera se previene el ingreso de información innecesaria.

Los datos que se moverán al stage del almacén de datos incluirán datos referenciales y transaccionales. Por ejemplo, en un almacén de datos de ventas, los datos referenciales se relacionarán con la información del cliente y los transaccionales serán la información asociada con la venta a un cliente. La información que no se trasladará corresponde a las que se encuentran en las tablas del administrador del sistema RDBMS y a las tablas de aplicación del sistema de origen que contenga metadatos o datos de proceso temporal.

Lo más importante es entender dónde se ubicarán los datos y cuáles se reubicarán. No se debe subestimar esta tarea; en caso de duda, es mejor que se deje los datos afuera, a menos que se sepa qué se hará con ellos.

#### 6.9.4.4 Depuración de datos (control de prevención)

La depuración de datos es corregir para estandarizar el formato y completar cualquier valor requerido por el almacén de datos. Este proceso contribuye con la identificación de los datos redundantes que, durante el proceso de carga, no se ingresarán en el almacén de datos. Para ello, se utilizan herramientas de software que migren, depuren y conviertan los datos. El retorno de la inversión justifica la compra de herramientas de software en lugar de desarrollar scripts en SQL.

Los costos asociados a mantener y ampliar desarrollos propios de scripts SQL excederán significativamente al de comprar herramientas de software desarrolladas por terceros.

El siguiente es un ejemplo simple de depuración de datos.

##### Antes de la depuración:

La tabla de información de cliente que aparece abajo contiene datos que están en un formato no estándar. En los elementos de datos NOMBRE, APELLIDO y EMPRESA la información está en un formato inconsistente. Incluso, al elemento de datos PROVINCIA le falta información.

| Nombre | Apellido | Empresa     | Área | Teléfono  | Provincia |
|--------|----------|-------------|------|-----------|-----------|
| Andres | Perez    | IBM         | 11   | 5355-2299 | BA        |
| Andres | Perez    | I.B.M,      | 11   | 5355-2299 | BA        |
| Andrés | Pérez    | IBM SA      | 11   | 5355-2299 |           |
| Andes  | Peres    | I.B.M. S.A. | 11   | 5355-2299 | BA        |
| Martín | López    | Bunge S.A.  | 351  | 272-2700  | OO        |

### Después de la depuración:

El proceso de depuración estandarizó los datos de la tabla de información de clientes a un formato que ha identificado información redundante y que será excluida del almacén de datos durante el proceso de carga.

| Nombre | Apellido | Empresa     | Área | Teléfono  | Provincia |
|--------|----------|-------------|------|-----------|-----------|
| Andrés | Pérez    | I.B.M. S.A. | 11   | 5355-2299 | BA        |
| Andrés | Pérez    | I.B.M. S.A. | 11   | 5355-2299 | BA        |
| Andrés | Pérez    | I.B.M. S.A. | 11   | 5355-2299 | BA        |
| Andrés | Pérez    | I.B.M. S.A. | 11   | 5355-2299 | BA        |
| Martín | López    | Bunge S.A.  | 351  | 272-2700  | OO        |

#### 6.9.4.5 Conversión de datos (control de prevención)



El objetivo de la conversión de datos es combinar los datos con el formato y la estructura requeridos por el almacén de datos.

El objetivo de la conversión de datos es combinar los datos con el formato y la estructura requeridos por el almacén de datos. El proceso de conversión debería reducir el número de elementos de datos que se cargan desde el stage del almacén de datos.

En el desarrollo de las reglas de conversión para este proceso, solo se utilizarán aquellos elementos de datos que se requieran para el almacén de datos. Si existieran otros que resultaran innecesarios, se prevendrá su ingreso en el almacén de datos y no se los incorporará en las sentencias de conversión o carga.

A continuación, se analizarán las dos maneras de cargar un almacén de datos.

#### 6.9.4.6 Renovación completa

La renovación completa comienza truncando (truncate) las tablas en Data Warehouse y luego cargándolas con todos los datos requeridos. Esta alternativa puede prevenir que datos no deseados ingresen al almacén de datos abarcando condiciones en las sentencias de carga.

Si en cada una de las sentencias de carga se coloca un amplio rango de condiciones, se evita que los datos no buscados ingresen al almacén de datos. Por ejemplo, en el caso de que un almacén de datos contenga información financiera solo en dólares estadounidenses, la sentencia necesitaría una condición que filtre por información financiera solo grabada en dólares estadounidenses; de esta manera, no ingresaría ninguna información en otra moneda.

Cuando el proceso de carga termina, se colocan, en cada una de las tablas del almacén de datos, índices únicos. En caso de que existan valores redundantes en los elementos de datos que se definen por estos índices, el RDBMS no los creará o mostrará un mensaje de error. En tal ocasión, se debe identificar y eliminar los registros de datos dentro del almacén de datos.

Para mejorar la performance del proceso en la sentencia de carga, no deberían existir índices en las tablas de destino, que se crearán una vez finalizado el proceso.

#### 6.9.4.7 Renovación incremental

La renovación incremental identifica los cambios que se produjeron en los datos origen desde la última vez que se cargó el *Data Warehouse* y, luego, inserta, actualiza o borra registros de datos en cada tabla del almacén de datos como se lo solicite.

La carga con renovación incremental puede prevenir el ingreso de datos no deseados al almacén de datos, que abarquen condiciones en las sentencias de carga e índices únicos en las tablas destino.

De la misma manera que con la renovación completa, las condiciones en cada una de las sentencias de carga previenen que los datos no buscados ingresen al almacén de datos; incluso, los índices que se colocan en las tablas destino dentro de este último evitarán que ingresen los datos redundantes. En cada caso, el RDBMS cargará solo los registros únicos de datos en el almacén de datos o se visualizará un mensaje de error.

#### 6.9.4.8 Conciliación del almacén de datos (control de detección)

El proceso de conciliación identifica los problemas de datos que, si no se les diera importancia, pasarían los controles de prevención. Este proceso se diseña para proveer veracidad y para la identificación de los datos que no concuerdan con la información que contiene el sistema de origen. La conciliación de los datos determina la precisión y la integridad de la información. Para ello, se debe analizar:

- La calidad de datos: la exactitud se evalúa con el uso de totales de control sobre los elementos de datos seleccionados, que luego se compararán con los resultados anticipados.
- La cantidad de datos: la integridad se determina cuantificando el número de registros y comparando los resultados con el número de registros anticipados.

Independientemente del enfoque que se utilice, la conciliación de un almacén de datos proveerá una red segura que identificará las excepciones de datos y asistirá con preguntas y perspectiva de direccionamiento en todas las partes interesadas dentro de la organización.

Existen dos enfoques principales para conciliar un almacén de datos:

##### 6.9.4.8.1 Conciliación completa

Al finalizar cada proceso de carga, se realiza una conciliación completa que compara la información del almacén de datos con la del sistema origen correspondiente.

##### 6.9.4.8.2 Conciliación por fase

La conciliación se realiza después de cada etapa del flujo del proceso de datos, cuando no es factible una conciliación completa, debido al número de sistemas de origen o a la complejidad de los procesos de depuración o conversión.

Con la conciliación por fase, se determinan la veracidad e integridad de los datos luego de cada una de las siguientes etapas:



El proceso de conciliación identifica los problemas de datos que, si no se les diera importancia, pasarían los controles de prevención. Este proceso se diseña para proveer veracidad e identificar los datos que no concuerdan con la información que contiene el sistema de origen.

- Migración de datos: después de que los datos del sistema origen han sido migrados al *stage* del almacén de datos, se realiza la conciliación entre los datos del sistema origen y los del *stage* del almacén de datos.
- Depuración: cuando termina el proceso de depuración, se realiza la conciliación entre los datos no depurados, el listado de excepciones y los datos depurados del *stage* del almacén de datos.
- Conversión: una vez que finaliza el proceso de conversión, se produce la conciliación entre los datos depurados, la lista de excepciones y los datos convertidos del *stage* del almacén de datos.
- Carga: después de terminado el proceso de carga, se hace la conciliación entre los datos convertidos del *stage* del almacén de datos.

A medida que los almacenes de datos se convierten en aplicaciones críticas para una organización, la necesidad de asegurar la integridad de los datos aumenta.

El éxito de un almacén de datos descansa en las percepciones que los usuarios tienen de él: si los datos son imprecisos o incompletos, la confianza y su uso disminuirán.

Al establecer un medio que incorpora controles de detección y prevención, se habrán creado los medios para proveer de confianza al usuario y se detectarán las anormalidades en los datos.

## 6.10 Costos y valor del almacén de datos

En todo proyecto se debe realizar un análisis que mida el costo y el valor de lo que se está haciendo; por ello, en este apartado se discutirán, por un lado, los beneficios que aporta a la organización el almacén de datos y el costo de su construcción, mantenimiento y operación. Por otro, el valor agregado que implica la obtención de información de calidad que contribuirá con una mejora en la toma de decisiones y en los procesos de la organización.

### 6.10.1 Costos de un almacén de datos

Como se mencionó en el párrafo precedente, los costos de un almacén de datos se pueden dividir en: por un lado, los relativos a su construcción y mantenimiento; por otro, los relacionados con su operación. A continuación, se analizarán en detalle:

#### 6.10.1.1 Costos de construcción

Son similares a los de cualquier proyecto de tecnología de información y se pueden clasificar en tres categorías:

- Recursos Humanos: se necesita personal idóneo y con un gran conocimiento del área en la que se implementará el almacén de datos y de los procesos de la organización. Además, deberán ser capaces de interactuar y compartir sus destrezas con los especialistas tecnológicos, para superar los desafíos que implica el desarrollo del almacén de datos.

- Tiempo: se debe establecer para la construcción y entrega de los resultados de un almacén de datos, un límite temporal. Este límite también es fundamental para el plan del proyecto y la definición de la arquitectura, puesto que los dos establecen el marco de referencia y los estándares que son críticos para la eficacia del almacén de datos.
- Tecnología: El almacén de datos introduce gran parte de las nuevas tecnologías. El costo de la nueva tecnología puede ser tan solo la inversión inicial del proyecto.

#### 6.10.1.2 Costos de operación

Luego de la construcción y entrega de un almacén de datos, se necesita, para que éste obtenga un valor organizacional, de ciertas actividades de soporte que son la fuente de continuos costos operacionales.

Se pueden distinguir tres tipos de costos de operación:

- Evolutivos: a medida que transcurre el tiempo, el almacén de datos necesitará ajustes continuos relacionados con los cambios de expectativa, que son el producto del aprendizaje de los integrantes del proyecto que van adquiriendo experiencia en su implementación.
- Crecimiento: se refiere al aumento en el tiempo de los volúmenes de datos y de la cantidad de usuarios del almacén de datos. Esto implica, también, el crecimiento de los recursos necesarios como la demanda de monitoreo, administración y sintonización del *Data Warehouse*. De esta manera, se evita el incremento en los tiempos de respuesta y de recuperación de datos.
- Cambios: se necesita que el almacén de datos soporte los cambios que ocurren en el origen de datos que usa y en sus necesidades de información.

Los costos evolutivos y de crecimiento son básicos en la manutención de cualquier sistema de información. En cambio, en los costos de operación por cambios se debe considerar cómo impactan el modelo OLTP y el ambiente organizacional con el almacén de datos.

Resulta esencial para llevar a cabo un proyecto de *Data Warehouse*, tener claridad en la forma en que se ve afectado por medio de cambios a nivel del modelo OLTP como del ambiente organizacional.

Cuando se implementa un almacén de datos, el impacto de cambios es compuesto. Existen dos orígenes primarios de cambios:

- Cambios en el ambiente organizacional: un cambio en el ambiente organizacional puede modificar las necesidades de información de los usuarios. De esta manera, el contenido del almacén de datos se puede ver afectado y las aplicaciones DSS (sistemas de soporte para las decisiones) y EIS (sistemas de información para ejecutivos) pueden requerir cambios.
- Cambios en la tecnología: un cambio en la tecnología puede afectar la manera en que se almacenan los datos operacionales. Esto implicaría un ajuste en los procesos de extracción, transporte y carga para adaptar las variaciones presentadas.



Los costos evolutivos y de crecimiento son básicos en la manutención de cualquier sistema de información. Sin embargo, en los costos de operación por cambios se debe considerar cómo impactan el modelo OLTP y el ambiente organizacional con el almacén de datos.

Cualquier modificación que se produzca en los orígenes primarios de cambio afectará en los sistemas operacionales. En el caso de que se produzcan en el ambiente organizacional, es posible que varíe el formato, la estructura o el significado de los datos operacionales que se utilizan como origen para un almacén de datos. Por esta razón, se afectarían los procesos de extracción, conversión y carga de datos.



El valor de un almacén de datos se describe en tres dimensiones: mejorar la entrega de información, mejorar el proceso de toma de decisiones y el impacto positivo sobre los procesos organizacionales.

### 6.10.2 Valor del almacén de datos

El valor de un almacén de datos se describe en tres dimensiones:

1. Mejorar la entrega de información: información completa, correcta, consistente, oportuna y accesible. Información que la gente necesita, en el tiempo y en el formato que la necesita.
2. Mejorar el proceso de toma de decisiones: con un mayor soporte de información se obtienen decisiones más rápidas; la gente de negocios adquiere mayor confianza en sus propias decisiones y las del resto y, además, logra un mayor entendimiento de los impactos de sus decisiones.
3. Impacto positivo sobre los procesos organizacionales: cuando a la gente se le da acceso a una mejor calidad de información, la organización se puede lograr por sí sola:
  - Eliminar los retardos de los procesos organizacionales que resultan de la información incorrecta, inconsistente y/o no existente.
  - Integrar y optimizar procesos organizacionales a través del uso compartido e integrado de las fuentes de información.
  - Eliminar la producción y el procesamiento de datos que no son usados ni son necesarios, producto de aplicaciones mal diseñadas o que no se utilizan.



Un proyecto de almacén de datos implica una estrategia a largo plazo. En consecuencia, se deben evaluar sus costos y beneficios en un período razonable de tiempo. En general, el retorno de la inversión no es inmediato. Por eso, no se aconseja un cálculo de costo/valor para el corto plazo, ya que los costos serán mucho más elevados que el valor.

### 6.10.3 Balance entre los costos y el valor

Determinar la ponderación de los factores de valor no es una tarea sencilla ya que, muchas veces, puede resultar complejo y con un alto grado de subjetividad. Una manera de cuantificarlos es desde el punto de vista de los costos evitables. Es decir, por el precio que paga la organización por no contar con la información adecuada en los niveles técnicos y en los procesos de la organización, como la toma de decisiones.

Un proyecto de almacén de datos implica una estrategia a largo plazo. En consecuencia, se deben evaluar sus costos y beneficios en un período razonable de tiempo. En general, el retorno de la inversión no es inmediato y, por lo tanto, no se aconseja un cálculo de costo/valor para el corto plazo, puesto que los costos serán mucho más altos que el valor.

## 6.11 Impactos de la implementación de un almacén de datos

Como se observó en las páginas precedentes, para que un almacén de datos sea exitoso debe mejorar los procesos de la organización y contribuir con la toma de

decisiones. Para su correcta implementación, es necesario que se comprenda cómo impactará en los siguientes ámbitos:

### 6.11.1 Recursos Humanos

- Construcción del almacén de datos: necesita la participación de los integrantes de las áreas involucradas en el proyecto. La construcción de un almacén de datos está estrechamente relacionada con la realidad de la empresa y de las condiciones objetivas de ese momento que determinarán las variables que se incluirán.
- Anexando al almacén de datos: los usuarios pueden agregar información cuando lo requieran. Esto tiene varias consecuencias:
  - a. La gente de la organización puede necesitar aprender nuevas destrezas.
  - b. Los análisis extensos y demoras de programación para obtener información serán eliminados. Como la información estará lista para ser agregada, las expectativas probablemente aumentarán.
  - c. Pueden existir nuevas oportunidades en la comunidad que compone la organización para los especialistas de información.
  - d. La gran cantidad de reportes en papel será reducida o eliminada.
  - e. La madurez de un almacén de datos dependerá del uso activo y de la retroalimentación de sus usuarios.
- Usando aplicaciones DSS/EIS: usuarios de aplicaciones DSS y EIS necesitarán menos experiencia para construir su propia información y desarrollar nuevas destrezas.

### 6.11.2 Impactos organizacionales

Provocan determinados efectos en la organización, particularmente en:

- Procesos y decisiones organizacionales: se deben considerar los beneficios organizacionales potenciales de los siguientes impactos:
  - a. Los procesos de toma de decisiones se pueden mejorar gracias a la información disponible. De esta forma, se consiguen decisiones organizacionales más rápidas y con gente más informada.
  - b. Los procesos organizacionales pueden ser optimizados. El tiempo perdido esperando por información que finalmente es incorrecta, o que no se encuentra, es eliminado.
  - c. Conexiones y dependencias entre procesos organizacionales se vuelven más claros y entendibles. Las secuencias de los procesos organizacionales se optimizan para ganar eficiencia y reducir costos.
  - d. Se utilizan y se examinan los procesos y datos de los sistemas operacionales, así como los datos en el *Data Warehouse*. Cuando los datos se organizan y se estructuran para tener un significado organizacional, la gente aprende mucho de los sistemas de información. Pueden quedar expuestos posibles defectos en aplicaciones actuales y es posible entonces mejorar la calidad de nuevas aplicaciones.

- Comunicación e impactos organizacionales: apenas el almacén de datos comienza a ser fuente primaria de información organizacional consistente, se pueden presentar los siguientes impactos:
  - a. La gente tiene mayor confianza en las decisiones organizacionales que se toman. Ambos, quienes toman las decisiones como los afectados, conocen que está basada en buena información.
  - b. Las organizaciones y los recursos humanos que la componen quedan determinados por el acceso a la información. De esta manera, la gente queda mejor habilitada para entender su propio rol y responsabilidad y también los efectos de sus contribuciones; a la vez, desarrollan un mejor entendimiento y apreciación de las contribuciones de otros.
  - c. La información compartida conduce a un lenguaje y conocimiento común, que afianza la comunicación en la organización. Se mejora la confianza y cooperación entre distintos sectores de la organización, viéndose reducida la sectorización de funciones.
  - d. Visibilidad, accesibilidad y conocimiento de los datos producen mayor confianza en los sistemas operacionales.

#### 6.11.3 Impactos técnicos del almacén de datos

La construcción y soporte del almacén de datos y el soporte de sistemas operacionales tienen los siguientes impactos técnicos:

- Nuevas destrezas de desarrollo: la construcción de un almacén de datos, implica la adquisición de nuevas destrezas; por ejemplo:
  - a. Conceptos y estructura del *Data Warehouse*.
  - b. El almacén de datos introduce muchas tecnologías nuevas, como la carga y acceso de datos, el catálogo de metadatos, la implementación de aplicaciones DSS e EIS y cambia la manera en la que se utiliza la tecnología. Nuevas responsabilidades de soporte, nuevas demandas de recursos y nuevas expectativas son los efectos de estos cambios.
  - c. Destrezas de diseño y análisis en los que los requerimientos organizacionales no son posibles de definir de una forma estable a través del tiempo.
  - d. Técnicas de desarrollo incremental y evolutivo.
  - e. Trabajo en equipo con el personal del área de negocios, como participantes activos en el desarrollo del proyecto.
- Nuevas responsabilidades de operación: las modificaciones en los sistemas y en los datos operacionales se deben estudiar con cautela para determinar el impacto que tienen sobre ellos y sobre el almacén de datos.

#### 6.11.4 Consideraciones finales

Por último, un proyecto de almacenamiento de datos que consigue satisfacer las necesidades organizacionales es exitoso. Su implementación siempre trae aparejado los siguientes cambios en la empresa:

- La gente de la organización depende del almacén de datos como un recurso primario de información.
- La gente de organización se vuelve menos dependiente de los sistemas operacionales y de sus bases de datos para sus necesidades de información.
- Se ve reducida o eliminada la demanda por programación especializada para encontrar la información necesaria.
- Los usuarios y uso de un Data Warehouse crecen, con un correspondiente incremento en la demanda de soporte.
- La complejidad de cambios en los sistemas operacionales se incrementa y su efecto sobre un almacén de datos debe ser considerado.

## 6.12 Estrategia recomendada para la implementación de un almacén de datos

### 6.12.1 Prototipo

El prototipo tiene la finalidad de acercar a los usuarios finales una aproximación de lo que un almacén de datos puede hacer por la organización en un período razonable. El grupo de trabajo de un almacén de datos necesita demostrar en este tiempo los beneficios a los usuarios finales y obtener de ellos un *feedback*.

El prototipo es una simulación del producto que se entregará a los usuarios. En *Data Warehouse* debe tener las siguientes funcionalidades:

- Se deben entregar integrados y cargados en estructuras de datos apropiadas.
- Deben ser distribuidas las herramientas de acceso de datos a los usuarios finales y sus aplicaciones para realizar consultas.
- Se deben crear herramientas de soporte en la decisión, si es aplicable. Sin embargo, el proceso de integrar y convertir los datos no será completamente automatizado. En la mayoría de los casos, el prototipo contemplará una carga no repetible (de una sola vez) de los datos de las estructuras del almacén de datos. La plataforma y la base de datos para el almacenamiento pueden también diferir de aquellas para la arquitectura definitiva del almacén de datos.
- Debe existir una presentación de los datos al usuario final que sea tan fel como sea posible para que sea igualmente presentada en posteriores etapas del almacén de datos. En la mayoría de los casos, la herramienta que será utilizada en el desarrollo es la misma que la que se ha utilizado para el prototipo.

### 6.12.2 Piloto

El piloto simplemente es el primero de muchos esfuerzos iterativos que se harán para llegar a la construcción de un almacén de datos. Se debe tener especial cuidado porque es la primera fase del proyecto, en la que el equipo de trabajo utilizará los métodos, técnicas y herramientas que serán la base para un almacén de datos completo. Por esta razón, el proyecto piloto debe tener un pequeño alcance y una consideración de tiempo adicional comparativamente grande para los esfuerzos sucesivos.



El prototipo es una simulación del producto que se entregará a los usuarios que debe tener las siguientes funcionalidades: entregar integrados y cargados en estructuras de datos apropiadas, distribuir las herramientas de acceso de datos a los usuarios finales y aplicaciones para realizar consultas, crear herramientas de soporte en la decisión y presentar los datos al usuario final.



El piloto es la primera fase del proyecto en la construcción de un almacén de datos. En esta etapa se utilizarán los métodos, técnicas y herramientas que serán la base para un almacén de datos completo.



La arquitectura de un almacén de datos se diseña de acuerdo a la estructura interna de los datos y del tipo de consultas que se realizarán. Es importante extraer, con periodicidad, los datos de los sistemas de aplicación y de otras fuentes del almacén de datos que luego deben guardarse en la capa de *Data Scrubbing*.

### 6.12.3 Prueba del concepto tecnológico

Si bien es un paso opcional, esta prueba determina si la arquitectura del almacén de datos será un fiel reflejo de la que se quiere implementar. También sirve como prueba de concepto para la arquitectura técnica. Sin embargo, no se recomienda que esta prueba sea cercana al prototipo, puesto que el objetivo de este último es que los usuarios tengan datos lo más rápido posible. La factibilidad técnica durante el prototipo puede traer aparejado grandes riesgos.

### 6.12.4 Arquitectura de un almacén de datos

La arquitectura de un almacén de datos se diseña en función de la estructura interna de los datos y, especialmente, del tipo de consultas que se realizarán. En primer lugar, es importante que se extraigan con cierta periodicidad los datos de los sistemas de aplicación y de otras fuentes del Data Warehouse; luego se los debe guardar en la capa de Data Scrubbing. Aquí se normalizan los paquetes de datos para que no se produzcan fragmentaciones y ambigüedades. En muchos casos, la extracción se realizará con los programas específicos que acompañan estas tareas. El Data Scrubbing se debe hacer con los programas desarrollados para este fin o con la ayuda de scrubbing.

Es importante que el corazón del almacén de datos se organice teniendo en cuenta las características del negocio. Por esta razón, se necesita que la estructura de datos se encuentre ligeramente normalizada a nivel del modelo entidad/relación. No obstante, si se observaran los atributos, se podría dar el caso en que la estructura de datos estuviera sin normalización. Esta aproximación asegura que el corazón del almacén de datos siempre representa el negocio en su totalidad y sus datos, independientemente de la cantidad de usuarios que observen o ingresen a esos datos en particular. Esto tiene especial relevancia porque el modo en que se utilizará la información sufrirá cambios frecuentes y hace falta una base de datos estable para soportarlos.

El corazón del almacén de datos es un conjunto estable de datos; el almacén de datos mantiene las reglas lógicas del negocio implícitas en los datos, cuyo acceso es difícil para los usuarios finales. Para que éstos puedan acceder, se necesita una serie de Data Marts, que consisten en datos extraídos del corazón del DW y reorganizados y/o formateados para hacer más fácil su uso para diferentes propósitos. Sin embargo, como éstos cambian a largo del tiempo, los Data Marts se deben concebir sobre estructuras de datos temporales que se puedan modificar.

Cuando los usuarios no requieran ver más los datos en la forma como están presentados en un Data Mart en particular, éste se debe remover para no generar confusiones ni almacenamientos adicionales a los requeridos.

De igual forma, cuando los usuarios desarrollen nuevas formas de hacer búsquedas y observar los datos, se deben crear nuevos Data Marts para satisfacer dichas búsquedas de una manera más sencilla y con un mejor rendimiento de acceso.

Los Data Marts incluyen una gran variedad de estilos de tablas; algunas de ellas pueden ser un subconjunto de datos del almacén de datos que describa, por ejemplo, una zona geográfica, una unidad de negocios específica, entre otros. También, los Data Marts pueden provenir de la recolección de información de distintas tablas dentro del corazón del almacén de datos: por ejemplo, en una tabla desnormalizada.

Otros pueden contener elementos que se haya calculado y derivado porque no estaban explícitamente almacenados en el corazón del almacén de datos.

En cuanto al almacenamiento físico, cabe mencionar que el uso de estructuras de datos multidimensionales debería estar reservado para los *Data Marts*. Esto significa que los datos que están en el corazón del almacén de datos deberían almacenarse en forma relacional sobre un DBMS, para disminuir el espacio de almacenamiento y potenciar su velocidad de acceso y, luego, ser extraídos en un *Data Mart* multidimensional con el fin de optimizar el proceso de desglose de dimensiones aportado por las BDM.

Las herramientas de acceso de usuario final son un componente crítico de los almacenes de datos. Para la mayoría de los usuarios finales, la herramienta de acceso es el *Data Warehouse*; sin embargo, no lo son, ni tampoco deberían; esto se debe a la compleja arquitectura de datos y análisis que está detrás de la información que ellos ven en sus pantallas. Por lo tanto, es crítico proveer a los usuarios de un método apropiado para utilizar la información de los almacenes de datos.

No se debe esperar que un usuario novato negocie una compleja y poderosa herramienta, solo para hacer una simple pregunta de *Data Warehouse*, en forma similar un usuario avanzado. Rápidamente quedará frustrado si espera realizar un complejo análisis de negocio usando una herramienta de acceso con menos poder del que se necesita. Es importante reconocer que hay diferentes estilos de usuarios finales, cada uno con su propio nivel de conocimiento y necesidades, para así proveer de apropiados mecanismos de acceso para cada clase de usuarios.

### 6.12.5 Acceso a datos de usuario finales

Para estos casos, se deberá desarrollar una aplicación destinada a obtener consultas directas sobre el almacén de datos, de forma tal que el usuario tenga diferentes opciones intuitivas para elegir y, de esta forma, realizar consultas. Esto puede realizarse a través de pantallas de consulta genérica donde el usuario solo elija patrones de interés a buscar, los campos o las columnas que desea ver, el periodo de tiempo y el rango de información.

### 6.12.6 Factores de riesgo

En la construcción de un almacén de datos, es necesario que se identifiquen los factores de riesgo porque impactarán a la hora de implementarlos.

Los principales factores de riesgo son:

- Expectativas de los usuarios: como se dijo repetidas veces, el éxito depende de las expectativas del usuario. Por esta razón, es muy importante que el equipo de trabajo explique qué entregará para que no se cree una percepción errónea acerca del almacén de datos. Además, el usuario debe entender la naturaleza iterativa de la construcción de un almacén de datos.
- Experiencia con *Data Warehouse*: la falta de experiencia puede traer como consecuencia el uso inadecuado por parte de los proveedores y de los consultores.
- Dirección estratégica: es relativamente lógico definir un punto de inicio para el almacén de datos. Sin embargo, cuando esta primera área se haya completado,



En la construcción de un almacén de datos se deben identificar los diferentes factores de riesgo ya que impactarán a la hora de implementarlos. Los principales factores de riesgo son: expectativas de los usuarios, experiencia con *Data Warehouse* y dirección estratégica.

será más difícil identificar áreas para esfuerzos futuros y asegurar que esos esfuerzos estén alineados con los objetivos y necesidades del negocio. El riesgo se puede mitigar siguiendo la estrategia recomendada, para entender las necesidades y prioridades de la información del negocio y desarrollar una implementación de almacenes de datos a largo plazo que cumpla con estas prioridades.

### 6.13 Resumen

---

De lo expuesto en este capítulo podemos concluir que el almacén de datos no es un producto que pueda adquirirse en el mercado, sino la combinación de conceptos y tecnología que modifican la manera en la que se entrega la información a los usuarios de negocios.

Actualmente, el almacén de datos es una de las tecnologías fundamentales para la obtención de información. Por lo tanto, la automatización de los procesos, el control centralizado de un gran caudal de datos y el uso de sistemas de información en línea son los principales pilares dentro de una organización constituyendo un recurso primario trascendental. El manejo eficiente de esta información es primordial en el desarrollo de acciones adecuadas que permitirán un mayor control de la producción.

### 6.14 Contenido de la página Web de apoyo

---



El material marcado con asterisco (\*) solo está disponible para docentes.

#### 6.14.1 Mapa conceptual del capítulo

#### 6.14.2 Autoevaluación

#### 6.14.3 Presentaciones\*

# 7

## Minería de datos

### Contenido

|   |     |
|---|-----|
| 7.1 Introducción.....   | 218 |
| 7.2 Concepto.....   | 218 |
| 7.3 Características de la minería de datos.....                             | 219 |
| 7.4 Capacidades de la minería de datos.....                                 | 220 |
| 7.5 Herramientas algorítmicas de la minería de datos.....                   | 221 |
| 7.6 Modelado de la minería de datos.....                                    | 228 |
| 7.7 Integración entre almacén de datos y minería de datos.....              | 229 |
| 7.8 Ventajas de la minería de datos.....                                    | 230 |
| 7.9 Diferencias entre el análisis estadístico y<br>la minería de datos..... | 230 |
| 7.10 Aplicaciones de la minería de datos.....                               | 232 |
| 7.11 Resumen.....   | 233 |
| 7.12 Contenido de la página Web de apoyo.....                               | 233 |

### Objetivos

- Comprender el *Data Mining* como técnica.
- Obtener información desconocida.
- Identificar ventajas y diferencias de distintos tipos de algoritmos con el Análisis Estadístico.

## 7.1 Introducción

En este capítulo se abordará uno de los tópicos más complejos dentro del campo de la gestión de datos: la minería de datos o Data Mining. Si bien existe una amplia bibliografía acerca de este tema, la intención de este apartado es introducir al lector en su concepto y fundamento y, también, en los componentes y en las técnicas de esta tecnología para que obtenga una aproximación global de su funcionalidad. Por esta razón, estos conceptos y herramientas tecnológicas no se tratarán de manera exhaustiva.



La minería de datos es un conjunto de técnicas que se utilizan para obtener información implícita de las grandes bases de datos. Es una tecnología poderosa que permite a las organizaciones la recolección de información desconocida de sus propias bases.

## 7.2 Concepto

La minería de datos es un conjunto de técnicas que se utilizan para la obtención de la información implícita de las grandes bases de datos. En otras palabras, es una tecnología poderosa y de gran potencial que permite a las organizaciones la recolección de información desconocida de sus propias bases (almacén de datos).

Mediante el análisis completo de la información —en el que se prueban todas las combinaciones posibles que existen entre todos los parámetros en el almacén de datos—, las herramientas de la minería de datos predicen tendencias y comportamientos que contribuyen, gracias al conocimiento exhaustivo que brindan, a una toma de decisiones proactiva que beneficia el desempeño de los negocios.

En esta clase de análisis, las herramientas de la minería de datos están un paso adelante de otras herramientas retrospectivas provistas por los sistemas de soporte de decisión, puesto que responden a preguntas de negocios cuya resolución conlleva, tradicionalmente, un tiempo considerable. Es usual, además, que estas respuestas traigan, como consecuencia, cierta información que los usuarios no están muy predispuestos a aceptar.

La bibliografía especializada en la materia compara los procesos de Data Mining con los que utilizan en la industria minera, que realiza explosiones para encontrar el mineral buscado. De esta analogía nace el término “minería de datos” porque el Data Mining trabaja de manera similar: mediante una “explosión” de datos encuentra la información que éstos ocultan a través de la combinación de todos los patrones entre sí.

Es importante destacar que la minería de datos no es un modelo de almacenamiento: es una tecnología conformada por un conjunto de técnicas algorítmicas que se vinculan directamente con la programación avanzada.

Es usual que se realice una asociación directa de la minería de datos con el almacén de datos. Esta asociación es correcta ya que, como se explicó en el capítulo anterior, modela e integra la información y construye un modelo integrado de datos que la unifica y estandariza; de esta manera, facilita la predicción de futuros acontecimientos que guiarán la toma de decisiones.

La minería de datos se encarga —a través de un conjunto de herramientas y técnicas algorítmicas— de buscar los patrones de interés ocultos, que son los que permiten la anticipación de futuros acontecimientos gracias a la predicción de acontecimientos o al pronóstico de situación con cierto grado de probabilidad.



La minería de datos no es un modelo de almacenamiento: es una tecnología conformada por un conjunto de técnicas algorítmicas que se vinculan directamente con la programación avanzada.

Estas herramientas exploran las bases de datos en busca de patrones ocultos, encontrando información predecible que un experto no puede llegar a encontrar porque está fuera de sus expectativas.

Una de las cualidades de las técnicas de la minería de datos es que se pueden implementar en plataformas ya existentes de software y de hardware que acrecientan el valor de las fuentes de información y, además, también tienen la capacidad para integrarse a los nuevos productos y sistemas recolectados en línea (on-line).

El avance en las computadoras cliente/servidor de alta performance de procesamiento, en las que se implementan las herramientas de la minería de datos, permiten el análisis de bases de datos masivas con gran versatilidad y velocidad.

## 7.3 Características de la minería de datos

La minería de datos es resultado de una larga evolución que se inició el día que las computadoras pudieron almacenar los datos de negocios y continuó cuando se implementaron las primeras mejoras en el acceso de datos hasta llegar —más recientemente— a la implementación de tecnologías generadas que brindan la posibilidad a los usuarios de navegar a través de los datos en tiempo real. De esta manera, y tras sucesivas investigaciones y desarrollos de productos, se llegó a las técnicas de la minería de datos, que van más allá del acceso y la navegación retrospectiva de los datos, ya que entregan información prospectiva y proactiva.

Los procesos de la minería de datos corren sobre bases de datos de gran volumen; esto se produce por dos aspectos fundamentales que se analizarán a continuación:

- **Gran cantidad de columnas:** cuantas más columnas se especifiquen en el almacén de datos, mayor será el nivel de análisis y de detalle en la minería de datos, dado que realiza diferentes combinaciones entre los patrones especificados —en este caso, las columnas predefinidas. Entonces, la cantidad de conclusiones que entregue estará en estrecha relación con el nivel de combinación que realice.
- **Gran cantidad de filas:** para que la minería de datos pueda contrastar los resultados con más tiempo para que, de esta manera, disminuya la cantidad de errores de estimación y desvíos, se necesita que las tablas tengan la mayor cantidad de filas posibles que provean toda la información histórica disponible.

Para que la minería de datos se pueda ejecutar y cumplir con su objetivo, debe tener las siguientes características:

- **Recolección de datos en gran escala:** unifica el contenido de la información de todas las bases de datos disponibles, internas o externas. Como ya se mencionó, se disminuyen los errores y desvíos si la información disponible contiene amplitud y profundidad porque, de este modo, mayor será la aproximación o proyección que se obtenga de la tecnología.
- **Alta tecnología y gran almacenamiento:** como la minería de datos procesa un volumen de información considerable y realiza un importante número de combinaciones, necesita múltiples y veloces procesadores; también requiere



Los procesos de la minería de datos se manejan sobre bases de datos de gran volumen que se producen por dos aspectos fundamentales: gran cantidad de columnas y filas.



Para que la minería de datos se pueda ejecutar y cumplir con su objetivo, debe tener: recolección de datos a gran escala, alta tecnología y un gran almacenamiento y algoritmos de minería de datos.

una gran capacidad de memoria RAM y secundaria (almacenamiento de disco), debido a los procesos intermedios de recolección y combinación de datos e información que ejecuta esta tecnología.

- Algoritmos de minería de datos: se explicarán en este capítulo. Aquí solo se dirá que la minería de datos funciona con la aplicación de diversas herramientas algorítmicas que son las que permiten la búsqueda de información oculta.

Cada paso nuevo se basa en el anterior, ya que es la forma de llegar desde los datos de negocios hasta la información de negocios.

Los componentes esenciales de la tecnología de la minería de datos se han desarrollado —durante décadas— en áreas de investigación como estadística, inteligencia artificial y aprendizaje de máquinas.

En la actualidad, gracias a la madurez de estas técnicas y a los motores de bases de datos relacionales de alta *performance*, este tipo de tecnologías resultan prácticas para los entornos del almacén de datos.

Del mismo modo, los algoritmos de la minería de datos utilizan técnicas que han existido por lo menos desde hace diez años, pero que solo han sido implementadas recientemente como herramientas más maduras, confiables, entendibles y eficientes que los métodos estadísticos clásicos.



El término *Data Mining* deriva de las similitudes que existen entre la búsqueda de información de negocios en las grandes bases de datos y la explotación de una montaña para encontrar una veta de metales valiosos.

## 7.4 Capacidades de la minería de datos

Como ya se mencionó, el término *Data Mining* deriva de las similitudes que existen entre la búsqueda de información de negocios en las grandes bases de datos —como, por ejemplo, las ventas de un determinado producto en los grandes montos de gigabytes almacenados— y la exploración de una montaña para encontrar una veta de metales valiosos.

Estos procesos necesitan del examen de inmensas cantidades de material o de la investigación inteligente que encuentre el lugar en el que residen los valores.

La tecnología de la minería de datos, con bases de datos de suficiente tamaño y calidad, genera nuevas oportunidades de negocios que proveen las siguientes capacidades:

- Predicción automatizada de tendencias y comportamientos. La minería de datos, al automatizar la búsqueda de información predecible en grandes bases de datos, puede inferir, ante una nueva situación o estímulo determinado, cuál sería el comportamiento y la tendencia de respuesta probable de los clientes, proveedores y otros sujetos de la organización. Esto es de vital importancia para los sectores comerciales y de *marketing*, puesto que permite tomar decisiones que disminuirán —al contar con información probable del marco de comportamiento de los diferentes actores— el riesgo de fracaso.
- Obtención automatizada de modelos previamente desconocidos. Si bien una organización cuenta con parámetros conocidos que permiten la evaluación del comportamiento de sus clientes y productos, esto forma parte de la historia de sus bases de datos con respecto a sus ventas y a su comercialización.

Para la identificación de los nuevos patrones de tendencia, es necesaria la utilización de la minería de datos para que barra de un solo paso —a través de sus herramientas algorítmicas de búsqueda de información oculta— las bases de datos e identifique los patrones desconocidos por la organización. Por ejemplo: una empresa puede conocer el patrón de conducta de una clase clientes que adquiere un producto determinado; sin embargo, quizás ignore que el interés por ese producto se debe a otro patrón común que todavía no descubrió. Para descubrirlo, la minería de datos evalúa todos los parámetros que conforman el comportamiento de los actores y los combina con el fin de obtener una heurística de comportamiento que identifique, con una determinada probabilidad, el interés de los clientes por ese producto. De esta manera, las campañas de publicidad y marketing disminuyen sus costos, ya que se dirigen específicamente a ese sector de consumidores.

Las técnicas de la minería de datos pueden redituar los beneficios de automatización en las plataformas de hardware y software existentes y, también, implementarse en sistemas nuevos a medida que las actualicen. Además, facilita el desarrollo de nuevos productos.

Si las herramientas de la minería de datos se implementan en sistemas de procesamiento paralelo de alta performance, analizan bases de datos masivas en minutos.

Esta velocidad de procesamiento ayuda a que el usuario, de manera automática, experimente con más modelos de interpretación de datos complejos. La rapidez permite a los usuarios analizar inmensas cantidades de datos. Como ya se indicó, las grandes bases de datos producen mejores predicciones.

## 7.5 Herramientas algorítmicas de la minería de datos

En las páginas precedentes, se definió a la minería de datos como “el conjunto de técnicas y herramientas algorítmicas que permiten la predicción y el pronóstico del comportamiento de un modelo determinado”.

Dentro de estas técnicas, las más utilizadas son:

- **Redes neuronales artificiales:** son modelos predecibles, de características no lineales que aprenden a través del entrenamiento y semejan la estructura de una red neuronal biológica.
- **Árboles de decisión:** estructuras cuya forma representa la copa de un árbol y que representan conjuntos de decisiones. Estas decisiones son las que generan las reglas que clasifican un conjunto de datos, que se segmentan mediante búsquedas arboladas. Dentro de los métodos específicos de árboles de decisión se incluyen, también, los árboles de clasificación y regresión.
- **Algoritmos genéticos:** son técnicas de optimización con un diseño basado en el concepto de evolución y que utilizan procesos como las combinaciones genéticas, las mutaciones y la selección natural.

Las herramientas de análisis especializadas —que trabajan con volúmenes de datos relativamente pequeños— han utilizado muchas de estas herramientas en los últimos



Las técnicas de la minería de datos pueden redituar los beneficios de automatización en las plataformas de hardware y software existentes e implementarse en sistemas nuevos a medida que las actualicen.

diez años. Sin embargo, la evolución de sus capacidades hace que, en la actualidad, se puedan integrar directamente con las herramientas de OLAP y almacenamiento de datos que, a diferencia de las anteriores, están conformadas por grandes volúmenes de datos de diferente origen, como se comentó en los capítulos anteriores.

A continuación, se hará una descripción global de las técnicas mencionadas. Para un análisis detallado, se recomienda al lector consultar la bibliografía específica sobre minería de datos.

### 7.5.1 Redes neuronales artificiales

La red neuronal artificial es un método de resolución de problemas que, como indica su nombre, emula el modo de conexión de las neuronas del cerebro. Esta red posee capas de unidades procesadoras —nodos— que se unen por conexiones direccionales; es decir: una capa de entrada se conecta a cero o más capas ocultas que, a la vez, se vinculan con la de salida. Un patrón inicial estimula la capa de entrada y, luego, sus nodos transmiten una señal a los de la capa siguiente.

Si la suma de todas las entradas que ingresan en una de estas neuronas virtuales es mayor que el umbral de activación de la neurona, ésta se activa y transmite su propia señal a las de la siguiente capa.

Por lo tanto, el patrón de activación se propaga hasta que llega a la capa de salida, que lo devuelve como solución a la entrada presentada. De la misma manera que en el sistema nervioso de los organismos biológicos, con el transcurso del tiempo, una red neuronal aprende y afina su rendimiento gracias a la repetición de rondas en las que ajusta sus umbrales hasta que, para cualquier entrada, la salida real coincide con la deseada.

A este proceso —denominado entrenamiento de la red— lo puede supervisar un experimentador humano o puede correr automáticamente con un algoritmo de aprendizaje que lo optimice de manera constante.

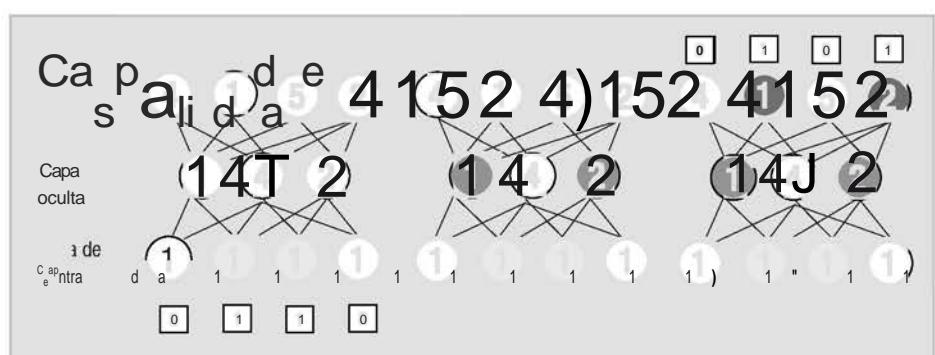


Fig. 7.1 Red Neuronal.

Dado que la activación se produce por los 1 recibidos, solo se activan en la capa oculta la primera y la tercera neurona que contienen un 1 y un 2. La 4 no se activa por no recibir al menos su valor como estímulo. Del mismo modo, en la capa de salida solo se activan las neuronas con contenido 1 y 2 porque el estímulo recibido no es suficiente para las que tienen valor 4 y 5.

# S

La red neuronal artificial es un método de resolución de problemas. Esta red posee capas de unidades procesadoras que se unen por conexiones direccionales.

El umbral de activación de cada neurona se representa por su número. Para que se excite debe recibir, por lo menos, esa cantidad de entradas. El diagrama muestra cómo la red neuronal recibe una cadena de entrada y cómo la activación se extiende por la red hasta producir una salida.

En el apartado siguiente se tratarán los algoritmos genéticos del tipo evolutivo, que son los que se utilizan para la construcción y el entrenamiento de redes neuronales. Sin embargo, existen otros algoritmos de optimización, como los de ascenso a colina o voraces y el recocido simulado. A continuación, se hará una breve descripción del funcionamiento de estas técnicas:

#### 7.5.1.1 Ascenso a colina

Son similares a los genéticos pero con una mayor sistematización y menor aleatoriedad. Un algoritmo de ascenso a colina comienza con la solución de un problema que tiene a mano que, normalmente, se elige al azar. Después, la cadena se muta y, si ésta proporciona una solución con mayor amplitud que la anterior, se conserva la nueva; en caso contrario, la actual.

Este algoritmo se repite hasta que no se pueda encontrar una mutación que provoque un incremento en la aptitud de la solución actual y ésta se devuelve como resultado.

Esta técnica se denomina ascenso a colina porque, en general, se representa con un paisaje en el que se encuentran todas las soluciones posibles de un problema particular. Cada solución, a la vez, se constituye por un conjunto de coordenadas de ese paisaje. La mejores soluciones están a mayor altitud y forman colinas y picos; las peores, a menor altitud y forman valles. Un “trepacolinas” es, en consecuencia, un algoritmo que se inicia en un punto del paisaje y se mueve hacia arriba de la colina.

Este tipo de algoritmo también se lo denomina “voraz” porque siempre hace la mejor elección en cada paso, con la esperanza de que se obtendrá el mejor resultado global. Otros métodos —como los algoritmos genéticos y los de recocido simulado—, en cambio, no son voraces. Es usual que, de tanto en tanto, realicen, al principio, elecciones menos óptimas con la ilusión de que conducirán a una mejor.

#### 7.5.1.2 Recocido simulado

El recocido simulado es otra de las técnicas de optimización parecida a los algoritmos evolutivos. Su nombre proviene del proceso industrial que consiste en calentar un material por encima de su punto de fusión y, luego, enfriarlo para eliminar los defectos en su estructura cristalina, que produce un entramado de átomos estable y regular. En el recocido simulado —de la misma manera que en los algoritmos genéticos— una función de aptitud define un paisaje adaptativo aunque, en este caso, solo existe una solución candidata. Esta clase de recocido añade, además, el concepto de temperatura, que es una cantidad numérica global que disminuye de manera gradual. En cada uno de sus pasos, esta solución muta, que es lo mismo que desplazarse hacia un punto adyacente en el paisaje adaptativo.

La aptitud de la nueva solución se compara con la anterior y, si es mayor, se la conserva. Si ocurre lo contrario, el algoritmo decide si la conserva o la descarta sobre la base de la temperatura. Si ésta es alta, se conservan incluso los cambios que causan disminuciones significativas en la aptitud y se utilizan para la siguiente ronda. Sin embargo, a medida que disminuye la temperatura, el algoritmo tiende a aceptar solo los cambios

Un algoritmo de ascenso a colina comienza con la solución de un problema se elige al azar. Después, la cadena se muta y, si ésta proporciona una solución con mayor amplitud que la anterior, se conserva la nueva; en caso contrario, la actual.



John Henry Holland. Científico y profesor de Psicología, Electromecánica y Ciencias de la Computación. También es un pionero en sistemas complejos y ciencia no lineal. Además, es considerado el padre del Algoritmo genético (AG).

que aumentan la aptitud. Finalmente, cuando la temperatura alcanza el cero y el sistema se “congela”, la configuración que exista en ese punto se convierte en la solución.

El recocido simulado también se aplica en la ingeniería del diseño y puede determinar, por ejemplo, la disposición física de los componentes en un chip informático.

### 7.5.2 Algoritmos Genéticos

Los Algoritmos Genéticos (AG) fueron inventados en 1975 por John Holland, de la Universidad de Michigan. Los AG son algoritmos de optimización, es decir, tratan de encontrar la mejor solución a un problema dado entre un conjunto de soluciones posibles. Los mecanismos que utilizan los AG para llevar a cabo esa búsqueda consisten en procesos que se asemejan a la evolución biológica, de allí el nombre de algoritmos genéticos.

Por esta razón, un AG opera de la siguiente manera: dada una determinada función objetivo, realiza una búsqueda en el espacio de soluciones e intenta encontrar la más eficiente.

Como conclusión, se puede afirmar que esta clase de algoritmos sirven para resolver los problemas de optimización. Solo es necesario que se encuentre la representación adecuada para cada una de las soluciones y la función que se deberá optimizar.

Básicamente, los AG operan de la siguiente manera: dada una población de soluciones, y sobre la base del valor de la función objetivo para cada uno de los individuos (soluciones) de esa población, se seleccionan los mejores individuos (que son aquellos que minimizan la función objetivo) y se combinan para generar otros nuevos. Este proceso se repite cíclicamente hasta probar todas las combinaciones y encontrar la óptima. Como se observa, existe cierta similitud con el proceso que se da en la naturaleza, en el que los individuos compiten por su supervivencia. Los que mejor están adaptados al medio —es decir, los que pueden optimizar la función objetivo— sobreviven y transmiten su material genético a las futuras generaciones.

#### 7.5.2.1 Funcionamiento

##### Bases biológicas

Como se manifestó en el apartado anterior, los algoritmos genéticos se basan en los principios y en la evolución de la naturaleza. Para que el lector comprenda su funcionamiento, se explicará, brevemente, cómo funcionan los mecanismos de la evolución desde una perspectiva biológica.

Como primera medida, es importante que se consideren las condiciones generadas por el medio ambiente: la escasez de recursos y la lucha por obtenerlos beneficia a los más fuertes en detrimento de los más débiles que, al estar peor adaptados al medio, tienen menos posibilidades de subsistencia. Por esta razón, se produce un proceso de selección natural en el que sobreviven los más aptos. Este proceso se denomina “adaptación al medio”.

Esta adaptación es la que transmite la información genética de generación en generación a través de diferentes mutaciones y que se codifica en el ADN. Esta codificación —que describe únicamente a un individuo de forma completa— traslada

a los hijos el material genético de sus progenitores en una suerte de “ensayo” que se basa en la recombinación de nuevas características en la que la descendencia hereda las más ventajosas. De esta manera, el material genético de los individuos mejor adaptados prevalece.

### Descripción algorítmica

Lo primero que se definirá en un AG es el problema. Para que el AG funcione es necesario que cuente con una función que evalúe una solución respecto de otra para luego desechar la que menos se adapte. En otras palabras, el problema se debe definir como “un problema de minimización respecto a la solución final”, que realizará aproximaciones sucesivas a la solución final y que representan las situaciones del medio en el que se encuentra la población.

Luego se especificará la forma de codificar la solución: en los AG es usual que se la represente en función de una cadena de bits. Esta cadena se interpretará de acuerdo con la naturaleza de la solución que puede ser la representación de un entero, un carácter, un real o un conjunto de valores booleanos que simbolizan algo específico.

Entonces, cuando se conoce el problema y la manera de representar sus soluciones, se implementa el AG mediante una sucesión de pasos lógicos:

1. Definir la solución.
2. Filtrar:
  - a. Aplicarle la función objetivo.
  - b. Ordenar los individuos en función de los valores obtenidos.
  - c. Seleccionar los mejores individuos (soluciones) para el cruce.
3. Cruzar los individuos.
4. Mutación de los descendientes.
5. Inserción.
6. Si se cumple la función objetivo “terminar”, de lo contrario volver al paso 2.

En primer lugar, se define la población (solución) y, para cada uno de los individuos, se selecciona un valor aleatorio que lo represente. En segundo lugar, se les aplica una función objetivo (2.a), cuyo resultado determinará el nivel de adaptación del individuo al medio. En tercer lugar, y en función del valor que se obtuvo, se ordena la población de mayor a menor (2.b); de esta manera, quedan en primer lugar los más adaptados. A continuación, se seleccionan los individuos que generarán una nueva descendencia (2.c). Suponiendo que se han seleccionado dos, se los cruza (3); se escoge un punto de corte y la cadena de bits que representa a cada progenitor se la divide en dos por ese punto. Luego, se generan los nuevos individuos uniendo la subcadenas. Por ejemplo:

Si tomamos como progenitores a 00110011 y a 01110101, se los divide en dos subcadenas compuestas por 001-10011 y 011-10101; luego se cruzan las subcadenas, conformando la descendencia 00110101 y 01110011.



Para que el AG funcione es necesario que cuente con una función que evalúe una solución respecto de otra para luego desechar la que menos se adapte.

En el paso cuatro, se realiza en la descendencia una mutación (cambio) de algún bit aleatorio; por ejemplo, el primer bit pasando de 00110101 a 10110101.

Una vez que se ha generado y mutado la descendencia, deberá insertarse en la población (5). Para llevar a cabo la inserción, existen diversas políticas que no se analizarán en detalle; lo más común es eliminar los que tuvieron menor adaptación al medio y reemplazarlos por la nueva descendencia.

Por último, en el paso 6, se comprueba si alguno de los individuos disponibles satisface los criterios establecidos y se puede considerar como solución al problema. En este paso, también se corrobora si se ha excedido un número de iteraciones o un límite de tiempo. Si esto no ocurre y ningún individuo cumple los criterios de parada, se vuelve al paso 2 para seguir iterando y probando nuevas soluciones mutadas.

Cabe señalar que esto es solo una breve narración de la forma, característica y ejecución de un AG; quedan muchas cosas en el tintero como, por ejemplo, los distintos tipos de representación, las formas de selección, el cruzamiento y la mutación, que exceden el tratamiento de este libro.



### 7.5.3 Árboles de decisión

Los árboles de decisión son una técnica de programación que permite analizar decisiones secuenciales basadas en el uso de resultados y probabilidades asociadas.

Los árboles de decisión son una técnica de programación que permite analizar decisiones secuenciales basadas en el uso de resultados y probabilidades asociadas, es decir, en una heurística de ocurrencia.

Las ventajas de un árbol de decisión son:

- Resume los ejemplos de partida y permite la clasificación de nuevos casos, siempre y cuando no existan modificaciones sustanciales en las condiciones que generaron los ejemplos que sirvieron para su construcción.
- Facilita la interpretación de la decisión adoptada, ya que permite regenerar el camino decisorio aplicado.
- Proporciona un alto grado de comprensión del conocimiento utilizado en la toma de decisiones.
- Explica el comportamiento respecto a una determinada tarea de decisión.
- Reduce el número de variables independientes.
- Es una magnífica herramienta para el control de la gestión empresarial.

En general, los árboles de decisión son binarios; esto significa que a partir de un punto tienen dos opciones. Es importante aclarar que también existen árboles de grado tres o más.

Estos árboles cuentan con una aplicación específica: los árboles de juegos. Normalmente, en un juego se toman decisiones constantes de movimientos o jugadas que se realizarán.

Por ejemplo: el Ta-Te-Ti se puede resolver a través de un árbol de decisión. En este juego participan dos contendientes: uno representado por cruces y el otro, por círculos. Cada jugador marca sus posiciones en un tablero —que se representa por una matriz de 3 x 3— e intenta colocar tres de sus fichas en línea horizontal, vertical o diagonal.

Entonces, cuando se inicia el juego, el primer participante tiene nueve opciones (todo el tablero); el segundo, ocho. A medida que el juego avanza, se van quedando con menos opciones. El árbol de decisión permite, desde una posición determinada, la elección de la mejor jugada. Esta operativa se realiza aplicando la heurística (probabilidad) para ganar en la menor cantidad de jugadas o que su rival no gane antes. Es decir, de acuerdo con el estado del tablero, se debe decidir si se juega para ganar o para evitar que el otro gane. Para ello, se utiliza la opción que tiene menos longitud de paso en el árbol y que se dirija al triunfo de uno de los contrincantes o el de sus oponentes.

Esta función también se puede implementar si se consideran el número de filas, columnas y diagonales restantes abiertas para un jugador y se las resta por el número de éstas para su oponente. A continuación, se ilustra una posición y sus posibles combinaciones:

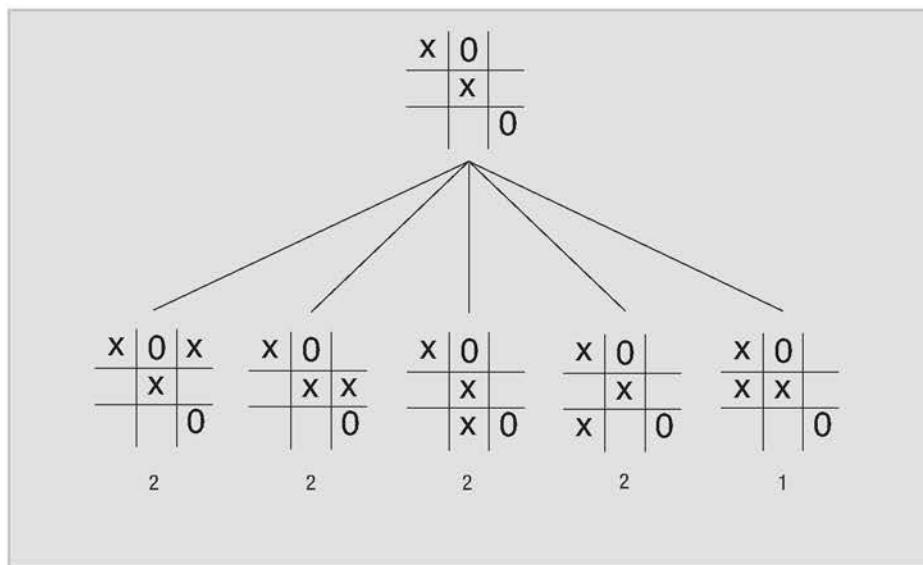


Fig. 7.2 Árbol de juego del Ta-Te-Ti.

Dada una posición en el tablero, para determinar la próxima jugada (la más adecuada), se deben considerar todos los movimientos posibles y las futuras posiciones. Sin embargo, en este análisis no se llega al mejor movimiento, como en el ejemplo anterior en el que las cuatro primeras posibilidades dan el mismo valor de evaluación. En cambio, sin lugar a dudas, la cuarta es la mejor; entonces, es necesario mejorarla aún más. Ahora existe la posibilidad de prever varios movimientos. De esta manera, la función mejorará en gran medida, ya que se inicia en cualquier posición y se determinan todos los posibles movimientos en un árbol hasta un cierto nivel de previsión. A este árbol se lo conoce como “árbol de juego”, cuya profundidad es igual a la profundidad de ese árbol.

Al turno del primer jugador se le designa una X; al del segundo, una O. Es obvio que el árbol empieza con el turno de X, que estará evaluado de acuerdo con su

conveniencia. En el árbol anterior, el mejor primer punto para X será la cruz central; entonces el jugador realizará este movimiento. Cuando le llega su turno, O seleccionará la jugada que tenga el menor valor porque ésta será la que perjudicará más a X.

Así funciona un árbol de juego que, como ya se indicó, es una aplicación de un árbol de decisión. De acuerdo con el nivel de previsión, se genera un árbol en el que cada jugador decidirá la jugada más conveniente, que se ajustará a la evaluación de una determinada posición.

Los árboles de decisión son más precisos que el hombre a la hora de desarrollar un diagnóstico de una situación. Por esta razón, se utilizan en la minería de datos. El hombre se puede equivocar y dejar algunos detalles importantes; en cambio, la máquina, mediante el uso de estos árboles, puede llegar a un resultado exacto.

Si bien se considera que esta técnica resulta, muchas veces, más lenta, debido a que analiza todas las combinaciones posibles, se debe destacar que esto es lo que la vuelve más precisa que el hombre.

La minería de datos y, en particular, los árboles de decisión se utilizan en la Inteligencia Artificial, especialmente en los denominados sistemas expertos, que se basan en grandes bases de datos, en las que se cargan reglas de decisión que encuentran su fundamento en la experiencia de los expertos en una ciencia determinada sobre la que versará el sistema. De esta manera, se lo puede utilizar para establecer un diagnóstico determinado, en el que se evalúa todos los caminos posibles dentro del árbol.



El funcionamiento de la minería de datos se basa en una aplicación técnica denominada modelado. Ésta construye un modelo en una situación determinada, en la que ya se conoce la respuesta; después, se lo aplica a otra situación en la que no se sabe cuál será la respuesta.

## 7.6 Modelado de la minería de datos

En principio, el funcionamiento de la minería de datos se basa en una aplicación técnica denominada modelado. Esta técnica construye un modelo en una situación determinada, en la que ya se conoce la respuesta; después, se lo aplica a otra situación en la que no se sabe cuál será la respuesta.

Si una persona busca un tipo de información en particular, es probable que parte de ciertos supuestos de los que ya tiene conocimiento. En el caso del petróleo, por ejemplo, si se intentara explorar una plataforma submarina, lo primero que hará es identificar las características comunes al modelo de los lugares en los que se encontró petróleo.

Para obtener mejores probabilidades de ocurrencia, estas características se deben cumplir en el modelo de situación. En el ejemplo del petróleo, en el caso en que la profundidad de los pozos descubiertos fuera la misma, este parámetro se debería utilizar en la búsqueda. Entonces, la combinación de todos los patrones de esta clase originaría la aplicación de técnicas de la minería de datos, que pronosticarán o planearán la búsqueda con un ahorro importante de tiempo y gastos de pruebas.

La humanidad ha construido modelos desde hace muchísimo tiempo, seguramente, antes de que las computadoras y la tecnología Data Mining aparecieran. Las nuevas tecnologías tienen una manera de construir modelos que no se diferencia mucho del modo en que lo hacen los humanos.

A las computadoras se las carga con información de situaciones en las que ya se conoce la respuesta; luego, el software de la minería de datos correrá por los datos

y distinguirá sus características principales para la construcción del modelo, que se utilizará en situaciones similares en las que no se sabe la respuesta.

Por esta razón, es de vital importancia que se realice una prueba de validez del nuevo modelo. En este caso, se prueban los parámetros sobre una base conocida, en la que ya se conoce la respuesta correcta. En el ejemplo de la exploración de la plataforma submarina, la forma de evaluar sería el procesamiento de todas las exploraciones exitosas para que el modelo las entregue como lugares aptos para la exploración. Si esto fuera correcto, el modelo podría procesar otras superficies no exploradas.

## 7.7 Integración entre almacén de datos y minería de datos

Para la correcta aplicación de estas técnicas algorítmicas avanzadas, se las debe integrar con el almacén de datos y con otras herramientas flexibles e interactivas para el análisis de negocios.

En la actualidad, algunas de las herramientas de la minería de datos operan fuera del almacén de datos y necesitan realizar más pasos para extraer, importar y analizar datos. La integración con Data Warehouse facilita la aplicación de los resultados desde Data Mining cuando nuevos conceptos requieren una implementación operacional.

El almacén de datos analítico resultante contribuye a un mejoramiento de los procesos en toda la organización como, por ejemplo, campañas de marketing, detección de fraudes, etcétera.

El almacén de datos debe contener, además de una combinación de datos de seguimiento interno, datos externos del mercado que informen sobre la actividad de los competidores. Este análisis es una manera excelente de proyectar.

Como se recordará, en los capítulos anteriores, se comentó que el almacén de datos se puede implementar en una variedad de sistemas de bases relacionales y que es necesario optimizarlo para un acceso a los datos flexible y rápido.

Si se trata de un modelo de negocios muy sofisticado, el almacén de datos se puede implementar mediante un servidor multidimensional OLAP. Esto permitirá que el modelo se aplique cuando se navega por Data Warehouse. Gracias a las estructuras multidimensionales, el usuario —a través de las diferentes dimensiones— podrá analizar los datos desde diferentes puntos de vista; por ejemplo, resumido por líneas de producto.

El servidor de minería de datos se debe integrar con el almacén de datos y el servidor OLAP para insertar el análisis de negocios directamente en esta infraestructura.

Un avanzado metadata centrado en procesos define los objetivos de la minería de datos para resultados específicos, tales como perspectivas, pronósticos y predicciones que permitan optimizar las promociones y las campañas de marketing.

La integración con el almacén de datos permite la implementación y el monitoreo directo de las decisiones operacionales. Con el tiempo, el almacén de datos obtendrá nuevas decisiones y resultados. La empresa podrá “minar” las mejores prácticas para utilizarlas en el futuro.



El almacén de datos se puede implementar en una variedad de sistemas de bases relacionales y es necesario optimizarlo para un acceso a los datos flexible y rápido.

Este diseño representa una transferencia fundamental desde los sistemas de soporte de decisión convencionales, puesto que, además de proporcionar datos a los usuarios finales mediante un software de consultas y reportes, el servidor tiene la capacidad de aplicar los modelos de negocios del usuario al almacén de datos y devuelve un análisis proactivo con la información más importante. Esto contribuye a que se mejoren los metadatos en el servidor OLAP y provean un nuevo estrato de metadatos que los represente con una vista fraccionada y estratégica.

## 7.8 Ventajas de la minería de datos

---

- Contribuye con la toma de decisiones estratégicas y proporciona un sentido automatizado para identificar información clave desde volúmenes de datos generados por procesos tradicionales y de Business Intelligence.
- Permite a los usuarios dar prioridad a decisiones y acciones e indica los factores que tienen una mayor incidencia, qué segmentos de clientes son desecharables y qué unidades de negocio son sobrepasadas y por qué.
- Proporciona poderes de decisión a los usuarios del negocio que mejor entienden el problema y el entorno y es capaz de medir las acciones y los resultados de la mejor forma.
- Genera modelos descriptivos en un contexto de objetivos definidos en los negocios, permite a las organizaciones, sin que se considere la industria o el tamaño, explorar automáticamente, visualizar y comprender los datos e identificar patrones, relaciones y dependencias que impactan en los resultados finales, tales como el aumento de los ingresos, incremento de los beneficios, disminución de costos y gestión de riesgos.
- Genera modelos predictivos: permite que relaciones no descubiertas e identificadas a través del proceso de la minería de datos se expresen como reglas de negocio o modelos predictivos. Estas salidas de la minería de datos pueden comunicarse en formatos tradicionales como presentaciones, informes, información electrónica, estar embebidos dentro de aplicaciones, etc., para que guien la estrategia y la planificación de la organización.

## 7.9 Diferencias entre el análisis estadístico y la minería de datos

---

En este apartado, se diferenciará la minería de datos del análisis que se desarrolla para la evaluación de situaciones transcurridas dentro de una organización.

En los dos casos lo que se quiere es mejorar la toma de decisiones a través de un mejor conocimiento del entorno, que se puede estudiar gracias a los datos que se encuentran almacenados en la organización —que pueden ser cuantitativos o cualitativos— y, también, mediante información propia o del medio en que ésta se desenvuelve.

A continuación, se establecen las situaciones en las cuales es preferible el uso de técnicas de la minería de datos antes que el análisis estadístico:

- Las técnicas estadísticas se centran generalmente en técnicas confirmatorias, esto es que a partir de patrones conocidos, demuestran la ocurrencia de los hechos en el paso del tiempo. En cambio, las técnicas de la minería de datos son exploratorias, es decir, rastrean el universo de posibilidades para encontrar nuevos patrones desconocidos. Entonces, ambas ciencias se pueden utilizar si lo que se quiere es confirmar o refutar alguna hipótesis; sin embargo, cuando el objetivo es meramente exploratorio como, por ejemplo, definir las variables más interesantes en un sistema de información, se necesita delegar parte del conocimiento analítico de la empresa en técnicas de aprendizaje utilizando la minería de datos. Aquí se observa la primera diferencia de aplicación entre las dos herramientas, a saber: se recurrirá a la minería de datos cuando no se parte de supuestos conocidos y se busca algún conocimiento nuevo y susceptible de proporcionar información novedosa en la toma de decisiones.
- La minería de datos provee mejores soluciones cuando la dimensión del problema es mayor. En un problema con una gran cantidad de variables, resulta muy complicado encontrar hipótesis de partida interesantes. En este caso, las técnicas de la minería de datos —como los árboles de decisión— encontrarán nuevas relaciones para luego investigar sobre las variables más interesantes. Las técnicas de la minería de datos tienen menos limitaciones que las estadísticas. Después de encontrar un punto de interés desde el que se quiere realizar un análisis estadístico en particular como, por ejemplo, distinguir los segmentos de mercado, se podría dar el caso en el que los datos con los que se cuenta no satisfagan los requerimientos específicos. Por esta razón, será necesario que se examinen todas las variables para establecer el modo de adecuarlas al análisis que, en algunos casos, no será conveniente y, en otros, imposible. La minería de datos, en cambio, al ser menos restrictivo que la estadística, se puede utilizar con supuestos mínimos.
- Las técnicas de la minería de datos son muy útiles si los datos de la organización son muy dinámicos. En los casos en los que el almacén de datos es más estático, se justifica una inversión en análisis estadístico. En cambio, en uno dinámico, se necesita que las técnicas de la minería de datos exploren los cambios y determinen si una regla de negocio ha cambiado para abordar ciertas cuestiones relacionadas con el corto y el mediano plazo. De esta manera, la minería de datos incidirá en la inversión y en la actualización del negocio.

A continuación, se expondrán las situaciones en las cuales es preferible el análisis estadístico en lugar de la minería de datos.

- Las técnicas de la minería de datos poseen relaciones ocultas que imponen la interpretación efectiva de los diagramas de causa y efecto. Por esta razón, cuando se quiere encontrar en una investigación las causas de ciertos efectos —como, por ejemplo, determinar qué es más aconsejable para que un producto venda más, si invertir en publicidad u ofrecer un descuento—, se recurrirá a las técnicas estadísticas.
- En ciertos casos, en los que se quiere realizar una generalización de poblaciones que se desconocen en su totalidad, si se necesita que las conclusiones se extiendan a otros elementos de poblaciones similares, se utilizarán técnicas de inferencia estadística. Esto se relaciona con situaciones en las que



Las técnicas de la minería de datos son exploratorias rastrean el universo de posibilidades para encontrar nuevos patrones desconocidos.



Las técnicas de la minería de datos son muy útiles si los datos de la organización son muy dinámicos.

solo se cuenta con muestras. En la minería de datos —como ya se indicó en este capítulo— se deben generar modelos que luego se validarán con otros casos conocidos de la población, que se utilizarán para el ajuste de predicción que, en poblaciones desconocidas, no se posee.

Hasta aquí, se ha detallado en qué circunstancias es inconveniente la utilización de una técnica u otra. Sin embargo, no son excluyentes entre sí y se puede establecer entre ellas cierta sinergia. En algunos casos, la metodología de un proyecto de la minería de datos necesita referencias del análisis estadístico:

- En la preparación de los datos, durante el tratamiento de valores erróneos y omitidos y en la aproximación a las variables de estudio conocidas.
- Durante la implementación del proyecto, cuando se realiza la generación de hipótesis que se refutará con una metodología y técnica estadística.

## 7.10 Aplicaciones de la minería de datos

En la actualidad, el manejo electrónico de datos es cada vez más importante. Los millones de bytes que se procesan a diario en las casas y en las oficinas son el refejo de lo que ocurre en el mundo. La complejidad y competitividad de los mercados mundiales necesitan cada vez más de bases de datos que contribuyan con la toma de decisiones de las organizaciones. A las empresas ya no les alcanza con alimentar los sistemas transaccionales (contables, financieros, de recursos humanos, de producción, etc.). La correcta interpretación de los consumidores actuales requiere de análisis muy complejos porque se ha perdido la masividad que caracterizaba a los mercados unos años atrás. Hoy, la heterogeneidad domina el mercado.

Las grandes empresas son un enorme depósito de información que, muchas veces, no se utiliza. Por esta razón, se necesita automatizar los procesos de información y descubrir los datos valiosos para que no se desperdicien. La minería de datos permite que esta información se transforme en futuros planes de negocios que generen una mayor rentabilidad en las organizaciones.

Otro ejemplo de la aplicación de esta herramienta surge a la hora de evaluar los comportamientos climáticos, dado que hay extensas bases de datos que contienen información de lo ocurrido en determinados lugares. Estos datos pueden correlacionarse de forma tal que pueden llegar a predecir comportamientos futuros para iguales condiciones. Esta técnica es utilizada para la proyección de eventos en función del análisis exhaustivo realizado por Data Mining de la información contenida en un extenso almacén de datos.

Gracias a esta nueva herramienta, las organizaciones segmentan su mercado, detectan fraudes o definen los nuevos perfiles de clientes, etcétera. En el caso de la segmentación de mercado, por ejemplo, una vez que la minería de datos seleccionó un segmento de clientes en su almacén de datos, podrá extraer los perfiles de sus consumidores. Cada vez que se agregan clientes se aplica un nuevo conjunto de modelos estadísticos y se corre el programa para comparar contra los segmentos existentes o crear otros nuevos. De esta forma, se está enfocando la atención a predecir la lealtad de marca y el comportamiento de los consumidores.



La minería de datos contribuye con el análisis de los factores de influencia en determinados procesos, realiza predicciones sobre futuros comportamientos o segmenta y agrupa ítems similares. Además, obtiene secuencias de eventos que inducen ciertos comportamientos.

En conclusión, la minería de datos contribuye con el análisis de los factores de influencia en determinados procesos; puede, también, realizar predicciones sobre futuros comportamientos o segmentar y agrupar ítems similares y, además, obtiene secuencias de eventos que inducen ciertos comportamientos.

Se considera a la minería de datos como la última etapa de la introducción de métodos cuantitativos y científicos en el universo de la industria, los negocios y el comercio.

## 7.11 Resumen

Luego de lo expuesto en este capítulo podemos destacar que la minería de datos no es un modelo de almacenamiento sino una tecnología formada por un conjunto de técnicas algorítmicas que se relacionan con la programación avanzada en forma directa.

A su vez, la minería de datos utiliza la información implícita de las grandes bases de datos. Es decir que se trata de una tecnología poderosa que le permite a las organizaciones recolectar información desconocida de sus propias bases de datos.

## 7.12 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) solo está disponible para docentes.

### 7.12.1 Mapa conceptual del capítulo

### 7.12.2 Autoevaluación

### 7.12.3 Presentaciones\*



# 8

## Bases de Datos Orientadas a Objetos

### Contenido

|  |     |
|--|-----|
| 8.1 Introducción.....  | 236 |
| 8.2 Historia y origen de las BDOO.....                                 | 237 |
| 8.3 Conceptos fundamentales.....                                       | 238 |
| 8.4 Bases de la orientación a objetos.....                             | 239 |
| 8.5 Características de las Bases de Datos Orientadas<br>a Objetos..... | 240 |
| 8.6 Sistema de gestión de BDOO (SGBDOO).....                           | 252 |
| 8.7 Rendimiento de las BDOO.....                                       | 256 |
| 8.8 Ventajas de las BDOO.....  | 256 |
| 8.9 Desventajas de las BDOO.....                                       | 257 |
| 8.10 Bases de datos Objeto-Relacionales.....                           | 258 |
| 8.11 Resumen.....  | 273 |
| 8.12 Contenido de la página Web de apoyo.....                          | 273 |

### Objetivos

- Introducir al lector en el uso de las Bases de Datos Orientadas a Objetos.
- Conocer las ventajas y las desventajas de los nuevos modelos de base de datos.
- Conocer el funcionamiento y la utilización de las bases de datos objeto-relacionales.

## 8.1 Introducción

En este capítulo se hablará de la evolución de los diferentes tipos de bases de datos y, por consiguiente, del surgimiento de las Bases de Datos Orientadas a Objetos (BDOO). Las BDOO almacenan y manipulan información respetando las características de la Programación Orientada a Objetos (POO); de esta forma, proporcionan una estructura flexible con acceso ágil, rápido, con gran capacidad de modificación. Además, combinan las mejores cualidades de los archivos planos, las bases jerárquicas y las relacionales. Como veremos a continuación, las BDOO representan el siguiente paso en la evolución de las bases de datos para soportar el análisis, el diseño y la programación orientada a objetos.

Estas bases de datos permiten el desarrollo y mantenimiento de aplicaciones complejas, ya que se puede utilizar un mismo modelo conceptual y así aplicarlo al análisis, diseño y programación sin necesidad de "mapear" o modelar en forma relacional un desarrollo que naturalmente fue diseñado y programado en objetos. Esto reduce el problema de conversión entre los diferentes modelos a través de todo el ciclo de vida, con un costo significativamente inferior.

Una base de datos orientada a objetos es una colección de datos que puede constituirse de forma que sus contenidos tengan la capacidad para encapsular, transmitir y renovar sencillamente elementos de datos, sus características, atributos y el código que opera sobre ellos en elementos complejos denominados objetos. Las bases de datos están constituidas por objetos, que pueden ser de muy diversos tipos, y sobre los cuales se encuentran definidas unas operaciones donde interactúan y se integran con las de un lenguaje de programación orientado a objetos, es decir, que los componentes de la base de datos son objetos de los lenguajes de programación. Además, este tipo de base de datos se diseña para trabajar con lenguajes orientados a objetos y para que manipulen datos complejos de forma rápida y segura.

Las Bases de Datos Orientadas a Objetos se crearon para tratar de satisfacer las necesidades de estas nuevas tecnologías. La orientación a objetos ofrece flexibilidad para manejar algunos de estos requisitos y no está limitada por los tipos de datos y los lenguajes de consulta de los sistemas de bases de datos tradicionales.

Los objetos estructurados se agrupan en clases. Las clases utilizadas en un determinado lenguaje de programación orientado a objetos son las mismas que se usarán en una base de datos, de tal manera que no es necesaria una transformación del modelo de objetos para ser utilizado. De forma contraria, el modelo relacional requiere abstraerse lo suficiente como para adaptar los objetos del mundo real a tablas. El conjunto de las clases se estructura en subclases y superclases, los valores de los datos también son objetos.

Como en cualquier base de datos, una Base de Datos Orientada a Objetos (BDOO) da un ambiente para el desarrollo de aplicaciones con un depósito persistente listo para su explotación; además permiten que el mismo modelo conceptual se aplique al análisis, diseño, programación, definición y acceso a la base de datos. Esto reduce el problema del operador de traducción entre los diferentes modelos, dado que, actualmente, todos los desarrollos se diseñan sobre la base de la orientación a objetos y, a su vez, se desarrollan sobre lenguajes de programación orientados a objetos. Es por ello que uno de los sustentos de las BDOO es el modelo conceptual, que debe estar



La orientación a objetos ofrece flexibilidad para manejar algunos de los requisitos y no está limitada por los tipos de datos y los lenguajes de consulta de los sistemas de bases de datos tradicionales.



Como en cualquier base de datos, una BDOO da un ambiente para el desarrollo de aplicaciones con un depósito persistente listo para su explotación; además permite que el mismo modelo conceptual se aplique al análisis, diseño, programación, definición y acceso a la base de datos.

basado en herramientas CASE orientadas a objetos y totalmente integradas a las BDOO, ya que ayudan a generar la estructura de datos y los métodos sin necesidad de realizar mapeos o traducciones de modelos.

Se podría decir que las BDOO ofrecen un mejor rendimiento del computador que las bases de datos relacionales, sobre todo para aplicaciones o clases con estructuras complejas de datos, dado que no requiere descomponer dicha clase compleja para almacenarla, sino que todos los objetos se almacenan en la forma en que fueron creados. Sin embargo, dado que corren sobre sistemas operativos que responden mucho más a un modelo relacional que a uno orientado a objetos, este rendimiento decrece a la hora de realizar la conversión de modelos para el almacenamiento final de los datos. Este tema del rendimiento será tratado en detalle más adelante en este capítulo.



Las BDOO ofrecen un mejor rendimiento del computador que las bases de datos relacionales, sobre todo para aplicaciones o clases con estructuras complejas de datos, dado que no requiere descomponer dicha clase compleja para almacenarla, sino que todos los objetos se almacenan en la forma en que fueron creados.

## 8.2 Historia y origen de las BDOO

Los lenguajes de programación orientados a objetos tienen sus inicios en el lenguaje SIMULA 67, propuesto a finales de la década de 1960. En SIMULA, el concepto de clase agrupa la estructura de datos interna de un objeto en una declaración de clase; este lenguaje es fuertemente tipado por el hecho de ser compilado. Sin embargo, el primer lenguaje que realmente popularizó la aproximación de la programación orientada a objetos fue Smalltalk, que surgió en 1976. Este lenguaje ofrece una gran flexibilidad que se basa, principalmente, en que es interpretado y no compilado, de ahí el hecho de no ser tipado. Además, este lenguaje incorpora el concepto de metaclasa.

Desde mediados de los años ochenta, con el advenimiento de las computadoras personales, crecieron numerosos lenguajes orientados a objetos que se basaron en los antes mencionados como, por ejemplo, C++, Objective C y Java.

El origen de las BDOO se sustenta en la existencia de problemas para representar cierta información y modelar ciertos aspectos del “mundo real”, puesto que los modelos clásicos permiten representar gran cantidad de datos, pero las operaciones y representaciones que se pueden realizar sobre ellos son bastante simples.

El paso del modelo de objetos al modelo relacional genera un “mapeo” o conversión que trae aparejado dificultades, situación que no ocurre en el caso de una Base de Datos Orientada a Objetos, ya que el modelo por implementar es el mismo. Por lo tanto, las Bases de Datos Orientadas a Objetos surgen básicamente para tratar de evitar las deficiencias de los modelos anteriores y para proporcionar eficiencia y sencillez a las aplicaciones.

A partir de ello, en años recientes, han aparecido muchos prototipos experimentales y sistemas de bases de datos comerciales orientados a objetos. Entre los primeros se encuentran los sistemas ORION, OpenOODB, IRIS, ODE y el proyecto ENCORE/ObServer. Y entre los sistemas disponibles en el mercado están: GESTONE/OPAL de ServioLogic, ONTOS de Ontologic, Objectivity de Objectivity Inc., Versant de Versant Technologies, ObjecStore de ObjectDesign y O2 de O2 Technology.

### 8.3 Conceptos fundamentales

A efectos de estandarizar los conceptos y la terminología utilizada en el resto del capítulo, se definirán, a continuación, los términos básicos y los conceptos fundamentales de la Programación Orientada a Objetos (POO) y su aplicación a BDOO.

**Objeto:** es cualquier cosa real o abstracta acerca de la cual se almacenan datos y los métodos que marcan el comportamiento de dichos datos. También se define como objeto a una instancia de una clase. Por ejemplo, en un sistema de ventas de una compañía, se define una clase CLIENTES y se aplica a esta clase los diferentes clientes que compran productos a la empresa (dentro de ellos PP S.A. o XX S.A. podrían ser una instancia de dicha clase siendo estos objetos). Las características fundamentales que permiten conocer, observar e identificar los objetos son la identidad, el comportamiento y el estado.

**Identidad:** es la propiedad que permite diferenciar a cada objeto de los restantes; esta propiedad permite comparar objetos para detectar si son o no iguales. En programación, la identidad de un objeto está dada por la posición de memoria que ocupa en la computadora que lo alberga.

**Comportamiento:** esta propiedad determina la funcionalidad del objeto, es decir, las tareas que puede realizar o a las que puede responder a partir de un mensaje recibido de otros objetos.

**Estado:** es el valor de los atributos que componen al objeto en un momento determinado; el comportamiento de un objeto puede modificar su estado.

**Clase:** es un tipo de objeto que agrupa una categoría de objetos con las mismas características inherentes y de comportamiento, por ejemplo, la clase CLIENTES utilizada en el ejemplo anterior, en el que se tipifica a determinados objetos bajo la misma forma. Es decir, un objeto es una instancia específica de un tipo de objeto o clase.

**Mensaje:** invoca una operación específica, con uno o más objetos como parámetros. Es decir, es la forma en la que se solicita que se lleve a cabo la operación indicada y que se encuentre el resultado deseado. En consecuencia, las implementaciones se refieren a los objetos como solicitudes.

**Encapsulamiento:** es el acto de ocultar los detalles de implementación de un objeto respecto de otros objetos o del usuario.

**Herencia:** es el mecanismo que permite la reutilización y extensibilidad del software. Una clase implementa el tipo de objeto. Una subclase hereda propiedades de su clase padre: una subclase puede heredar la estructura y todos o algunos de los métodos.

**Herencia Múltiple:** es la posibilidad de que una clase herede atributos o métodos de varias clases diferentes.

**Polimorfismo:** es la propiedad de una operación mediante la cual puede ser aplicada a distintos tipos de objetos.

Abstracción: es la posibilidad de aislar un elemento de su contexto o del resto de los elementos que lo acompañan; denota las características esenciales de un objeto que lo distinguen de todos los demás tipos de objeto y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador. Una abstracción se centra en la visión externa de un objeto y, por tanto, sirve para separar el comportamiento esencial de un objeto de su implementación.

Modularidad: se basa en el concepto de fragmentación de los programas en componentes individuales para reducir su complejidad en algún grado y para crear, además, una serie de fronteras bien definidas y documentadas dentro del programa, donde estas fronteras o interfaces tienen un alto valor para su comprensión.

Jerarquía: es la que denota una clasificación u ordenamiento de abstracciones.

Genericidad: es lo que permite construir clases genéricas para otras clases.

Objetos complejos: son aquellos que están construidos mediante el uso de algunos objetos más simples o mediante la aplicación de constructores a ellos. Los objetos más simples son objetos básicos como: Integer, Carácter, String, booleanos o punto flotante y algunos pueden ser de tipo atómico. Hay varios constructores de objetos complejos como las listas y arreglos. De este modo, el juego mínimo de constructores que el sistema debe tener es una lista y un arreglo.

Las listas y los arreglos son importantes porque pueden capturar órdenes que ocurren en el mundo real y, también, se pueden levantar en muchas especificaciones científicas donde las necesidades de la gente son matrices, series de tiempo, de información o de datos. El objeto de constructores debe ser ortogonal y cualquier constructor se aplicará a cualquier objeto.

## 8.4 Bases de la orientación a objetos

En la orientación a objetos, el conocimiento se descentraliza en todos los objetos que lo componen, de esta forma, cada objeto sabe hacer lo suyo y no le interesa saber cómo otro objeto hace su trabajo, pero conoce que lo hace y qué es lo que puede hacer. Dan Ingalls, de Smalltalk, afirmó: "La orientación a objetos proporciona una solución que conduce a un Universo de Objetos 'bien educados' que se piden de manera cortés, concederse mutuamente sus deseos".

El objetivo principal de este tipo de desarrollo es dejar la etapa en la que la construcción del software es una labor de artesanos y pasar a la etapa en la que se pueda tener fábricas de software, con gran capacidad de reutilización de código y con metodologías eficientes y efectivas que se apliquen al proceso de producción.

De esta forma, se pretende encontrar la solución de un problema, identificando los objetos que participan y delegando en cada uno de ellos las cosas que deben y pueden hacer, sin seguir un hilo conductor para resolver el qué, como lo hace el paradigma imperativo, sino que, por el contrario, identifica quién participa, analizando el comportamiento y las funciones que realizarán estos objetos participantes de forma tal que, interactuando entre ellos, lleguen al objetivo deseado.



El objetivo principal de este tipo de desarrollo es pasar a la etapa en la que se pueda tener fábricas de software, con gran capacidad de reutilización de código y con metodologías eficientes y efectivas que se apliquen al proceso de producción.

En este contexto aparecieron las primeras BDOO, que son lo suficientemente inteligentes para soportar el paradigma orientado a objetos, almacenando datos y métodos y no solo datos como en las bases de datos relacionales.

Por este motivo, están diseñadas para ser eficaces y, desde el punto de vista físico, para almacenar objetos complejos y evitar el acceso a los datos mediante los métodos almacenados en ellos. De igual forma, son más seguras, ya que no permiten tener acceso a los datos, debido a que para poder acceder a ellos se tiene que hacer a través de los métodos que posea cada uno de los objetos que la componen.

## 8.5 Características de las Bases de Datos Orientadas a Objetos

A continuación, se analizarán las características esenciales de las BDOO. Inicialmente se tendrá en cuenta la unificación del modelo lógico propuesto por estas bases de datos para unificar el desarrollo con los lenguajes de programación modernos, para después analizar las demás características.

### 8.5.1 Modelo conceptual

Las técnicas OO utilizan los mismos modelos conceptuales para el análisis, diseño y construcción. La tecnología de las BDOO da un paso más hacia la unificación; el modelo conceptual de las BDOO es el mismo que se utiliza para la programación orientada a objetos, lo que permite evitar la traducción, mapeo o reconversión del modelo, simplificando el desarrollo completo del software.

El uso del mismo modelo conceptual para todos los aspectos del desarrollo lo simplifica, particularmente con las herramientas CASE (Computer Aided Software Engineering) orientadas a objetos, que mejoran la comunicación entre usuarios, analistas y programadores, además de que reducen las posibilidades de error.

Actualmente, en las etapas de análisis y diseño se realizan relevamientos que luego se plasman en gráficas de UML y definiciones de casos de uso, como así también en un diagrama de clases. Todas estas herramientas no son compatibles con las bases de datos relacionales y obligan a la conversión de dichas herramientas en un diagrama Entidad-Relación que permita modelar el almacenamiento y la persistencia de los objetos utilizados en la programación a través de tablas, atributos y relaciones.

El modelo conceptual unificado implementado por las BDOO permite que, tal como fue diseñado el sistema, persista en la base de datos almacenado como objetos y no como tablas, atributos y relaciones.

De acuerdo con lo expresado en el párrafo anterior, es fundamental en la construcción de una BDOO incluir una herramienta CASE orientada a objetos que permita que el diseño se realice directamente en la propia base de datos y que no requiera conversiones de modelos.

El diseño de las BDOO actuales debe aprovechar al máximo el CASE e incorporar métodos creados con cualquier técnica poderosa, que incluya enunciados declarativos, generadores de códigos e inferencias basados en reglas.



CASE (Computer Aided Software Engineering) es un conjunto de herramientas de aplicaciones informáticas. Entre sus objetivos principales, podemos destacar que aumentan la productividad del software en múltiples tareas como la compilación automática, la documentación o la localización de errores, reducción del tiempo y costo de desarrollo. En 1984, se lanzó la primera de estas herramientas que trabajaba bajo una plataforma PC.



El diseño de las BDOO actuales debe aprovechar al máximo el CASE e incorporar métodos creados con cualquier técnica poderosa, que incluya enunciados declarativos, generadores de códigos e inferencias basados en reglas.

### 8.5.2 Modelo de datos orientado a objetos

En las bases de datos relacionales, los objetos se corresponden con las entidades, mientras que en el paradigma orientado a objetos se basa en el encapsulamiento de los datos. De esta forma, cada objeto se asocia con un conjunto de variables o atributos que contienen los datos del objeto, un conjunto de mensajes a los que responde y un conjunto de métodos que son bloques de código que responden los mensajes con un valor.

En una BDOO, hay objetos similares que se caracterizan por responder a los mismos mensajes, estar compuestos por los mismos atributos y contener los mismos métodos; a estos objetos se los agrupa en clases, donde cada objeto, como ya fue mencionado, es una instancia de esa clase. Todos los objetos que pertenecen a una clase se caracterizan por tener los mismos atributos y métodos y se diferencian por los valores que contienen cada uno de sus atributos.

Se podría decir que el concepto de clases en las BDOO se corresponde con el concepto de entidad en el modelo Entidad-Relación aplicado en las bases de datos relacionales. Por ejemplo, suponiendo un sistema para una universidad, seguramente, en una base de datos relacional, existiría una tabla "Alumnos" que contendría los datos de cada uno de los alumnos que la integran, donde cada una de las filas se correspondería con un alumno. En una BDOO, en lugar de una entidad, se definiría una clase "Alumnos", donde cada alumno sería un objeto que representaría una instancia de esta clase y contendría los datos de cada uno de los alumnos de la universidad.

A menudo, existen objetos que se parecen en algunas cosas, pero difieren en otras y éstas no hacen posible que pertenezcan a la misma clase; en ese caso, el concepto de herencia aplicado por las BDOO permite que una clase pueda heredar de otra de mayor nivel parte de su composición, lo cual permite la reutilización de código que evita tener que codificar todos los métodos de la clase de menor jerarquía, ya que los heredan de la de mayor jerarquía. Este concepto es parecido al aplicado por la especialización en el modelo relacional.

### 8.5.3 Persistencia de los datos

Un paso importante para tener en cuenta en una BDOO es proporcionar la manera de hacer persistentes los objetos que en los lenguajes de programación están en la memoria. Para ello, se han propuesto distintos enfoques de persistencia que se describen a continuación:

- **Por clases:** es el más sencillo, pero el menos conveniente; radica en definir las clases como persistentes, de forma tal que todos los objetos que pertenecen a ella son persistentes en forma predeterminada. Además, los objetos que pertenezcan a una clase que no se ha definido como persistente son transitorios.
- **Por creación:** en este enfoque los objetos son persistentes o transitorios, según se hayan creado o no así. Esto se realiza mediante una extensión en la sintaxis de creación de cada objeto pero, en este caso, sin que importe la clase a la cual pertenezcan.
- **Por marcas:** es una variante de la persistencia por creación; todos los objetos se crean como transitorios y, luego, a los que se quiera convertir en persistentes se los marca para que puedan persistir más allá de la ejecución del programa que lo está utilizando.

- Por alcance: algunos objetos se declaran de manera explícita como objetos persistentes (comúnmente denominados objetos raíz). El resto de los objetos serán persistentes si y solo si son alcanzados a través de una o varias referencias por alguno de estos objetos raíz que fueron creados persistentes.

#### 8.5.4 Almacenamiento y acceso de los objetos persistentes en una BDOO

Hay varias formas que pueden ser implementadas por las BDOO para acceder a los objetos almacenados en ellas; a continuación se analizarán cada una de ellas:

- Asignándole un nombre a los objetos, del mismo modo que se le asigna un nombre a un archivo. Esta forma es válida cuando la cantidad de objetos almacenados es relativamente chica, pero es impráctico si en la base de datos se almacenan millones de objetos.
- Exponiendo los identificadores de los objetos o los punteros al lugar físico donde están almacenados; a diferencia de la forma anterior, pueden almacenarse en forma externa, no tienen que ser recordados y pueden ser punteros físicos dentro de la base de datos.
- Guardando las colecciones de objetos y permitiendo que el programa acceda a dichas colecciones e iterando sobre ellas encuentre el objeto deseado. Estas colecciones, a su vez, pueden modelarse como objetos del tipo colección que incluyen conjuntos, listas, multiconjuntos, etcétera. Un caso particular de colección son las extensiones de clase que representan la colección de todos los objetos pertenecientes a una clase.

La mayoría de los gestores de BDOO aceptan las tres formas de almacenar y acceder a los objetos. En todas, los objetos se identifican; generalmente solo se le da nombre a las extensiones de clase y a otros objetos del tipo colección, pero normalmente no se nominan todos los objetos.



Malcolm Atkinson expresó un manifiesto con las características obligatorias y opcionales con las que debe contar un SGBDOO. Para ello, se basó en dos criterios: debe tratarse de un Sistema de Gestión de Base de Datos y debe estar orientados a objetos.

#### 8.5.5 Manifiesto de Atkinson

En 1989, Malcolm Atkinson expresó un manifiesto que contiene las características obligatorias y opcionales con las que debe contar un Sistema de Gestión de Base de Datos Orientado a Objetos (SGBDOO). Para ello, se basó en dos criterios: el primero es que debe ser un Sistema de Gestión de Base de Datos, como ocurre con los gestores de base de datos relacionales y, el segundo, es que debe ser orientado a objetos.

A continuación, se enumeran y describen las características obligatorias según los dos criterios y las característicasopcionales.

##### 8.5.5.1 Características obligatorias de orientación a objetos

- Debe soportar objetos complejos: esto es que debe tener la posibilidad de almacenar y acceder directamente a objetos complejos, esto es, objetos conformados por otros objetos simples, sin necesidad de desagregar los objetos complejos en objetos simples como sí lo requiere el modelo relacional.
- Debe soportar mecanismos de identidad de los objetos: como ya se mencionó en el apartado anterior, debe permitir almacenar y acceder a los objetos a través de su identificación.

- Debe soportar el encapsulamiento: permitir que la información del objeto esté oculto a otros objetos o al usuario.
- Debe soportar tipos o clases: dado que si no soportara el concepto de clase y tipo no sería orientado a objetos.
- Debe soportar que los tipos o clases se hereden: lo cual significa que acepte y resuelva la herencia y permita la reutilización de código.
- Debe soportar el enlace dinámico: de forma tal que enlace cuando el programa se ejecuta y no en el momento de la compilación.
- El DML (*Data Manipulation Language*) debe ser computacionalmente complejo: esto significa que debe permitir operar directamente con objetos complejos sin necesidad de desagregarlos en objetos simples.
- El conjunto de todos los tipos de datos debe ser ampliable: indica que permitirá la creación de nuevos tipos de datos a partir de los existentes para su ampliación.

#### 8.5.5.2 Características obligatorias de un Sistema de Gestión de Base de Datos

- Debe permitir la persistencia de datos: esto implica almacenar y acceder a los objetos persistidos, como fue ya explicado en párrafos anteriores.
- Debe ser capaz de administrar bases de datos de gran tamaño: permitiendo que se mantenga la integridad, persistencia y valor de los valores almacenados, independientemente de la cantidad de cada uno de ellos.
- Debe soportar usuarios concurrentes: que varios usuarios puedan acceder a la base de datos y a los mismos objetos si así lo requirieran.
- Debe ser capaz de recuperarse de fallas: esto implica tener opciones de recuperación que puedan retornar la información almacenada al último punto íntegro conocido, ya sea por fallas de hardware o de software.
- Debe proporcionar una forma simple de consultar datos: es decir, poseer algún lenguaje de consulta que permita consultar sin inconvenientes la información almacenada.

#### 8.5.5.3 Características opcionales

- Herencia múltiple: que los objetos persistidos en la base de datos puedan heredar más de una clase.
- Comprobación de tipos e inferencia de tipos: es la posibilidad de comprobar tipos de datos y asignar automáticamente un tipo de datos sin que lo defina el programador.
- Sistema de base de datos distribuido: es la posibilidad de que el sistema de gestión se encuentre distribuido en varios servidores.
- Soporte de versiones: la mayoría de los sistemas de bases de datos solo permiten que exista una representación de un ente de la base de datos dentro de ésta. Las versiones autorizan que las representaciones alternas existan en forma simultánea.

### 8.5.6 Intento de estandarización ODMG

Dado que la mayor limitación de las Bases de Datos Orientadas a Objetos es la carencia de un estándar, algunos fabricantes de la industria conformaron un ODMG (*Object Database Management Group*), con el objeto de llegar a definir un estándar para la construcción y utilización de un SGBDOO.

Este modelo adopta una arquitectura que consta de un sistema de gestión que soporta un lenguaje de bases de datos orientado a objetos, con una sintaxis similar a un lenguaje de programación también orientado a objetos, como puede ser C++ o Smalltalk. El lenguaje de bases de datos se especifica mediante un lenguaje de definición de datos (ODL), un lenguaje de manipulación de datos (OML) y un lenguaje de consulta (OQL), siendo todos ellos portables a otros sistemas con el fin de conseguir la portabilidad de la aplicación completa.

Con ello, lo que se intenta es definir un SGBDOO que integre las capacidades de las bases de datos con las capacidades de los lenguajes de programación, de forma que los objetos de la base de datos aparezcan como objetos del lenguaje de programación, intentando, de esta manera, eliminar la falta de correspondencia existente entre los sistemas de tipos de ambos lenguajes. De este modo, el SGBDOO extiende el lenguaje con persistencia, concurrencia, recuperación de datos, consultas asociativas, etcétera.

A continuación, se expondrán los componentes del modelo estándar para la semántica de los objetos en una base de datos.

#### 8.5.6.1 Objetos

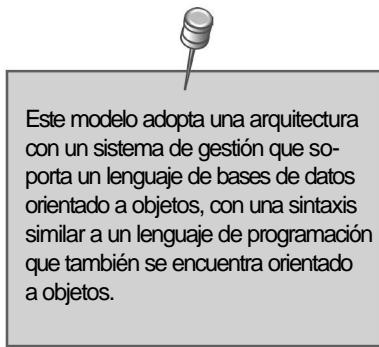
Los tipos de objetos se descomponen en atómicos, colecciones y tipos estructurados. La propuesta del estándar para las clases contenedora son los tipos colección, que se derivan de la interfaz *Collection*.

Los objetos colección identificados por el estándar son los siguientes:

- *Set <tipo>*: es un conjunto de objetos desordenados del mismo tipo, en el que se permiten duplicados.
- *Bag <tipo>*: es un conjunto de objetos desordenados del mismo tipo, en el que no se permiten duplicados.
- *List <tipo>*: es un conjunto ordenado de objetos del mismo tipo, en el que se permiten duplicados.
- *Array <tipo>*: es un conjunto ordenado de objetos del mismo tipo a los que se puede acceder por su posición. Su tamaño es dinámico y los elementos se pueden insertar y borrar de cualquier posición.
- *Dictionary <clave,valor>*: es como un índice. Está formado por claves ordenadas, cada una de ellas aparejada con un solo valor.

Los tipos estructurados son los siguientes:

- *Date*: utilizados para representar una fecha del tipo día, mes, año.
- *Time*: utilizados para representar una hora en formato hora, minutos, segundos.



Este modelo adopta una arquitectura que consta de un sistema de gestión que soporta un lenguaje de bases de datos orientado a objetos, con una sintaxis similar a un lenguaje de programación también orientado a objetos, como puede ser C++ o Smalltalk. El lenguaje de bases de datos se especifica mediante un lenguaje de definición de datos (ODL), un lenguaje de manipulación de datos (OML) y un lenguaje de consulta (OQL), siendo todos ellos portables a otros sistemas con el fin de conseguir la portabilidad de la aplicación completa.

- *Timestamp*: es una hora de una fecha con precisión de microsegundos.
- *Interval*: es un intervalo de tiempo.

Estos tipos tienen la misma definición que los tipos con el mismo nombre del estándar de SQL. La creación de los objetos se realiza utilizando el método new( ) y todos heredan la interfaz que se detalla a continuación:

```
interface Object
{
    enum Lock Type {read, write, upgrade};
    void lock(in Lock_Type mode) raises(LockNotGranted);
    boolean try_lock(in Lock_Type mode);
    boolean same_as(in Object anObject);
    Object copy();
    void delete();
};
```

Dentro del estándar se establece que cada objeto tiene un identificador de objeto único generado por el sistema de gestión, que no cambia y que no es reutilizado cuando se borra el objeto, pero no existe un estándar para la generación de los identificadores, dado que cada uno de los gestores genera los identificadores a su forma.

#### 8.5.6.2 Literales

Los tipos literales se descomponen en atómicos, colecciones y estructurados. No tienen identificadores, no pueden aparecer solos como objetos y tampoco referenciarse de modo individual, ya que están embebidos en objetos.

Los literales atómicos son boolean, short, long, float, double, octet, char, string y enum.

Los literales estructurados contienen un número fijo de elementos heterogéneos, cada uno de ellos es un par conformado por <nombre, valor>, donde valor puede ser cualquier tipo literal. Estos tipos estructurados son: date, time, timestamp, interval y struct. Los tipos de colección son: set <tipo>, bag <tipo>, list <tipo>, array <tipo> y dictionary <clave,valor>.

#### 8.5.6.3 Tipos

Uno de los fundamentos que persigue el paradigma orientado a objetos es la distinción entre la interfaz pública de una clase y sus elementos privados, de allí el cumplimiento de la propiedad de encapsulamiento.

El estándar propuesto hace esta distinción hablando de la especificación externa de un tipo y de sus implementaciones. Una interfaz es una especificación del comportamiento abstracto de un tipo de objeto y contiene las signaturas de las operaciones. Aunque una interfaz puede tener propiedades como los atributos y las relaciones como parte de su especificación, éstas no se pueden heredar desde la

## S

Una interfaz es una especificación del comportamiento abstracto de un tipo de objeto y contiene las signaturas de las operaciones. Aunque una interfaz puede tener propiedades como los atributos y las relaciones como parte de su especificación, éstas no se pueden heredar desde la interfaz. Además, no se puede instanciar de forma tal que impida la creación de objetos a partir de ella.

# S

Una clase es una especificación del comportamiento abstracto y del estado abstracto de un tipo de objeto. Las clases son instanciables, por lo tanto, a partir de ellas, se pueden crear instancias que son objetos individuales.

interfaz. Además, una interfaz no se puede instanciar de forma tal que impida la creación de objetos a partir de ella (es el equivalente a una clase abstracta en la mayoría de los lenguajes de programación).

Una clase es una especificación del comportamiento abstracto y del estado abstracto de un tipo de objeto. Las clases son instanciables, por lo tanto, a partir de ellas, se pueden crear instancias que son objetos individuales (es el equivalente a una clase concreta en los lenguajes de programación).

El estándar propuesto soporta la herencia simple y la herencia múltiple mediante las interfaces, ya que éstas no son instanciables y se suelen utilizar para especificar operaciones abstractas que pueden ser heredadas por clases o por otras interfaces. A esto se lo denomina herencia de comportamiento y se especifica mediante el símbolo ":". Esta herencia de comportamiento requiere que el supertipo sea una interfaz, mientras que el subtipo puede ser una clase o una interfaz.

La interfaz o clase más baja de la jerarquía es el tipo más específico, ya que hereda los comportamientos de todos los tipos que tiene por encima en la jerarquía; es la interfaz o clase más completa.

Dado el siguiente ejemplo,

```
interface ProductoVenta...;
interface Vehiculos : ProductoVenta ...;
class Automovil: Vehiculos ...;
class Moto : Vehiculos ...;
class Camioneta : Vehiculos ...;
```

los tipos más específicos son: automóvil, moto y camioneta.

Uno de los beneficios prácticos de la herencia es que se puede hacer referencia a los subtipos como su supertipo. Por ejemplo, un programa de aplicación puede hacer referencia a automóvil, moto y camioneta como vehículos o, incluso, como producto de venta; esto hace que sea más sencillo tratar los subtipos como un grupo cuando sea necesario.

Los subtipos se especializan como sea necesario añadiéndoles comportamientos. Los subtipos de un subtipo especializado heredan también los comportamientos añadidos.

El modelo orientado a objetos utiliza la relación extends para indicar la herencia de estado y de comportamiento. En este tipo de herencia tanto el subtipo como el supertipo deben ser clases. Las clases que extienden a otra clase ganan acceso a todos los estados y comportamientos del supertipo, incluyendo cualquier cosa que el supertipo haya adquirido a través de la herencia de otras interfaces.

Una clase puede extender, como máximo, a otra clase. Sin embargo, si se construye una jerarquía de extensiones, las clases de más abajo en la jerarquía heredan todo lo que sus supertipos adquieren de las clases que tienen por encima.

De esta forma, el modelo le permite al diseñador que declare una extensión (extent) para cada tipo de objeto definido como una clase. La extensión de un tipo tiene un

nombre e incluye todas las instancias de objetos persistentes creadas a partir de dicho tipo.

Por ejemplo, declarar una extensión denominada “Empleados” para el tipo de objeto “Empleado” es similar a crear un objeto de tipo Set <Empleado> denominado “Empleados”. Una extensión se puede indexar para que el acceso a su contenido sea más rápido.

Una clase con una extensión puede tener una o más claves (key). Una clave es un identificador único; cuando una clave está formada por una sola propiedad es una clave simple; si está formada por varias propiedades es compuesta.

Una implementación de un tipo consta de dos partes: la representación y los métodos. La representación es una estructura de datos dependiente de un lenguaje de programación que contiene las propiedades del tipo. Las especificaciones de la implementación vienen de una conexión con un lenguaje (language binding), esto quiere decir que la representación interna de un tipo será diferente, dependiendo del lenguaje de programación que se utilice y de que un mismo tipo pueda tener más de una representación.

Los detalles de las operaciones de un tipo se especifican mediante un conjunto de métodos. En la especificación externa de cada operación debe haber al menos un método, sin embargo, un tipo puede incluir métodos que nunca se ven desde fuera del mismo. Estos métodos realizan algunas funciones necesarias para otros métodos del tipo.

Los métodos se escribirán en el mismo lenguaje de programación utilizado para expresar la representación del tipo. Si una base de datos soporta aplicaciones programadas en C++, Java y Smalltalk, entonces necesitará tres implementaciones para cada tipo, una para cada lenguaje, aunque cada programa de aplicación utilizará solo la implementación que le corresponda.

#### 8.5.6.4 Propiedades

El modelo ODMG establece dos tipos de propiedades, atributos y relaciones, donde un atributo se define del tipo de un objeto, pero considerando que no es un objeto. Por ello, no tiene un identificador, pero toma como valor un literal o el identificador de un objeto.

Las relaciones se definen entre tipos. El modelo actual solo soporta relaciones binarias con todas las cardinalidades: 1:1, 1:N y N:M.

Una relación no tiene nombre y tampoco es un objeto, pero define caminos transversales en la interfaz de cada dirección. En el lado del muchos de la relación, los objetos pueden estar desordenados (set o bag) u ordenados (list). El gestor de bases de datos mantiene automáticamente la integridad de las relaciones y se genera una excepción cuando se intenta atravesar una relación en la que uno de los objetos participantes se ha borrado. El modelo aporta operaciones para formar (form) y eliminar (drop) miembros de una relación.

#### 8.5.6.5 Transacciones

El modelo estándar soporta el concepto de transacciones, que son unidades lógicas de trabajo que llevan a la base de datos de un estado consistente a otro estado consistente. El modelo asume una secuencia lineal de transacciones que se ejecutan



La representación es una estructura de datos dependiente de un lenguaje de programación que contiene las propiedades del tipo. Las especificaciones de la implementación vienen de una conexión con un lenguaje, esto quiere decir que la representación interna de un tipo será diferente, dependiendo del lenguaje de programación que se utilice y de que un mismo tipo puede tener más de una representación.

de modo controlado. La concurrencia se basa en bloqueos estándar de lectura/escritura con un protocolo pesimista de control de concurrencia. Todos los accesos, creación, modificación y borrado de objetos persistentes se realizarán dentro de una transacción. El modelo especifica operaciones para iniciar, terminar (commit) y abortar transacciones, así como la operación de checkpoint.

Esta última operación hace permanentes los cambios realizados por la transacción en curso, sin liberar ninguno de los bloqueos adquiridos.



El lenguaje de definición de datos (ODL) en un SGBDOO se emplea para facilitar la portabilidad de los esquemas de las bases de datos. El ODL no es un lenguaje de programación completo; define las propiedades y los prototipos de las operaciones de los tipos, pero no los métodos que las implementan. Además, define los atributos y las relaciones entre tipos y especifica la firma de las operaciones.

#### 8.5.6.6 Lenguaje ODL

El lenguaje de definición de datos (ODL) en un SGBDOO se emplea para facilitar la portabilidad de los esquemas de las bases de datos. Es importante señalar que el ODL no es un lenguaje de programación completo; define las propiedades y los prototipos de las operaciones de los tipos, pero no los métodos que las implementan.

El ODL intenta definir tipos que puedan implementarse en diversos lenguajes de programación, no está, por tanto, ligado a la sintaxis concreta de un lenguaje de programación particular. De esta forma, un esquema especificado en ODL puede ser soportado por cualquier SGBDOO que sea compatible con ODMG.

La sintaxis de ODL es una extensión de la del IDL (Interface Definition Language) desarrollado por OMG como parte de CORBA (Common Object Request Broker Architecture).

El ODL define los atributos y las relaciones entre tipos y especifica la firma de las operaciones. A continuación, se muestra la sintaxis de creación:

```

<definición de tipo> ::= 
interfase <nombre del tipo>(:<lista de supertipos>)
{
  (<lista de propiedades del tipo>
  (<lista de propiedades>
  (<lista de operaciones>
}

<propiedad del tipo> ::= 
extent <nombre de la extensión>
|key(s) <lista de claves>
<especificación del atributo> ::= 
(attribute)
<tipo de dominio> (tamaño) <nombre del atributo>

<especificación de interrelación> ::= 
(relationship)
<destino del camino><nombre del camino>
inverse<camino inverso>
```

(order by <lista de atributos>)

especificación de operación>::=  
<tipo devuelto> <nombre de la operación>  
(<lista de argumentos>) (raises (<excepciones levantadas>)  
<argumento>::=  
<papel> (<nombre del argumento>:<tipo del argumento>)

A continuación, se muestra un ejemplo donde se crea una clase “Persona” con sus datos personales y los datos de esposos e hijos:

```
class Persona (extent gente)
{
    attribute string nombre;
    attribute struct Direccion {unsigned short number, string calle, string ciudad} address;
    relationship Persona esposo inverse Persona::esposo;
    relationship set<Persona> hijos inverse
        Persona::padres; relationship list<Persona> padres inverse
        Persona::hijos; void nacimiento (in string nombre);
    boolean casamiento (in string persona_nombre)
    raises (ninguna_persona);
    unsigned short ancestros (out set<Persona> todos_ancestros)
    raises (ninguna_persona);
    void mueve (in string nueva_direccion);
};
```

#### 8.5.6.7 Lenguaje OML

El lenguaje de manipulación se emplea para la elaboración de programas que permitan crear, modificar y borrar datos que constituyen la base de datos. ODMG sugiere que este lenguaje sea la extensión de un lenguaje de programación, de forma que se pueden realizar, entre otras, las siguientes operaciones sobre la base de datos: creación, borrado, modificación e identificación de un objeto

#### 8.5.6.8 Lenguaje OQL

El lenguaje de consulta propuesto por ODMG presenta las siguientes características:

- No es computacionalmente completo. Sin embargo, las consultas pueden invocar métodos e, inversamente, los métodos escritos en cualquier lenguaje de programación pueden incluir consultas.



OQL es un lenguaje declarativo del tipo de SQL, que permite realizar consultas de modo eficiente sobre bases de datos orientadas a objetos, incluyendo primitivas de alto nivel para conjuntos de objetos y estructuras.

- Tiene una sintaxis abstracta.
- Su semántica formal puede definirse fácilmente.
- Proporciona un acceso declarativo a los objetos.
- Proporciona una sintaxis para mezclar las consultas con C++, Smalltalk y Java.
- Tiene una sintaxis concreta al estilo SQL, pero puede cambiarse con facilidad.
- Puede optimizarse fácilmente.
- No proporciona operadores explícitos para la modificación, se basa en las operaciones definidas sobre los objetos para ese fn.
- Proporciona primitivas de alto nivel para tratar con conjuntos de objetos, pero no restringe su utilización con otros constructores de colecciones.

OQL es un lenguaje declarativo del tipo de SQL, que permite realizar consultas de modo eficiente sobre bases de datos orientadas a objetos, incluyendo primitivas de alto nivel para conjuntos de objetos y estructuras.

No posee primitivas para modificar el estado de los objetos, ya que las modificaciones se pueden realizar mediante los métodos que éstos poseen. La sintaxis básica de OQL es una estructura SELECT...FROM...WHERE..., como en SQL.

A continuación, se mostrarán algunos ejemplos de las diferentes combinaciones para ejecutar consultas de OQL. Para ello, se tomará el siguiente modelo:

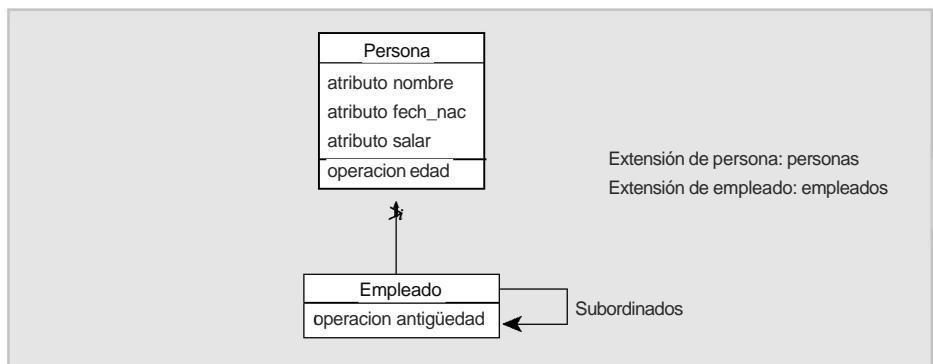


Fig. 8.1 Esquema de Datos.

Este ejemplo retorna la edad de Ana:

```

select distinct x.edad
from x in personas
where x.nombre="Ana"
literal del tipo set <integer>
  
```

Este ejemplo retorna la edad y el salario de Ana:

```
select distinct struct (a:x.edad, s:x.salario)
from x in personas
where x.nombre="Ana"
literal del tipo set <struct (a:integer, s:integer)>
```

Este ejemplo retorna, para cada empleado, el nombre del empleado y sus subordinados que ganan más de 3000.

```
select distinct struct (a:x.nombre, smp:
(select y
from y in x.subordinados
where y.salario>3000))
from x in empleados
literal del tipo set <struct (nombre:string, smp:bag<empleado>)>
```

Retorna la edad y el salario de los empleados con diez años de antigüedad:

```
select struct (a:x.edad, s:x.salario)
from x in
(select y
from y in empleados
where y.antiguedad="10")
literal del tipo bag <struct (a:integer, s:integer)>
```

### 8.5.7 Enfoques para la construcción de BDOO

Dependiendo del fabricante y la forma de encarar los proyectos vigentes de desarrollo de las BDOO, se puede decir que existen tres enfoques distintos de construcción:

- Se puede utilizar el código actual altamente complejo de los sistemas de administración de las bases de datos relacionales, de modo que una BDOO se implante más rápido sin tener que iniciar de cero. Las técnicas orientadas a objetos se pueden utilizar como medios para el diseño sencillo de sistemas complejos. De esta forma, los sistemas se construyen a partir de componentes ya probados con un formato definido para las solicitudes de las operaciones del componente.

- Se considera a la BDOO como una extensión de la tecnología de las bases de datos relacionales. De este modo, las herramientas, las técnicas y la amplia experiencia de la tecnología relacional se utilizan para construir un nuevo Sistema de Gestión de Base de Datos. Se pueden añadir apuntadores a las tablas relacionales para ligarlas con objetos binarios de gran tamaño (BLOB). La base de datos también debe proporcionar a las aplicaciones del tipo cliente un acceso aleatorio y por partes a grandes objetos.
- Producir una nueva arquitectura optimizada, que cumple las necesidades de la tecnología orientada a objetos. Muchos fabricantes como Versant y Objectivity utilizan este enfoque y afirman que la tecnología de las bases de datos relacionales es un subconjunto de una capacidad más general, agregando que las BDOO son casi dos veces más rápidas que las bases de datos relacionales para almacenar y recuperar la información compleja.

## 8.6 Sistema de gestión de BDOO (SGBDOO)

Como ya se ha mencionado, el objetivo principal de las BDOO es permitir el almacenamiento y acceso a objetos complejos para simplificar la programación orientada a objetos. Para ello, es necesario que se almacenen los objetos directamente en la base de datos y se empleen las mismas estructuras y relaciones que los lenguajes de programación orientados a objetos.



Un SGBD es un conjunto de datos relacionados entre sí y un grupo de programas para tener acceso a ellos. Es el encargado de administrar el almacenamiento y el acceso a los datos. Un SGBDOO es un SGBD que almacena objetos y cumple con las mismas propiedades mandatorias y opcionales de un SGBD, pero se diferencia porque trata directamente con objetos y no necesita traducirlos a tablas o filas.

### 8.6.1 Concepto de un SGBDOO

Un Sistema de Gestión de Bases de Datos (SGBD) es un conjunto de datos relacionados entre sí y un grupo de programas para tener acceso a ellos. Es decir, es el encargado de administrar el almacenamiento y el acceso a los datos, tal como ya fue desarrollado en capítulos anteriores, en los que se exponía el funcionamiento de las bases de datos relacionales.

Un Sistema de Gestión de Bases de Datos Orientadas a Objetos (SGBDOO) es un SGBD que almacena objetos y cumple con las mismas propiedades mandatorias y opcionales de un SGBD, pero con la diferencia que este sistema de gestión debe tratar directamente con objetos y no necesita traducirlos a tablas o filas.

La diferencia sustancial entre los Sistemas de Gestión es que las bases de datos relacionales almacenan solo datos, mientras que las Bases de Datos Orientadas a Objetos almacenan objetos, con una estructura arbitraria y un comportamiento. Supongamos como ejemplo que se modela un sistema de ventas de una organización; cuando el sistema emite una factura que se almacenará en la base de datos, tanto para quien emite la factura como para quien la recibe, la factura emitida es un objeto. Ahora bien, para almacenar dicho objeto en una base de datos relacional se lo debe descomponer, dado que es un objeto complejo conformado por objetos simples, por lo cual, por ejemplo, deberá almacenarse como una fila en la tabla "Facturas" y otro conjunto de filas en la tabla "Renglones", sin registrar únicamente una entrada a la base de datos, sino tantas como filas se ingresen. Por el contrario, en una BDOO directamente se almacenará la factura como un objeto complejo y luego se accederá a él también como una unidad.

Normalmente, el SGBDOO se construye sobre una máquina abstracta persistente y totalmente orientada a objetos. El hecho de basarse en una máquina abstracta permite que cualquier aplicación sobre este SGBDOO sea portable a cualquier plataforma, sin necesidad —obviamente— de sobrescribir el código.

El SGBDOO debe ser configurable y modular para que permita la elección del mecanismo de indexación, aunque en la actualidad, en los primeros prototipos existentes, se implementa una técnica de indexación específica, que no impide que se vayan incorporando a dicho sistema técnicas ya existentes o innovadoras. De esta manera, se puedan realizar comparativas para ver cuál de ellas se comporta mejor y en qué situaciones.

#### 8.6.2 Objetivo

Uno de los objetivos del desarrollo del SGBDOO tiene como finalidad generar un desarrollo más sencillo del propio SGBDOO. Esto es lógico, ya que algunas de las funciones como, por ejemplo, la persistencia que debería implementar el SGBD ya están disponibles dentro del propio sistema operativo.

Al tratarse de un sistema integral orientado a objetos, se obtienen las ventajas de la orientación a objetos: por ejemplo, es posible reutilizar el código de persistencia existente o extenderlo añadiendo únicamente la funcionalidad adicional necesaria para el SGBDOO. Esta funcionalidad puede proporcionarse mediante un motor de base de datos adecuado que complemente las características proporcionadas por el sistema operativo.

Otra finalidad es permitir una mayor integración en el sistema, es decir, los objetos de la base de datos son simplemente unos objetos más dentro de los objetos del sistema operativo que proporciona servicios. Es más, puede pensarse el SGBDOO como el elemento que cumple el papel de los sistemas de archivos en los sistemas operativos tradicionales. De esta forma, el SGBDOO no se utilizaría como un sistema independiente del sistema operativo, si no que el usuario lo utilizaría como sistema de gestión de los objetos del sistema operativo y realizaría una consulta sobre los mismos.

También se busca mayor rendimiento, dado que el propio sistema integral ya se orienta a objetos. No existe la necesidad de desarrollar capas superpuestas a un sistema operativo tradicional para salvar el espacio existente entre el paradigma del sistema operativo y el de la base de datos.

Por último, se persigue una mayor productividad, debido a que la programación de aplicaciones de bases de datos es más productiva, ya que no es necesario que el programador cambie constantemente de filosofías: una para trabajar con la base de datos y otra para manejar el sistema operativo, ambos elementos utilizan ahora el mismo paradigma de orientación a objetos.

#### 8.6.3 Características de los SGBDOO

Como se mencionó anteriormente, un SGBDOO debe ser un SGBD que admite la orientación a objetos; por este motivo, debe cumplir con estos dos criterios: ser un sistema orientado a objetos y, también, un sistema de gestión de bases de datos.

Por ello, para ser un sistema orientado a objetos debe cumplir, al menos, con las siguientes características mínimas:



El SGBDOO debe ser configurable y modular para que permita la elección del mecanismo de indexación, aunque en los primeros prototipos, se implementa una técnica de indexación específica, que no impide que se vayan incorporando técnicas ya existentes o innovadoras.

Abstracción: consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan y separar el comportamiento esencial de un objeto de su implementación.

Persistencia: se refleja en el hecho de conservar el estado del objeto, como así también el de la clase a la que pertenece, que debe trascender a un programa en particular de forma tal que todos los programas interpreten el estado en el cual está almacenada.

Concurrencia: es la capacidad de identificar un objeto activo de otro que no lo está.

Encapsulamiento: es la posibilidad de ocultar los detalles de implementación de un objeto determinado del resto de los objetos o de los usuarios que los acceden.

Modularidad: debe permitir la aplicación de la programación modular, es decir, permitir que los procedimientos con una funcionalidad común se almacenen en módulos separados.

Jerarquía: es la que permite la clasificación y el ordenamiento de las diferentes abstracciones creadas.

Genericidad: es la propiedad que permite crear clases genéricas para otras clases.

Polimorfismo: es la posibilidad de que una misma operación sea aplicada a objetos de diferente tipo.

Herencia: es el mecanismo que permite la reusabilidad del código y que un objeto determinado herede propiedades y comportamientos de otro de mayor jerarquía.

Por otra parte, para ser un Sistema de Gestión de Base de Datos, debe cumplir con las siguientes características:

Persistencia: es la capacidad que tiene el programador para que sus datos se conserven al finalizar la ejecución de un proceso, para que se puedan reutilizar en otros.

Concurrencia: se relaciona con la existencia de muchos usuarios, interactuando concurrentemente en el sistema. Este debe controlar la interacción entre las transacciones concurrentes para evitar que se destruya la consistencia de la base de datos.

Recuperación: proporcionar, como mínimo, el mismo nivel de recuperación que los sistemas de bases de datos actuales. De forma que, tanto en caso de fallo de hardware como de fallo de software, el sistema pueda retroceder hasta un estado coherente de los datos.

Gestión del almacenamiento secundario: es soportada por un conjunto de mecanismos que no son visibles al usuario, tales como gestión de índices, agrupación de datos, selección del camino de acceso, optimización de consultas, etcétera. Estos mecanismos evitan que los programadores tengan que escribir programas para mantener índices, asignar el almacenamiento en disco o trasladar los datos entre el disco y la memoria principal, creándose, de esta forma, una independencia entre los niveles lógicos y físicos del sistema.

Facilidad de Consultas: permitir al usuario hacer consultas sencillas a la base de datos. Este tipo de consultas tiene como misión proporcionar la información solicitada por el usuario de una forma correcta y rápida.

A continuación, se analizarán las diferentes arquitecturas implementadas actualmente.

### 8.6.4 Estructura de un SGBDOO

En la actualidad existen diferentes arquitecturas utilizadas para la construcción de sistemas gestores de bases de datos orientadas a objetos.

Algunos de los Sistemas de Gestión de Bases de Datos Orientadas a Objetos se crean como simples gestores de almacenamiento persistente, tanto en el nivel comercial como en el de investigación en proyectos realizados por diferentes universidades. Lo que no se han encontrado son SGBDOO integrados con máquinas abstractas persistentes y sistemas operativos orientados a objetos.

Algunos de ellos representan una mezcla entre las tecnologías de Bases de Datos Orientadas a Objetos y los sistemas de archivos tradicionales, de forma tal que todo dato persistente es un objeto y todo objeto tiene una identidad especificada por un identificador de objeto único. Todos los datos persistentes son descritos mediante el lenguaje ODL propuesto por ODMG.

Estos SGBDOO permiten el control de la concurrencia, la recuperación de caídas, el manejo de transacciones y las consultas optimizadas sobre ciertos tipos. Además, implementan un espacio de nombres, un modelo de control de acceso similar a los de Unix y los servicios tradicionales de toda base de datos, como el acceso asociativo a los datos, la indexación y el agrupamiento.

Otros han sido desarrollados como un sistema de almacenamiento persistente orientado a objetos, desarrollado e implementado como una librería de C++, de tal forma que cualquier aplicación enlazada con dicha librería puede crear y manipular tanto objetos persistentes como transitorios.

Emplean un mecanismo de traducción de punteros (pointer swizzling) cuando se produce el fallo de página. Los objetos almacenan las referencias a otros objetos como identificadores de objetos persistentes sobre disco, pero cuando se trae una página a la memoria todas las referencias sobre esa página son traducidas a punteros en la memoria virtual. Todo esto se realiza de una forma transparente al programa cliente y, una vez que las referencias a los objetos han sido traducidas, los programas clientes pueden acceder rápidamente a los objetos en la memoria.

Por último, otros gestores han desarrollado un sistema de base de datos distribuido orientado a objetos, que proporciona un sistema de almacenamiento persistente de objetos y que permite acceder a ellos mediante transacciones.

En este esquema un objeto se convierte en persistente cuando es alcanzado por el objeto raíz. La arquitectura se organiza en torno a un repositorio de objetos residente en el servidor y a unas aplicaciones del tipo cliente almacenados en los clientes. Estos procesos front-end son los que hacen las veces de intermediarios entre el programa cliente y el repositorio de objetos.

De esta forma, se proporciona un programa cliente para cada lenguaje soportado, que incluye una librería de código que implementa la interfaz entre el cliente y el front-end.

Los objetos dentro se especifican e implementan usando un nuevo lenguaje procedural de programación orientado a objetos, extensible y fuertemente tipado, que permite tanto la especificación de la interfaz de un tipo como su implementación.



Los objetos almacenan las referencias a otros objetos como identificadores de objetos persistentes sobre disco, pero cuando se trae una página a la memoria todas las referencias sobre esa página son traducidas a punteros en la memoria virtual.

## 8.7 Rendimiento de las BDOO

En cuanto al rendimiento de las BDOO en comparación con las bases de datos relacionales, cabe señalar que las primeras permiten que los objetos hagan referencia directamente a otro mediante apuntadores suaves. Esto les permite que comuniquen más rápido del objeto A al objeto B que las relacionales, dado que estas últimas deben utilizar comandos JOIN para lograr esa comunicación y, aunque dicho comando JOIN se optimice, es más lento que un recorrido de los objetos. Así, incluso, sin alguna afinación especial, una BDOO es en general más rápida en esta mecánica que una relacional.

Además, las BDOO hacen que el agrupamiento sea más eficiente. La mayoría de los sistemas de bases de datos permiten que el operador coloque cerca las estructuras relacionadas entre sí, en el espacio de almacenamiento en disco. Esto reduce en forma radical el tiempo de recuperación de los datos relacionados, puesto que todos los datos se leen con una lectura de disco en lugar de varias. Sin embargo, en una base de datos relacional, los objetos de la implementación se traducen en representaciones tabulares que generalmente se dispersan en varias tablas. Así, en una base de datos relacional, estos renglones relacionados deben quedar agrupados, de modo que todo el objeto se pueda recuperar mediante una única lectura del disco. Esto es automático en una BDOO. Asimismo, el agrupamiento de los datos relacionados, como todas las subpartes de un ensamble, puede afectar radicalmente el rendimiento general de una aplicación. Esto es relativamente directo en una BDOO, puesto que representa el primer nivel de agrupamiento. Por el contrario, el agrupamiento físico es imposible en una base de datos relacional, ya que esto requiere un segundo nivel de agrupamiento: un nivel para agrupar las filas que representan a los objetos individuales y otro para los grupos de filas que representan a los objetos relacionados.

## 8.8 Ventajas de las BDOO

Dentro de las ventajas de una BDOO, se encuentran las siguientes características:

- Mayor capacidad de modelado: permite modelar el mundo real sin tener que desagregarlo, esto se debe a que un objeto permite encapsular su estado y su comportamiento, puede almacenar todas las relaciones que tenga con otros objetos y agruparse para formar objetos complejos.
- Flexibilidad: esto ocurre, por ejemplo, en una base de datos relacional. Si una empresa obtiene clientes por referencia de otros que ya posee, pero la base de datos existente —que mantiene la información de clientes y sus compras— no tiene un campo para registrar quién proporcionó la referencia, de qué manera fue dicho contacto o si debe compensarse con una comisión, será necesario reestructurar la base de datos para añadir este tipo de modificaciones. Por el contrario, en una BDOO, el usuario puede añadir una subclase de la clase de clientes para manejar las modificaciones que representan los clientes por referencia. La subclase heredará todos los atributos, características de la definición original. Además, se especializará en especificar los nuevos campos que se requieren, así como los métodos para manipularlos.

Naturalmente, se generan los espacios para almacenar la información adicional de los nuevos campos. Esto presenta la ventaja adicional: que una BDOO puede ajustarse a usar siempre el espacio de los campos que son necesarios, eliminando el espacio desperdiciado en registros con campos que nunca se utilizaron.

- Manipula datos complejos en forma ágil: la estructura de la base de datos está dada por referencias (o apuntadores lógicos) entre objetos, esto hace que su acceso sea más rápido.
- Extensibilidad: esto se debe a que se pueden construir nuevos tipos de datos a partir de los existentes; también permite agrupar propiedades comunes de diferentes clases en una superclase, lo que disminuye la redundancia y también permite la reusabilidad de clases. De esta manera, se disminuye el tiempo de desarrollo y se facilita el mantenimiento.
- Acceso navegacional: es la forma más común de acceso en una BDOO, desde un objeto al siguiente, mientras que un sistema relacional utiliza un acceso asociativo. El acceso navegacional es superior al asociativo para gestionar algunas de las operaciones comunes en una base de datos, como el despiece de un elemento o las consultas recursivas.

## 8.9 Desventajas de las BDOO

Dentro de las desventajas de una BDOO, se pueden mencionar:

- Ausencia de un modelo de datos universal: a diferencia de lo que ocurre con el modelo relacional, que está totalmente unificado, no existe un modelo de datos unificado para las BDOO y la mayoría de ellos carecen de una base teórica fuerte.
- Inmadurez del mercado: no se cuenta con la experiencia suficiente, como en los modelos relacionales.
- Falta de estándares: a diferencia de lo que ocurre con SQL, si bien hay aproximaciones a estándares, todavía no se cuenta con uno asumido por todos los fabricantes.
- Falta de base teórica: el modelo de objetos no posee una teoría matemática que le proporcione sustento, mientras que el modelo relacional, sí, dado que se basa en el Álgebra relacional.
- Optimización de consultas: la optimización requiere una mejor implementación y un acceso a los objetos más eficiente. Este concepto va en contra del encapsulamiento aplicado en las BDOO.
- Competencia: por sus experiencias, las bases de datos relacionales han construido estándares de construcción y uso; poseen un lenguaje SQL, que es un estándar aprobado por todos los fabricantes, y han construido una gran cantidad de herramientas que simplifican la tarea a los desarrolladores y se aproximan a la idea de un modelo unificado. Todo esto atenta contra el avance de las BDOO.

# S

Los sistemas de bases de datos relacionales orientadas a objetos son sistemas basados en el modelo relacional, pero que además proporcionan las ventajas del paradigma orientado a objetos para el tratamiento de los datos.

# S

Una BDOR es una base de datos que, desde el modelo relacional, evoluciona hacia una base de datos más extensa y compleja e incorpora conceptos del modelo orientado a objetos.

## 8.10 Bases de Datos Objeto-Relacionales

### 8.10.1 Concepto

Los sistemas de bases de datos relacionales orientadas a objetos son sistemas basados en el modelo relacional, pero que además proporcionan las ventajas del paradigma orientado a objetos para el tratamiento de los datos.

Una de las principales metas de este nuevo modelo es mejorar la representación y el acceso a los datos mediante la aplicación de la orientación a objetos, manteniendo el sistema de almacenamiento y operatoria del modelo relacional.

Una Base de Datos Objeto-Relacional (BDOR) es una base de datos que, desde el modelo relacional, evoluciona hacia una base de datos más extensa y compleja e incorpora, para obtener este fin, conceptos del modelo orientado a objetos.

Se puede afirmar por ello que un Sistema de Gestión de Base de Datos Objeto-Relacional (SGBDOR) contiene dos tecnologías: la relacional y la de objetos.

En una Base de Datos Objeto-Relacional se siguen almacenando tuplas como en la relacional, aunque la estructura de las tuplas no está restringida a contener escalares, sino que las relaciones se pueden definir en función de otras, que es lo que se denomina herencia directa.

El modo en que los objetos han entrado en el mundo de las bases de datos relacionales es en forma de dominios, actuando como el tipo de datos de una columna.

Hay dos características muy importantes por el hecho de utilizar una clase como un dominio. En la primera, es posible almacenar múltiples valores en una columna de una misma fila, ya que un objeto suele contener múltiples valores; sin embargo, si se utiliza una clase como dominio de una columna, en cada fila esa columna, solo se puede contener un objeto de la clase, debido a que se sigue manteniendo la restricción del modelo relacional que contiene los valores atómicos en la intersección de cada fila con cada columna. En la segunda, es posible almacenar procedimientos en las relaciones, porque un objeto está enlazado con el código de los procesos que sabe realizar, es decir, los métodos de su clase.

Otro modo de incorporar objetos en las Bases de Datos Relacionales es construyendo tablas de objetos, donde cada fila es un objeto, dado que un sistema objeto-relacional es un sistema relacional que permite almacenar objetos en sus tablas. La base de datos sigue sujeta a las restricciones que se aplican en todas las Bases de Datos Relacionales y conserva la capacidad de utilizar operaciones de concatenación (join) para implementar las relaciones, pero ahora cada fila contiene un objeto completo y no valores escalares simples.

### 8.10.2 Características de las BDOR

Con las Bases de Datos Objeto-Relacionales, se pueden crear nuevos tipos de datos, que permiten gestionar aplicaciones más complejas con una gran riqueza de dominios.

Estos nuevos tipos pueden ser compuestos, lo que implica que se debe definir, al menos, dos métodos transformadores: uno para convertir el tipo nuevo a ASCII y otro que convierte de ASCII al nuevo tipo.

Dentro de las características con las que cuentan este tipo de base de datos, se encuentran:

- Soporta tipos complejos como registros, conjuntos, referencias, listas, pilas, colas y arreglos.
- Se pueden crear funciones que tengan un código en algún lenguaje de programación independientemente de SQL, como Java, C++, C#, etcétera.
- Existe una mayor capacidad expresiva para los conceptos y asociaciones.
- Se pueden crear operadores asignándole un nombre y existencia de nuevas consultas con mayor capacidad consultiva.
- Se soporta el encadenamiento dinámico y herencia en los tipos tupla o registro.
- Se pueden compartir varias bibliotecas de clases ya existentes, esto es lo que conocemos como reusabilidad.
- Mantienen la posibilidad de incluir el chequeo de las reglas de integridad referencial a través de los triggers.
- Contiene soporte adicional para seguridad y activación de la versión cliente-servidor.

#### 8.10.3 Implementación en Oracle

A continuación, se analizará la forma de implementación de una BDOR sobre Oracle, como exemplificación de cómo pueden implementarse soluciones en este tipo de base de datos.

Los tipos de objetos en Oracle son tipos de datos definidos por el usuario. La tecnología de objetos que proporciona es una capa de abstracción construida sobre su propia tecnología relacional como una capa superior a la externa, por lo cual los datos siguen siendo almacenados en el formato fila, columna y en tablas.

A partir de aquí, se resumirá la orientación a objetos que soportan las versiones de Oracle actuales.

##### 8.10.3.1 Tipos de objetos y referencias

Para crear tipos de objetos en Oracle se utiliza la sentencia CREATE TYPE. A continuación, se enumeran algunos ejemplos de su utilización:

```
CREATE TYPE persona AS OBJECT (
    nombre    VARCHAR2 (30),
    direccion VARCHAR2 (50),
    telefono  VARCHAR2 (20)
);
```

```
CREATE TYPE renglon_pedido AS OBJECT (
    nombre_producto  VARCHAR2 (30),
```



Los tipos de objetos en Oracle son tipos de datos definidos por el usuario. La tecnología de objetos que proporciona es una capa de abstracción construida sobre su propia tecnología relacional como una capa superior a la externa, por lo cual los datos siguen siendo almacenados en el formato fila, columna y en tablas.

```

        cantidad      NUMBER,
        precio_unitario  NUMBER (12, 2)
);

CREATE TYPE renglon_pedido_tabla AS TABLE OF renglon_pedido;

CREATE TYPE pedido AS OBJECT
(
    id NUMBER,
    contacto persona,
    lineaspedido renglon_pedido_tabla,

    MEMBER FUNCTION obtener_valor RETURN NUMBER
);

```

*lineaspedido* es lo que se denomina una tabla anidada (*nested table*), que es un objeto de tipo colección. Una vez creados los objetos, se pueden utilizar como un tipo de dato de la misma manera que NUMBER o VARCHAR2.

Por ejemplo, se puede definir una tabla relacional para guardar información de personas de contacto:

```

CREATE TABLE contactos
(
    contacto persona,
    fecha DATE
);

```

Ésta es una tabla relacional que tiene una columna cuyo tipo es un objeto. Cuando los objetos se utilizan de este modo, se los denomina *objetos columna*.

Cuando se declara una columna como un tipo de objeto o como una tabla anidada, se puede incluir una cláusula DEFAULT para asignar valores por defecto:

```

CREATE TYPE persona AS OBJECT
(
    Id_persona NUMBER,
    nombre VARCHAR2(30),
    direccion VARCHAR2(50),
);

```

```
CREATE TYPE gente AS TABLE OF persona;

CREATE TABLE Sector
(
    Numero_sector VARCHAR2(5) PRIMARY KEY,
    nombre_sector VARCHAR2(20),
    director persona DEFAULT persona(1,'Jose Rodriguez',NULL),
    empleados gente DEFAULT gente( persona(2,'Juan Perez','Corrientes 30'),
    persona(3,'Pedro Rodriguez', 'Callao 3'))
)
NESTED TABLE empleados STORE AS empleados_tab;
```

Las columnas que son tablas anidadas y los atributos que son tablas de objetos requieren una tabla aparte donde almacenar las filas de dichas tablas. Esta tabla de almacenamiento se especifica mediante la cláusula NESTED TABLE...STORE AS.

Para recorrer las filas de una tabla anidada se utilizan cursorse anidados. Sobre las tablas de objetos también se pueden definir restricciones. En el siguiente ejemplo, se muestra cómo definir una clave primaria sobre una tabla de objetos:

```
CREATE TYPE ubicacion AS OBJECT
(
    num_edificio NUMBER,
    ciudad VARCHAR2(30)
);
```

```
CREATE TYPE persona2 AS OBJECT
(
    id NUMBER,
    nombre VARCHAR2(30),
    direccion VARCHAR2(30),
    oficina ubicacion
);
```

```
CREATE TABLE empleados OF persona2
(
    id PRIMARY KEY
);
```

El siguiente ejemplo define restricciones sobre atributos escalares de un objeto columna:

```

CREATE TABLE departamento
(
    num_dept VARCHAR2(5) PRIMARY KEY,
    nombre_dept VARCHAR2(20),
    director persona,
    despacho ubicacion,
    CONSTRAINT despacho_cons1
    UNIQUE (despacho.num_edificio, despacho.ciudad),
    CONSTRAINT despacho_cons2
    CHECK (despacho.ciudad IS NOT NULL)
);

```

Sobre las tablas de objetos también se pueden definir *triggers*, pero sobre las tablas de almacenamiento definidas mediante NESTED TABLE, no.

```

CREATE TABLE traslado
(
    id NUMBER,
    despacho_antiguo ubicacion,
    despacho_nuevo ubicacion
);

CREATE TRIGGER disparador
AFTER UPDATE OF despacho_nuevo ON traslado
FOR EACH ROW
--WHEN :NEW.despacho_nuevo.ciudad = 'Castellon'
BEGIN
    IF (:NEW.despacho_nuevo.num_edificio = 600)
    THEN
        INSERT INTO traslado
        (ID, despacho_antiguo, despacho_nuevo
        )
        VALUES (:OLD.ID, :OLD.despacho_nuevo, :NEW.despacho_nuevo
        );
    END IF;
END;

```

Las relaciones se establecen mediante columnas o atributos REF. Estas relaciones pueden estar restringidas mediante la cláusula SCOPE o mediante una restricción de integridad referencial (REFERENTIAL).

Cuando se restringe mediante SCOPE, todos los valores almacenados en la columna REF apuntan a objetos de la tabla especificada en la cláusula; sin embargo, puede ocurrir que haya valores que apunten a objetos que no existen.

La restricción mediante REFERENTIAL es similar a la especificación de claves foráneas. En este caso, la regla de integridad referencial se aplica a estas columnas, por lo que las referencias a objetos que se almacenen en estas columnas deben ser siempre de objetos que existen en la tabla referenciada.

Para evitar ambigüedades con los nombres de atributos y de métodos al utilizar la notación punto, Oracle obliga, en la mayoría de las ocasiones, a utilizar alias para las tablas.

Por ejemplo:

```
CREATE TYPE persona3 AS OBJECT (dni VARCHAR2(9));
```

```
CREATE TABLE ptab1 OF persona3;
```

```
CREATE TABLE ptab2 (c1 persona3);
```

donde las siguientes consultas muestran modos correctos e incorrectos de referenciar el atributo DNI:

Modo Correcto:

```
SELECT dni FROM ptab1;  
SELECT p.c1.dni FROM ptab2 p;
```

S

El motor arrojará el error: c1.dni identificador no válido, porque no puede referenciar al DNI desde c1.

Modo Incorrecto:

```
SELECT c1.dni FROM ptab2;  
SELECT ptab2.c1.dni FROM ptab2;
```

S

El motor arrojará el error: ptab2.c1.dni identificador no válido, porque no puede referenciar al DNI desde ptab2.c1.

### 8.10.3.2 Métodos

Los métodos son funciones o procedimientos que se pueden declarar en la definición de un tipo de objeto para implementar el comportamiento que se desea para el mismo.

Las aplicaciones llaman a los métodos para invocar su comportamiento; para ello, se utiliza también la notación punto de la forma objeto.metodo(lista parámetro). Aunque un método no tenga parámetros, Oracle obliga a utilizar los paréntesis en las llamadas objeto.metodo().

Hay tres clases de métodos: miembros, estáticos y constructores, que el propio sistema define para cada tipo de objeto.

Los métodos miembro son los que se utilizan para ganar acceso a los datos de una instancia de un objeto. Se debe definir un método para cada operación que se desea que haga el tipo de objeto. Estos métodos tienen un parámetro denominado SELF que denota la instancia del objeto sobre la que se está invocando el método. Los métodos miembro pueden hacer referencia a los atributos y a los métodos de SELF, sin necesidad de utilizar el calificador.

```

CREATE TYPE racional AS OBJECT
(
    num INTEGER,
    den INTEGER,
    MEMBER PROCEDURE normaliza
);

CREATE TYPE BODY racional
AS

    MEMBER PROCEDURE normaliza
    AS

        g INTEGER;

    BEGIN

        g := gcd ( SELF.num, SELF.den);
        g := gcd(num, den);
        num := num / g;
        den := den / g;

    END

```

SELF no necesita declararse, aunque lo puede hacer. Si no se declara, en las funciones se pasa como IN y en los procedimientos como IN OUT.

Los valores de los tipos de datos escalares siguen un orden y, por lo tanto, se pueden comparar; sin embargo, con los tipos de objetos que pueden tener múltiples atributos de distintos tipos, no hay un criterio predefinido de comparación.

Para poder comparar objetos se debe establecer este criterio mediante métodos de mapeo o de orden.

Un método de mapeo (MAP) permite comparar objetos mapeando instancias de objetos con tipos escalares DATE, NUMBER, VARCHAR2 o cualquier tipo ANSI SQL como CHARACTER o REAL.

Un método de mapeo es una función sin parámetros que devuelve uno de los tipos anteriores.

Si un tipo de objeto define uno de estos métodos, el método se llama automáticamente para evaluar comparaciones del tipo obj1 > obj2 y para evaluar las comparaciones que implican DISTINCT, GROUP BY y ORDER BY.

```
CREATE TYPE rectangulo AS OBJECT (
    alto NUMBER,
    ancho NUMBER,
    MAP MEMBER FUNCTION area
        RETURN NUMBER
);
CREATE TYPE BODY rectangulo AS
    MAP MEMBER FUNCTION area RETURN NUMBER IS
    BEGIN
        RETURN alto*ancho;
    END
    /
```

Los métodos de orden ORDER hacen comparaciones directas objeto–objeto; son funciones con un parámetro declarado para otro objeto del mismo tipo. El método se debe escribir para que devuelva un número negativo, cero o un número positivo, lo que significa que el objeto SELF es menor, igual o mayor que el otro objeto que se pasa como parámetro.

Los métodos de orden se utilizan cuando el criterio de comparación es muy complejo como para implementarlo con un método de mapeo.

Un tipo de objeto puede declarar solo un método de mapeo o solo un método de orden, de manera que cuando se comparan dos objetos, se llama automáticamente al método que se haya definido, sea de uno u otro tipo.

Los métodos estáticos son los que pueden ser invocados por el tipo de objeto y no por sus instancias. Estos métodos se utilizan para operaciones que son globales al tipo y que no necesitan referenciar datos de una instancia determinada. Los métodos estáticos no requieren el parámetro SELF porque no mencionan un objeto determinado.

Para invocar estos métodos, se utiliza la notación punto sobre el tipo del objeto, de la forma: tipo objeto.metodo().

Cada tipo de objeto tiene un método constructor implícito definido por el sistema, que crea un nuevo objeto, es decir, una instancia de ese tipo, y pone valores en sus atributos.

El método constructor es una función y devuelve el nuevo objeto como su valor.

Como en casi todos los lenguajes de programación orientada a objetos, el nombre del método constructor es, precisamente, el nombre del tipo de objeto y sus parámetros tienen los nombres y los tipos de los atributos del tipo.

```
CREATE TABLE departamento2
(
    num_dept VARCHAR2(5) PRIMARY KEY,
    nombre_dept VARCHAR2(20),
    despacho ubicacion
);

INSERT INTO departamento2
VALUES ( '233', 'Ventas', ubicacion(200,'Borriol') );
```

#### 8.10.3.3 Colecciones

Para representar colecciones, Oracle soporta dos tipos de datos colección: las tablas anidadas y los varray.

Un varray es una colección ordenada de elementos; la posición de cada uno de ellos viene dada por un índice que permite acceder a los mismos. De la misma manera que en cualquier array, cuando se define un varray se debe especificar el número máximo de elementos que puede contener, aunque este número puede ser modificado luego, del mismo modo que en lenguajes como C++. Los varray se almacenan como objetos RAW o BLOB.

Una tabla anidada puede tener cualquier número de elementos, dado que no se especifica ningún máximo cuando se la define. En ellas no se mantiene el orden de los elementos.

En las tablas anidadas se consultan y actualizan datos del mismo modo que se hace con las tablas relacionales. Los elementos de una tabla anidada se almacenan en una tabla aparte, en la que hay una columna denominada NESTED TABLE ID, que referencia la tabla padre o el objeto al que pertenece.

```
CREATE TYPE precios AS VARRAY(10) OF NUMBER(12,2)
/
CREATE OR REPLACE TYPE lineaped_tabla AS TABLE OF renglon_pedido
/
```



Un varray es una colección ordenada de elementos; la posición de cada uno de ellos viene dada por un índice que permite acceder a los mismos. Cuando se define un varray se debe especificar el número máximo de elementos que puede contener, aunque este número puede ser modificado luego, del mismo modo que en lenguajes como C++.

Cuando se utiliza una tabla anidada como una columna de una tabla o como un atributo de un objeto, es preciso especificar cuál sería su tabla de almacenamiento mediante NESTED TABLE...STORE AS....

Se pueden crear tipos colección multinivel, que son tipos colección cuyos elementos son colecciones.

```
CREATE TYPE satelite AS OBJECT (
    nombre  VARCHAR2 (20),
    diametro NUMBER
);

CREATE TYPE tab_satelite AS TABLE OF satelite;

CREATE TYPE planeta AS OBJECT (
    nombre  VARCHAR2 (20),
    masa    NUMBER,
    satelites tab_satelite
);

CREATE TYPE tab_planeta AS TABLE OF planeta;
```

En este caso, la especificación de las tablas de almacenamiento se debe hacer para todas y cada una de las tablas anidadas.

```
CREATE TABLE estrellas
(
    nombre VARCHAR2(20),
    edad NUMBER,
    planetas tab_planeta )
NESTED TABLE planetas STORE AS tab_alm_planetas
(NESTED TABLE satelites STORE AS tab_alm_satelites);
```

Para crear una instancia de cualquier tipo de colección, también se utiliza el método constructor, tal y como se hace con los objetos.

```
INSERT INTO estrellas
VALUES ('Sol', 23,
        tab_planeta (planeta ('Neptuno',
                               10,
```

```

        tab_satelite (satelite ('Proteus', 67),
                      satelite ('Triton', 82)
                    )
      ),
    planeta ('Jupiter',
              189,
              tab_satelite (satelite ('Calisto', 97),
                            satelite ('Ganimedes', 22)
                          )
            )
  ));

```

Las colecciones se pueden consultar con los resultados anidados de la siguiente manera:

```

SELECT e.nombre, e.planetas
FROM estrellas e;

```

O también con los resultados sin anidar, como sigue:

```

SELECT e.nombre, p.*
FROM estrellas e, TABLE(e.planetas) p;

```

La expresión que aparece en TABLE puede ser tanto el nombre o una subconsulta de una colección.

Las dos consultas que se muestran a continuación obtienen el mismo resultado:

```

SELECT p.*
FROM estrellas e, TABLE (e.planetas) p
WHERE e.edad = 23;

SELECT *
FROM TABLE (SELECT e.planetas
            FROM estrellas e
            WHERE e.edad = 23);

```

También es posible hacer consultas con resultados no anidados sobre colecciones multinivel.

```
SELECT s.nombre  
FROM estrellas e, TABLE (e.planetas) p, TABLE (p.satellites) s;
```

#### 8.10.3.4 Herencia de tipos

Oracle soporta herencia de tipos; cuando se crea un subtipo a partir de un tipo, el subtipo hereda todos los atributos y los métodos del tipo padre. Cualquier cambio en los atributos o en los métodos del tipo padre se reflejan automáticamente en el subtipo que lo heredó.

Un subtipo se convierte en una versión especializada del tipo padre cuando al subtipo se le añaden atributos o métodos, o cuando se redefinen los métodos que ha heredado, de modo que al subtipo que ejecuta el método a su propia forma se lo denomina polimorfismo ya que, dependiendo del tipo del objeto sobre el que se invoca el método, se ejecuta uno u otro código.

Cada tipo puede heredar de un solo tipo, no de varios a la vez; esto indica que no soporta la herencia múltiple, pero se pueden construir jerarquías de tipos y subtipos.

Cuando se define un tipo de objeto, se determina si de él se pueden derivar subtipos mediante la cláusula NOT FINAL. Si no se incluye esta cláusula, se considera que es FINAL, es decir, que no puede tener subtipos.

Del mismo modo que para los objetos, también los métodos pueden ser FINAL o NOT FINAL.

Si un método es final, los subtipos no pueden redefinirlo con una nueva implementación. Por defecto, los métodos son NOT FINAL, es decir que son redefinibles.

```
CREATE TYPE t1 AS OBJECT (  
    ID NUMBER,  
    MEMBER PROCEDURE imprime,  
    FINAL MEMBER FUNCTION fun (x NUMBER)  
        RETURN NUMBER  
)  
NOT FINAL;
```

Para crear un subtipo se utiliza la cláusula UNDER.

```
CREATE TYPE PERSONA_NOT_FINAL AS OBJECT  
(  
    nombre VARCHAR2(30),  
    direccion VARCHAR2(50),  
    telefono VARCHAR2(20)  
) NOT FINAL;
```

```
CREATE TYPE estudiante UNDER PERSONA_NOT_FINAL (
    titulacion    VARCHAR2 (30),
    fecha_ingreso DATE
)
NOT FINAL;
```

El nuevo tipo, además de heredar los atributos y métodos del tipo padre, define dos nuevos atributos. A partir del subtipo se pueden derivar otros subtipos y del tipo padre, también. Para redefinir un método, se utilizará la cláusula **OVERRIDING**.

Los tipos y los métodos se pueden declarar como no instanciables.

Si un tipo no es instanciable, no tiene método constructor, dado que no se pueden crear instancias a partir de él.

Un método no instanciable se utiliza cuando no se le dará una implementación en el tipo en el que se declara sino que cada subtipo va a proporcionar una implementación distinta.

```
CREATE TYPE t1 AS OBJECT (
    ID NUMBER,
    NOT INSTANTIABLE MEMBER FUNCTION fun ()RETURN NUMBER
) NOT INSTANTIABLE NOT FINAL;
```



La firma es la combinación del nombre de un método, el número de parámetros, los tipos de estos parámetros y el orden formal.

Un tipo puede definir varios métodos con el mismo nombre, pero con distinta firma. La firma es la combinación del nombre de un método, el número de parámetros, los tipos de estos parámetros y el orden formal. A esto se lo denomina sobrecarga de métodos (*overloading*).

En una jerarquía de tipos, los subtipos son variantes de la raíz. Por ejemplo, el tipo estudiante y el empleado son clases de persona. Normalmente, cuando se trabaja con jerarquías, a veces se quiere trabajar a un nivel más general, por ejemplo, seleccionar o actualizar todas las personas y, a veces, se quiere trabajar solo con los estudiantes o únicamente con los que no son estudiantes.

La habilidad para seleccionar todas las personas juntas, pertenezcan o no a algún subtipo, es lo que se denomina sustituibilidad.

Un supertipo es sustituible si uno de sus subtipos puede sustituirlo en una variable, columna, etc., declarada del tipo del supertipo.

En general, los tipos son sustituibles, esto es: un atributo definido como *REF susTipo* puede contener una *REF* a una instancia de *susTipo* o a una instancia de cualquier subtipo de *susTipo*. Del mismo modo, un atributo definido de tipo *susTipo* puede contener una instancia de *susTipo* o una instancia de cualquier subtipo de *susTipo*.

En cuanto a las colecciones, una colección de elementos de tipo *susTipo* puede contener instancias de *susTipo* o instancias de cualquier subtipo de *susTipo*.

Dado el tipo libro:

```
CREATE TYPE libro AS OBJECT (
    titulo  VARCHAR2 (30),
    autor   persona
);
```

Se puede crear una instancia de libro especificando un título y un autor de tipo persona o de cualquiera de sus subtipos, estudiante o empleado:

```
libro('Bases de datos',
estudiante(123,'Carlos Lopez','Castro Barros 30','II','10-OCT-10')
```

A continuación, se muestra un ejemplo de la sustituibilidad en las tablas de objetos.

```
CREATE TYPE persona5 AS OBJECT (
    ID      NUMBER,
    nombre  VARCHAR2 (30),
    direccion  VARCHAR2 (30)
)
NOT FINAL;

CREATE TYPE estudiante2 UNDER persona5 (
    titulacion  VARCHAR2 (10),
    especialidad  VARCHAR2 (30)
)
NOT FINAL;

CREATE TYPE estudiante_doctorado UNDER estudiante2 (
    programa  VARCHAR2(10)
);

CREATE TABLE personas_tab OF persona5;

INSERT INTO personas_tab
    VALUES (persona5 (1234, 'Ana', 'Callao 23'));

INSERT INTO personas_tab
    VALUES (estudiante2 (2345, 'Jose', 'Mitre 33', 'ITDI', 'Mecánica'));

INSERT INTO personas_tab
    VALUES (estudiante_doctorado (3456, 'Luis', 'Vidt 45', 'IInf,
```

```
NULL,
'CAA'
));
```

*VALUE*: esta función toma como parámetro un alias de tabla (de una tabla de objetos) y devuelve instancias de objetos correspondientes a las filas de la tabla.

```
SELECT VALUE (p)
FROM personas_tab p
WHERE p.direccion LIKE 'C/Corrientes %';
```

La consulta devuelve todas las personas que viven en la calle Corrientes, sean o no de algún subtipo.

*REF*: es una función que toma como parámetro un alias de tabla y devuelve una referencia a una instancia de un objeto de dicha tabla.

*DEREF*: es una función que devuelve la instancia del objeto correspondiente a una referencia que se le pasa como parámetro.

*IS OF*: permite formar predicados para comprobar el nivel de especialización de instancias de objetos.

```
SELECT VALUE (p)
FROM personas_tab p
WHERE VALUE (p) IS OF (estudiante2);
```

De este modo, se obtienen las personas que son del subtipo estudiante o que son de alguno de sus subtipos. Para obtener solamente aquellas personas cuyo tipo más específico es estudiante, se utiliza la cláusula *ONLY*:

```
SELECT VALUE (p)
FROM personas_tab p
WHERE VALUE (p) IS OF (ONLY estudiante2);
```

*TREAT*: es una función que trata una instancia de un supertipo como una instancia de uno de sus subtipos:

```
SELECT TREAT (VALUE (p) AS estudiante2)
FROM personas_tab p
WHERE VALUE (p) IS OF (ONLY estudiante2);
```

## 8.11 Resumen

Como puede observarse, los sistemas de bases de datos relacionales han avanzado fuertemente sobre la incorporación de objetos en su estructura de desarrollo, almacenamiento y acceso. Si bien aún no llegan a aplicar todas las propiedades y fundamentos de la orientación a objetos, están muy próximos a realizarlo.

Esto debilita fuertemente a los nuevos desarrollos sobre objetos que están realizando otros fabricantes, dado que por la gran experiencia de los sistemas gestores de bases de datos relacionales, su gran robustez y presencia en el mercado, los hacen fuertes competidores de cualquier nuevo desarrollo que pueda originar cambios profundos en el mercado actual.

Por lo dicho precedentemente, se debe esperar un futuro con la convivencia de ambos modelos, entendiendo que el lugar adonde se va, es a la interpretación de la orientación a objetos como una capa de abstracción superior en la construcción de gestores de base de datos que permita al usuario operar con la base como si ésta almacenara en objetos, aunque realmente no lo haga.

## 8.12 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) solo está disponible para docentes.

### 8.12.1 Mapa conceptual del capítulo

### 8.12.2 Autoevaluación

### 8.12.3 Presentaciones\*



# 9

## Implementando el modelo en Oracle Express Edition XE

### Contenido

|   |     |
|---|-----|
| 9.1. Introducción.....  | 276 |
| 9.2 Creando y probando nuestro modelo de<br>Estudiante - Universidad..... | 283 |
| 9.3 Creación de las tablas y sus relaciones.....                          | 285 |
| 9.4 Contenido de la página Web de apoyo.....                              | 296 |

### Objetivos

- Saber dónde conseguir el instalador oracleXE.exe de la base Oracle edición Express.
- Entender cómo es el proceso de instalación.
- Conectar a la base instalada como usuario administrador.
- Crear el usuario dueño de las tablas del modelo visto en el libro.
- Crear las tablas del modelo y un conjunto de datos de prueba.
- Crear las funciones PL/SQL de la aplicación.
- Ejecutar algunas pruebas del código almacenado en la base.

## 9.1. Introducción

### 9.1.1 Obteniendo el software e inicializando la instalación del Motor Oracle 10g XE

En primer lugar se bajará el software para instalarlo en la máquina. Una vez instalado, se podrá contactar desde los programas clientes incluidos en la instalación.



En la Web de apoyo, encontrará el vínculo que le brindará mayor información acerca del producto Oracle Express Edition XE.

### 9.1.2 Descarga

La edición Express del Motor Oracle 10g es de distribución gratuita y su programa instalador puede descargarse desde el centro de recursos de Oracle.com, la Red técnica o Technet. El link de la red es [otn.oracle.com](http://otn.oracle.com)

Desde la pantalla principal, luego de registrarse sin ningún costo, se deberá recordar el usuario obtenido y su contraseña porque, en el momento de iniciar la descarga, se les pedirá para su registro. En la página de descarga, que se encuentra en los links ubicados a la derecha, se accederá al link específico del sistema operativo en el cual se instalará y allí se advertirá la versión que se usará; en este caso, la de soporte Universal, que incluye varios idiomas. Las pantallas pertenecen a la instalación con la versión de ejecutable que responde al nombre de Universal.

Ya descargado, se encuentra un archivo ejecutable denominado OracleXE.exe, con un tamaño de, aproximadamente, 160 Mb. Cuando se inicia comienza la instalación, que se describirá en el siguiente punto.

La ayuda quedará instalada en un ícono en el escritorio con la leyenda “Introducción a Base de Datos Oracle 10g Express Edition”. En las propiedades del ícono del acceso directo a la documentación, se observará el directorio en el que se encuentra el punto de inicio de la gestión de la ayuda (Fig. 9.1).



Fig. 9.1 Ícono de documentación.

### 9.1.3 Instalación

Al iniciar el programa de instalación, las pantallas de las Figs. 9.2 y 9.3 mostrarán las distintas etapas y requerirán la información necesaria para la instalación inicial del motor de la base y sus archivos ejecutables. Además, definirán dónde se instalarán los archivos de la base propiamente dicha y configurarán el entorno de servidor http para las posteriores conexiones al motor.

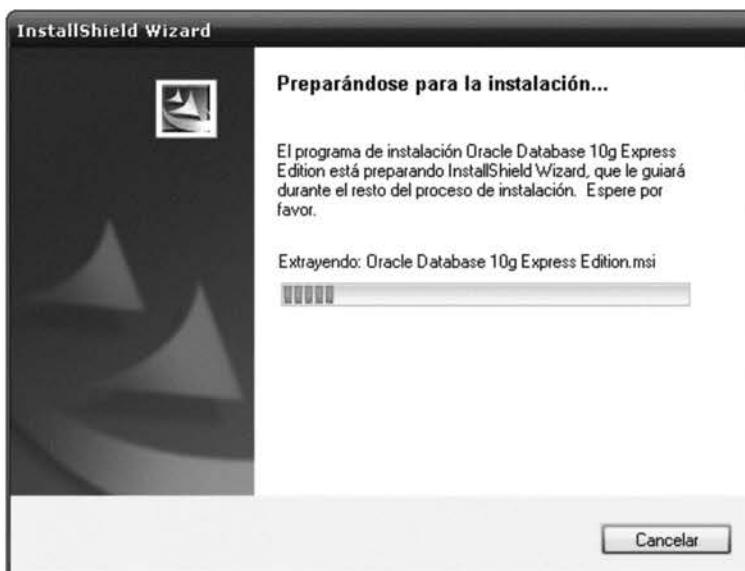


Fig. 9.2 Primera pantalla de avance de la instalación.

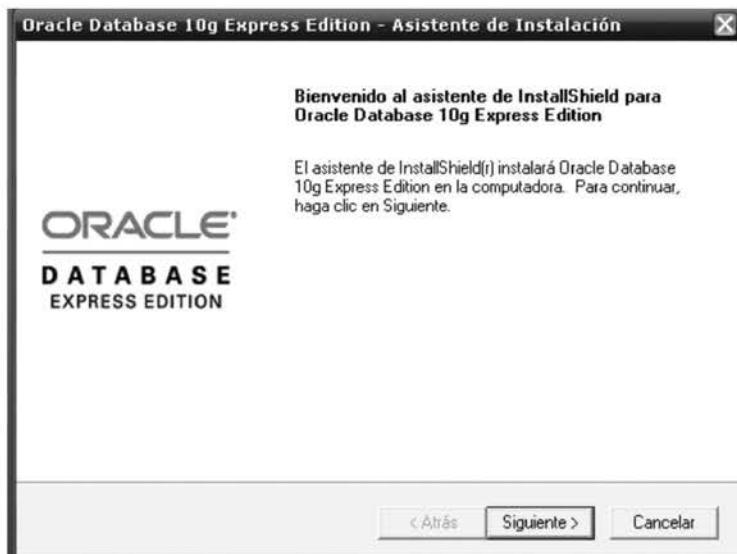


Fig. 9.3 Inicio de la instalación.

Al iniciar, el programa solicita la aprobación de los términos de las condiciones de uso de este software. Es necesario que se comprenda antes de continuar con la instalación. Por esta razón, se recomienda la lectura del contrato de licencia.



Fig. 9.4 Aceptación de la licencia de uso del software.

A partir de este punto, se inicia el proceso de copia de archivos y el programa solicitará que se le indique la ubicación de los archivos de datos que se crearán como parte del proceso de instalación para que, al terminar, se pueda instalar en una base inicial estándar. En la Fig. 9.5 y, en la siguiente, se verá cómo se modifica la dirección propuesta por el instalador (Fig. 9.6).



Fig. 9.5 Sugerencia de lugar de instalación de los archivos de la base de datos inicial.

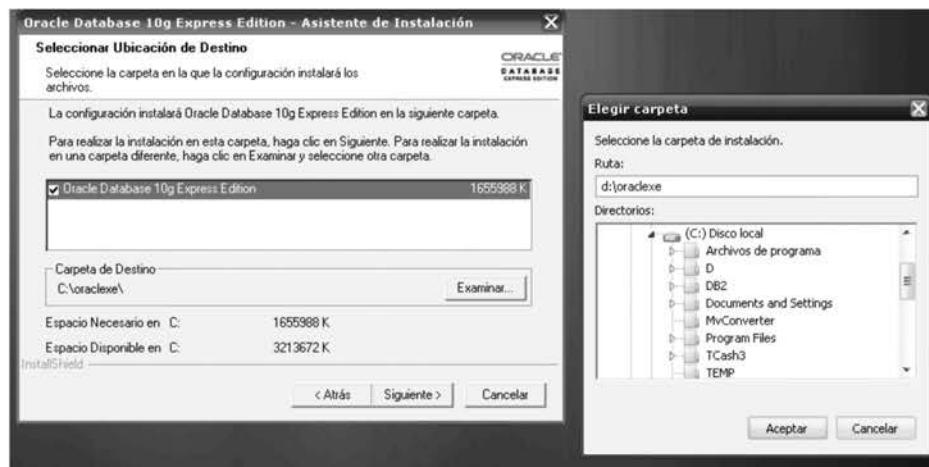


Fig. 9.6 Para cambiar la ubicación de los archivos, con el botón Examinar podremos elegir una nueva.

Luego de la definición de la ubicación de los archivos de la base de datos estándar que se instalará, el programa solicitará la contraseña de los usuarios de administración SYS y SYSTEM (Fig. 9.7), que permitirá realizar las tareas de administración de la base de datos, la creación de los usuarios para poder instalar las tablas de ejemplo y para hacer otras tareas importantes. Esta contraseña se debe registrar y recordar para, posteriormente, completar las tareas de instalación del modelo de datos del libro.

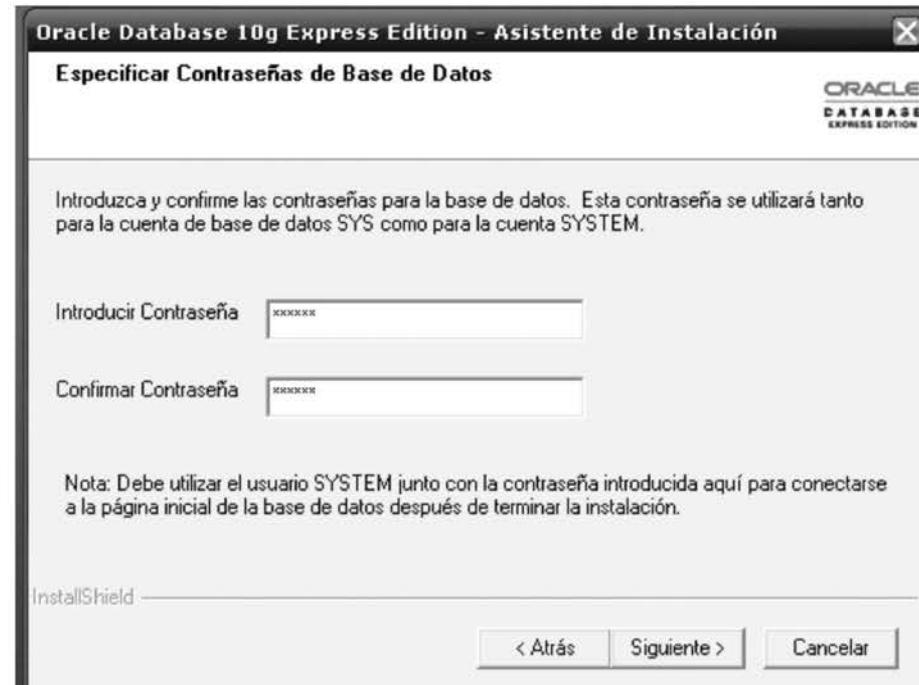


Fig. 9.7 Pantalla de definición de contraseñas.

Otra validación que solicita el programa de instalación son los parámetros de puertos del Listener, que es el proceso que atiende en un puerto http del servidor y que tiene como valor por defecto el 1521. En este punto, se podría definir otro número de puerto que se usaría en el acceso de esta base de datos (Fig. 9.8).

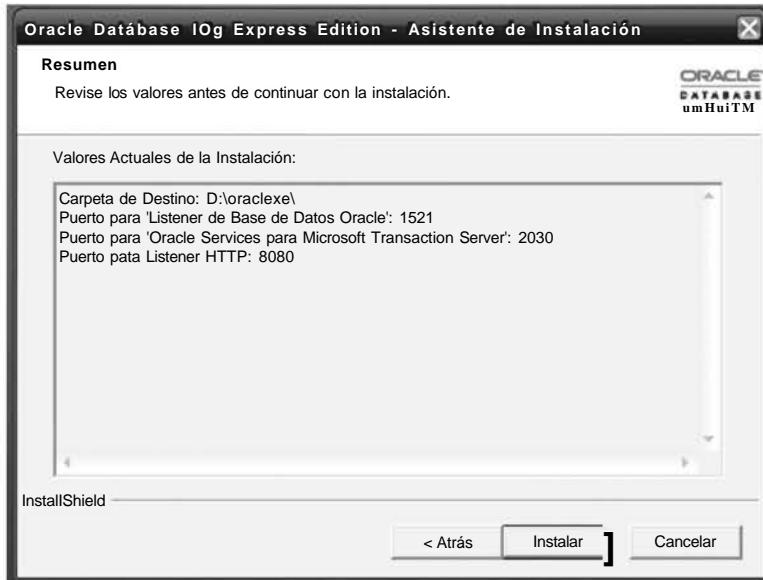


Fig. 9.8 Consulta sobre los valores de instalación para confirmación.

Una vez confirmados los valores de configuración, se iniciará el proceso que indicará el estado de avance del trabajo de instalación y que se controlará con la siguiente pantalla, que indicará el estado de avance del trabajo de instalación (Fig. 9.9).

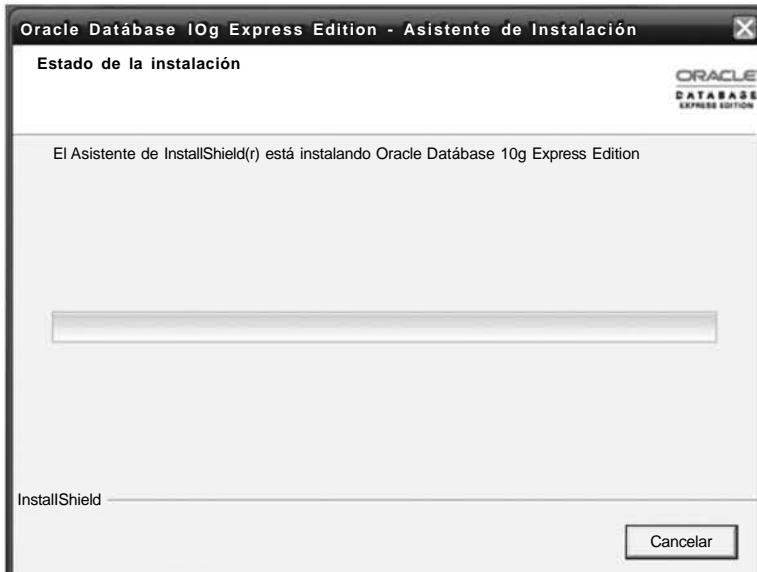


Fig. 9.9 Avance del proceso.

Al finalizar el proceso de instalación del software y de la creación de la base de datos inicial, denominada XE —por Express Edition—, aparecerá la pantalla de la Fig. 9.10, que permitirá seleccionar, una vez aceptado el final del proceso, y se conectará a través del browser predeterminado con los parámetros adecuados (observe la línea de la url en la imagen).



Fig. 9.10 Final del proceso con la indicación para iniciar la conexión.



Fig. 9.11 Fin de la instalación y paso previo para conectarse a la Base XE.

Si se ingresa el usuario SYSTEM y la contraseña elegida en los pasos anteriores, se accederá al menú de operación de la base en el que se podrán configurar los parámetros de funcionamiento de la base y, también, conectarse a ella para ejecutar los scripts de creación de tablas, constraints, procedimientos y funciones (Fig. 9.12).

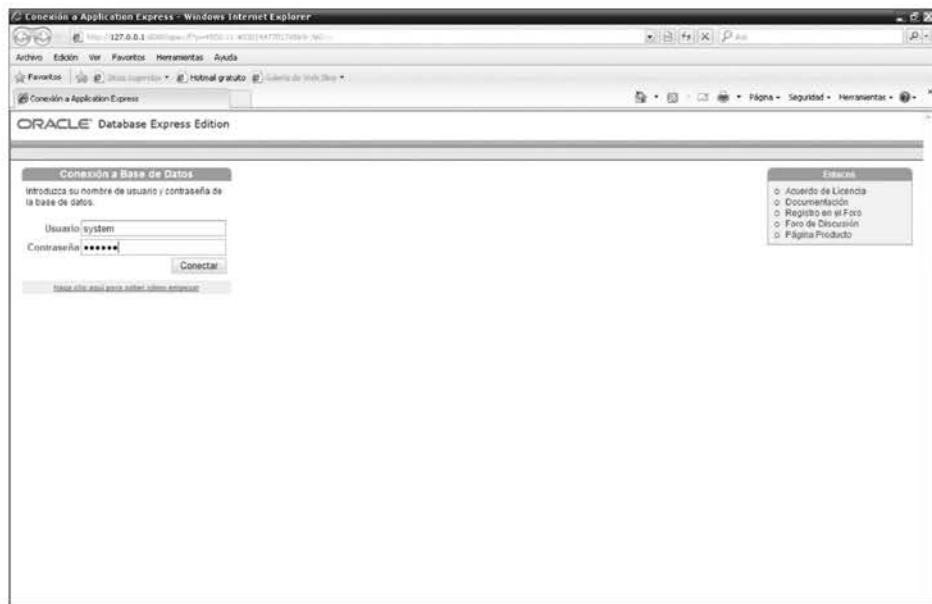


Fig. 9.12 Ingreso de la contraseña.

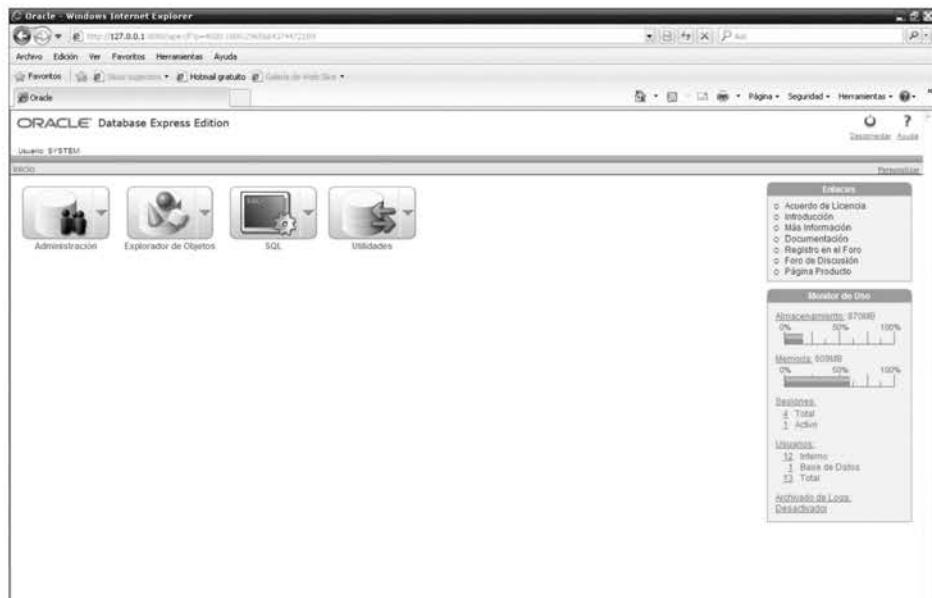


Fig. 9.13 Menú principal de la administración de la Base XE.

## 9.2 Creando y probando nuestro modelo de Estudiante – Universidad

El lector ya está en condiciones de interactuar con el cliente de la base de datos Oracle Express Edition 10g. Por esta razón, creará un usuario denominado “Libro” (Fig. 9.14) y, una vez conectado como tal, ejecutará cada sentencia de creación (Fig. 9.15).

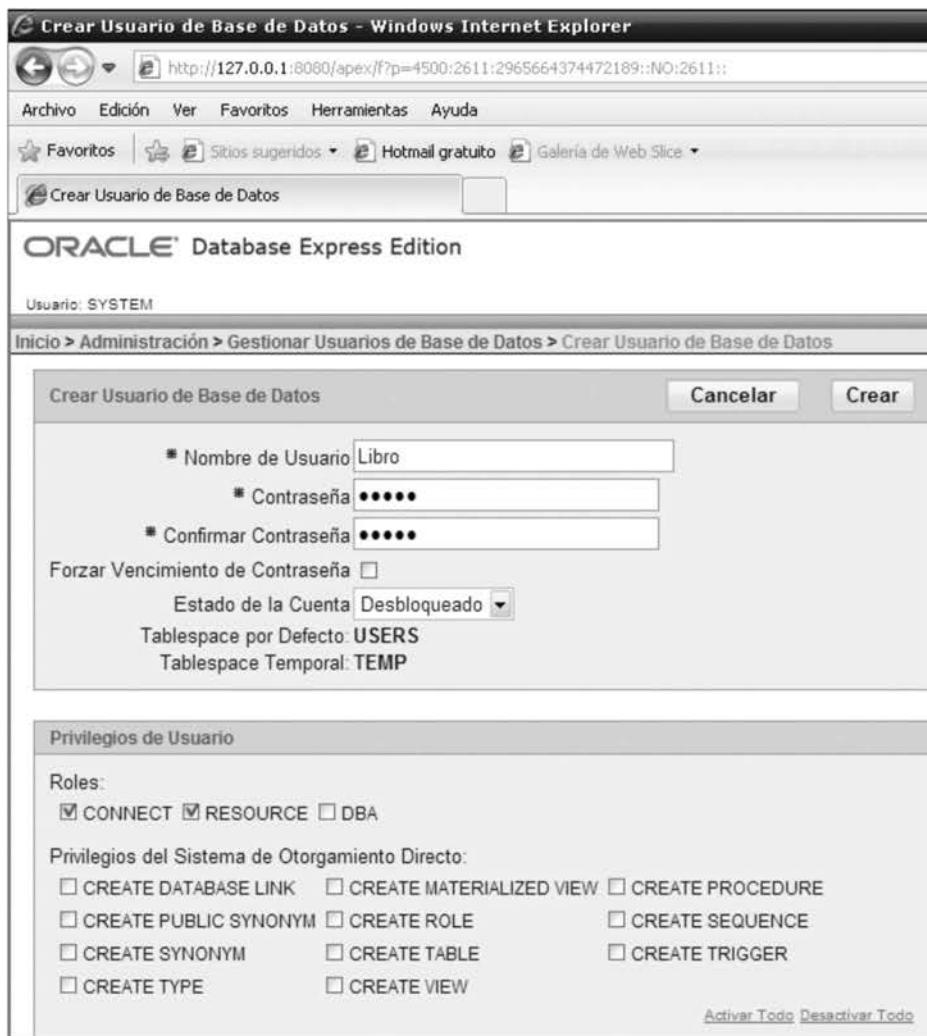


Fig. 9.14 Creación del usuario “Libro”.

Luego de la creación del usuario, realizará un *login* con el usuario “Libro” y con su contraseña, para que pueda crear la tabla en el esquema del usuario “Libro”, tal como se muestra en la Fig. 9.15. Nótese que la pantalla tiene dos partes: en la primera, se pegará la sentencia de creación de la tabla “Carreras” y, luego, con el botón “Ejecutar”

(que no se muestra en la figura porque se encuentra en el browser, en el extremo derecho de la pantalla) aparecerá el mensaje de resultado de la ejecución; en este caso, se aprecia que la tabla se creó en 0,2 segundos (Fig. 9.15).

The screenshot shows a Windows Internet Explorer window titled "Comandos SQL - Windows Internet Explorer". The URL is <http://127.0.0.1:8080/apex/F?p=4500:1003:2965664374472189::NO:1003::>. The menu bar includes Archivo, Edición, Ver, Favoritos, Herramientas, and Ayuda. The toolbar includes Favoritos, Sitios sugeridos, Hotmail gratuito, and Galería de Web Slice. The title bar says "Comandos SQL". The main content area displays the Oracle Database Express Edition logo and the text "Usuario: SYSTEM". Below that, it shows the navigation path "Inicio > SQL > Comandos SQL". A checkbox labeled "Confirmación Automática" is checked, and a dropdown menu shows "Mostrar 10". The SQL command entered is:

```
CREATE TABLE carreras(
    id_carrera int NOT NULL,
    nombre_carrera varchar2(50) NOT NULL,
    titulo varchar2(50) ,
    CONSTRAINT PK_carreras PRIMARY KEY (id_carrera)
)
```

At the bottom of the results pane, the message "Tabla creada." is displayed, followed by "0,34 segundos".

Fig. 9.15 Creación de una tabla.

## 9.3 Creación de las tablas y sus relaciones

### 9.3.1 Ejecución de script

Ahora, que ya se cuenta con la base de datos, se procederá a crear las tablas correspondientes al modelo y sus respectivas claves para asegurar la integridad de los datos. Posteriormente, y a modo de ejemplo, se crearán una función y un procedimiento almacenado de programación en la base de datos.

Como referencia para la interpretación del script, se podría agregar que las primeras sentencias son las de creación de tablas con sus claves primarias y, luego, las de modificación de las tablas creadas recientemente para la incorporación de las claves foráneas; finalmente, la inserción de datos de prueba y la creación de la función y el procedimiento almacenado. El motivo en que se fundamenta la decisión de modificar las tablas para incorporar las claves foráneas (FK) y no crearlas junto con la definición de cada tabla, es que ambas tablas, relacionadas por dicha clave, deben existir cuando ésta se crea. De este modo, se evitan las complicaciones originadas por esta causa.

### 9.3.2 Modelo de datos

La Fig. 9.16 representa el diagrama del modelo de datos utilizado. Corresponde mencionar que el asistente para la creación de dicho diagrama viene incorporado en el SSMS y que su proceso de definición es muy sencillo. Para una instrucción práctica en la utilización del asistente para el diagrama, diríjase, dentro del SMSS, al grupo “Diagramas de bases de datos” para la base de datos de la que desea crear un diagrama y, tras seleccionar la opción de nuevo diagrama de base de datos del menú contextual, se iniciará el asistente que lo guiará en la definición.

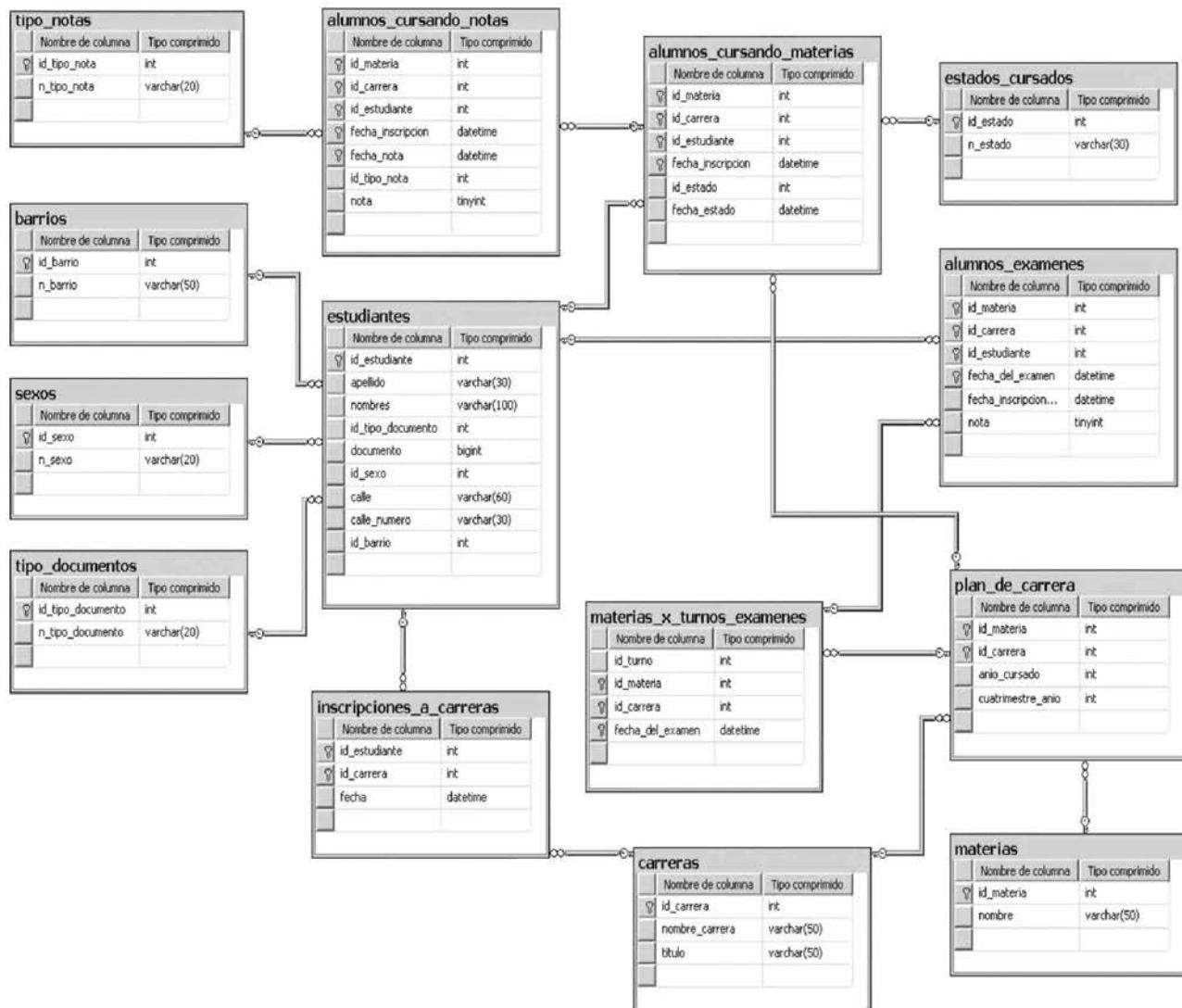


Fig. 9.16 Modelo de datos completo.

### 9.3.3 Script (sentencias DDL)

A continuación, se precisarán las sentencias de definición de datos que se ejecutarán para disponer del modelo completo de la Fig. 9.16.

```
-- Creación de Tablas con sus claves primarias
```

```
CREATE TABLE carreras(
```

```
    id_carrera int NOT NULL,
```

```
nombre_carrera varchar2(50) NOT NULL,
    título varchar2(50),
CONSTRAINT PK_carreras PRIMARY KEY (id_carrera)
);

CREATE TABLE sexos(
    id_sexo integer NOT NULL,
    n_sexo varchar2(20) NOT NULL,
CONSTRAINT PK_sexos PRIMARY KEY (id_sexo)
);

CREATE TABLE estados_cursados(
    id_estado integer NOT NULL,
    n_estado varchar2(30) NOT NULL,
CONSTRAINT PK_estados_cursados PRIMARY KEY (id_estado)
);

CREATE TABLE barrios(
    id_barrio integer NOT NULL,
    n_barrio varchar2(50) NOT NULL,
CONSTRAINT PK_barrios PRIMARY KEY (id_barrio)
);

CREATE TABLE tipo_notas(
    id_tipo_nota integer NOT NULL,
    n_tipo_nota varchar2(20) NOT NULL,
CONSTRAINT PK_tipo_notas PRIMARY KEY (id_tipo_nota)
);

CREATE TABLE tipo_documentos(
    id_tipo_documento integer NOT NULL,
    n_tipo_documento varchar2(20) NOT NULL,
CONSTRAINT PK_tipo_documentos PRIMARY KEY (id_tipo_documento)
);

CREATE TABLE materias(
    id_materia integer NOT NULL,
    nombre VARCHAR2(50) NOT NULL,
CONSTRAINT PK_materias PRIMARY KEY (id_materia)
);

CREATE TABLE alumnos_cursando_materias(
    id_materia integer NOT NULL,
```

```
    id_carrera integer NOT NULL,
    id_estudiante integer NOT NULL,
    fecha_incripcion date NOT NULL,
    id_estado integer NOT NULL,
    fecha_estado date NOT NULL,
CONSTRAINT PK_alumnos_cursando_materias PRIMARY KEY (id_materia, id_carrera, id_estudiante, fecha_incripcion)
);
CREATE TABLE alumnos_examenes(
    id_materia integer NOT NULL,
    id_carrera integer NOT NULL,
    id_estudiante integer NOT NULL,
    fecha_del_examen date NOT NULL,
    fecha_incripcion_a_examenes date NOT NULL,
    nota integer,
CONSTRAINT PK_alumnos_examenes PRIMARY KEY (id_materia, id_carrera, id_estudiante, fecha_del_examen)
);
CREATE TABLE alumnos_cursando_notas(
    id_materia integer NOT NULL,
    id_carrera integer NOT NULL,
    id_estudiante integer NOT NULL,
    fecha_incripcion date NOT NULL,
    fecha_nota date NOT NULL,
    id_tipo_nota integer NOT NULL,
    nota integer NULL,
CONSTRAINT PK_alumnos_cursando_notas PRIMARY KEY (id_materia, id_carrera, id_estudiante, fecha_incripcion,
fecha_nota)
);
CREATE TABLE inscripciones_a_carreras(
    id_estudiante integer NOT NULL,
    id_carrera integer NOT NULL,
    fecha date NOT NULL,
CONSTRAINT PK_inscripciones_a_carreras PRIMARY KEY (id_estudiante,id_carrera)
);
CREATE TABLE plan_de_carrera(
    id_materia integer NOT NULL,
    id_carrera integer NOT NULL,
    anio_cursado integer,
    cuatrimestre_anio integer,
CONSTRAINT PK_plan_de_carrera PRIMARY KEY (id_materia,id_carrera));
```

```
CREATE TABLE estudiantes(
    id_estudiante integer NOT NULL,
    apellido varchar2(30) NOT NULL,
    nombres varchar2(100) NOT NULL,
    id_tipo_documento integer NOT NULL,
    documento integer NOT NULL,
    id_sexo integer,
    calle varchar2(60) NOT NULL,
    calle_numero varchar2(30) NOT NULL,
    id_barrio integer,
    CONSTRAINT PK_estudiantes PRIMARY KEY (id_estudiante ));
CREATE TABLE materias_x_turnos_examenes(
    id_turno integer NOT NULL,
    id_materia integer NOT NULL,
    id_carrera integer NOT NULL,
    fecha_del_examen date NOT NULL,
    CONSTRAINT PK_materias_x_turnos_examenes PRIMARY KEY (id_materia, id_carrera, fecha_del_examen)
);
-- Creación de claves foráneas
ALTER TABLE alumnos_cursando_materias ADD CONSTRAINT FK_alu_cur_mat_estados FOREIGN KEY(id_estado)
REFERENCES estados_cursados (id_estado);
ALTER TABLE alumnos_cursando_materias ADD CONSTRAINT FK_alu_cur_mat_estudiantes FOREIGN KEY(id_estudiante)
REFERENCES estudiantes (id_estudiante);
ALTER TABLE alumnos_cursando_materias ADD CONSTRAINT FK_alu_cur_mat_plan_carrera FOREIGN KEY(id_materia,
id_carrera)
REFERENCES plan_de_carrera (id_materia, id_carrera);
ALTER TABLE alumnos_examenes ADD CONSTRAINT FK_alu_exam_estudiantes FOREIGN KEY(id_estudiante)
REFERENCES estudiantes (id_estudiante);
ALTER TABLE alumnos_examenes ADD CONSTRAINT FK_alu_exam_mat_x_tur_exam FOREIGN KEY(id_materia, id_carrera,
fecha_del_examen)
REFERENCES materias_x_turnos_examenes (id_materia, id_carrera, fecha_del_examen);
ALTER TABLE alumnos_cursando_notas ADD CONSTRAINT FK_alu_curs_notas_t_notas FOREIGN KEY(id_tipo_nota)
REFERENCES tipo_notas (id_tipo_nota);
ALTER TABLE alumnos_cursando_notas ADD CONSTRAINT FK_alu_curs_notas_alu_cur_mat FOREIGN KEY(id_materia,
id_carrera, id_estudiante, fecha_inscripcion)
REFERENCES alumnos_cursando_materias (id_materia, id_carrera, id_estudiante, fecha_inscripcion);
ALTER TABLE inscripciones_a_carreras ADD CONSTRAINT FK_insc_a_carr_carr FOREIGN KEY(id_carrera)
REFERENCES carreras (id_carrera);
```

```

ALTERTABLE inscripciones_a_carreras ADD CONSTRAINT FK_insc_a_carr_est FOREIGN KEY(id_estudiante)
REFERENCES estudiantes (id_estudiante);
ALTERTABLE plan_de_carrera ADD CONSTRAINT FK_p_de_carrera_carr FOREIGN KEY(id_carrera)
REFERENCES carreras (id_carrera);
ALTERTABLE plan_de_carrera ADD CONSTRAINT FK_p_de_carrera_mate FOREIGN KEY(id_materia)
REFERENCES materias (id_materia);
ALTERTABLE estudiantes ADD CONSTRAINT FK_estudiantes_barrios FOREIGN KEY(id_barrio)
REFERENCES barrios (id_barrio);
ALTERTABLE estudiantes ADD CONSTRAINT FK_estudiantes_sexos FOREIGN KEY(id_sexo)
REFERENCES sexos (id_sexo);
ALTERTABLE estudiantes ADD CONSTRAINT FK_estudiantes_tipo_doc FOREIGN KEY(id_tipo_documento)
REFERENCES tipo_documentos (id_tipo_documento);

```

### 9.3.4 Datos de prueba

Se insertarán un conjunto de datos de manera de poder probar la función y el procedimiento almacenado que se programarán en los ítems subsiguientes. De la misma manera que en la creación de tablas, las sentencias de inserción de datos (INSERT) se escribirán en la ventana de documento del SSMS; previo a la ejecución, se asegurará de que se tiene seleccionada la base de datos correcta.

```

INSERT INTO barrios (id_barrio,n_barrio) VALUES (1,'Alto Alberdi');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (2,'Yofre');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (3,'Nueva Córdoba');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (4,'Centro');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (5,'Los Boulevares');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (6,'Guemes');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (7,'Ipona');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (8,'Alta Córdoba');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (9,'General Paz');

INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (1,'Parcial 1');
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (2,'Parcial 2');
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (3,'Final');
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (4,'Trabajo Practico');

INSERT INTO tipo_documentos (id_tipo_documento,n_tipo_documento) VALUES (1,'DNI');
INSERT INTO tipo_documentos (id_tipo_documento,n_tipo_documento) VALUES (2,'LC');

```

```
INSERT INTO carreras (id_carrera,nombre_carrera,titulo) VALUES (1,'Ingeniería en Sistemas de Información','Ingeniero en Sistemas de Informacion');

INSERT INTO sexos (id_sexo,n_sexo) VALUES (1,'Masculino');
INSERT INTO sexos (id_sexo,n_sexo) VALUES (2,'Femenino');

INSERT INTO estados_cursados (id_estado,n_estado) VALUES (1,'Inscripto');
INSERT INTO estados_cursados (id_estado,n_estado) VALUES (2,'Regular');
INSERT INTO estados_cursados (id_estado,n_estado) VALUES (3,'Libre');

INSERT INTO materias (id_materia,nombre) VALUES (1,'Algoritmos y Estructuras de Datos');
INSERT INTO materias (id_materia,nombre) VALUES (2,'Análisis Matemático I');
INSERT INTO materias (id_materia,nombre) VALUES (3,'Análisis Matemático II');
INSERT INTO materias (id_materia,nombre) VALUES (4,'Álgebra y Geometría Analítica');
INSERT INTO materias (id_materia,nombre) VALUES (5,'Matemática Discreta');
INSERT INTO materias (id_materia,nombre) VALUES (6,'Sistemas y Organizaciones');
INSERT INTO materias (id_materia,nombre) VALUES (7,'Ingeniería y Sociedad');
INSERT INTO materias (id_materia,nombre) VALUES (8,'Paradigmas de Programación');

INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (1,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (2,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (3,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (4,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (5,1,1,2);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (6,1,1,2);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (7,1,1,2);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (8,1,1,2);

INSERT INTO estudiantes (id_estudiante,apellido,nombres,id_tipo_documento,documento,id_sexo,calle,calle_numero,id_barrio) VALUES (1,'Abrutsky','Maximiliano Adrián',1,23852963,1,'Av. Colón','1035',1);
INSERT INTO estudiantes (id_estudiante,apellido,nombres,id_tipo_documento,documento,id_sexo,calle,calle_numero,id_barrio) VALUES (2,'DAMIANO','Luis',1,19147258,1,'Dean Funes','903',1);

INSERT INTO inscripciones_a_carreras (id_estudiante,id_carrera,fecha) VALUES (1,1,'26/01/2009');

INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (1,1,1,'01/02/2009',1,'01/02/2009');
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (2,1,1,'01/02/2009',1,'01/02/2009');
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (3,1,1,'01/02/2009',1,'01/02/2009');
```

```

INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_incripcion,id_estado,fecha_estado) VALUES (4,1,1,'01/02/2009',1,'01/02/2009') ;

INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_incripcion,fecha_nota,id_tipo_nota/nota) VALUES (1,1,1,'01/02/2009','15/03/2009',1,5);
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_incripcion,fecha_nota,id_tipo_nota/nota) VALUES (1,1,1, '01/02/2009', '26/05/2009',2,7);
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_incripcion,fecha_nota,id_tipo_nota/nota) VALUES (2,1,1,'01/02/2009','18/03/2009',1,3);
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_incripcion,fecha_nota,id_tipo_nota/nota) VALUES (2,1,1,'01/02/2009','29/05/2009',2,4);

```

### 9.3.5 Función escalar definida por el usuario

El objetivo de esta función ejemplo es recibir por parámetro el id\_estudiante y devolver el nombre completo del estudiante con el formato 'Apellido, nombres'. Es importante recalcar que solamente la primera letra del apellido será mayúscula y todas las de/los nombre/s, en minúscula.

La sentencia de creación está comentada para facilitar la comprensión. Asimismo, el funcionamiento consiste en declarar una variable, asignarle como valor la primera letra del apellido en mayúsculas, concatenándole el resto de las letras en minúsculas, una coma con espacio en blanco y, finalmente, el o los nombres, todo en minúsculas. Se concluye cuando se retorna dicha variable como resultado.

```

CREATE or replace FUNCTION obtener_Nombre_Completo(p_id_estudiante INTEGER) --cabecera de la función (nombre y parámetros)
RETURNS VARCHAR2 -- tipo de valor que retorna
AS
nombreCompleto VARCHAR2(150); -- definición de variable que será
retornada
BEGIN
-- Las funciones UPPERCASE y LOWERCASE convierten a mayús. o minús. respectivamente.

    SELECT initcap(apellido)||', '|initcap(nombres)
    INTO nombrecompleto
    FROM estudiantes
    WHERE id_estudiante = p_id_estudiante;

    RETURN nombreCompleto; -- retorno del valor
END;

```

Esta función, como en cualquier otra que también sea escalar, retornará solo un valor, por lo que puede invocarse cuando se utilizan expresiones escalares (en el SELECT, INSERT, UPDATE, etcétera).

#### Ejemplos de invocación

```
SELECT id_estudiante, apellido, nombres, obtener_Nombre_Completo(id_estudiante)
```

```
as NombreCompleto
```

```
FROM estudiantes
```

Resultado:

| id_estudiante | apellido | nombres            | NombreCompleto        |
|---------------|----------|--------------------|-----------------------|
| 1<br>adrián   | Abrutsky | Maximiliano Adrián | Abrutsky, maximiliano |
| 2             | DAMIANO  | Luis               | Damiano, luis         |

#### 9.3.6 Procedimiento almacenado definido por el usuario

La funcionalidad de este procedimiento consistirá en recibir por parámetro un estudiante, la materia y la carrera (id\_estudiante, id\_materia e id\_carrera) y actualizar el estado de cursado (tabla alumnos\_cursando\_materias) de dicha materia - carrera; por ejemplo: de inscripto a regular o libre de acuerdo con si tiene, o no, al menos dos parciales aprobados (tabla alumnos\_cursando\_notas).

Consideraciones:

- Un alumno puede cursar nuevamente una materia, por lo que se toma la última inscripción.
- La nota de aprobación es mayor o igual a cuatro.
- Al menos debe tener dos parciales rendidos para que se actualice el estado.
- Si se tiene al menos dos parciales aprobados, se lo considera regular.
- Menos de dos parciales aprobados, se lo considera libre.
- Por sencillez, en este modelo, no se consideran los exámenes de recuperación de aplazos.

```
CREATE OR REPLACE PROCEDURE actualizar_estado_materia (p_id_estudiante
INTEGER, p_id_materia INTEGER, p_id_carrera INTEGER)
```

```
AS
```

```
fecha_inscripcion_cursado DATE;
cant_parciales INTEGER;
cant_parciales_aprobados INTEGER;
cant_parciales_reprobados INTEGER;
```

```

BEGIN
SELECT MAX(fecha_inscripcion)
INTO fecha_inscripcion_cursado
FROM alumnos_cursando_materias
WHERE id_materia = p_id_materia
AND id_estudiante = p_id_estudiante;

SELECT COUNT(*)
INTO cant_parciales
FROM alumnos_cursando_notas notas JOIN materias mat ON notas.id_materia = mat.id_materia
JOIN tipo_notas tip ON notas.id_tipo_nota = tip.id_tipo_nota
WHERE notas.id_estudiante = p_id_estudiante
AND notas.id_materia = p_id_materia
AND notas.id_carrera = p_id_carrera
AND notas.fecha_inscripcion = fecha_inscripcion_cursado
AND LOWER(tip.n_tipo_nota) LIKE '%parcial%';

SELECT COUNT(*)
INTO cant_parciales_aprobados
FROM alumnos_cursando_notas notas JOIN materias mat ON notas.id_materia = mat.id_materia
JOIN tipo_notas tip ON notas.id_tipo_nota = tip.id_tipo_nota
WHERE notas.id_estudiante = p_id_estudiante
AND notas.id_materia = p_id_materia
AND notas.id_carrera = p_id_carrera
AND notas.fecha_inscripcion = fecha_inscripcion_cursado
AND LOWER(tip.n_tipo_nota) LIKE '%parcial%'
AND notas.nota >= 4;

cant_parciales_reprobados := cant_parciales - cant_parciales_aprobados;

DBMS_OUTPUT.PUT_LINE('Cantidad de parciales rendidos = '||cant_parciales);
DBMS_OUTPUT.PUT_LINE('Cantidad de parciales aprobados = '||cant_parciales_aprobados);
DBMS_OUTPUT.PUT_LINE('Cantidad de parciales reprobados = '||cant_parciales_reprobados);

IF cant_parciales < 2 THEN
    DBMS_OUTPUT.PUT_LINE('La cantidad rendida de parciales es menor a 2, no se cambia el estado');
ELSIF cant_parciales_aprobados >= 2 THEN
    BEGIN

```

```
DBMS_OUTPUT.PUT_LINE('2 parciales aprobados, REGULAR');
UPDATE alumnos_cursando_materias
SET id_estado = (SELECT id_estado FROM estados_cursados WHERE n_estado LIKE 'Regular')
WHERE id_estudiante = p_id_estudiante
AND id_materia = p_id_materia
AND id_carrera = p_id_carrera
AND fecha_incripcion = fecha_incripcion_cursado;
END;

ELSE
BEGIN
    DBMS_OUTPUT.PUT_LINE( 'Menos de 2 parciales aprobados, LIBRE');
    UPDATE alumnos_cursando_materias
    SET id_estado = (SELECT id_estado FROM estados_cursados WHERE n_estado LIKE 'Libre')
    WHERE id_estudiante = p_id_estudiante
    AND id_materia = p_id_materia
    AND id_carrera = p_id_carrera
    AND fecha_incripcion = fecha_incripcion_cursado;
END;

END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('El alumno no se ha inscripto a la materia – carrera especificada');
END;
```

### 9.3.7 Ejemplos de ejecución

Invocación:

Si se realiza desde Oracle SQL\*PLUS, debe definirse con el comando set para que muestre los mensajes con la siguiente línea de código

```
SQL> SET SERVEROUTPUT ON.
```

Y, luego,

```
SQL> EXECUTE actualizar_estado_materia(1,8,1)
```

Resultado:

"La cantidad rendida de parciales es menor a dos, por lo que si no se actualizan los valores, no se cambia el estado." Procedimiento PL/SQL terminado correctamente".

Invocación:

```
SQL> execute actualizar_estado_materia(1,2,1);
```

Resultado:

Cantidad de parciales rendidos = 2  
Cantidad de parciales aprobados = 1  
Cantidad de parciales reprobados = 1  
Menos de 2 parciales aprobados = LIBRE

Procedimiento PL/SQL terminado correctamente.

Invocación:

```
EXEC actualizar_estado_materia 1,1,1
```

Resultado:

Cantidad de parciales rendidos = 2  
Cantidad de parciales aprobados = 2  
Cantidad de parciales reprobados = 0  
2 parciales aprobados = REGULAR

Procedimiento PL/SQL terminado correctamente.

#### 9.4 Contenido de la página Web de apoyo.....



El material marcado con asterisco (\*) solo está disponible para docentes.

##### 9.4.1 Mapa conceptual del capítulo

##### 9.4.2 Autoevaluación

##### 9.4.3 Presentaciones\*

# 10

## Implementando el modelo en IBM DB2

### Contenido

|  |     |
|--|-----|
| 10.1 Introducción.....   | 298 |
| 10.2 Instalando e inicializando el servidor.....                           | 298 |
| 10.3 Creando y probando nuestro modelo de<br>Estudiante - Universidad..... | 299 |
| 10.4 Contenido de la página Web de apoyo.....                              | 310 |

### Objetivos

- Saber dónde conseguir el instalador de la base DB2 edición “Express - C” de IBM.
- Entender cómo es el proceso de instalación.
- Tener referencia sobre más bibliografía de esta edición de la base DB2.
- Usar el centro de control para crear las tablas del modelo visto en el libro.



DB2 Express-C forma parte de servidores de datos IBM DB2 que, además de almacenar datos en tablas relacionales, puede manejar información almacenada jerárquicamente como XML. DB2 Express-C tiene una forma de licenciamiento libre, sin límites y fácil de usar.

## 10.1 Introducción

DB2 Express-C es un integrante de la familia de servidores de datos IBM DB2 que, además de almacenar datos en tablas relacionales, también puede manejar información almacenada jerárquicamente como XML. DB2 Express-C tiene una forma de licenciamiento libre, sin límites y fácil de usar. La 'C' es porque se dirige a la comunidad que se ha creado con los usuarios de esta base de datos.

La comunidad DB2 Express-C consiste en una variedad de personas y compañías que diseñan, desarrollan, implementan o utilizan soluciones de base de datos, como desarrolladores de aplicaciones, vendedores de hardware e infraestructura y proveedores de otros tipos de solución que quieran incluir o empotrar un completo servidor de datos como parte de sus soluciones.

DB2 Express-C comparte el mismo conjunto base de funcionalidades y código, como las ediciones pagas de DB2 para Linux, UNIX y Windows. DB2 Express-C puede correr en sistemas de 32-bits y 64-bits, con sistemas operativos Windows o Linux y, también, en un sistema que tenga cualquier cantidad de núcleos y de memoria. No tiene ningún requerimiento especial de almacenamiento o de configuración del sistema que sea especial. DB2 Express-C también incluye pureXML sin costo, que es la tecnología única de DB2 para almacenar y procesar documentos XML nativamente.



En la Web de apoyo, encontrará el vínculo a la página Web que le brindará mayor información acerca del producto IBM DB2.

## 10.2 Instalando e inicializando el servidor

El primer paso que se realizará para instalar el software del motor DB2 en nuestro servidor será descargarlo para luego correr el archivo ejecutable de instalación de manera que, posteriormente, podamos conectarnos desde las herramientas incluidas en la instalación como así también desde otros programas "clientes" que pueden ejecutarse en el mismo equipo o bien desde otro nodo de la red.

### 10.2.1 Download

En la dirección <http://www-01.ibm.com/software/data/db2/express/download.html>, se puede descargar la versión adecuada para nuestro Sistema Operativo, entre los que figuran actualmente Windows, Linux, Solaris e incluso, MAC OS X de Apple.

### 10.2.2 Instalación

La arquitectura del procesador está disponible para 32-bit, 64-bit y PowerPC (Linux). Si se necesita correr el DB2 sobre otra plataforma, como UNIX, se debe comprar una de las ediciones de servidor de datos diferente a la Express-C.

Los requerimientos del sistema operativo, para todas las ediciones de DB2, se describen en este documento:

<http://www.ibm.com/software/data/db2/udb/sysreqs.html>

El DB2 Express-C se puede instalar sobre sistemas que contengan uno o varios núcleos de CPU y de memoria, pero solo utilizará hasta dos núcleos y 2GB de memoria para la licencia gratuita y sin garantía y, para la licencia de suscripción de doce meses y versión con soporte, hasta cuatro núcleos y 4GB de memoria. El DB2 Express-C se puede utilizar en sistemas reales y en máquinas virtuales. Obviamente, también corre sobre sistemas más pequeños como, por ejemplo, un sistema de un solo procesador con 1GB de memoria.

La última información sobre el requerimiento de hardware para DB2 Express-C se encuentra en su página Web:

<http://www-306.ibm.com/software/data/db2/express/getstarted.html>

Luego de bajar y descomprimir la imagen DB2 Express-C, el asistente guiará la instalación. En Windows, se debe ejecutar el archivo setup.exe en el directorio EXP/image; en Linux, alcanzará con ejecutar el programa db2setup del directorio exp/disk1 DB2.

Al arrancar el programa de instalación, se verá la pantalla de inicio del proceso o Setup Launchpad del DB2. En las opciones que éste muestra se deberá hacer clic en “Install a product” y, luego, se elegirá la opción “Install New” para instalar otra copia de DB2 Express-C en el sistema. En los pasos siguientes, se verá cómo crear una base de datos para este ejemplo, las tablas asociadas y las filas del sistema académico.

Para una referencia completa, se recomienda descargar el libro provisto para la comunidad Conociendo el DB2 Express-C de Raúl Chong, Ian Hakes y Rav Ahuja, que es un recurso gratuito disponible en múltiples idiomas en:

<http://www.ibm.com/developerworks/wikis/display/DB2/FREE+Book-Getting+Started+with+DB2+Express-C>

## 10.3 Creando y probando nuestro modelo de Estudiante – Universidad

### 10.3.1 Creación de la base de datos

Una vez instalado el software, se accederá a los programas que permitirán operar la base de datos desde el botón inicio y las opciones que se muestran en la Fig. 10.1. Específicamente, se ejecutará el centro de control. Allí se consultará qué vista del centro de control se usará, si la básica o la avanzada, según se ve en la Fig. 10.2.

Para trabajar se elegirá la opción de vista avanzada del centro de control y, a continuación, se creará la base de datos para, posteriormente, avanzar sobre la creación de tablas y la inserción de filas.

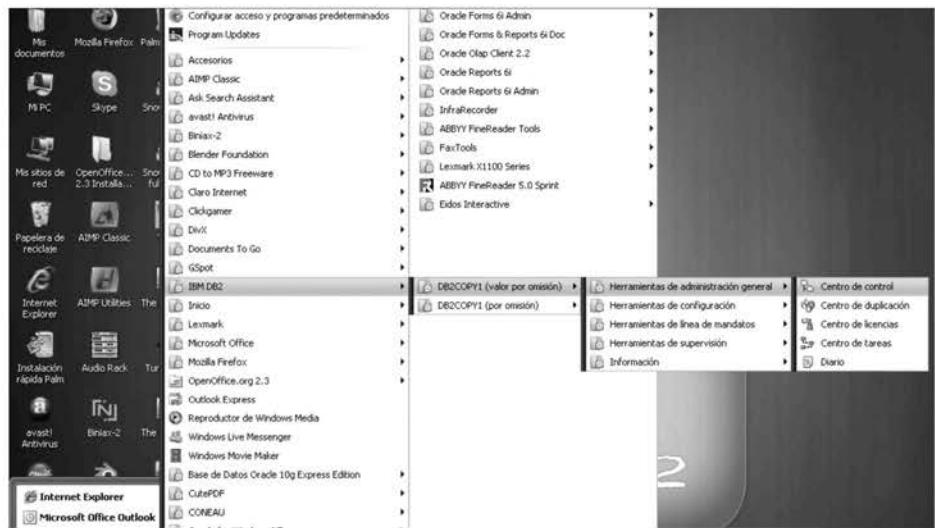


Fig. 10.1 Creación de una base de datos.

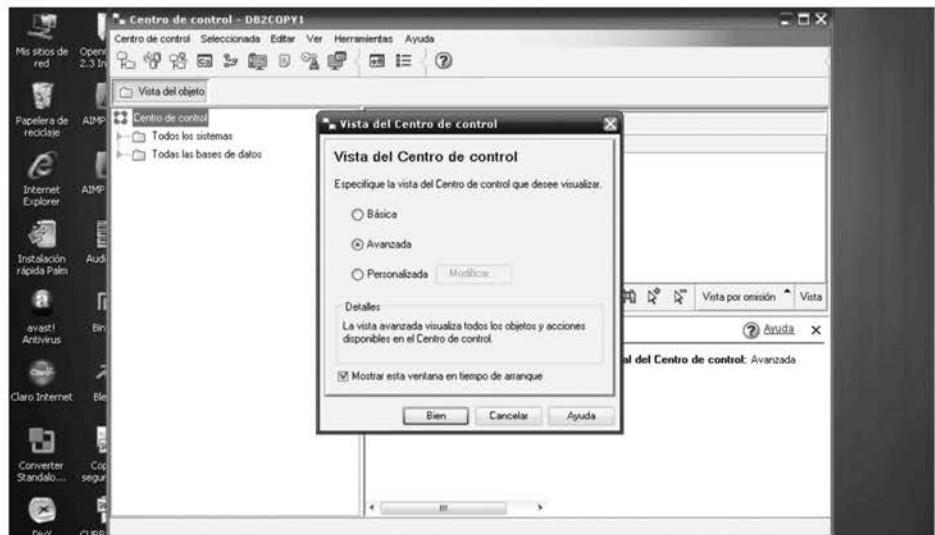


Fig. 10.2 Inicio en el uso del centro de control en el que se especifica la vista que se utilizará.

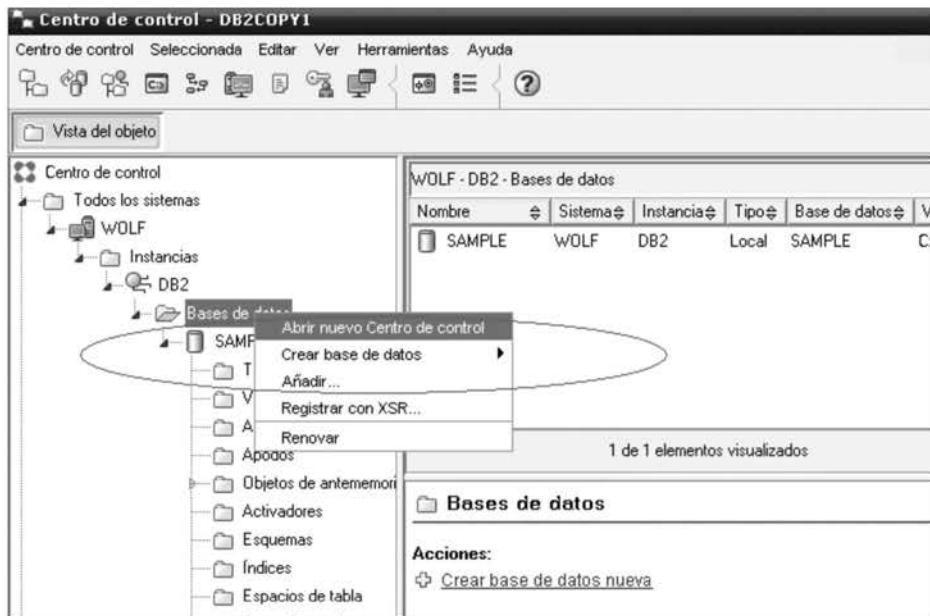


Fig. 10.3 Creación de una base de datos.

En este paso, es necesario que se elija el nombre de la base y en qué ubicación del sistema de archivos se guardarán todos los archivos propios de la base de datos, según lo visto en la Fig. 10.4.

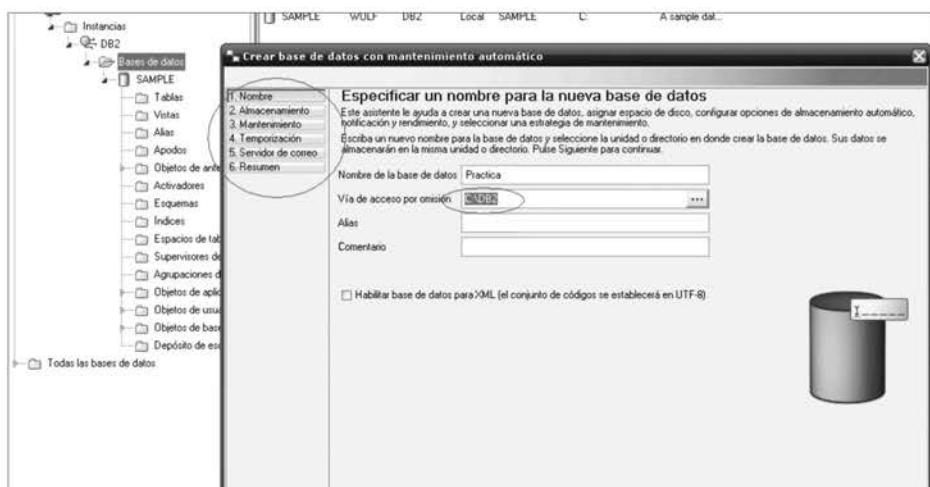


Fig. 10.4 Creación de una base de datos en la que se define el nombre y la ubicación de los archivos en el sistema.

En la figura anterior, además del nombre de la base de datos y la ubicación, se ve, a la izquierda, el submenú de los pasos que se cubrirán con esta herramienta como, por ejemplo, “Mantenimiento”, en el que se podrá especificar la ventana de tiempo, donde el motor realizará tareas como la reorganización de índices y el respaldo de la base de datos. Estas tareas son importantes para el buen funcionamiento general de las aplicaciones. El nombre elegido para la base de datos es “PRACTICA”, como se observa en la Fig. 10.5.

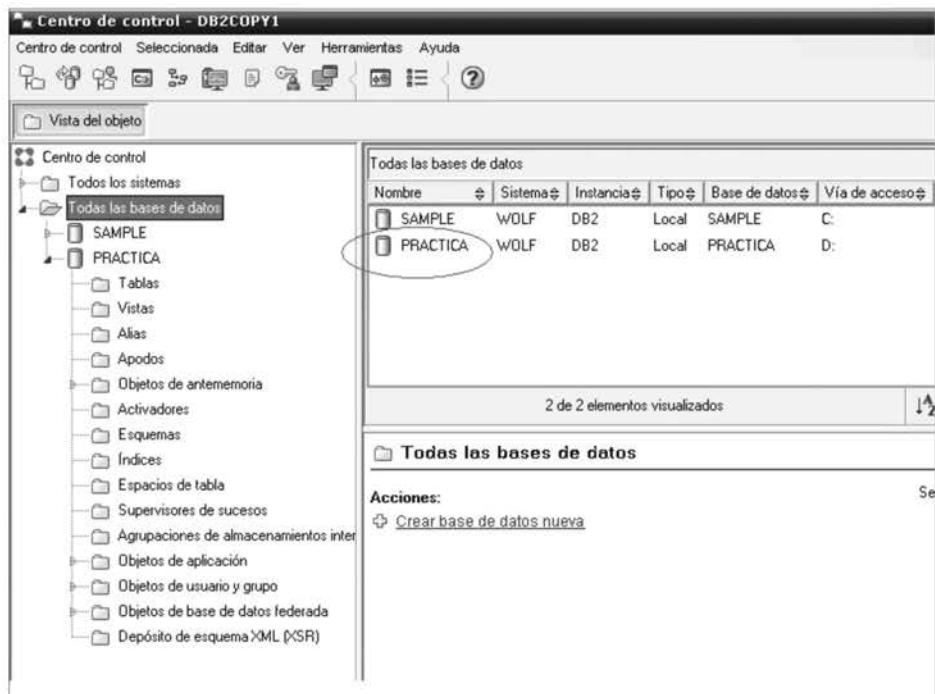


Fig. 10.5 Base de datos de ejemplo “PRACTICA” creada.

Desde la vista avanzada, si se abren los nodos de las bases de datos a la izquierda, se verán las carpetas en las que se guardarán las definiciones de objetos contenidos en la base de datos.

En el menú del centro de control, la opción Herramientas permitirá acceder al programa Editor de mandatos, en el que se ejecutarán las sentencias SQL y se observará el resultado, ya sean los mensajes de ejecución, la descripción de un error que ha ocurrido en la ejecución y las filas resultantes de una consulta.

Para ejecutar los comandos, es necesario conectarse a una base de datos. Esto se realiza desde la opción “Destino” del Editor de mandatos, tal como se muestra en la Fig. 10.6.

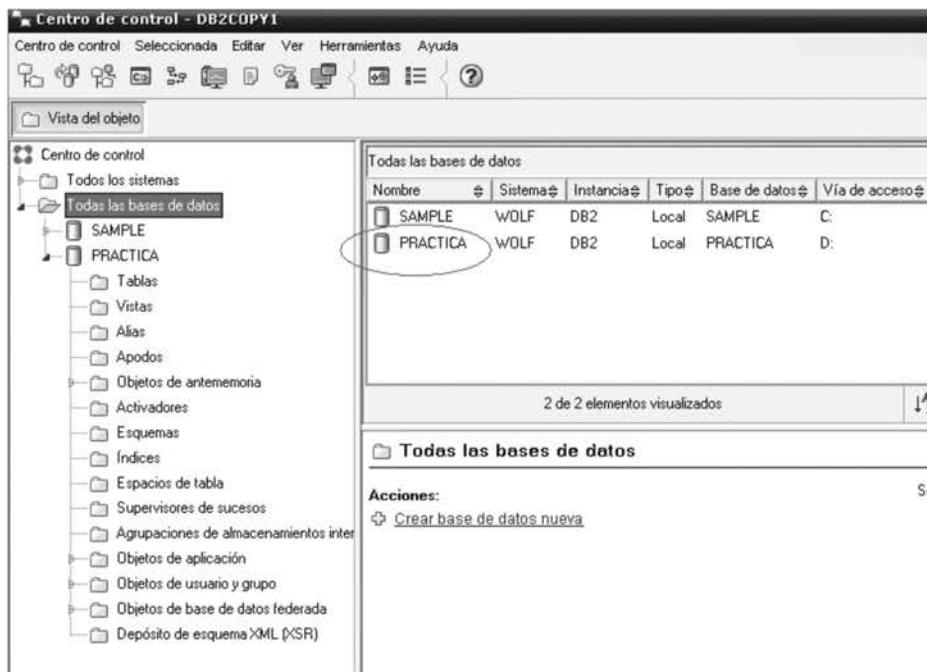


Fig. 10.6 Para ejecutar los comandos en el Editor de comandos, debe conectarse a una base de datos previamente.

### 10.3.2 Creación de las tablas y sus relaciones

#### 10.3.2.1 Ejecución de scripts

Ahora que ya se cuenta con una base de datos conectada, se procederá a crear las tablas correspondientes al modelo y sus respectivas claves para asegurar la integridad de los datos. Posteriormente, se establecerán, a modo de ejemplo, una función y un procedimiento almacenado de programación en la base de datos.

El modelo de ejemplo se ha creado con un script similar al de los capítulos anteriores que se corresponden con otros motores de bases de datos. A continuación, se analizarán las diferencias que surgen al instalar en DB2. Como referencia, se deben tomar las primeras sentencias para la creación de las tablas con sus claves primarias y, luego, las de modificación de las tablas recientemente creadas para incorporar las claves foráneas. Finalmente, la inserción de datos de prueba y de creación de la función y del procedimiento almacenado. El motivo en el cual se fundamenta la decisión de modificar las tablas para incorporar las claves foráneas (FK) y no crearlas junto con la definición de cada tabla, es que ambas tablas relacionadas por dicha clave deben existir cuando ésta se crea. De esta manera, se evitarán las complicaciones originadas por esta causa.

Para poder crear los objetos con las sentencias usadas en las otras bases, se tuvo que adaptar el nombre del tipo de datos de VARCHAR2 a VARCHAR, y el constraint

'NULL' ha sido eliminado. Otro detalle que hubo que ajustar es la longitud del nombre de los constraints, para que la creación de los modelos de datos se pudiera ejecutar sin error.

### 10.3.3 Modelo de datos

La Fig. 10.7 representa el diagrama del modelo de datos utilizado.

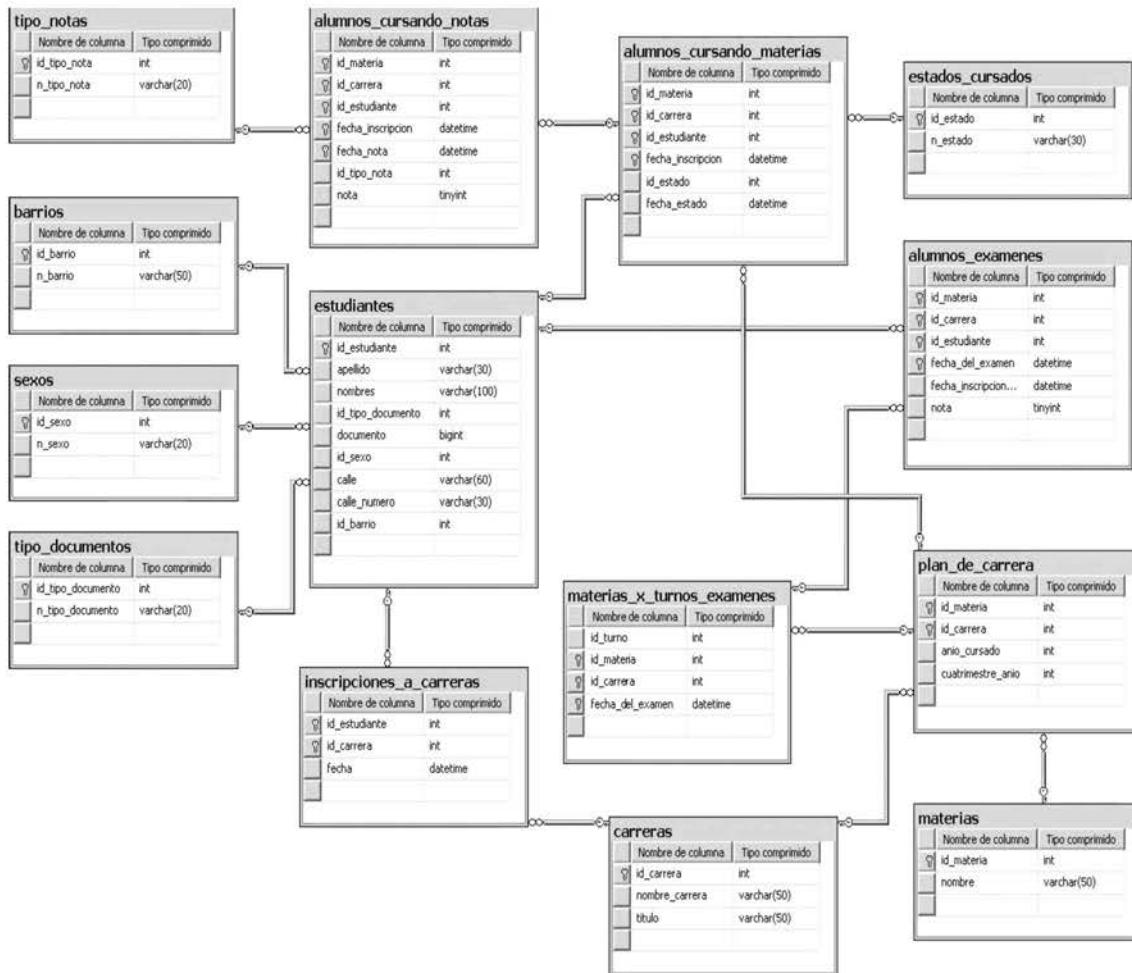


Fig. 10.7 Modelo de datos completo.

#### 10.3.3.1 Script (sentencias DDL)

A continuación, se precisarán las sentencias de definición de datos por ejecutarse, para disponer del modelo completo de la Fig. 10.7.

```
-- Creación de Tablas con sus claves primarias
CREATE TABLE carreras(
    id_carrera int NOT NULL,
    nombre_carrera varchar(50) NOT NULL,
    titulo varchar(50),
    CONSTRAINT PK_carreras PRIMARY KEY (id_carrera)
);

CREATE TABLE sexos(
    id_sexo integer NOT NULL,
    n_sexo varchar(20) NOT NULL,
    CONSTRAINT PK_sexos PRIMARY KEY (id_sexo)
);

CREATE TABLE estados_cursados(
    id_estado integer NOT NULL,
    n_estado varchar(30) NOT NULL,
    CONSTRAINT PK_estados_cur PRIMARY KEY (id_estado)
);

CREATE TABLE barrios(
    id_barrio integer NOT NULL,
    n_barrio varchar(50) NOT NULL,
    CONSTRAINT PK_barrios PRIMARY KEY (id_barrio)
);

CREATE TABLE tipo_notas(
    id_tipo_nota integer NOT NULL,
    n_tipo_nota varchar(20) NOT NULL,
    CONSTRAINT PK_tipo_notas PRIMARY KEY (id_tipo_nota)
);

CREATE TABLE tipo_documentos(
    id_tipo_documento integer NOT NULL,
    n_tipo_documento varchar(20) NOT NULL,
    CONSTRAINT PK_tipo_documentos PRIMARY KEY (id_tipo_documento)
);

CREATE TABLE materias(
    id_materia integer NOT NULL,
    nombre VARCHAR(50) NOT NULL,
    CONSTRAINT PK_materias PRIMARY KEY (id_materia)
);
```

```
CREATE TABLE alumnos_cursando_materias(
    id_materia integer NOT NULL,
    id_carrera integer NOT NULL,
    id_estudiante integer NOT NULL,
    fecha_inscripcion date NOT NULL,
    id_estado integer NOT NULL,
    fecha_estado date NOT NULL,
    CONSTRAINT PK_al_curs_mat PRIMARY KEY (id_materia, id_carrera, id_estudiante, fecha_inscripcion)
);
CREATE TABLE alumnos_examenes(
    id_materia integer NOT NULL,
    id_carrera integer NOT NULL,
    id_estudiante integer NOT NULL,
    fecha_del_examen date NOT NULL,
    fecha_inscripcion_a_examdate NOT NULL,
    nota integer,
    CONSTRAINT PK_alu_exam PRIMARY KEY (id_materia, id_carrera, id_estudiante, fecha_del_examen)
);
CREATE TABLE alumnos_cursando_notas(
    id_materia integer NOT NULL,
    id_carrera integer NOT NULL,
    id_estudiante integer NOT NULL,
    fecha_inscripcion date NOT NULL,
    fecha_nota date NOT NULL,
    id_tipo_nota integer NOT NULL,
    nota integer,
    CONSTRAINT PK_alu_curs_not PRIMARY KEY (id_materia, id_carrera, id_estudiante, fecha_inscripcion, fecha_nota)
);
CREATE TABLE inscripciones_a_carreras(
    id_estudiante integer NOT NULL,
    id_carrera integer NOT NULL,
    fecha date NOT NULL,
    CONSTRAINT PK_inscr_a_carr PRIMARY KEY (id_estudiante,id_carrera)
);
```

```
CREATE TABLE carreras(
    id_carrera int NOT NULL,
    nombre_carrera varchar(50) NOT NULL,
    titulo varchar(50),
CONSTRAINT PK_carreras PRIMARY KEY (id_carrera)
);

CREATE TABLE estudiantes(
    id_estudiante int NOT NULL,
    apellido varchar(30) NOT NULL,
    nombres varchar(100) NOT NULL,
    id_tipo_documento int NOT NULL,
    documento bigint NOT NULL,
    id_sexo int,
    calle varchar(60) NOT NULL,
    calle_numero varchar(30) NOT NULL,
    id_barrio int,
CONSTRAINT PK_estudiantes PRIMARY KEY (id_estudiante)
);

CREATE TABLE materias_x_turnos_examenes(
    id_turno integer NOT NULL,
    id_materia integer NOT NULL,
    id_carrera integer NOT NULL,
    fecha_del_examen date NOT NULL,
CONSTRAINT PK_matxturnexam PRIMARY KEY (id_materia, id_carrera, fecha_del_examen)
);
```

Creación de claves foráneas (en DB2 la longitud de los nombres de constraint ha sido disminuida para que sea aceptado)

```
ALTER TABLE alumnos_cursando_materias ADD CONSTRAINT FK_alucurma_esta FOREIGN KEY(id_estado)
REFERENCES estados_cursados (id_estado);
ALTER TABLE alumnos_cursando_materias ADD CONSTRAINT FK_alucurma_est FOREIGN KEY(id_estudiante)
REFERENCES estudiantes (id_estudiante);
ALTER TABLE alumnos_cursando_materias ADD CONSTRAINT FK_alcurma_plan FOREIGN KEY(id_materia, id_carrera)
REFERENCES plan_de_carrera (id_materia, id_carrera);
ALTER TABLE alumnos_examenes ADD CONSTRAINT FK_alu_exam_est FOREIGN KEY(id_estudiante)
REFERENCES estudiantes (id_estudiante);
```

```

ALTER TABLE alumnos_examenes ADD CONSTRAINT FK_alexamatxturexa
FOREIGN KEY(id_materia, id_carrera, fecha_del_examen)
REFERENCES materias_x_turnos_examenes (id_materia, id_carrera, fecha_del_examen);
ALTER TABLE alumnos_cursando_notas ADD CONSTRAINT FK_alcurnotnotas FOREIGN KEY(id_tipo_nota)
REFERENCES tipo_notas (id_tipo_nota);
ALTER TABLE alumnos_cursando_notas ADD CONSTRAINT FK_alcunoAlucurmat FOREIGN KEY(id_materia, id_carrera,
id_estudiante, fecha_inscripcion)
REFERENCES alumnos_cursando_materias (id_materia, id_carrera, id_estudiante, fecha_inscripcion);
ALTER TABLE inscripciones_a_carreras ADD CONSTRAINT FK_inscarcarr FOREIGN KEY(id_carrera)
REFERENCES carreras (id_carrera);
ALTER TABLE inscripciones_a_carreras ADD CONSTRAINT FK_insccarrest FOREIGN KEY(id_estudiante)
REFERENCES estudiantes (id_estudiante);
ALTER TABLE plan_de_carrera ADD CONSTRAINT FK_pcarreracarr FOREIGN KEY(id_carrera)
REFERENCES carreras (id_carrera);
ALTER TABLE plan_de_carrera ADD CONSTRAINT FK_pcarremate FOREIGN KEY(id_materia)
REFERENCES materias (id_materia);
ALTER TABLE estudiantes ADD CONSTRAINT FK_estbarrios FOREIGN KEY(id_barrio)
REFERENCES barrios (id_barrio);
ALTER TABLE estudiantes ADD CONSTRAINT FK_estudsexos FOREIGN KEY(id_sexo)
REFERENCES sexos (id_sexo);
ALTER TABLE estudiantes ADD CONSTRAINT FK_esttipodoc FOREIGN KEY(id_tipo_documento)
REFERENCES tipo_documentos (id_tipo_documento);

```

#### 10.3.4 Datos de prueba

Se insertará un conjunto de datos para probar la función y el procedimiento almacenado que se programarán en los ítems subsiguientes. De la misma manera que en la creación de tablas, las sentencias de inserción de datos (INSERT) se escribirán en la ventana del Editor del comando, disponible independientemente o desde la opción de Menú Herramientas del Centro de control. Previo a la ejecución, es necesario que se tenga seleccionada la base de datos correcta.

```

INSERT INTO barrios (id_barrio,n_barrio) VALUES (1,'Alto Alberdi');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (2,'Yofre');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (3,'Nueva Córdoba');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (4,'Centro');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (5,'Los Boulevares');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (6,'Guemes');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (7,'Ipona');

```

```
INSERT INTO barrios (id_barrio,n_barrio) VALUES (8,'Alta Córdoba');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (9,'General Paz');

INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (1,'Parcial 1');
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (2,'Parcial 2');
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (3,'Final');
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (4,'Trabajo Práctico');

INSERT INTO tipo_documentos (id_tipo_documento,n_tipo_documento) VALUES (1,'DNI');
INSERT INTO tipo_documentos (id_tipo_documento,n_tipo_documento) VALUES (2,'LC');

INSERT INTO carreras (id_carrera,nombre_carrera,titulo) VALUES (1,'Ingeniería en Sistemas de Información','Ingeniero en Sistemas de Información');

INSERT INTO sexos (id_sexo,n_sexo) VALUES (1,'Masculino');
INSERT INTO sexos (id_sexo,n_sexo) VALUES (2,'Femenino');

INSERT INTO estados_cursados (id_estado,n_estado) VALUES (1,'Inscripto');
INSERT INTO estados_cursados (id_estado,n_estado) VALUES (2,'Regular');
INSERT INTO estados_cursados (id_estado,n_estado) VALUES (3,'Libre');

INSERT INTO materias (id_materia,nombre) VALUES (1,'Algoritmos y Estructuras de Datos');
INSERT INTO materias (id_materia,nombre) VALUES (2,'Análisis Matemático I');
INSERT INTO materias (id_materia,nombre) VALUES (3,'Análisis Matemático II');
INSERT INTO materias (id_materia,nombre) VALUES (4,'Algebra y Geometría Analítica');
INSERT INTO materias (id_materia,nombre) VALUES (5,'Matemática Discreta');
INSERT INTO materias (id_materia,nombre) VALUES (6,'Sistemas y Organizaciones');
INSERT INTO materias (id_materia,nombre) VALUES (7,'Ingeniería y Sociedad');
INSERT INTO materias (id_materia,nombre) VALUES (8,'Paradigmas de Programación');

INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (1,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (2,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (3,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (4,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (5,1,1,2);
```

```
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (6,1,1,2);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (7,1,1,2);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (8,1,1,2);

INSERT INTO estudiantes (id_estudiante,apellido,nombres,id_tipo_documento,documento,id_sexo,calle,calle_numero,id_barrio) VALUES (1,'Abrutsky','Maximiliano Adrián',1,23852963,1,'Av. Colón','1035',1);
INSERT INTO estudiantes (id_estudiante,apellido,nombres,id_tipo_documento,documento,id_sexo,calle,calle_numero,id_barrio) VALUES (2,'DAMIANO','Luis',1,19147258,1,'Dean Funes','903',1);

INSERT INTO inscripciones_a_carreras (id_estudiante,id_carrera,fecha) VALUES (1,1,'26/01/2009');

INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (1,1,1,'01/02/2009',1,'01/02/2009');
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (2,1,1,'01/02/2009',1,'01/02/2009');
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (3,1,1,'01/02/2009',1,'01/02/2009');
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (4,1,1,'01/02/2009',1,'01/02/2009');

INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_inscripcion,fecha_nota,id_tipo_nota,nota) VALUES (1,1,1,'01/02/2009','15/03/2009',1,5);
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_inscripcion,fecha_nota,id_tipo_nota,nota) VALUES (1,1,1,'01/02/2009','26/05/2009',2,7);
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_inscripcion,fecha_nota,id_tipo_nota,nota) VALUES (2,1,1,'01/02/2009','18/03/2009',1,3);
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_inscripcion,fecha_nota,id_tipo_nota,nota) VALUES (2,1,1,'01/02/2009','29/05/2009',2,4);
```

## 10.4 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) solo está disponible para docentes.

### 10.4.1 Mapa conceptual del capítulo

### 10.4.2 Autoevaluación

### 10.4.3 Presentaciones\*

# 11

## Implementando el modelo en SQL Server 2005

### Contenido

|  |     |
|--|-----|
| 11.1 Introducción.....   | 312 |
| 11.2 Inicializando el servidor.....                                      | 312 |
| 11.3 Inicializando el cliente.....                                       | 317 |
| 11.4 Creando y probando nuestro modelo de<br>Estudiante-Universidad..... | 318 |
| 11.5 Contenido de la página Web de apoyo.....                            | 334 |

### Objetivos

- Saber dónde conseguir el instalador del sistema de gestión de bases de datos SQL Server 2005, edición Express de Microsoft.
- Comprender el proceso de instalación y poder realizar la parametrización básica de la instancia.
- Adquirir los conocimientos para inicializar el servidor de bases de datos y la herramienta cliente.
- Utilizar el *SQL Server Management Studio* para crear la base de datos, todas las tablas y relaciones del modelo visto en el libro y los objetos de programación de la base de datos.

## 11.1 Introducción

SQL Server es el Sistema de Gestión de Bases de Datos Relacional (SGBDR) de Microsoft. Esta empresa tuvo su ingreso en el mercado de bases de datos de nivel corporativo, por medio de una asociación con las firmas Sybase y Ashton-Tate. De esa asociación, surgió SQL Server 1.0 en 1989.

Si bien al principio sus características y su compatibilidad con Sistemas Operativos eran limitadas, los sucesivos lanzamientos de SQL Server lograron incrementar la porción de mercado hasta obtener una posición vital en el panorama actual de bases de datos corporativas.

En la actualidad, las dos últimas versiones comerciales de SQL Server son la 2005 y 2008. A partir de la versión 2005, existe una edición sin costos de licenciamientos denominada “MS SQL Server Express Edition”; anteriormente, en la versión 2000, existía una edición de licenciamiento similar MSDE “Ms SQL Server 2000 Desktop Engine”.

Aquí se utilizará la edición Express de la versión 2005 de Ms SQL Server.



En la Web de apoyo, encontrará el vínculo a la página Web que le brindará información acerca del producto SQL Server 2005.

## 11.2 Inicializando el servidor

El primer paso que se realizará es la instalación de SGBDR en el servidor, de manera que, posteriormente, pueda conectarse desde los clientes que se pueden ejecutar en el mismo equipo o bien desde otro nodo de la red.

### 11.2.1 Download

Como se mencionó anteriormente, la edición Express es de distribución gratuita; por lo tanto, se puede descargar desde el centro de descargas oficial de Microsoft. El link válido al momento de la publicación de esta obra es:

<http://www.microsoft.com/downloadS/details.aspx?displaylang=es&FamilyID=220549b5-0b07-4448-8848-dcc397514b41>

Este link podría estar obsoleto en el momento de la lectura, su búsqueda debería hacerse utilizando “Microsoft SQL Server 2005 Express Edition”.

### 11.2.2 Instalación

La instalación puede llevarse a cabo utilizando un archivo de configuración en el que se especifican todos los datos necesarios (rutas, tipo de autenticación, propiedades de la instancia, etc.), o bien a través del asistente que lo guiará paso a paso. Para usuarios principiantes, se recomienda la segunda alternativa.

Los requisitos mínimos de hardware para esta edición son: procesador Pentium 600 Mhz. o mayor, 192Mb de memoria RAM y, al menos, 150Mb de espacio libre en el disco duro. El Sistema Operativo puede ser una edición servidor como Windows 2000 Server SP4, o superior, o bien de uso doméstico como Windows XP SP2 home edition o superior. (En la edición empresarial, es necesario un sistema operativo Windows servidor).

En la instalación guiada por el asistente, el usuario deberá definir algunas características como el nombre de la instancia, el tipo de intercalación, etcétera. Para evitar la

profundización de los aspectos técnicos, se recomienda dejar todas las opciones por defecto; si se enfatiza en que se preste la máxima atención en el nombre de la instancia, en el modo de autenticación y en la intercalación.

#### 11.2.2.1 Nombre de la instancia

Una instancia de SQL Server es básicamente una instalación del motor de bases de datos de SQL Server, que se materializa como un servicio de Windows. Es decir: una instancia sería un servidor lógico, puesto que pueden instalarse varias instancias en un equipo o servidor físico. Cada instancia o servidor lógico contendrá varias bases de datos.

Normalmente, se instala solo una instancia por cada equipo, pero suelen darse situaciones en las que se utilizan varias instancias para economizar recursos; por ejemplo, una instancia para cada ciclo de vida del desarrollo, es decir, una instancia de desarrollo, otra de prueba y otra de preproducción, las tres en el mismo equipo físico o servidor y, luego, una única instancia de producción en otro servidor.

En el caso de tener varias instancias, es importante destacar que competirán por los recursos de memoria, disco y procesamiento, situación que motiva tener la instancia de producción en un servidor dedicado (solo con una instancia). Cada instancia se diferenciará del resto a través de un nombre único.

En la instalación, se tienen dos posibilidades respecto del nombre de la instancia: la primera es que se defina un nombre, práctica recomendada si se tiene previsto instalar más de una instancia en el equipo. La segunda alternativa es utilizar la instancia predeterminada, o sin nombre, lo que es común cuando se instala una sola instancia en el equipo. Solo se puede instalar una instancia predeterminada, o sin nombre, y puede coexistir con varias instancias con nombres en el mismo equipo.

La Fig. 11.1 muestra la ventana de diálogo del asistente de instalación, en la que el usuario definirá el nombre de la instancia o instalará la instancia predeterminada. Se recomienda esta segunda opción, tal cual se visualiza en la figura mencionada.



Fig. 11.1 Nombre de la instancia.

### 11.2.2.2 Modo de autenticación

La autenticación es el proceso por el cual se comprueba alguna identidad, se verifica que quién dice ser alguien realmente lo sea, para, de esta manera, controlar el acceso.

SQL Server tiene dos modos de autenticación: uno es el modo de autenticación Windows, en el que los usuarios se autentican por el sistema operativo, desligándose el motor de bases de datos de esta tarea, dado que confía en el sistema operativo (con este modo de autenticación, SQL Server no pedirá ninguna contraseña). El otro es el modo de autenticación mixta: los usuarios también se acreditan por SQL Server; el motor de bases de datos solicitará la contraseña para realizarla.

En el proceso de instalación (Fig. 11.2), se define el modo de autenticación para el inicio de sesión 'sa', que tiene el rol de System Administrator (sysadmin) o administrador de sistema. Es el rol con el que se realizarán todas las tareas relacionadas al servidor de bases de datos, ya que agrupa todos los permisos posibles y, con esto, se hará la primera conexión al motor de bases de datos. Entonces, si se elige el modo de autenticación mixta, definirá una contraseña que no se olvidará. Si el modo elegido es autenticación Windows, se logeará al sistema operativo con el mismo usuario que hizo la instalación y, luego, se accederá al servidor sin definir ninguna contraseña.

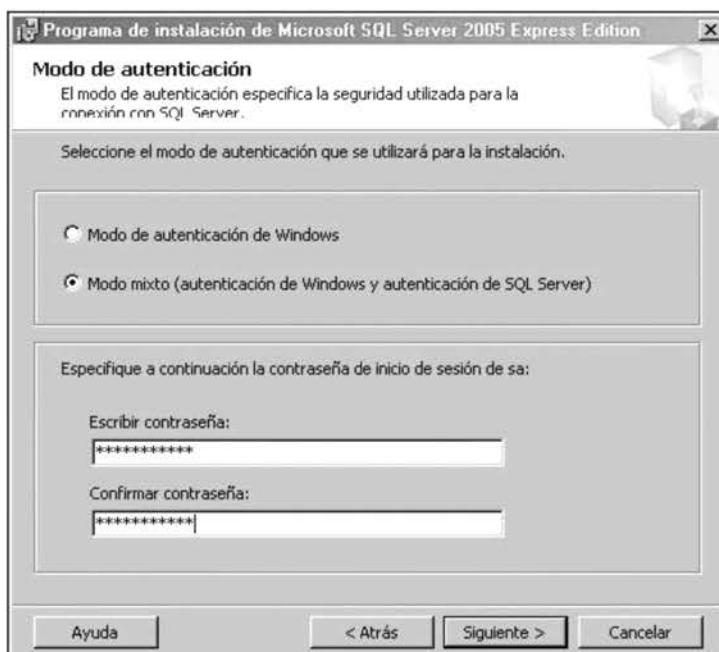


Fig. 11.2 Modo de autenticación.

En la Fig. 11.2, se visualiza la posibilidad de seleccionar los dos modos de autenticación mencionados con anterioridad; se recomienda elegir el modo de autenticación mixta y recordar la contraseña para poder lograr una conexión efectiva. Fundamentos de sencillez conceptual motivan esta recomendación, no porque sea más seguro o mejor en todos los aspectos que el otro modo. Hay circunstancias en las que resulta

conveniente la utilización de uno u otro, según las necesidades. Queda a criterio del lector profundizar acerca de la seguridad en SQL Server 2005, donde podrá ahondar en las ventajas y desventajas de cada modo.

#### 11.2.2.3 Intercalación

También es importante dedicar atención a la definición de la intercalación para el servidor, ya que define las reglas de comparación y de ordenamiento entre cadenas de caracteres. ¿Usted considera que la siguiente comparación es verdadera? Papa = papa = papá. Esto dependerá de la intercalación definida, si es o no binaria, si es sensible a mayúsculas / minúsculas, etc. Ahora, suponga que ese conjunto de valores es retornado desde una consulta SQL con una cláusula que lo ordene, ¿cómo considera que se debería ordenar? Esto también depende de la intercalación, es decir, define reglas de comparación y de ordenamiento. Este tema puede profundizarse si se aborda desde una intercalación que depende de un juego de caracteres (conjunto de todos los pares símbolo/codificación), pero esto queda sujeto a interés del lector.

Durante la instalación, se define la intercalación en el nivel servidor (Fig. 11.3), pero, también, puede definirse otra para cada base de datos (en el momento de su creación) o para cada tabla dentro de cada base, o bien hasta para cada columna (del tipo carácter) de cada tabla (también cuando se la crea). En esta jerarquía de definiciones de intercalaciones, se utiliza siempre la de menor granularidad (la de menor nivel), esto implica que si no se determinó una intercalación específica en una columna, se recurrirá a la de la tabla; si tampoco se la definió, se utilizará la definida en la base de datos; y si ésta tampoco se definió, entonces se empleará la del servidor. Normalmente, sucede esta situación si se descuida esta propiedad y, por ende, se aplica la definida en el servidor o, lo que es lo mismo, la que se define en el momento de la instalación.

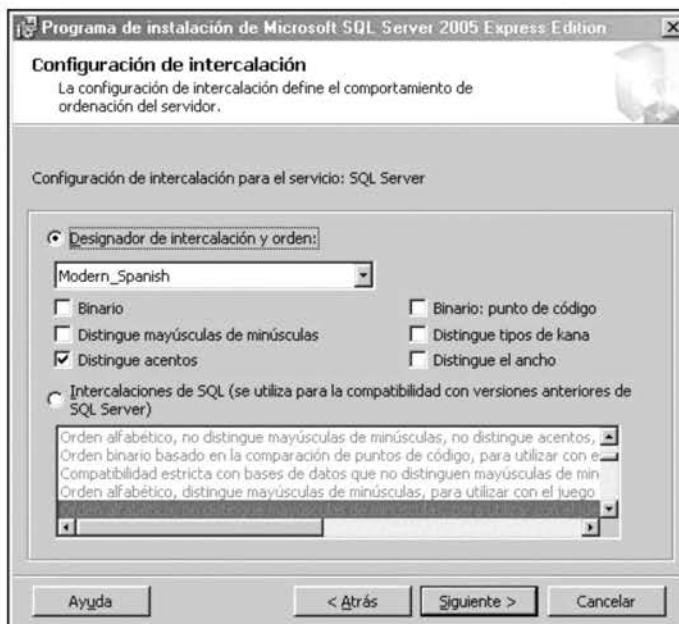


Fig. 11.3 Intercalación de la instancia.

### 11.2.3 Inicialización

SQL Server, tal como lo hacen otras aplicaciones en el sistema operativo Windows, se ejecuta a través de servicios (procesos de segundo plano). Distintos componentes de SQL Server se ejecutan como distintos servicios. Este capítulo centrará su interés en el servicio principal de SQL Server, que representa la accesibilidad de la instancia.

Si se considera que la instalación se ejecutó con los valores por defecto, el servicio se denomina SQLSERVER (dicho nombre puede cambiar si durante la instalación se define un nombre para la instancia). El servidor estará ejecutándose y accesible, si y solo si, dicho servicio se está ejecutando. Esto se puede supervisar tanto desde las herramientas administrativas de Windows (Fig. 11.4), como desde el Manejador de configuraciones (Fig. 11.5), que es una herramienta que se instala automáticamente con SQL Server con la finalidad de configurar los servicios y las conexiones.

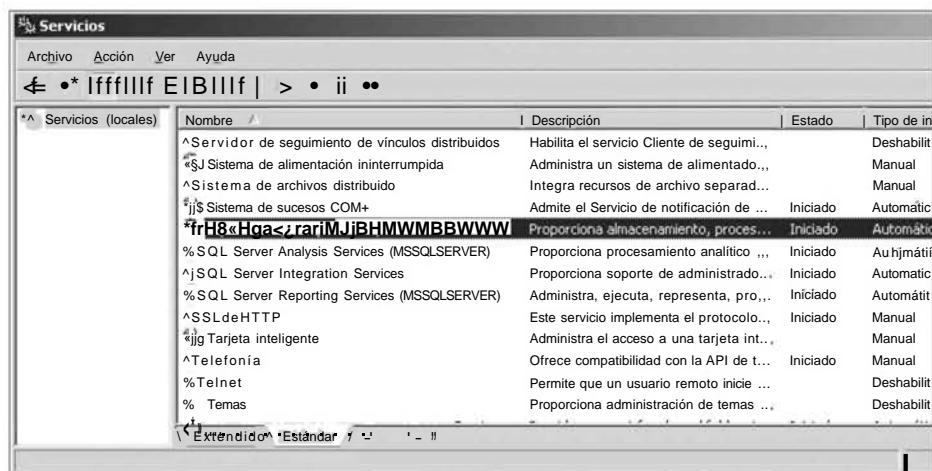


Fig. 11.4 Herramientas administrativas -> Servicios.

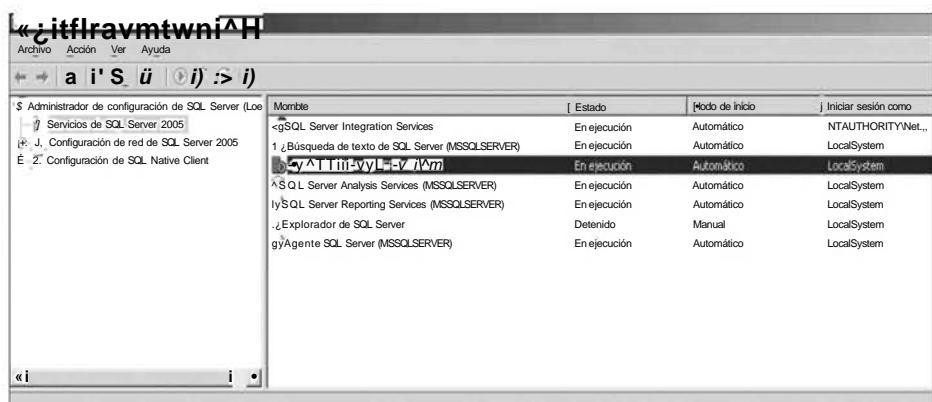


Fig. 11.5 Manejador de configuraciones SQL Server.

## 11.3 Inicializando el cliente

Finalizada la instalación de nuestro servidor, se continuará con la instalación del SQL Server Management Studio 2005 (SSMS), que es la herramienta gráfca de Microsoft de libre distribución, que brinda un entorno integrado para obtener el acceso a todos los componentes de SQL Server, para configurarlos, administrarlos y desarrollarlos. El entorno visual y la sencillez de uso, es la motivación principal para utilizar esta herramienta. Es válido mencionar que si optase por no instalarlo, la forma de interactuar con el SGBDR SQL Server 2005, sería a través de sqlcmd, que es la herramienta de línea de comando para la ejecución de consultas en línea.

### 11.3.1 Download

La descarga, también gratuita, se hace igualmente desde el centro de descarga oficial de Microsoft, pero de manera independiente a la descarga del servidor de SQL Server 2005. El link válido en el momento de la publicación de esta obra es:

<http://www.microsoft.com/downloadS/details.aspx?familyid=C243A5AE-4BD1-4E3D-94B8-5A0F62BF7796&displaylang=es>

En caso de que el enlace esté obsoleto durante la lectura de este libro, se realizará la búsqueda utilizando “Microsoft SQL Server Management Studio Express 2005”.

### 11.3.2 Instalación

Es un proceso muy sencillo, dado que el paquete de instalación de Windows Installer (archivo .msi) realiza la instalación prácticamente de manera automática. Sí hay que tener presente que es necesario tener instalado el Framework.Net 2.0 y Microsoft Core XML Services (MSXML) 6.0 en el equipo en el que se realizará la instalación. Ambas instalaciones no son necesarias si se instala en el mismo equipo donde se instaló el servidor SQL Server 2005 (ya tiene esos componentes).

### 11.3.3 Utilizando el SSMS

Para utilizar esta herramienta, diríjase al menú Inicio, seleccione Todos los programas, Microsoft SQL Server 2005 y, a continuación, haga clic en SQL Server Management Studio.

En la siguiente imagen, se señalan los dos componentes principales del SSMS: el explorar de objetos y la ventana de documentos (Fig. 11.6). Existen más vistas que pueden seleccionarse desde el menú Ver.

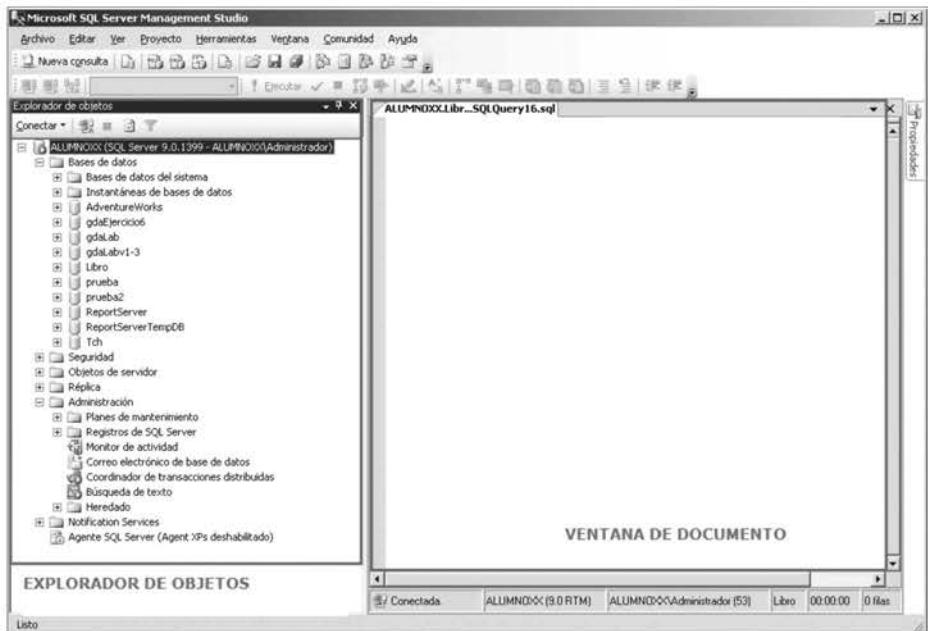


Fig. 11.6 Pantalla de SQL Server Management Studio.



Para mayor información, se sugiere remitirse a los tutoriales de esta herramienta en la biblioteca MSDN (Microsoft Developer Network):  
[http://msdn.microsoft.com/es-ar/library/ms167593\(SQL\\_90\).aspx](http://msdn.microsoft.com/es-ar/library/ms167593(SQL_90).aspx)

**Explorador de objetos:** es una vista en árbol con todos los objetos de la base de datos que contiene un servidor (en la imagen, servidor ALUMNOXX). Obviamente, el explorador necesita conectarse a los servidores para poder mostrar sus contenidos, de los cuales solo se mostrarán aquellos para los que se tiene permiso con la cuenta con la que se realiza la conexión.

**Ventana de documento:** es el área de mayor tamaño que puede contener editores de consulta y ventanas del explorador.

## 11.4 Creando y probando nuestro modelo de Estudiante – Universidad

En el momento en que el lector está en condiciones de interactuar con SQL Server 2005, el servidor está corriendo y ya conoce una breve descripción de la herramienta SSMS, con la que puede realizar las tareas administrativas y de programación de bases de datos SQL Server.

Como primer paso, se procederá a crear una base de datos, luego a crear todas las tablas utilizando el script (conjunto de sentencias que se ejecutan de forma conjunta).

### 11.4.1 Creación de la base de datos

Tanto la creación de la base de datos como las demás actividades se llevarán a cabo utilizando la cómoda interfaz del SSMS.

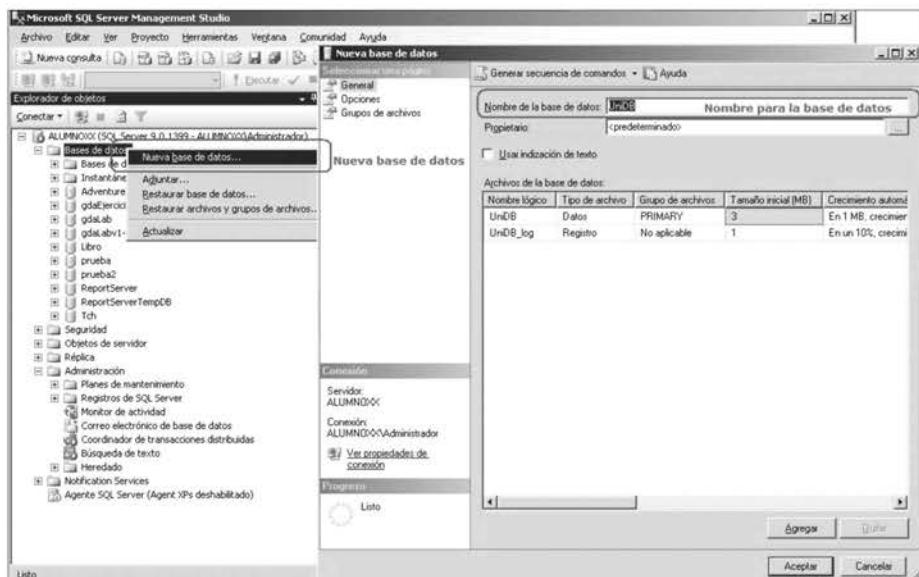


Fig. 11.7 Creación de una base de datos.

El proceso de creación de la base de datos desde el SSMS es muy sencillo: hay que dirigirse al explorador de objetos, seleccionar el grupo “Bases de Datos” y desplegar su menú contextual (haciendo clic en el botón secundario del mouse), para acceder a la opción “Nueva Base de Datos” (Fig. 11.7).

En la creación, uno puede especificar determinadas opciones a la base de datos que está creando como, por ejemplo, la intercalación utilizada en la base que puede diferir de la del servidor (ver instalación), el modelo de recuperación lo que impacta en cómo se administra el archivo *log* de transacciones, y de esto dependen los posibles tipos de copias de seguridad disponibles, si la base de datos es de solo lectura, la ubicación física en la que estarán los archivos que componen la base de datos, etcétera.

Pero con solamente especificar el nombre de la nueva base de datos, ésta se crearía con toda la configuración por *default* (u omisión). Es el único valor requerido que el usuario deberá ingresar, tal como se visualiza en la Fig. 11.6, en la que se especificó el nombre UniDB.

Los valores por omisión provienen de la base de datos de sistema model, que se crea automáticamente al instalar SQL Server 2005. Es decir que los valores de cada propiedad que se omiten (no se especifican) para cada nueva base de datos adoptará el valor que tiene la DB model. Por esto es también llamada base de datos de plantilla.

## 11.4.2 Creación de las tablas y sus relaciones

### 11.4.2.1 Ejecución de script

Ahora que ya se cuenta con la base de datos, se procederá a crear las tablas correspondientes al modelo y sus respectivas claves para asegurar la integridad de datos. Posteriormente, se crearán, a modo de ejemplo, una función y un procedimiento almacenado de programación en la base de datos.

Referencia para la interpretación del script: las primeras sentencias son las de creación de tablas con sus claves primarias; luego, las de modificación de las tablas recientemente creadas para incorporar las claves foráneas y, finalmente, la inserción de datos de prueba y de creación de la función y del procedimiento almacenado. El motivo por el que se fundamenta la decisión de modificar las tablas para incorporar las claves foráneas (FK), y no crearlas junto con la definición de cada tabla, es que ambas tablas relacionadas por dicha clave deben existir en el momento de su creación. De esta manera, se evitan complicaciones originadas por esta causa.

Las sentencias que se ejecutarán se especifican en la ventana de documento del SSMS. Hay que seleccionar la base de datos en la que se ejecutarán las sentencias DDL que crearán los objetos del modelo. Esto puede hacerse seleccionándola desde el combo de bases de datos (Fig. 11.8), o bien escribiendo la sentencia USE nombre\_de\_la\_base\_a\_seleccionarse antes de la primera sentencia. En el punto anterior, se la llamó UniDB; por ende, se debe seleccionar UniDB en el combo o escribir USE uniDB como primera sentencia en la ventana de documento (previo al primer CREATE TABLE).

```

Microsoft SQL Server Management Studio
Archivo Editar Ver Consulta Proyecto Herramientas Ventana Comunidad Ayuda Ejecutar las sentencias (F5)
Nueva consulta Ejecutar
Explorador de objetos BD seleccionada
Unidb
Alumnoxx (SQL Server 9.0.1399 - ALUMNOXX\Administrador)
Bases de datos
Instantáneas de bases de datos
AdventureWorks
gdsejercicios
gdslab
gdslabv1-3
Libro
prueba
prueba2
ReportServer
ReportServerTempDB
Tch
UniDB
Seguridad
Objetos de servidor
Riplica
Administración
Planes de mantenimiento
Registros de SQL Server
Monitor de actividad
Correo electrónico de base de datos
Coordinador de transacciones distribuidas
Búsqueda de texto
Heredado
Notification Services
Agente SQL Server (Agent XPs deshabilitado)

ALUMNOXX\JULIARMADEADO.sql [ ALUMNOXX\LIBR...SQLQuery16.sql ]
-- Creación de Tablas con sus claves primarias
CREATE TABLE Carreras(
    id_carrera int NOT NULL,
    nombre_carrera varchar(50) NOT NULL,
    titulo varchar(50) NULL,
    CONSTRAINT PK_Carreras PRIMARY KEY CLUSTERED (id_carrera ASC)
)
GO

CREATE TABLE Sexos(
    id_sexo int NOT NULL,
    n_sexo varchar(20) NOT NULL,
    CONSTRAINT PK_Sexos PRIMARY KEY CLUSTERED (id_sexo ASC)
)
GO

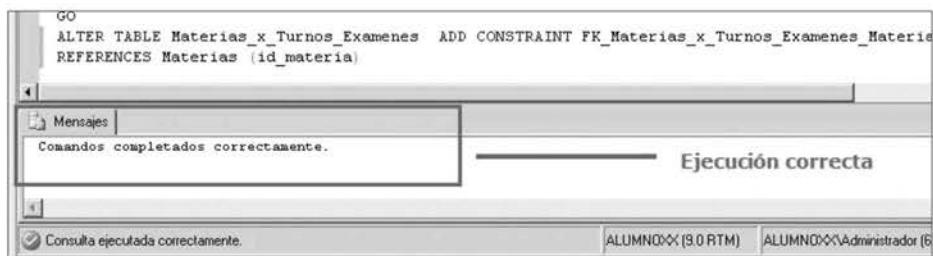
CREATE TABLE Estados_Cursados(
    id_estado int NOT NULL,
    n_estado varchar(30) NOT NULL,
    CONSTRAINT PK_Estados PRIMARY KEY CLUSTERED (id_estado ASC)
)
GO

CREATE TABLE Barrios(
    id_barrio int NOT NULL,
    n_barrio varchar(50) NOT NULL,
    CONSTRAINT PK_Barrios PRIMARY KEY CLUSTERED (id_barrio ASC)
)
GO

```

Fig. 11.8 Ejecución de sentencias.

Luego de asegurar que los objetos se crearán en la base de datos correspondiente, se ejecutarán las sentencias con el botón destinado a tal fin, o bien utilizando la tecla de función F5. En la consola de mensajes (bajo la ventana de documentos), se muestra el resultado de la Fig. 11.9. Se podrán visualizar los objetos creados desde el explorador de objetos; en el caso de las tablas, hay que desplegar la base de datos y, luego, el grupo “Tablas”.



The screenshot shows the SSMS interface. In the top pane, there is T-SQL code: GO and ALTER TABLE Materias\_x\_Turnos\_Examenes ADD CONSTRAINT FK\_Materias\_x\_Turnos\_Examenes\_Materias REFERENCES Materias (id\_materia). Below this, the 'Mensajes' (Messages) window is open, displaying the message 'Comandos completados correctamente.' (Commands completed successfully.). To the right of this window, the text 'Ejecución correcta' (Execution successful) is written. At the bottom of the screen, there is a status bar with the text 'Consulta ejecutada correctamente.' (Query executed successfully.), the version 'ALUMNOXXX (9.0 RTM)', and the user 'ALUMNOXXX\Administrador (6)'.

Fig. 11.9 Ejecución correcta.

#### 11.4.2.2 Modelo de datos

La Fig. 11.10 representa el diagrama del modelo de datos utilizado. Corresponde mencionar que el asistente para la creación de dicho diagrama viene incorporado en el SSMS y que su proceso de definición es muy sencillo. Quien deseara instruirse prácticamente en la utilización del asistente para diagrama, se dirigirá, dentro del SSMS, al grupo “Diagramas de bases de datos” para la base de datos de la que desea crear un diagrama y, tras seleccionar la opción de nuevo diagrama de base de datos del menú contextual, se iniciará el asistente que lo guiará en la definición.

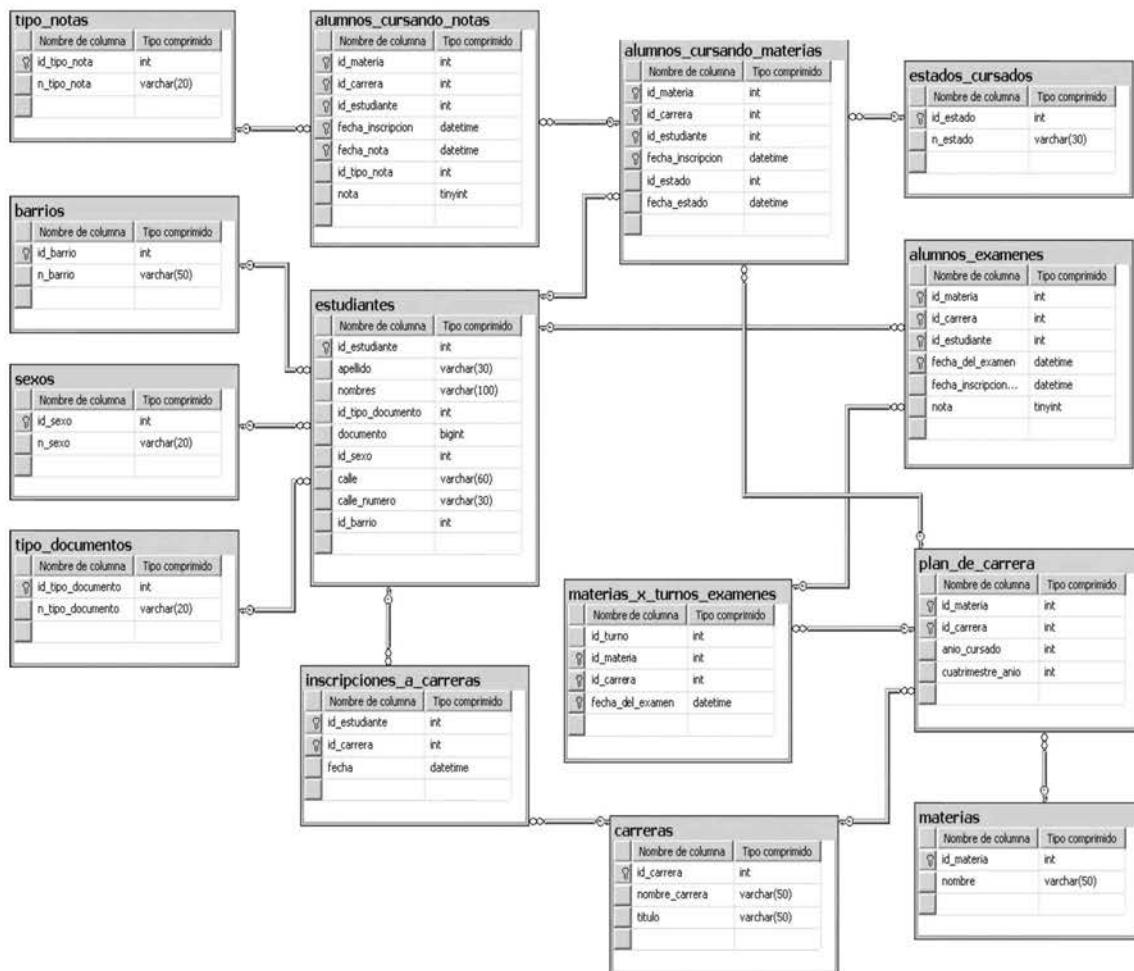


Fig. 11.10 Modelo de datos completo.

#### 11.4.2.3 Script (sentencias DDL)

A continuación, se precisan las sentencias de definición de datos que se ejecutarán para disponer del modelo completo de la Fig. 11.10.

```
-- Creación de Tablas con sus claves primarias
CREATE TABLE carreras(
    id_carrera int NOT NULL,
    nombre_carrera varchar(50) NOT NULL,
    título varchar(50) NULL,
```

```
CONSTRAINT PK_carreras PRIMARY KEY CLUSTERED (id_carrera ASC)
)
GO

CREATE TABLE sexos(
    id_sexo int NOT NULL,
    n_sexo varchar(20) NOT NULL,
CONSTRAINT PK_sexos PRIMARY KEY CLUSTERED (id_sexo ASC)
)
GO

CREATE TABLE estados_cursados(
    id_estado int NOT NULL,
    n_estado varchar(30) NOT NULL,
CONSTRAINT PK_estados_cursados PRIMARY KEY CLUSTERED (id_estado ASC)
)
GO

CREATE TABLE barrios(
    id_barrio int NOT NULL,
    n_barrio varchar(50) NOT NULL,
CONSTRAINT PK_barrios PRIMARY KEY CLUSTERED (id_barrio ASC)
)
GO

CREATE TABLE tipo_notas(
    id_tipo_nota int NOT NULL,
    n_tipo_nota varchar(20) NOT NULL,
CONSTRAINT PK_tipo_notas PRIMARY KEY CLUSTERED (id_tipo_nota ASC)
)
GO

CREATE TABLE tipo_documentos(
    id_tipo_documento int NOT NULL,
    n_tipo_documento varchar(20) NOT NULL,
CONSTRAINT PK_tipo_documentos PRIMARY KEY CLUSTERED (id_tipo_documento ASC)
)
GO
```

```
CREATE TABLE materias(
    id_materia int NOT NULL,
    nombre VARCHAR(50) NOT NULL,
    CONSTRAINT PK_materias PRIMARY KEY CLUSTERED (id_materia ASC)
)
GO

CREATE TABLE alumnos_cursando_materias(
    id_materia int NOT NULL,
    id_carrera int NOT NULL,
    id_estudiante int NOT NULL,
    fecha_inscripcion datetime NOT NULL,
    id_estado int NOT NULL,
    fecha_estado datetime NOT NULL,
    CONSTRAINT PK_alumnos_cursando_materias PRIMARY KEY CLUSTERED (id_materia ASC, id_carrera ASC, id_estudiante ASC, fecha_inscripcion ASC)
)
GO

CREATE TABLE alumnos_examenes(
    id_materia int NOT NULL,
    id_carrera int NOT NULL,
    id_estudiante int NOT NULL,
    fecha_del_examen datetime NOT NULL,
    fecha_inscripcion_a_examene datetime NOT NULL,
    nota tinyint NULL,
    CONSTRAINT PK_alumnos_examenes PRIMARY KEY CLUSTERED (id_materia ASC, id_carrera ASC, id_estudiante ASC, fecha_del_examen ASC)
)
GO

CREATE TABLE alumnos_cursando_notas(
    id_materia int NOT NULL,
    id_carrera int NOT NULL,
    id_estudiante int NOT NULL,
    fecha_inscripcion datetime NOT NULL,
    fecha_nota datetime NOT NULL,
    id_tipo_nota int NOT NULL,
    nota tinyint NULL,
```

```
CONSTRAINT PK_alumnos_cursando_notas PRIMARY KEY CLUSTERED (id_materia ASC,id_carrera ASC,id_estudiante ASC,fecha_inscripcion ASC,fecha_nota ASC)
)
GO

CREATE TABLE inscripciones_a_carreras(
    id_estudiante int NOT NULL,
    id_carrera int NOT NULL,
    fecha datetime NOT NULL,
CONSTRAINT PK_inscripciones_a_carreras PRIMARY KEY CLUSTERED (id_estudiante ASC,id_carrera ASC)
)
GO

CREATE TABLE plan_de_carrera(
    id_materia int NOT NULL,
    id_carrera int NOT NULL,
    anio_cursado int NULL,
    cuatrimestre_anio int NULL,
CONSTRAINT PK_plan_de_carrera PRIMARY KEY CLUSTERED (id_materia ASC,id_carrera ASC)
)
GO

CREATE TABLE estudiantes(
    id_estudiante int NOT NULL,
    apellido varchar(30) NOT NULL,
    nombres varchar(100) NOT NULL,
    id_tipo_documento int NOT NULL,
    documento bigint NOT NULL,
    id_sexo int NULL,
    calle varchar(60) NOT NULL,
    calle_numero varchar(30) NOT NULL,
    id_barrio int NULL,
CONSTRAINT PK_estudiantes PRIMARY KEY CLUSTERED (id_estudiante ASC)
)
GO

CREATE TABLE materias_x_turnos_examenes(
    id_turno int NOT NULL,
    id_materia int NOT NULL,
```

```
    id_carrera int NOT NULL,
    fecha_del_examen datetime NOT NULL,
CONSTRAINT PK_materias_x_turnos_examenes PRIMARY KEY CLUSTERED (id_materia ASC, id_carrera ASC, fecha_
del_examen ASC)
)
GO

-- Creación de claves foráneas
ALTERTABLE alumnos_cursando_materias ADD CONSTRAINT FK_alumnos_cursando_materias_estados FOREIGN
KEY(id_estado)
REFERENCES estados_cursados (id_estado)
GO
ALTERTABLE alumnos_cursando_materias ADD CONSTRAINT FK_alumnos_cursando_materias_estudiantes FOREIGN
KEY(id_estudiante)
REFERENCES estudiantes (id_estudiante)
GO
ALTERTABLE alumnos_cursando_materias ADD CONSTRAINT FK_alumnos_cursando_materias_plan_carrera FOREIGN
KEY(id_materia, id_carrera)
REFERENCES plan_de_carrera (id_materia, id_carrera)
GO
ALTERTABLE alumnos_examenes ADD CONSTRAINT FK_alumnos_examenes_estudiantes FOREIGN KEY(id_estudian-
te)
REFERENCES estudiantes (id_estudiante)
GO
ALTER TABLE alumnos_examenes ADD CONSTRAINT FK_alumnos_examenes_materias_x_turnos_examenes FOREIGN
KEY(id_materia, id_carrera, fecha_del_examen)
REFERENCES materias_x_turnos_examenes (id_materia, id_carrera, fecha_del_examen)
GO
ALTERTABLE alumnos_cursando_notas ADD CONSTRAINT FK_alumnos_cursando_notas_tipo_notas FOREIGN KEY(id_-
tipoNota)
REFERENCES tipo_notas (id_tipoNota)
GO
ALTER TABLE alumnos_cursando_notas ADD CONSTRAINT FK_alumnos_cursando_notas_alumnos_cursando_materias
FOREIGN KEY(id_materia, id_carrera, id_estudiante, fecha_incripcion)
REFERENCES alumnos_cursando_materias (id_materia, id_carrera, id_estudiante, fecha_incripcion)
GO
ALTERTABLE inscripciones_a_carreras ADD CONSTRAINT FK_inscripciones_a_carreras_carreras FOREIGN KEY(id_ca-
rrera)
REFERENCES carreras (id_carrera)
GO
```

```
ALTERTABLE inscripciones_a_carreras ADD CONSTRAINT FK_inscripciones_a_carreras_estudiantes FOREIGN KEY(id_estudiante)
REFERENCES estudiantes (id_estudiante)
GO
ALTERTABLE plan_de_carrera ADD CONSTRAINT FK_plan_de_carrera_carreras FOREIGN KEY(id_carrera)
REFERENCES carreras (id_carrera)
GO
ALTERTABLE plan_de_carrera ADD CONSTRAINT FK_plan_de_carrera_materias FOREIGN KEY(id_materia)
REFERENCES materias (id_materia)
GO
ALTERTABLE estudiantes ADD CONSTRAINT FK_estudiantes_barrios FOREIGN KEY(id_barrio)
REFERENCES barrios (id_barrio)
GO
ALTERTABLE estudiantes ADD CONSTRAINT FK_estudiantes_sexos FOREIGN KEY(id_sexo)
REFERENCES sexos (id_sexo)
GO
ALTERTABLE estudiantes ADD CONSTRAINT FK_estudiantes_tipo_documentos FOREIGN KEY(id_tipo_documento)
REFERENCES tipo_documentos (id_tipo_documento)
GO
ALTERTABLE materias_x_turnos_examenes ADD CONSTRAINT FK_materias_x_turnos_examenes_plan_carrera FOREIGN KEY(id_materia, id_carrera)
REFERENCES plan_de_carrera (id_materia, id_carrera)
GO
```

## 11.4.3 Programación de la base de datos

### 11.4.3.1 Datos de prueba

Se insertarán un conjunto de datos para probar la función y el procedimiento almacenado, que se programarán en los ítems subsiguientes. De la misma manera que en la creación de tablas, las sentencias de inserción de datos (INSERT) se escribirán en la ventana de documento del SSMS; previo a la ejecución, se debe asegurar que la base de datos seleccionada es la correcta.

```
INSERT INTO barrios (id_barrio,n_barrio) VALUES (1,'Alto Alberdi')
INSERT INTO barrios (id_barrio,n_barrio) VALUES (2,'Yofre')
INSERT INTO barrios (id_barrio,n_barrio) VALUES (3,'Nueva Córdoba')
INSERT INTO barrios (id_barrio,n_barrio) VALUES (4,'Centro')
INSERT INTO barrios (id_barrio,n_barrio) VALUES (5,'Los Boulevares')
INSERT INTO barrios (id_barrio,n_barrio) VALUES (6,'Guemes')
```

```
INSERT INTO barrios (id_barrio,n_barrio) VALUES (7,'Ipona')
INSERT INTO barrios (id_barrio,n_barrio) VALUES (8,'Alta Córdoba')
INSERT INTO barrios (id_barrio,n_barrio) VALUES (9,'General Paz')

INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (1,'Parcial 1')
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (2,'Parcial 2')
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (3,'Final')
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (4,'Trabajo Practico')

INSERT INTO tipo_documentos (id_tipo_documento,n_tipo_documento) VALUES (1,'DNI')
INSERT INTO tipo_documentos (id_tipo_documento,n_tipo_documento) VALUES (2,'LC')

INSERT INTO carreras (id_carrera,nombre_carrera,titulo) VALUES (1,'Ingeniería en Sistemas de Información','Ingeniero en Sistemas de Informacion')

INSERT INTO sexos (id_sexo,n_sexo) VALUES (1,'Masculino')
INSERT INTO sexos (id_sexo,n_sexo) VALUES (2,'Femenino')

INSERT INTO estados_cursados (id_estado,n_estado) VALUES (1,'Inscripto')
INSERT INTO estados_cursados (id_estado,n_estado) VALUES (2,'Regular')
INSERT INTO estados_cursados (id_estado,n_estado) VALUES (3,'Libre')

INSERT INTO materias (id_materia,nombre) VALUES (1,'Algoritmos y Estructuras de Datos')
INSERT INTO materias (id_materia,nombre) VALUES (2,'Análisis Matemático I')
INSERT INTO materias (id_materia,nombre) VALUES (3,'Análisis Matemático II')
INSERT INTO materias (id_materia,nombre) VALUES (4,'Algebra y Geometría Analítica')
INSERT INTO materias (id_materia,nombre) VALUES (5,'Matemática Discreta')
INSERT INTO materias (id_materia,nombre) VALUES (6,'Sistemas y Organizaciones')
INSERT INTO materias (id_materia,nombre) VALUES (7,'Ingeniería y Sociedad')
INSERT INTO materias (id_materia,nombre) VALUES (8,'Paradigmas de Programación')

INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (1,1,1,1)
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (2,1,1,1)
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (3,1,1,1)
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (4,1,1,1)
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (5,1,1,2)
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (6,1,1,2)
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (7,1,1,2)
```

```
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (8,1,1,2)

INSERT INTO estudiantes (id_estudiante,apellido,nombres,id_tipo_documento,documento,id_sexo,calle,calle_numero,id_barrio) VALUES (1,'Abrutsky','Maximiliano Adrián',1,23852963,1,'Av. Colón','1035',1)
INSERT INTO estudiantes (id_estudiante,apellido,nombres,id_tipo_documento,documento,id_sexo,calle,calle_numero,id_barrio) VALUES (2,'DAMIANO','Luis',1,19147258,1,'Dean Funes','903',1)

INSERT INTO inscripciones_a_carreras (id_estudiante,id_carrera,fecha) VALUES (1,1,CONVERT(datetime, '26/01/2009', 103))

INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (1,1,1,CONVERT(datetime, '01/02/2009', 103),1,CONVERT(datetime, '01/02/2009', 103))
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (2,1,1,CONVERT(datetime, '01/02/2009', 103),1,CONVERT(datetime, '01/02/2009', 103))
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (3,1,1,CONVERT(datetime, '01/02/2009', 103),1,CONVERT(datetime, '01/02/2009', 103))
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_inscripcion,id_estado,fecha_estado) VALUES (4,1,1,CONVERT(datetime, '01/02/2009', 103),1,CONVERT(datetime, '01/02/2009', 103))

INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_inscripcion,fecha_nota,id_tipo_nota,nota) VALUES (1,1,1,CONVERT(datetime, '01/02/2009', 103),CONVERT(datetime, '15/03/2009', 103),1,5)
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_inscripcion,fecha_nota,id_tipo_nota,nota) VALUES (1,1,1,CONVERT(datetime, '01/02/2009', 103),CONVERT(datetime, '26/05/2009', 103),2,7)
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_inscripcion,fecha_nota,id_tipo_nota,nota) VALUES (2,1,1,CONVERT(datetime, '01/02/2009', 103),CONVERT(datetime, '18/03/2009', 103),1,3)
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_inscripcion,fecha_nota,id_tipo_nota,nota) VALUES (2,1,1,CONVERT(datetime, '01/02/2009', 103),CONVERT(datetime, '29/05/2009', 103),2,4)
```

#### 11.4.3.2 Función Escalar definida por el usuario

La característica ilustrativa de esta función es recibir por parámetro el `id_estudiante` y devolver el nombre completo del estudiante con el formato ‘Apellido, nombres’. Nótese que solamente la primera letra del apellido será mayúscula y todas las del/los nombre/s en minúscula.

La sentencia de creación está comentada para facilitar la comprensión. Asimismo, el funcionamiento consiste en declarar una variable, asignarle como valor la primera letra del apellido en mayúsculas, concatenándole el resto de las letras en minúsculas, una coma con espacio en blanco y, finalmente, el o los nombres, todo en minúsculas. Se concluye cuando se retorna, como resultado, dicha variable.

```

CREATE FUNCTION obtener_Nombre_Completo(@id_estudiante INT) – cabecera de la función (nombre y parámetros)
RETURNS VARCHAR(150) – tipo de valor que retorna
AS
BEGIN
    DECLARE @nombreCompleto VARCHAR(150) – definición de variable que será retornada

    -- seteo de valor de la variable. UPPER y LOWER convierten a mayús. o minús. respectivamente. LEFT y RIGHT
    recortan la cadena retornando el lado izquiero . derecho respectivamente. LEN devuelve el largo de la cadena
    SELECT @nombreCompleto = UPPER(LEFT(apellido, 1)) + LOWER(RIGHT(apellido, LEN(apellido)-1))
        +
        +
        + LOWER(nombres)

    FROM estudiantes
    WHERE id_estudiante = @id_estudiante

    RETURN @nombreCompleto – retorno del valor
END

```

Esta función, como cualquier otra que también sea escalar, retornará solo un valor, por lo que pueden invocarse cuando se utilizan expresiones escalares (en el SELECT, WHERE, SET, etc.), o bien utilizando la instrucción EXECUTE / EXEC.

Ejemplos:

Invocación:

```

SELECT id_estudiante, apellido, nombres, dbo.obtener_Nombre_Completo(id_estudiante)
as NombreCompleto
FROM estudiantes

```

Resultado:

| id_estudiante | apellido | nombres            | NombreCompleto        |
|---------------|----------|--------------------|-----------------------|
| 1<br>adrián   | Abrutsky | Maximiliano Adrián | Abrutsky, maximiliano |
| 2             | DAMIANO  | Luis               | Damiano, luis         |

Invocación:

```

SELECT dbo.obtener_Nombre_Completo(2) as NombreCompleto

```

Resultado:

NombreCompleto

Damiano, luis

#### 11.4.3.3 Procedimiento almacenado definido por el usuario

La funcionalidad de este procedimiento consistirá en recibir por parámetro un estudiante, la materia y carrera (`id_estudiante`, `id_materia` e `id_carrera`) y actualizar el estado de cursado (tabla `alumnos_cursando_materias`) de dicha materia-carrera, por ejemplo, de inscripto a regular o libre acorde con si tiene, o no, al menos dos parciales aprobados (tabla `alumnos_cursando_notas`).

Consideraciones:

- Un alumno puede recursar una materia, por lo que se toma la última inscripción.
- La nota de aprobación es mayor o igual a cuatro.
- Al menos debe tener dos parciales rendidos para que se actualice el estado.
- Si se tiene al menos dos parciales aprobados, se lo considera regular.
- Menos de dos parciales aprobados, se lo considera libre.
- Por sencillez, no se consideran los recuperatorios.

```

CREATE PROCEDURE actualizar_estado_materia @id_estudiante INT, @id_materia INT, @id_carrera INT
AS
DECLARE @fecha_incripcion_cursado DATETIME

-- Obtención de la última inscripción de la materia (podría estar recursando y solo se procesa la última inscripción)
SELECT @fecha_incripcion_cursado = MAX(fecha_incripcion)
FROM alumnos_cursando_materias
WHERE id_materia = @id_materia
AND id_estudiante = @id_estudiante

-- Si no se obtiene ninguna fecha, significa que el alumno nunca se inscribió a la materia indicada.
IF @fecha_incripcion_cursado IS NULL
    PRINT 'El alumno no se ha inscripto a la materia – carrera especificada'
ELSE
BEGIN
    -- declaración de variables
DECLARE @cant_parciales INT
DECLARE @cant_parciales_aprobados INT
DECLARE @cant_parciales_reprobados INT

    -- cantidad de parciales rendidos (notas cuya descripción contengan 'parcial' sin importar mayúsculas y minúsculas)
    SELECT @cant_parciales = COUNT(*)
    FROM alumnos_cursando_notas
        INNER JOIN materias mat ON notas.id_materia = mat.id_materia

```

```

    INNER JOIN tipo_notas tip ON notas.id_tipo_nota = tip.id_tipo_nota
    WHERE notas.id_estudiante = @id_estudiante
    AND   notas.id_materia = @id_materia
AND   notas.id_carrera = @id_carrera
    AND   notas.fecha_inscripcion = @fecha_inscripcion_cursado
    AND   LOWER(tip.n_tipo_nota) LIKE '%parcial%'

-- cantidad de parciales aprobados (rendidos con nota mayor o igual a 4)
SELECT @cant_parciales_aprobados = COUNT(*)
FROM alumnos_cursando_notas notas
    INNER JOIN materias mat ON notas.id_materia = mat.id_materia
    INNER JOIN tipo_notas tip ON notas.id_tipo_nota = tip.id_tipo_nota
WHERE notas.id_estudiante = @id_estudiante
AND   notas.id_materia = @id_materia
AND   notas.id_carrera = @id_carrera
    AND   notas.fecha_inscripcion = @fecha_inscripcion_cursado
    AND   LOWER(tip.n_tipo_nota) LIKE '%parcial%'
    AND   notas.nota >= 4

-- a modo informativo se calculan los reprobados (parcial que no se aprobó, se reprobó)
SET @cant_parciales_reprobados = @cant_parciales - @cant_parciales_aprobados

PRINT 'Cantidad de parciales rendidos = ' + CAST(@cant_parciales AS CHAR)
PRINT 'Cantidad de parciales aprobados = ' + CAST(@cant_parciales_aprobados AS CHAR)
PRINT 'Cantidad de parciales reprobados = ' + CAST(@cant_parciales_reprobados AS CHAR)

IF @cant_parciales < 2
    PRINT 'La cantidad rendida de parciales es menor a 2, no se cambia el estado'
ELSE IF @cant_parciales_aprobados >= 2
    BEGIN
        PRINT '2 parciales aprobados, REGULAR'
        -- actualización al estado REGULAR
        UPDATE alumnos_cursando_materias
        SET id_estado = (SELECT id_estado FROM estados_cursados WHERE n_estado LIKE 'Regu-
lar')
        WHERE id_estudiante = @id_estudiante
        AND   id_materia = @id_materia
AND   id_carrera = @id_carrera
    
```

```
        AND fecha_incripcion = @fecha_incripcion_cursado
    END
ELSE
BEGIN
    PRINT 'Menos de 2 parciales aprobados, LIBRE'
    -- actualización al estado LIBRE
    UPDATE alumnos_cursando_materias
    SET id_estado = (SELECT id_estado FROM estados_cursados WHERE n_estado LIKE 'Libre')
    WHERE id_estudiante = @id_estudiante
    AND id_materia = @id_materia
AND id_carrera = @id_carrera
    AND fecha_incripcion = @fecha_incripcion_cursado
END
END
```

Ejemplos:

Invocación:

EXEC actualizar\_estado\_materia 1,8,1

Resultado:

El alumno no se ha inscripto a la materia – carrera especificada

Invocación:

EXEC actualizar\_estado\_materia 1,2,1

Resultado:

Cantidad de parciales rendidos = 2

Cantidad de parciales aprobados = 1

Cantidad de parciales reprobados = 1

Menos de 2 parciales aprobados, LIBRE

Invocación:

EXEC actualizar\_estado\_materia 1,1,1

Resultado:

Cantidad de parciales rendidos = 2

Cantidad de parciales aprobados = 2

Cantidad de parciales reprobados = 0

2 parciales aprobados, REGULAR



## 11.5 Contenido de la página Web de apoyo

El material marcado con asterisco (\*) solo está disponible para docentes.

11.5.1 Mapa conceptual del capítulo

11.5.2 Autoevaluación

11.5.3 Presentaciones\*

# 12

## Implementando el modelo en MySQL 5.1

### Contenido

|  |     |
|--|-----|
| 12.1 Introducción.....   | 336 |
| 12.2 Inicializando el servidor.....  | 336 |
| 12.3 Inicializando el cliente.....   | 342 |
| 12.4 Creando y probando nuestro modelo de<br>Estudiante - Universidad..... | 343 |
| 12.5 Contenido de la página Web de apoyo.....                              | 358 |

### Objetivos

- Saber dónde conseguir el instalador de MySQL.
- Comprender los tipos y el proceso de instalación en general, y poder realizar la configuración básica de la instancia.
- Adquirir los conocimientos para inicializar el servidor de bases de datos y la herramienta cliente.
- Utilizar el MySQL Workbench para crear la base de datos, todas las tablas y relaciones del modelo visto en el libro y los objetos de programación de la base de datos.



En la Web de apoyo, encontrará el vínculo a la página Web que le brindará mayor información acerca del producto MySQL 5.1.

## 12.1 Introducción

---

MySQL surgió en una empresa sueca MySQL AB, en la década del noventa. En la actualidad, la empresa es subsidiada por Sun Microsystems de Oracle Corporation. MySQL es, sin duda, uno de los Sistemas de Gestión de Bases de Datos Relacionales (SGBDR) open source más difundido y utilizado. Se puede obtener bajo la licencia GNU GPL (Software Libre) o mediante la distribución comercial, que se diferencia en el soporte, las herramientas de monitoreo y en la posibilidad de incorporarlo en productos privativos. En este capítulo se utilizará la versión libre.

En sus orígenes, este producto carecía de elementos básicos de un SGBDR como la integridad referencial y las transacciones, que se fueron incorporando a medida de que el proyecto logró una mayor difusión mundial. En la actualidad, existen millones de instalaciones impulsadas por el auge del open source y el conjunto de herramientas de desarrollo LAMP: Linux – Apache – MySQL – PHP/Perl/Python.

## 12.2 Inicializando el servidor

---

La instalación del servidor es la actividad inicial que se realizará; luego, se establecerán conexiones desde los clientes, ya sean locales o remotos, para interactuar con el servidor de bases de datos.

### 12.2.1 Download

El software libre implica la posibilidad de obtener una distribución sin costo del producto. MySQL puede ejecutarse en diferentes sistemas operativos, como las variadas distribuciones de Linux, Mac OS X, Sun Solaris, IBM AIX, MS Windows, etcétera.

Se descargará la distribución indicada a su plataforma. Dentro de las posibilidades para el sistema operativo Windows, se puede optar por un paquete de instalación MSI de Windows Installer, que lo guiará, paso a paso, con un asistente de instalación, o bien por el archivo comprimido en ZIP, en el que el usuario será el encargado de descomprimirlo y de la configuración manual. Las dos opciones se descargarán completas o solo con los componentes esenciales de MySQL. Esto excluye asistentes de configuración, librerías para desarrolladores y otras opciones. Todas las alternativas se encuentran en el siguiente enlace:

<http://dev.mysql.com/downloads/mysql/>

Como este enlace podría estar obsoleto en el momento de la lectura de esta obra, su búsqueda se debería ejecutar utilizando “Download MySQL Community Server”.

En este tutorial, se utilizará la distribución para Windows, con instalador y versión completa (no esencial) “Windows (x86, 32-bit), MSI Installer”; el nombre del archivo es “mysql-5.1.44-win32.msi”.

## 12.2.2 Instalación

La instalación en un sistema operativo Windows puede ser Windows 2000, Windows X P, Windows Server 2003 y Windows Server 2008. Son soportadas tanto las versiones en 32 como en 64 bits.

Como requisito mínimo, se recomienda unos 200 mb de espacio libre en el disco duro; es necesario el soporte TCP/IP. Es recomendable ejecutar la instalación con una cuenta con privilegios de administrador.

Para MySQL, también se cuenta con la posibilidad de ejecutar el paquete de instalación de forma silenciosa, utilizando un archivo de configuración con todos los parámetros necesarios. La otra alternativa, que es la recomendable para usuarios inexpertos, es través del asistente guía.

### 12.2.2.1 Tipos de instalación

Al ejecutarse el paquete de instalación de MySQL Server 5.1, se inicia el asistente con una pantalla de bienvenida; luego, se solicita que se seleccione el tipo de instalación que se realizará: típica, completa o personalizada, tal como se aprecia en la Fig. 12.1.

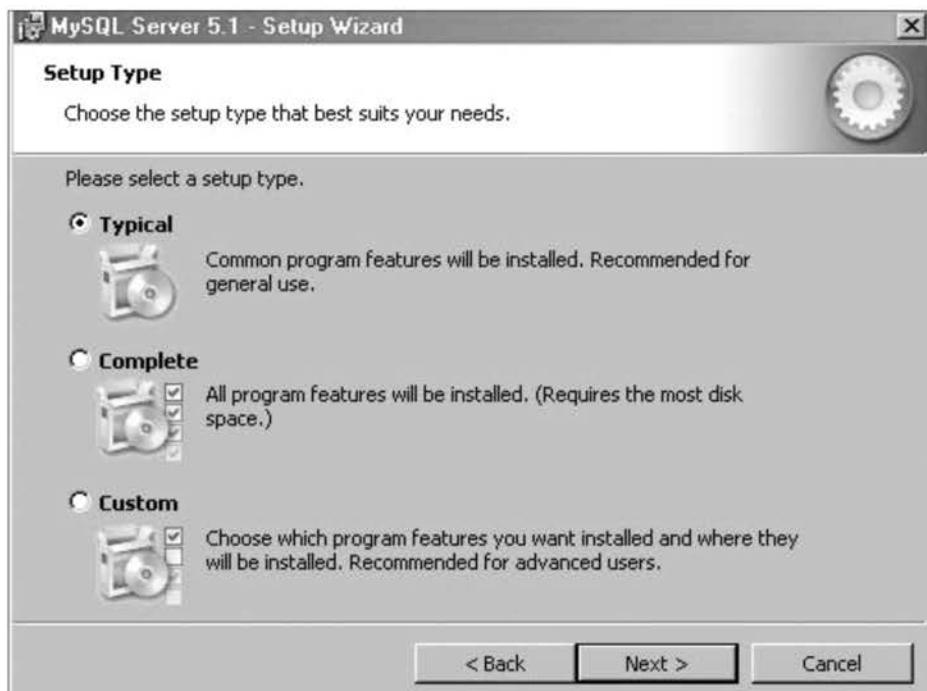


Fig. 12.1 Tipo de instalación.

La instalación típica consta del servidor MySQL, el cliente por línea de comando para interactuar con el servidor y las utilidades no gráficas para la administración del servidor y de sus bases de datos. La completa incluye todos los componentes

scripts de soporte, documentación, librería de servidor embebido, etcétera. El último tipo de instalación es la personalizada, que brinda la flexibilidad de definir los componentes a instalarse acorde con las necesidades particulares.

Se seleccionará la instalación completa para continuar con los siguientes pasos.

Más adelante, se presentará un resumen previo de la instalación que se realizará y, tras confirmarla, mostrará el progreso actual. Luego de completarse, se publicitarán las ventajas de MySQL Enterprise (distribución comercial paga) y se finalizará el asistente con la posibilidad de configurar el servidor recientemente instalado y, también, de registrar el producto con el fin de recibir notificaciones de actualizaciones, ofertas, etcétera (Fig. 12.2).



Fig. 12.2 Finalización del asistente de instalación del servidor MySQL.

Se seleccionará la configuración del servidor. En el siguiente apartado, se analizarán los aspectos importantes.

#### 12.2.2.2 Configurando la instancia. Opciones de Servicio Windows

Cabe recordar el concepto de instancia explicado para otro motor. Una instancia es, básicamente, una instalación del motor de bases de datos, que en el caso de MySQL se podrá implementar como un servicio de Windows, tal como se verá en este apartado. Es decir que, una instancia, sería un servidor lógico, puesto que pueden instalarse varias instancias en un equipo o servidor físico. Cada instancia o servidor lógico contendrá varias bases de datos.

Antes de definir si la instancia se implementará como un servicio del sistema operativo, el asistente de configuración brinda la posibilidad de optar por el modo detallado o el básico (Fig. 12.3).

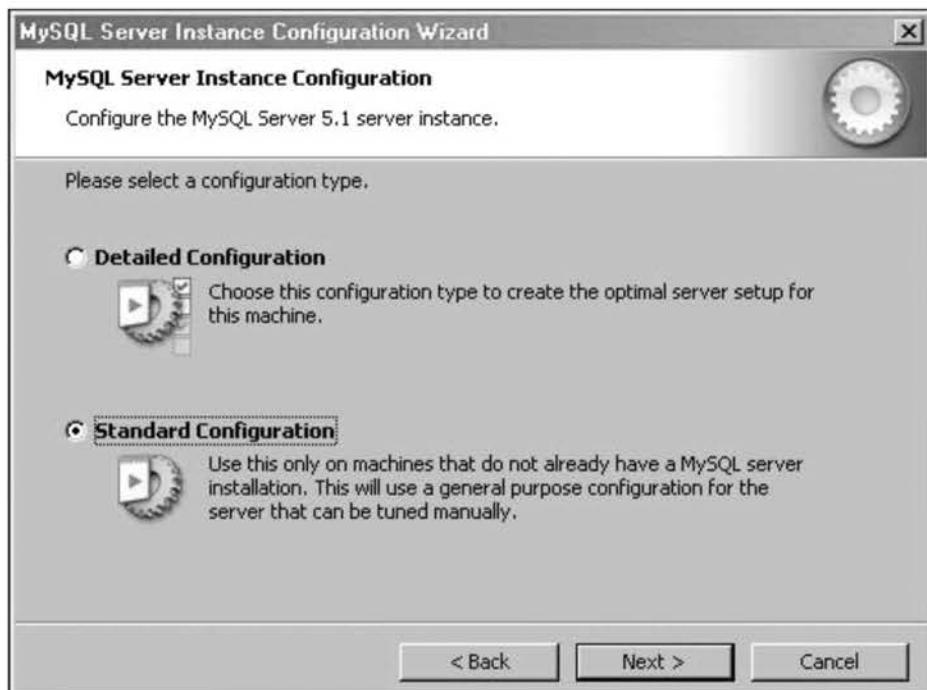


Fig. 12.3 Tipo de configuración de la instancia.

El modo básico se orienta a nuevos usuarios para que puedan, de manera fácil y rápida, poner en funcionamiento el servidor MySQL, sin algunas decisiones de configuración que sí se deben especificar en el modo detallado. Se optará por la vía más sencilla, en este caso, solo se definirán las opciones de servicio y de seguridad.

Continuando con el modo básico de configuración, se prosigue con la definición de instalar la instancia como un servicio (Fig. 12.4).

Un servicio, dentro del sistema operativo Windows, es un proceso que se ejecuta en segundo plano, que se puede iniciar automáticamente en el momento en que, también, se inicia el sistema. Para su configuración, hay que chequear la opción "Iniciar el servidor MySQL automáticamente". El nombre por defecto del servicio es MySQL, pero el usuario puede modificarlo. Si se realizaran instalaciones de más de una instancia, tendrá cada una su nombre de servicio propio.



Fig. 12.4 Opción de servicio en Windows.

#### 12.2.2.3 Configuración de la instancia. Opciones de seguridad

Durante el proceso de configuración, el usuario root tiene todos los privilegios en el servidor. Se puede especificar su contraseña y, también, crear una cuenta de acceso anónima, que no es lo recomendado porque disminuiría la seguridad.

La Fig. 12.5 es la pantalla en la que se define lo mencionado en el párrafo anterior. Cabe destacar que difiere si se trata de la primera instalación o si ya existía otra instancia MySQL ejecutándose en el equipo.

Por aspectos de seguridad, es altamente recomendable definir una contraseña para root, que se recordará para una posterior autenticación contra el servidor. Por defecto, el usuario root solo se puede logear localmente en el servidor. Para permitir que lo pueda hacer remotamente desde otros equipos, se tiene que chequear dicha opción.

Para finalizar, se mostrará un resumen de los pasos que realizará el asistente y, luego de que se confirme la ejecución, se visualizará el progreso de la instalación hasta que termine.

Se podrá ejecutar nuevamente el asistente de configuración, ya sea para reconfiguración de la instancia, o bien para removerla. Junto con el acceso directo, se creará el cliente por la línea de comando (Fig. 12.6).



Fig. 12.5 Seguridad de la instancia.

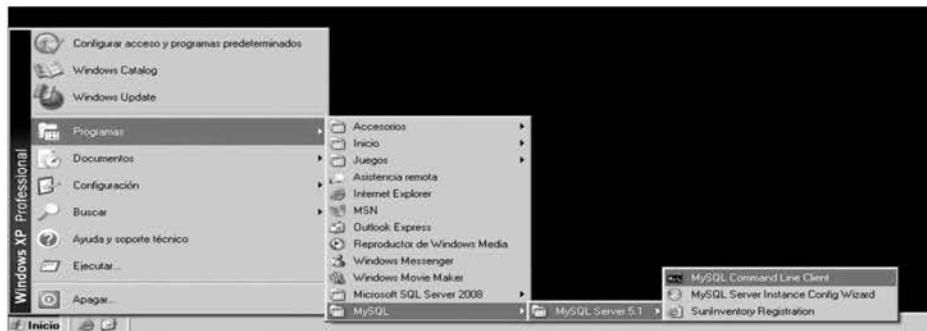


Fig. 12.6 Grupo de programas MySQL Server 5.1.

### 12.2.3 Inicialización

Si MySQL se instaló como servicio para que se inicie automáticamente, éste debería estar inicializado (ejecutándose) a la espera de conexiones. Se puede verificar o modificar el estado desde "Servicios" perteneciente al grupo de "Herramientas administrativas" de Windows (Fig. 12.7).



Fig. 12.7 Herramientas Administrativas -&gt; Servicios.

## 12.3 Inicializando el cliente

Luego de la puesta en marcha del servidor, el siguiente paso es la instalación de MySQL Workbench —que es la herramienta gráfica más reciente y de libre distribución— para que interactúe con el servidor MySQL. No es obligación instalar esta herramienta cliente, puesto que el servidor ya se encuentra funcionando y se podría administrar a través de sentencias que utilicen la línea de comando, como así también las aplicaciones desarrolladas podrían operar contra el motor de bases de datos. Se optará por su utilización debido a la sencillez operativa brindada por el entorno visual de la herramienta.

MySQL Workbench consta de tres componentes principales:

- Database Design & Modeling: diseño y modelado de bases de datos.
- SQL Development: programación en SQL, realización de consultas, ejecución de *scripts*, creación de procedimientos almacenados, etc.
- Database Administration: realización de las tareas inherentes a la administración del servidor, como la implementación de acceso y de seguridad, las políticas de *backup*, los controles de rendimiento, entre otros.

Workbench reemplaza a las obsoletas MySQL GUI Tools, compuestas por MySQL Administrator, Query Browser, Migration Tool y System Tray Monitor. Se recomienda no utilizarlas porque su mantenimiento y soporte están discontinuados.

### 12.3.1 Download

La descarga se puede hacer libremente desde el siguiente *link*, válido en el momento de la publicación de este libro (es conveniente que se seleccione una versión 5.2 o superior):

<http://dev.mysql.com/downloads/workbench/5.2.html>

En caso de que el enlace se encuentre obsoleto en el momento de la lectura, la búsqueda se realizará utilizando “Download MySQL Workbench”.

### 12.3.2 Instalación

El proceso de instalación es muy sencillo: si se seleccionó un paquete de instalación MSI, un asistente guiará la tarea. Solo se especificará si se desea una instalación completa que incluya todos los componentes, o bien una personalizada. En el caso

que se hubiese optado por descargar la herramienta para ser utilizada sin instalador, se deberá descomprimir el archivo ZIP en el directorio de destino.

### 12.3.3 Utilizando el MySQL Workbench

Para utilizar esta herramienta, hay que dirigirse al menú Inicio y seleccionar Todos los programas; luego, MySQL. A continuación, se hará clic en MySQL Workbench 5.2 OSS.

En la Fig. 12.8, se muestra la solapa inicial, desde la que se puede acceder a las funcionalidades principales mencionadas con anterioridad: Desarrollo, Modelado y Administración (se señalan con óvalos).

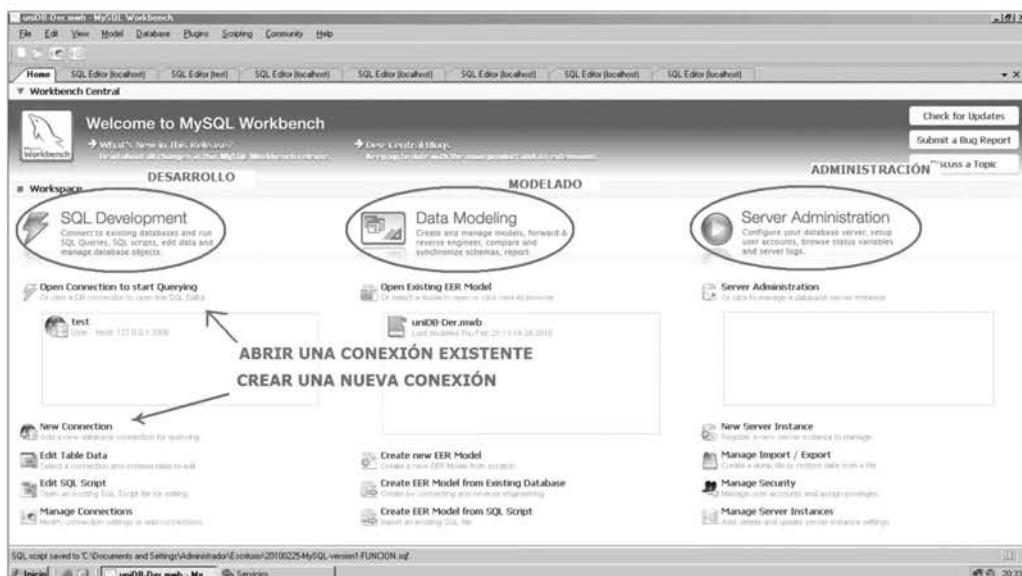


Fig. 12.8 Solapa inicial de MySQL Workbench.

## 12.4 Creando y probando nuestro modelo de Estudiante – Universidad

El lector ya está en condiciones de interactuar con MySQL, esto es: el servidor se está ejecutando con el entorno gráfico MySQL Workbench para realizar las tareas administrativas y de programación de forma amigable.

Como primer paso, se procederá a establecer una conexión al servidor desde MySQL Workbench; luego, se creará la base de datos y todas las tablas utilizando el script (conjunto de sentencias que se ejecutan de forma conjunta).



Para mayor información sobre el uso de MySQL Workbench y de todas sus funcionalidades, se sugiere la biblioteca de documentación oficial:  
<http://dev.mysql.com/doc/workbench/en/index.html>

### 12.4.1 Creación de una conexión al servidor

Tanto la creación de la conexión como las demás actividades se llevarán a cabo con el MySQL Workbench.

A la acción de crear de una conexión, como en tantas otras tareas frecuentes, se puede acceder desde la solapa inicial de MySQL Workbench, tal como se indica en la Fig. 12.8. La opción que se seleccionará es “new connection”, dentro del grupo de tareas frecuentes de “SQL Development”. Se desplegará el administrador de conexiones para crear esta nueva conexión al servidor MySQL (Fig. 12.9).

Hay tres métodos de conexión: TCP/IP estándar y sobre SSH (intérprete de órdenes seguras) o por medio de socket. En el siguiente ejemplo, se utilizará el primer método mencionado (TCP/IP estándar).

Además del nombre de la conexión (“connLocal” en el ejemplo de la Fig. 12.9), para el método TCP/IP, es necesario que se especifiquen el nombre de red o la dirección IP (hostname) y el puerto (port) en el que se ejecuta el servidor MySQL. En el caso de que MySQL Workbench se ejecute en el mismo equipo servidor, hay que especificarle localhost, o bien la dirección IP 127.0.0.1. El puerto por defecto es el 3306.

Luego se configurará el usuario para la conexión. Se utilizará ‘root’ de la misma manera en que se explicitó durante la instalación, ya que es el que tiene todos los privilegios sobre el servidor. Es opcional la definición del esquema (base de datos) al que se conecta por defecto. En el siguiente ítem, se creará una base de datos uniDB.

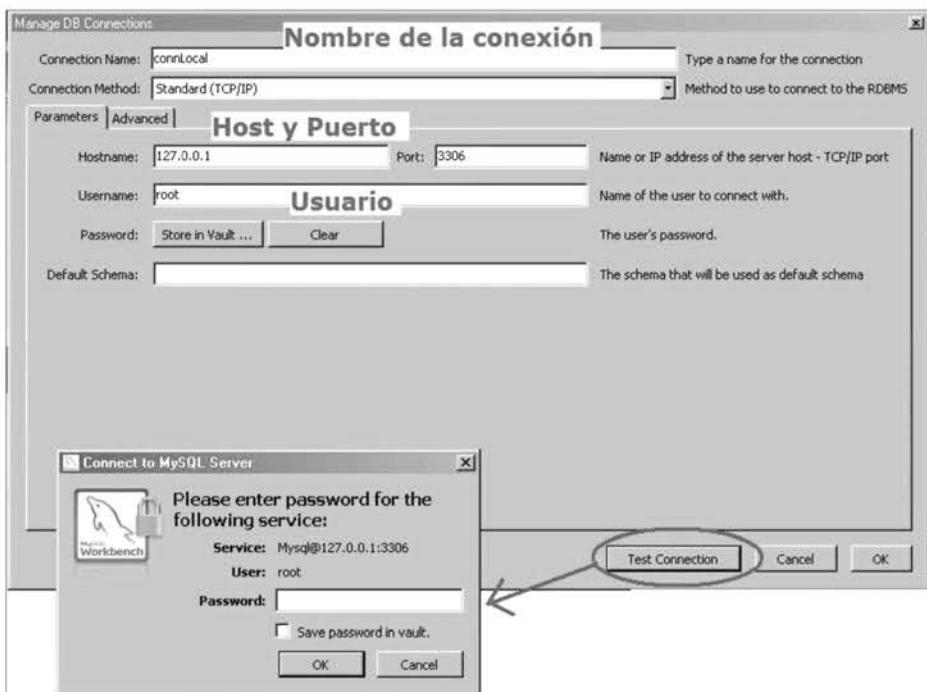


Fig. 12.9 Creación de conexión MySQL Workbench.

Antes de concluir con la conexión, se sugiere que se realice el test para verificar que los parámetros definidos sean los correctos. Esto se logra si se presiona “Test Connection”, que abrirá un mensaje de diálogo que solicitará la clave para el usuario especificado, en este caso, root, cuyo password se definió en la instalación del servidor. Se puede almacenar la clave, o bien ignorar esta opción y tipearla cada vez que se solicite.

## 12.4.2 Creación de la base de datos, las tablas y sus relaciones

### 12.4.2.1 Ejecución de script

Lo primero que se creará es la base de datos o esquema, luego se crearán las tablas correspondientes y sus respectivas claves para asegurar la integridad de los datos. Finalmente, se introducirá una función y un procedimiento almacenado a modo de ejemplo de programación.

Una aclaración importante: en MySQL, el concepto de esquema es sinónimo de base de datos y, normalmente, se usan estos términos indistintamente, tal es así que la sentencia de creación puede invocarse como CREATE DATABASE o CREATE SCHEMA. Esto es de suma relevancia, puesto que, en otros motores de bases de datos, el concepto es otro. En ellos, un esquema es una agrupación o un contenedor de varios objetos que existen por ventajas administrativas y de manejo de seguridad; una base de datos en SQL Server u Oracle podría tener varios esquemas.

Referencia para la interpretación del script: las primeras sentencias son las de creación de tablas con sus claves primarias y luego las de modificación de las tablas recientemente creadas para incorporar las claves foráneas; finalmente, la inserción de datos de prueba y de creación de la función y del procedimiento almacenado. El motivo en que se fundamenta la decisión de modificar las tablas para incorporar las claves foráneas (FK) y no crearlas junto con la definición de cada tabla es que ambas tablas relacionadas por dicha clave deben existir en el momento en que ésta se crea. De esta manera, se evitan complicaciones originadas por esta causa.

Para ejecutar las sentencias DDL que crearán la base de datos y los objetos del modelo, se abrirá una conexión desde “Open Connection to start Querying” (Fig. 12.8). Una ventana de diálogo solicitará que se seleccione una conexión (se utilizará la creada recientemente, ‘connLocal’, según el ejemplo) y, posteriormente, si es que no se hace uso de la opción de recordar claves, se pedirá el ingreso de la clave del usuario de dicha conexión (‘root’, según el ejemplo).

Posterior a copiar las sentencias se proseguirá con la ejecución de las mismas, para ello se hace uso de un botón destinado a tal fin, señalado en la próxima imagen (Fig. 12.10), notar que a través del mismo, se ejecutarán todas las sentencias, a diferencia con el botón que se encuentra a su derecha, que solo ejecutará la sentencia seleccionada.

Los resultados de la ejecución se visualizan por la consola de salida (Fig. 12.11). Se podrá navegar entre los objetos de la base de datos desde el explorador; en el caso de las tablas, hay que desplegar la base de datos y, luego, el grupo “Tables”. Es necesario actualizar el explorador (Refresh all) una vez finalizadas todas las sentencias.

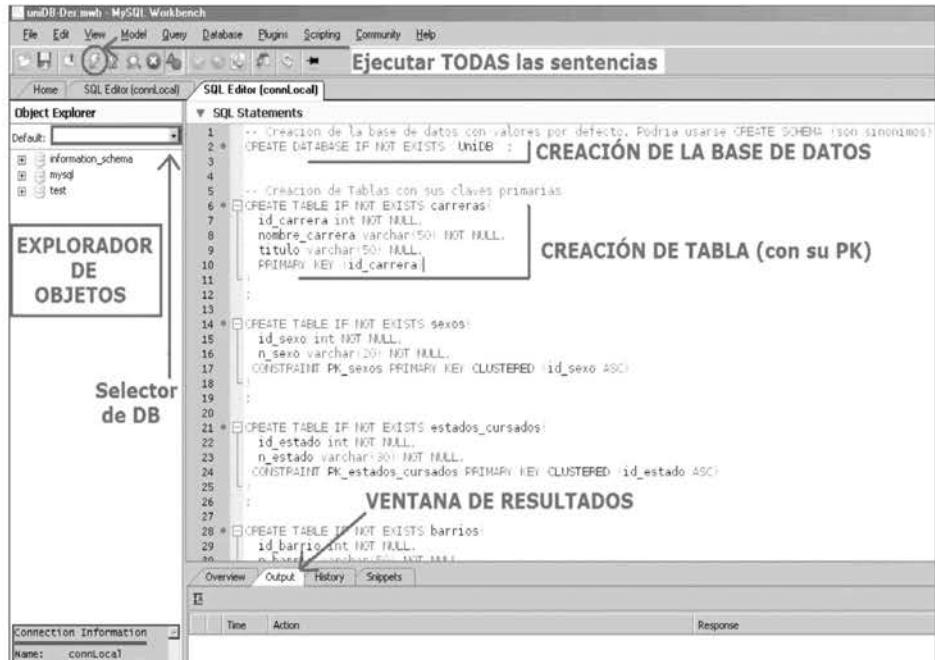


Fig. 12.10 Ejecución de sentencias.

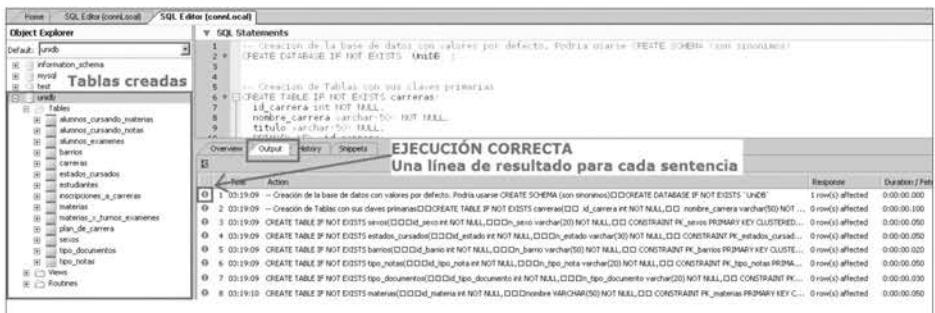


Fig. 12.11 Ejecución correcta.

#### 12.4.2.2 Modelo de datos

MySQL Workbench incorpora la funcionalidad de modelado de datos, tal como se mencionó en sus características y como se puede observar en la Fig. 12.8. El modelo de datos utilizado es el que se aprecia en el diagrama a continuación (Fig.12.12) y que se generó, prácticamente, de forma automática por la herramienta en cuestión. Queda a interés del lector profundizar acerca de su funcionalidad y utilización.

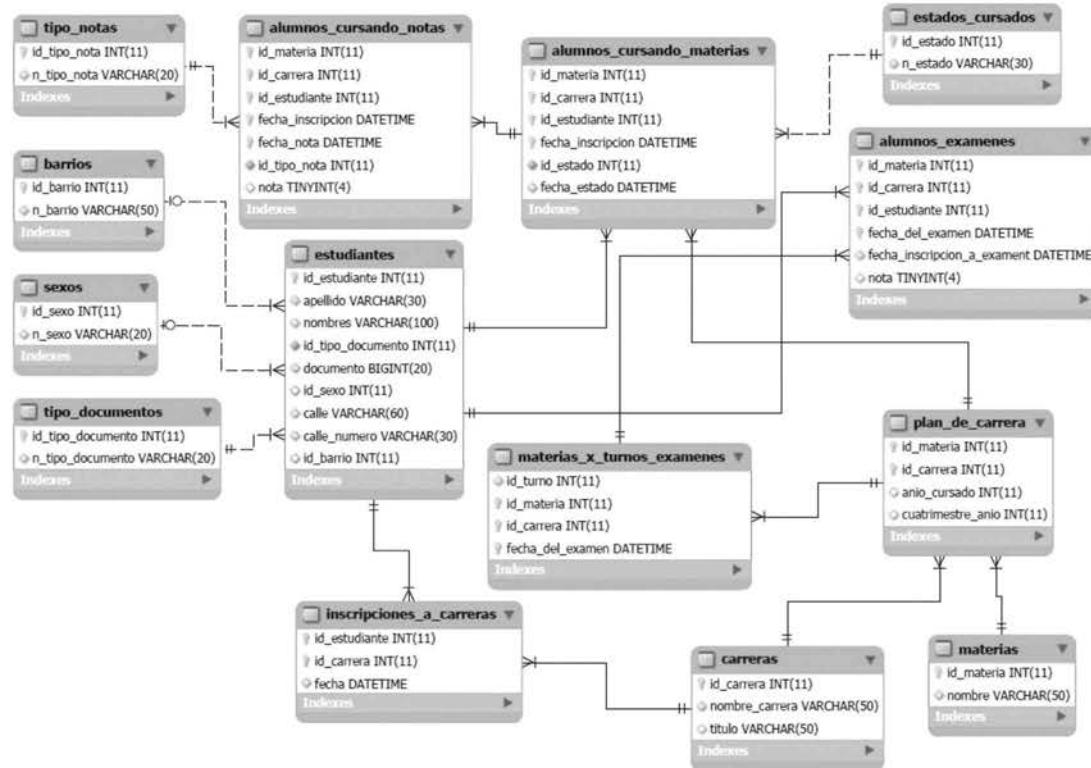


Fig. 12.12 Modelo de datos completo.

#### 12.4.2.3 Script (sentencias DDL)

A continuación, se precisan las sentencias de definición de datos que se ejecutarán para crear el modelo completo de la Fig. 12.12. Las sentencias son exactamente las mismas (ANSI) que para SQL Server, a excepción de la primera sentencia que crea la base de datos que, en SQL Server, se hizo uso de su herramienta visual para dicha acción. El separador de instrucción es ‘;’.

```
-- Creación de la base de datos con configuración por defecto. Podría haberse usado CREATE SCHEMA

CREATE DATABASE IF NOT EXISTS `UniDB`;
USE `UniDB`;

-- Creación de tablas con sus claves primarias
CREATE TABLE IF NOT EXISTS carreras(
    id_carrera int NOT NULL,
```

```
nombre_carrera varchar(50) NOT NULL,
    titulo varchar(50) NULL,
CONSTRAINT PK_carreras PRIMARY KEY CLUSTERED (id_carrera ASC)
)
;

CREATE TABLE IF NOT EXISTS sexos(
    id_sexo int NOT NULL,
    n_sexo varchar(20) NOT NULL,
CONSTRAINT PK_sexos PRIMARY KEY CLUSTERED (id_sexo ASC)
)
;

CREATE TABLE IF NOT EXISTS estados_cursados(
    id_estado int NOT NULL,
    n_estado varchar(30) NOT NULL,
CONSTRAINT PK_estados_cursados PRIMARY KEY CLUSTERED (id_estado ASC)
)
;

CREATE TABLE IF NOT EXISTS barrios(
    id_barrio int NOT NULL,
    n_barrio varchar(50) NOT NULL,
CONSTRAINT PK_barrios PRIMARY KEY CLUSTERED (id_barrio ASC)
)
;

CREATE TABLE IF NOT EXISTS tipo_notas(
    id_tipo_nota int NOT NULL,
    n_tipo_nota varchar(20) NOT NULL,
CONSTRAINT PK_tipo_notas PRIMARY KEY CLUSTERED (id_tipo_nota ASC)
)
;

CREATE TABLE IF NOT EXISTS tipo_documentos(
    id_tipo_documento int NOT NULL,
    n_tipo_documento varchar(20) NOT NULL,
CONSTRAINT PK_tipo_documentos PRIMARY KEY CLUSTERED (id_tipo_documento ASC)
)
;
```

```
CREATE TABLE IF NOT EXISTS materias(
    id_materia int NOT NULL,
    nombre VARCHAR(50) NOT NULL,
    CONSTRAINT PK_materias PRIMARY KEY CLUSTERED (id_materia ASC)
)
;

CREATE TABLE IF NOT EXISTS alumnos_cursando_materias(
    id_materia int NOT NULL,
    id_carrera int NOT NULL,
    id_estudiante int NOT NULL,
    fecha_inscripcion datetime NOT NULL,
    id_estado int NOT NULL,
    fecha_estado datetime NOT NULL,
    CONSTRAINT PK_alumnos_cursando_materias PRIMARY KEY CLUSTERED (id_materia ASC, id_carrera ASC, id_estudiante ASC, fecha_inscripcion ASC)
)
;

CREATE TABLE IF NOT EXISTS alumnos_examenes(
    id_materia int NOT NULL,
    id_carrera int NOT NULL,
    id_estudiante int NOT NULL,
    fecha_del_examen datetime NOT NULL,
    fecha_inscripcion_a_examenes datetime NOT NULL,
    nota tinyint NULL,
    CONSTRAINT PK_alumnos_examenes PRIMARY KEY CLUSTERED (id_materia ASC, id_carrera ASC, id_estudiante ASC, fecha_del_examen ASC)
)
;

CREATE TABLE IF NOT EXISTS alumnos_cursando_notas(
    id_materia int NOT NULL,
    id_carrera int NOT NULL,
    id_estudiante int NOT NULL,
    fecha_inscripcion datetime NOT NULL,
    fecha_nota datetime NOT NULL,
    id_tipo_nota int NOT NULL,
    nota tinyint NULL,
```

```
CONSTRAINT PK_alumnos_cursando_notas PRIMARY KEY CLUSTERED (id_materiaASC,id_carrera ASC,id_estudiante
ASC,fecha_inscripcion ASC,fecha_nota ASC)
)
;

CREATE TABLE IF NOT EXISTS inscripciones_a_carreras(
    id_estudiante int NOT NULL,
    id_carrera int NOT NULL,
    fecha datetime NOT NULL,
    CONSTRAINT PK_inscripciones_a_carreras PRIMARY KEY CLUSTERED (id_estudiante ASC,id_carrera ASC)
)
;

CREATE TABLE IF NOT EXISTS plan_de_carrera(
    id_materia int NOT NULL,
    id_carrera int NOT NULL,
    anio_cursado int NULL,
    cuatrimestre_anio int NULL,
    CONSTRAINT PK_plan_de_carrera PRIMARY KEY CLUSTERED (id_materia ASC,id_carrera ASC)
)
;

CREATE TABLE IF NOT EXISTS estudiantes(
    id_estudiante int NOT NULL,
    apellido varchar(30) NOT NULL,
    nombres varchar(100) NOT NULL,
    id_tipo_documento int NOT NULL,
    documento bigint NOT NULL,
    id_sexo int NULL,
    calle varchar(60) NOT NULL,
    calle_numero varchar(30) NOT NULL,
    id_barrio int NULL,
    CONSTRAINT PK_estudiantes PRIMARY KEY CLUSTERED (id_estudiante ASC)
)
;

CREATE TABLE IF NOT EXISTS materias_x_turnos_examenes(
    id_turno int NOT NULL,
    id_materia int NOT NULL,
    id_carrera int NOT NULL,
```

```
fecha_del_examen datetime NOT NULL,
CONSTRAINT PK_materias_x_turnos_examenes PRIMARY KEY CLUSTERED (id_materia ASC, id_carrera ASC, fecha_del_
examen ASC)
)
;

-- Creación de claves foráneas
ALTERTABLE alumnos_cursando_materias ADD CONSTRAINT FK_alumnos_cursando_materias_estados FOREIGN
KEY(id_estado)
REFERENCES estados_cursados (id_estado)
;
ALTERTABLE alumnos_cursando_materias ADD CONSTRAINT FK_alumnos_cursando_materias_estudiantes FOREIGN
KEY(id_estudiante)
REFERENCES estudiantes (id_estudiante)
;
ALTER TABLE alumnos_cursando_materias ADD CONSTRAINT FK_alumnos_cursando_materias_plan_carrera FOREIGN
KEY(id_materia, id_carrera)
REFERENCES plan_de_carrera (id_materia, id_carrera)
;
ALTERTABLE alumnos_examenes ADD CONSTRAINT FK_alumnos_examenes_estudiantes FOREIGN KEY(id_estudiante)
REFERENCES estudiantes (id_estudiante)
;
ALTER TABLE alumnos_examenes ADD CONSTRAINT FK_alumnos_examenes_materias_x_turnos_examenes FOREIGN
KEY(id_materia, id_carrera, fecha_del_examen)
REFERENCES materias_x_turnos_examenes (id_materia, id_carrera, fecha_del_examen)
;
ALTERTABLE alumnos_cursando_notas ADD CONSTRAINT FK_alumnos_cursando_notas_tipo_notas FOREIGN KEY(id_
tipoNota)
REFERENCES tipo_notas (id_tipoNota)
;
ALTER TABLE alumnos_cursando_notas ADD CONSTRAINT FK_alumnos_cursando_notas_alumnos_cursando_materias FOREIGN
KEY(id_materia, id_carrera, id_estudiante, fecha_incripcion)
REFERENCES alumnos_cursando_materias (id_materia, id_carrera, id_estudiante, fecha_incripcion)
;
ALTERTABLE inscripciones_a_carreras ADD CONSTRAINT FK_inscripciones_a_carreras_carreras FOREIGN KEY(id_carrera)
REFERENCES carreras (id_carrera)
;
ALTERTABLE inscripciones_a_carreras ADD CONSTRAINT FK_inscripciones_a_carreras_estudiantes FOREIGN KEY(id_
estudiante)
REFERENCES estudiantes (id_estudiante)
;
```

```

ALTERTABLE plan_de_carrera ADD CONSTRAINT FK_plan_de_carrera_carreras FOREIGN KEY(id_carrera)
REFERENCES carreras (id_carrera)
;
ALTERTABLE plan_de_carrera ADD CONSTRAINT FK_plan_de_carrera_materias FOREIGN KEY(id_materia)
REFERENCES materias (id_materia)
;
ALTERTABLE estudiantes ADD CONSTRAINT FK_estudiantes_barrios FOREIGN KEY(id_barrio)
REFERENCES barrios (id_barrio)
;
ALTERTABLE estudiantes ADD CONSTRAINT FK_estudiantes_sexos FOREIGN KEY(id_sexo)
REFERENCES sexos (id_sexo)
;
ALTERTABLE estudiantes ADD CONSTRAINT FK_estudiantes_tipo_documentos FOREIGN KEY(id_tipo_documento)
REFERENCES tipo_documentos (id_tipo_documento)
;
ALTERTABLE materias_x_turnos_examenes ADD CONSTRAINT FK_materias_x_turnos_examenes_plan_carrera FOREIGN KEY(id_materia, id_carrera)
REFERENCES plan_de_carrera (id_materia, id_carrera)
;

```

#### 12.4.2.4 Datos de prueba

Se insertará un conjunto de datos con la finalidad de probar la función y el procedimiento almacenado que se programarán en los ítems subsiguientes. Las sentencias de inserción de datos (INSERT) se ejecutarán de forma idéntica a la creación de tablas en el punto anterior. Se Asegurará la selección de la base de datos correcta, previo a la ejecución. (Verificar el combo de selector de DB en la Fig. 12.10 o escribir la sentencia USE 'UniDB' antes del primer INSERT).

```

INSERT INTO barrios (id_barrio,n_barrio) VALUES (1,'Alto Alberdi');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (2,'Yofre');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (3,'Nueva Córdoba');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (4,'Centro');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (5,'Los Boulevares');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (6,'Guemes');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (7,'Ipona');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (8,'Alta Córdoba');
INSERT INTO barrios (id_barrio,n_barrio) VALUES (9,'General Paz');

INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (1,'Parcial 1');

```

```
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (2,'Parcial 2');
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (3,'Final');
INSERT INTO tipo_notas (id_tipo_nota,n_tipo_nota) VALUES (4,'Trabajo Practico');

INSERT INTO tipo_documentos (id_tipo_documento,n_tipo_documento) VALUES (1,'DNI');
INSERT INTO tipo_documentos (id_tipo_documento,n_tipo_documento) VALUES (2,'LC');

INSERT INTO carreras (id_carrera,nombre_carrera,titulo) VALUES (1,'Ingeniería en Sistemas de Informacion','Ingeniero en Sistemas de Informacion');

INSERT INTO sexos (id_sexo,n_sexo) VALUES (1,'Masculino');
INSERT INTO sexos (id_sexo,n_sexo) VALUES (2,'Femenino');

INSERT INTO estados_cursados (id_estado,n_estado) VALUES (1,'Inscripto');
INSERT INTO estados_cursados (id_estado,n_estado) VALUES (2,'Regular');
INSERT INTO estados_cursados (id_estado,n_estado) VALUES (3,'Libre');

INSERT INTO materias (id_materia,nombre) VALUES (1,'Algoritmos y Estructuras de Datos');
INSERT INTO materias (id_materia,nombre) VALUES (2,'Analisis Matematico I');
INSERT INTO materias (id_materia,nombre) VALUES (3,'Analisis Matematico II');
INSERT INTO materias (id_materia,nombre) VALUES (4,'Algebra y Geometria Analitica');
INSERT INTO materias (id_materia,nombre) VALUES (5,'Matematica Discreta');
INSERT INTO materias (id_materia,nombre) VALUES (6,'Sistemas y Organizaciones');
INSERT INTO materias (id_materia,nombre) VALUES (7,'Ingenieria y Sociedad');
INSERT INTO materias (id_materia,nombre) VALUES (8,'Paradigmas de Programacion');

INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (1,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (2,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (3,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (4,1,1,1);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (5,1,1,2);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (6,1,1,2);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (7,1,1,2);
INSERT INTO plan_de_carrera (id_materia,id_carrera,anio_cursado,cuatrimestre_anio) VALUES (8,1,1,2);

INSERT INTO estudiantes (id_estudiante,apellido,nombres,id_tipo_documento,documento,id_sexo,calle,calle_numero,id_barrio) VALUES (1,'Abrutsky','Maximiliano Adrián',1,23852963,1,'Av. Colón','1035',1);
INSERT INTO estudiantes (id_estudiante,apellido,nombres,id_tipo_documento,documento,id_sexo,calle,calle_numero,id_barrio) VALUES (2,'DAMIANO','Luis',1,19147258,1,'Dean Funes','903',1);

INSERT INTO inscripciones_a_carreras (id_estudiante,id_carrera,fecha) VALUES (1,1,CAST('2009-01-26' AS DATETIME));
```

```

INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_incripcion,id_estado,fecha_estado) VALUES (1,1,1,CAST('2009-02-01' AS DATETIME),1,CAST('2009-02-01' AS DATETIME));
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_incripcion,id_estado,fecha_estado) VALUES (2,1,1,CAST('2009-02-01' AS DATETIME),1,CAST('2009-02-01' AS DATETIME));
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_incripcion,id_estado,fecha_estado) VALUES (3,1,1,CAST('2009-02-01' AS DATETIME),1,CAST('2009-02-01' AS DATETIME));
INSERT INTO alumnos_cursando_materias (id_materia,id_carrera,id_estudiante,fecha_incripcion,id_estado,fecha_estado) VALUES (4,1,1,CAST('2009-02-01' AS DATETIME),1,CAST('2009-02-01' AS DATETIME));

INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_incripcion,fecha_nota,id_tipo_nota/nota) VALUES (1,1,1,CAST('2009-02-01' AS DATETIME),CAST('2009-03-15' AS DATETIME),1,5);
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_incripcion,fecha_nota,id_tipo_nota/nota) VALUES (1,1,1,CAST('2009-02-01' AS DATETIME),CAST('2009-05-26' AS DATETIME),2,7);
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_incripcion,fecha_nota,id_tipo_nota/nota) VALUES (2,1,1,CAST('2009-02-01' AS DATETIME),CAST('2009-03-18' AS DATETIME),1,3);
INSERT INTO alumnos_cursando_notas (id_materia,id_carrera,id_estudiante,fecha_incripcion,fecha_nota,id_tipo_nota/nota) VALUES (2,1,1,CAST('2009-02-01' AS DATETIME),CAST('2009-05-29' AS DATETIME),2,4);

```

#### 12.4.2.5 Función escalar definida por el usuario

La utilidad de esta función ilustrativa llamada "obtener\_Nombre\_Completo()" es recibir, por parámetro, el id\_estudiante y devolver el nombre completo del estudiante con el formato 'Apellido, nombres'. Solamente la primera letra del apellido será mayúscula y todas las del/los nombre/s en minúscula.

Se comenta la sentencia de creación para facilitar la comprensión. Asimismo, el funcionamiento consiste en declarar una variable, asignarle como valor la primera letra del apellido en mayúsculas, concatenándole el resto de las letras en minúsculas, una coma con espacio en blanco y, finalmente, el o los nombres, todo en minúsculas. Se concluye retornando dicha variable como resultado.

```
DELIMITER $$
```

```
CREATE FUNCTION obtener_Nombre_Completo(p_id_estudiante INT) -- cabecera de la función (nombre y parámetros)
```

```
RETURNS VARCHAR(150) -- tipo de valor que retorna
```

```
BEGIN
```

```
    DECLARE v_nombreCompleto VARCHAR(150); -- definición de variable que será retornada
```

```
    -- seteo de valor de la variable. UPPER y LOWER convierten a mayús. o minús. respectivamente. LEFT y RIGHT recortan la cadena retornando el lado izquierdo . derecho respectivamente. CHAR_LENGTH devuelve el largo de la cadena, y CONCAT concatena las cadenas de caracteres recibidas por parámetro.
```

```
    SELECT CONCAT(UPPER(LEFT(apellido, 1))
```

```
        , LOWER(RIGHT(apellido, CHAR_LENGTH(apellido)-1))
```

```
        , ','
```

```
, LOWER(nombres))
INTO v_nombreCompleto
FROM estudiantes
WHERE id_estudiante = p_id_estudiante;

RETURN v_nombreCompleto; – retorno del valor
END$$$$
```

Esta función, como cualquier otra que también sea escalar, retornará solo un valor, por lo que se pueden invocar cuando se utilizan expresiones escalares (en el SELECT, WHERE, SET, etc...).

#### Ejemplos

Invocación:

```
SELECT id_estudiante, apellido, nombres, obtener_Nombre_Completo(id_estudiante) as NombreCompleto
FROM estudiantes ;
```

Resultado:

|   | id_estudiante | apellido | nombres            | NombreCompleto               |
|---|---------------|----------|--------------------|------------------------------|
| 1 |               | Abrutsky | Maximiliano Adrián | Abrutsky, maximiliano adrián |
| 2 |               | DAMIANO  | Luis               | Damiano, luis                |

Invocación:

```
SELECT obtener_Nombre_Completo(2) as NombreCompleto ;
```

Resultado:

```
NombreCompleto
```

```
Damiano, luis
```

#### 12.4.2.6 Procedimiento almacenado definido por el usuario

La funcionalidad de este procedimiento consistirá en que recibirá, por parámetro, un estudiante, la materia y la carrera (id\_estudiante, id\_materia e id\_carrera) y la actualización del estado de cursada (tabla alumnos\_cursando\_materias) de dicha materia-carrera; por ejemplo, de inscripto a regular o libre de acuerdo con si tiene, o no, al menos dos parciales aprobados (tabla alumnos\_cursando\_notas).

Consideraciones:

- Un alumno puede recursar una materia, por lo que se toma la última inscripción.
- La nota de aprobación es mayor o igual a cuatro.

- Al menos debe tener dos parciales rendidos para que se actualice el estado.
- Si se tiene al menos dos parciales aprobados, se lo considera regular.
- Menos de dos parciales aprobados, se lo considera libre.
- Por sencillez, no se consideran los recuperatorios en este modelo.

```
DELIMITER $$
```

```

CREATE PROCEDURE actualizar_estado_materia (p_id_estudiante INT, p_id_materia INT, p_id_carrera INT)
BEGIN
-- Declaración de variables
DECLARE v_fecha_inscripcion_cursado DATETIME;
DECLARE v_cant_parciales INT;
DECLARE v_cant_parciales_aprobados INT;
DECLARE v_cant_parciales_reprobados INT;

-- Obtención de la última inscripción de la materia (podría estar recursando y solo se procesa la última inscripción)
SELECT MAX(fecha_inscripcion)
INTO v_fecha_inscripcion_cursado
FROM alumnos_cursando_materias
WHERE id_materia = p_id_materia
AND id_carrera = p_id_carrera
AND id_estudiante = p_id_estudiante;

-- Si no se obtiene ninguna fecha, significa que el alumno nunca se inscribió a la materia indicada
IF v_fecha_inscripcion_cursado IS NULL THEN
    SELECT 'El alumno no se ha inscripto a la materia - carrera especificada';
ELSE
    -- cantidad de parciales rendidos (notas cuya descripción contengan 'parcial' sin importar mayúsculas y minúsculas)
    SELECT COUNT(*)
    INTO v_cant_parciales
    FROM alumnos_cursando_notas notas
    INNER JOIN materias mat ON notas.id_materia = mat.id_materia
    INNER JOIN tipo_notas tip ON notas.id_tipo_nota = tip.id_tipo_nota
    WHERE notas.id_estudiante = p_id_estudiante
    AND notas.id_materia = p_id_materia
    AND notas.id_carrera = p_id_carrera
    AND notas.fecha_inscripcion = v_fecha_inscripcion_cursado
    AND LOWER(tip.n_tipo_nota) LIKE '%parcial%';

```

```
-- cantidad de parciales aprobados (rendidos con nota mayor o igual a 4)
SELECT COUNT(*)
INTO v_cant_parciales_aprobados
FROM alumnos_cursando_notas notas
INNER JOIN materias mat ON notas.id_materia = mat.id_materia
INNER JOIN tipo_notas tip ON notas.id_tipo_nota = tip.id_tipo_nota
WHERE notas.id_estudiante = p_id_estudiante
AND notas.id_materia = p_id_materia
AND notas.id_carrera = p_id_carrera
AND notas.fecha_incripcion = v_fecha_incripcion_cursado
AND LOWER(tip.n_tipo_nota) LIKE '%parcial%'
AND notas.nota >= 4;

-- a modo informativo se calculan los reprobados (parcial que no se aprobó, se reprobó)
SET v_cant_parciales_reprobados = v_cant_parciales - v_cant_parciales_aprobados;

SELECT CONCAT('Cantidad de parciales rendidos = ', CAST(v_cant_parciales AS CHAR));
SELECT CONCAT('Cantidad de parciales aprobados = ', CAST(v_cant_parciales_aprobados AS CHAR));
SELECT CONCAT('Cantidad de parciales reprobados = ', CAST(v_cant_parciales_reprobados AS CHAR));

IF v_cant_parciales < 2 THEN
    SELECT 'La cantidad rendida de parciales es menor a 2, no se cambia el estado';
ELSE
    IF v_cant_parciales_aprobados >= 2 THEN
        SELECT '2 parciales aprobados, REGULAR';
        -- actualización al estado REGULAR
        UPDATE alumnos_cursando_materias
        SET id_estado = (SELECT id_estado FROM estados_cursados WHERE n_estado LIKE 'Regular')
        WHERE id_estudiante = p_id_estudiante
        AND id_materia = p_id_materia
        AND id_carrera = p_id_carrera
        AND fecha_incripcion = v_fecha_incripcion_cursado;
    ELSE
        SELECT 'Menos de 2 parciales aprobados, LIBRE';
        -- actualización al estado LIBRE
        UPDATE alumnos_cursando_materias
        SET id_estado = (SELECT id_estado FROM estados_cursados WHERE n_estado LIKE 'Libre')
        WHERE id_estudiante = p_id_estudiante
```

```
        AND      id_materia = p_id_materia
        AND  id_carrera = p_id_carrera
        AND  fecha_inscripcion = v_fecha_inscripcion_cursado;
    END IF;
END IF;
END$$
```

Ejemplos:

Invocación:

```
CALL actualizar_estado_materia (1,8,1)
```

Resultado:

El alumno no se ha inscripto a la materia - carrera especificada

Invocación:

```
CALL actualizar_estado_materia (1,2,1)
```

Resultado:

Cantidad de parciales rendidos = 2

Cantidad de parciales aprobados = 1

Cantidad de parciales reprobados = 1

Menos de 2 parciales aprobados, LIBRE

Invocación:

```
CALL actualizar_estado_materia (1,1,1)
```

Resultado:

Cantidad de parciales rendidos = 2

Cantidad de parciales aprobados = 2

Cantidad de parciales reprobados = 0

2 parciales aprobados, REGULAR

## 12.5 Contenido de la página Web de apoyo

El material marcado con asterisco (\*) solo está disponible para docentes.



### 12.5.1 Mapa conceptual del capítulo

### 12.5.2 Autoevaluación

### 12.5.3 Presentaciones\*

## Bibliografía

### Libros

bertino, e.; martino, l. Sistemas de bases de datos orientadas a objetos: Conceptos y arquitecturas, Ediciones Díaz de Santos, 1995.

codd, e. f. a Relational Model for Database Management: Version 2, Addison Wesley, 1990.

date, c. j. Introducción a los sistemas de base de datos, Pearson Education, 2001.

de miguel, a; piattini, m. Fundamentos y Modelos de Bases de Datos, México DF: Alfaomega, RA-MA, 1999.

elmasri, r.; navathe, s. b. Fundamentos de sistemas de bases de datos, México DF: Addison Wesley, 2000.

martin, j. Organización de las bases de datos, México DF: Prentice-Hall, 1977.

silberschatz, a.; korth, h.; sudarsha, s. Fundamentos de bases de datos, McGraw-Hill, 2002.

stocker, p. m.; gray, p. m. d.; atkinson, m. p. Databases, role and structure: an advanced course, Cambridge University Press, 1984.

### Artículos en línea

codd, e. f. "A Relational Model of Data for Large Shared Data Banks" en Communications of the ACM [en línea], junio de 1970, N.<sup>o</sup> 13, [consultado diciembre de 2009]. Disponible en: <http://www.cs.nott.ac.uk/~nza/G51DBS/codd.pdf>

codd, e. f. "Dr. E. F. Codd's 12 rules for defining a fully relational database" [en línea], s/f, [consultado diciembre de 2009]. Disponible en: <http://www.state.edu/~sgomori/570/coddsrules.html>

date, c. j. "Edgar F. Codd 08/23/1923 – 04/18/2003 A TRIBUTE" [en línea], 23/05/2003, [consultado enero de 2010]. Disponible en: <http://www.dbdebunk.com/page/page/621965.htm>

lorentz, d. "Oracle Database SQL Reference, 10g Rel 2 (10.2)" [en línea], junio de 2005, [consultado enero de 2011]. Disponible en: <http://www.tahiti.oracle.com>

lorentz, d. "SQL Reference Release 3 (8.1.7)" [en línea], septiembre de 2000, [consultado enero de 2011]. Disponible en: [http://www.docs.oracle.com/html/A85397\\_01/title.htm](http://www.docs.oracle.com/html/A85397_01/title.htm)

marqués, m. "Bases de datos orientadas a objetos" [en línea], 12/04/2002, [consultado febrero de 2010]. Disponible en: <http://www3.uji.es/~mmarques/e16-teoria/cap2.pdf>

oracle "Manual oficial Oracle Concepts" [en línea], s/f, [consultado agosto de 2009]. Disponible en: <http://www.adp-gmbh.ch/ora/concepts/transaction.html>

quiroz, j. "El modelo relacional de bases de datos" en Boletín de Política Informática [en línea], 2003, N.<sup>o</sup> 6, [consultado enero de 2010]. Disponible en: [http://www.inegi.gob.mx/prod\\_serv/contenidos/espanol/bvinegi/productos/integracion/pais/informatica/boletin/2003/num6.pdf](http://www.inegi.gob.mx/prod_serv/contenidos/espanol/bvinegi/productos/integracion/pais/informatica/boletin/2003/num6.pdf)

yager, r.; reese, g.; king, t. "SQL According to MySQL and mSQL" [en línea], julio de 1999, [consultado enero de 2010]. Disponible en: [http://www.docstore.mik.ua/oreilly/linux/sql/ch06\\_01.htm](http://www.docstore.mik.ua/oreilly/linux/sql/ch06_01.htm)



## Índice analítico

- %ROWTYPE 137
- anidamiento de funciones de grupo 117
- bloques anónimos 134, 135, 151
- cerramiento o bloqueos 140
- cláusulas de SELECT 112
- COMMENT 106
- CURSOR\_ALREADY\_OPEN EXCEPTION 150
- DB2
  - ajustes de sintaxis en DDL 304
  - centro de control 299
  - comunidad 298
  - definición de tareas de mantenimiento 302
  - Express-C, funcionalidades 298
  - modelo de datos Estudiante-Universidad DDL 306
  - modelo de datos Estudiante-Universidad DML 308
  - referencia a la documentación en línea 299
  - XML 298
- Descarga
  - MySQL Workbench 342
  - MySQL 336
  - SQL Server Management Studio 317
  - SQL Server 312
- DUP\_VAL\_ON\_INDEX EXCEPTION 150
- ELSIF 141
  - encapsulamiento 155
  - exactitud del tipo de dato number 113
  - excepciones 136
  - extensión procedural 132
- FETCH 147
- GROUP BY 116
- HINTS 129
  - historia del lenguaje 104
  - índices 110
- Instalación
  - MySQL Workbench 342
  - MySQL 337
  - SQL Server Management Studio 317
  - SQL Server 312
- INVALID\_CURSOR EXCEPTION 150
- MySQL Workbench
  - Conexión 344
  - creación de base de datos 345
  - utilización 343
- MySQL
  - ejemplo de función escalar en lenguaje procedural 354
  - ejemplo de Procedimiento almacenado en lenguaje procedural 355
  - modelo de datos Estudiante-Universidad 343
  - modo básico instalación 339
- sentencias DDL del ejemplo 347
- sentencias DML del ejemplo 352
- servidor lógico 338
- no procedural 105
- NO\_DATA\_FOUND EXCEPTION 150
- optimizador 112
  - basado en costos 127
  - basado en reglas 127
- Oracle
  - construcciones en PL/SQL 292
  - creación de usuario dueño de tablas 283
  - listener 280
  - modelo de datos Estudiante-Universidad DDL 286
  - modelo de datos Estudiante-Universidad DML 290
  - puerto de conexión 1521 280
  - SQL\*plus 295
  - usuarios de administración 279
- paquetes de optimización de performance 128
- PL/SQL
  - anomalías 139
  - cómo se administra 133
  - nivel de aislamiento 139
- RECORD 137
- rol 104
  - secciones de un bloque 134
- SQL Server Management Studio
  - creación de base de datos 319
  - explorador de objetos 317
  - modelo de datos Estudiante-Universidad 318
  - uso 317
  - ventana de documentos 317
- SQL Server
  - autenticación 314
  - ejemplo de función T-SQL 329
  - ejemplo de procedimiento T-SQL 331
  - initialización 316
  - instancia 313
  - intercalación 315
  - sentencias DDL del ejemplo 322
  - sentencias DML del ejemplo 327
- SQL
  - anomalías 111
  - nivel de aislamiento 111
- table 137
- TOO\_MANY\_ROWS EXCEPTION 150
- transaccional 132
- TRUNCATE 106
  - ventaja del uso 132
- ZERO\_DIVIDE EXCEPTION 150





# BASES DE DATOS

Esta obra desarrolla el concepto y la manipulación de un Motor de Base de Datos, la manera de interactuar con él a través de los diferentes lenguajes asociados y modelos de construcción lógica de datos.

Comienza en el Capítulo 1 con la presentación de las nociones generales de las bases de datos y sus Sistemas de Gestión. El Capítulo 2 describe el modelo relacional e incluye las relaciones entre tablas, claves, integración, etcétera. En el Capítulo 3, se detallan los comandos y utilización del lenguaje SQL. El Capítulo 4 agrega una extensión de SQL a los lenguajes procedimentales.

En el Capítulo 5, se tratan las características de las bases de datos multidimensionales, sus diferencias con las Bases de Datos Relacionales y las variantes de modelado designadas a las tecnologías OLAP. Los Capítulos 6 y 7 presentan una visión general del almacenamiento y la minería de datos. El Capítulo 8 introduce las características principales, las ventajas y las desventajas de las Bases de Datos Orientadas a Objetos.

La obra concluye con un modelo de datos, en los Capítulos 9, 10, 11 y 12 que se implementa en Oracle Express Edition XE, IBM DB2, SQL Server 2005 y MySQL 5.1 que permite a los estudiantes trabajar sobre la práctica real.

**Enrique José Reinoso:** Ingeniero en Sistemas de Información. Analista Universitario de Sistemas. Posgrado en Ingeniería Gerencial. Docente.

**Calixto Alejandro Maldonado:** Ingeniero en Sistemas de Información. Doctorando de Ingeniería en software basado en componentes reutilizables. Docente.

**Roberto Muñoz:** Ingeniero en Sistemas de Información. Maestrando en Ingeniería en Sistemas de Información. Docente.

**Luis Esteban Damiano:** Analista de Sistemas de Computación. Licenciado en Tecnología Educativa. Docente.

**Maximiliano Adrián Abrutsky:** Ingeniero en Sistemas de Información. Docente.

[www.alfaomega.com.mx](http://www.alfaomega.com.mx)



"Te acerca al conocimiento"

 **Alfaomega Grupo Editor**